Model Based Refinement and the Design of Retrenchments

Richard Banach
School of Computer Science, University of Manchester,
Manchester, M13 9PL, UK
Email: banach@cs.man.ac.uk

Abstract. ***The ingredients of typical methodologies for model based development via refinement are re-examined, and some well-known frameworks are reviewed, drawing out commonalities and differences. It is observed that the ingredients of these formalisms can frequently be 'mixed and matched' much more freely than is often imagined, resulting in semantic variations on the original formulations. It is also noted that similar alterations in the semantics of specific formalisms have taken place de facto due to applications pressures and for other reasons. This analysis suggests prioritising some criteria and proof obligations over others within this family of methods. These insights are used to construct a foundation for the design of notions of retrenchment appropriate for, and complementary to, given notions of refinement. The notions of retrenchment thus derived for the specific refinement formalisms examined earlier, namely Z, B, Event-B, ASM, VDM, RAISE, IO-automata and TLA+, are presented, and within the criteria given, all turn out to be very similar.***

Keywords: *Model Based Development, Refinement, Retrenchment, Z, B, Event-B, ASM, VDM, RAISE, IO-automata, TLA+.*

* Corresponding Author:
Richard Banach,
School of Computer Science, University of Manchester,
Manchester, M13 9PL, UK,
Email: banach@cs.man.ac.uk

1. Introduction

Refinement, as a model based methodology for developing systems from abstract specifications, has been around for a long time [1]. In this period, many variations on the basic idea have arisen, to the extent that an initiate can be bewildered by the apparently huge choice available. As well as mainstream refinement methodologies such as ASM, B, Z, VDM, RAISE, IO-automata, TLA+ etc., which have enjoyed significant applications use, there are a myriad of other related theories in the literature, too numerous to cite comprehensively. And at a detailed theoretical level, they are all slightly different.

From a developer's point of view, this variety can only be detrimental to the wider adoption of formal techniques in the real world applications arena — in the real world, developers have a host of things to worry about, quite removed from evaluating the detailed technical differences between diverse formal techniques in order to make the best choice regarding which one to use. In any event, such choice is often made on quite pragmatic grounds, such as the ready access to one or more experts and, crucially these days, availability of appropriate tool support. Anecdotally, the choice of one or another formalism appears to make little difference to the outcome of a real world project using such techniques — success seems to be much more connected with proper requirements capture, and with organising the development task in a way that is sympathetic to both the formal technique and to the developers' pre-existing development practices.

This is not to say that the designs of different refinement notions are poorly motivated. The description of a typical refinement notion is justified by a number of external goals, from which starting point the technical details of the particular notion follow quite logically. However, it is

frequently the case that the considerations in play here are somewhat orthogonal to the principal concerns of practical developers.

In this paper we examine what goes into a typical notion of model based refinement by examining a number of cases. Such an analysis has a number of benefits. For one thing, it can contribute towards an informed point of view regarding detailed differences between techniques, and how different techniques might to relate to one another, especially now that verification techniques and their tools can increasingly address mainstream industrial scale problems. This can guide the management of relationships between techniques going forwards into the future, and specifically can inform a perspective on how tools for different techniques might relate to one another, an issue relevant to the (currently active) Verification Grand Challenge [2-4]. This line has been briefly explored in [5].

For another thing, understanding the relationships between different refinement techniques can help guide the design of retrenchment notions [6-7] corresponding to different variants of refinement. Unlike refinement, which is designed to deliver guarantees about the relationship between abstract and concrete systems provided appropriate conditions are met, retrenchment, a weakening or liberalisation of model based refinement, is designed to offer maximal expressive flexibility when refinement is confronted with issues that do not fit within its constraints. Thus, designing a notion of retrenchment is not like designing a notion of refinement: rather than choosing a number of external goals that constitute a notion of correctness and then deriving the technical details of the refinement, retrenchment has to consider how issues that break strict conformity with such correctness criteria may fruitfully be described. In this context, considering a number of refinement variants together helps to highlight how each refinement notion ought to relate to its corresponding retrenchment notion. The latter is in turn crucial when we realise that to gain the maximum benefit from the two techniques, a close interplay between them is vital.

It is remarkable that, in large part, both of these things are supported by essentially the same body of evidence (at least if one approaches them in the way it is done in this paper). The things that diverse model based refinement notions have in common, and that thus provide a potential focus for convenient interworking between them at the operational level of tools and the like, also provide a focus for the design of retrenchment notions, since it is the common things that survive when one seeks to 'liberate' refinement, as one does with retrenchment. It is these implications for retrenchment that this paper explores in detail.

The rest of the paper is as follows. In Section 2 we discuss a number of features commonly found in model based formalisms. Our approach is very much structured round features that turn out to be of interest later, rather than the way one would evaluate refinement notions *per se*. Thus for instance, while a traditional discussion of notions of correctness in the context of refinement would focus on issues such as termination, and partial or total correctness, our discussion omits these since they play no role in retrenchment — rather, what we call 'notions of correctness' deals with some lower level issues that emerge from higher level considerations. Another typical key issue for refinement notions, which is connected with the preceding one, is the problem of soundness and completeness of a given collection of proof obligations with respect to a previously stated notion of partial or total correctness. Again, since these issues have no counterpart for retrenchment, there is no discussion of them below. Similar remarks apply to safety and liveness. Continuing this line of thought, in a traditional evaluation of refinement notions, one might well critically examine the external motivations for setting up a refinement notion in the way it was done, and compare the way that similar issues were addressed in different formalisms, or, taking into account that different refinement notions were conceived to address widely differing situations, why certain features are present in some notions but not others. But such considerations once more serve no purpose for us, so we do not dwell on them — it is sufficient for us to note the variety that we see.

In Section 3 we show how our generalities are reflected in a number of specific well known approaches, namely ASM, B, Event-B, Z, VDM, RAISE, IO-automata, TLA+. Considering that in many of these cases the same formal framework is supported by more than one detailed theory, we start the discussion of each approach by citing one or more standard references which sets the context for the remainder of the discussion. Although we discuss eight approaches, we make no claim of completeness of coverage, given the large number of such formalisms that are to be found in the literature. Our aim is to exhibit a variety of features and the way that they compare and contrast, rather than to give a fully comprehensive account of each approach. A consequence of this is that when an approach contains one or more features sufficiently similar to ones discussed already, we tend to be

brief in the extreme. Section 4 reflects on the evidence accumulated from all this, and draws some appropriate conclusions. Section 5 takes these points forward and shows how the insights gained should inform the design of a notion of retrenchment to complement a given notion of refinement. A strong element of this process is compatibility via the retrenchment *Tower Pattern*, discussed in Section 5.2. Section 6 then considers how the preceding can be instantiated in the context of the eight formalisms examined earlier, showing a large degree of commonality in the resulting retrenchment designs. The general analysis of refinement and retrenchment against broad criteria in Sections 3-6, and the detailed definitions of retrenchment in the specific methodologies discussed in this paper, constitute the main contributions of the present work. Section 7 concludes.

2. Model Based Refinement Methods: Generalities

A typical model based formal refinement method, whose aim is to formalize how an abstract model may be refined to a more concrete one, consists of a number of elements which interact in ways which are sometimes subtle. In this section we bring some of these facets into the light; the discussion may be compared to a similar one in [8].

Formal language. All formal refinement techniques need to be quite specific about the language in which the elements of the technique are formalised. This precision is needed for proper theoretical reasoning, and to enable mechanical tools with well-defined behaviour to be created for carrying out activities associated with the method. There are inevitably predicates of one kind or another to describe the properties of the abstract and concrete models, but technical means are also needed to express state change within the technique. Compared with the predicates used for properties, there is much more variety in the linguistic means used for expressing state change, although each has a firm connection with the predicates used for the modelling of properties.

Granularity and naming. All formal refinement techniques depend on relating concrete steps (or collections of steps) to the abstract steps (or collections of steps) which they refine. Very often, a single concrete step is made to correspond to a single abstract one, but occasionally more general schemes (in which sequences of abstract and concrete steps figure) are considered. The (1,1) scheme is certainly convenient to deal with theoretically, and it is often captured by demanding that the names of operations or steps that are intended to correspond at abstract and concrete levels are the same. However, in many applications contexts, such a simple naming scheme is far removed from reality, and if naively hardwired into the structure of a tool, makes the tool much less conveniently usable in practice.

Concrete-abstract fidelity. All formal refinement techniques demand that the concrete steps relate in a suitable manner to abstract ones. Almost universally, a retrieve relation (also referred to as a refinement mapping, abstraction relation, gluing relation, etc.) is used to express this relationship. It is demanded that the retrieve relation holds between the before-states of a concrete step (or sequence of steps) and the abstract step (or sequence of steps) which simulates it; likewise it must hold for the after-states of the simulating pair. In other words (sequences of) concrete steps must be faithful to (sequences of) abstract steps. (A special case, simple refinement, arises when the retrieve relation is an identity.) What we call concrete-abstract fidelity in this paper is more conventionally referred to as the forward simulation property for refinement notions. However, since in specific contexts this is often accompanied by other detailed criteria, we prefer a more neutral phrase here.

Concrete-abstract fidelity is the one feature that can be found in essentially the same form across the whole family of model based formalisms. It is also the case that this fidelity —usually expressed using a proof obligation (PO), the *fidelity PO*— is often derived as a *sufficient condition* for a more abstract formulation of refinement, concerning the overall behaviour of 'whole programs'. These sufficient conditions normally form the focus of the theory of model based refinement techniques,[1] since they offer what is usually the only route to proving refinement in practical cases.

[1] They are a key ingredient of the usual soundness and completeness arguments for refinement theories.

Notions of correctness. One of the responsibilities of a formal refinement technique is to dictate *when* there should be concrete steps that correspond to the existence of abstract ones. This (at least implicitly) is connected with the potential for refinement techniques to be used in a black-box manner. Thus if an abstract model has been drawn up which deals adequately with the requirements of the problem, then any refinement should *guarantee* that the behaviour expressed in the abstract model should be reflected appropriately in more concrete models, and ultimately in the implementation, so that the requirements coverage persists through to code. Note that this is again a much more narrowly drawn focus for the phrase 'notion of correctness' than is usual.

There is much variation among refinement techniques on how this is handled, particularly when we take matters of interpretation into account. Although the mainstream techniques we discuss below are reasonably consistent about the issue, some variation is to be found, and more variety can be found among refinement variants in the literature. The formal content of these perspectives gets captured in suitable POs, and often, the policy adopted has some impact on the fidelity PO too. A similar impact can be felt in initialisation (and finalisation) POs.

Interpretation. The preceding referred (rather obliquely perhaps) to elements of model based refinement theories that are expressed in the POs of the theory, i.e. *via logic*. However, this does not determine how the logical elements relate to phenomena in the real world. If transitions are to be described by logical formulae (involving before and after states, say), then those formulae can potentially take the value *false* as well as *true*. And while determining how the logical formulae correspond to the real world is usually fairly straightforward in the *true* case, determining the correspondence in the *false* case can be more subtle. These matters of logical-to-real-world correspondence constitute what we call here the *interpretation* aspects of a formal development technique.

Trace inclusion. Trace inclusion, i.e. the criterion that every execution sequence of the system (i.e. the concrete model) is as permitted by the specification (i.e. the abstract model), is of immense importance in the real world. When an implemented system behaves unexpectedly, the principal *post hoc* method of investigation amounts to determining how the preceding behaviour failed to satisfy the trace inclusion criterion. This importance is further underlined by the role that trace inclusion plays in model checking. The 'whole program' starting point of the derivation of many sufficient conditions for refinement is also rooted in trace inclusion. Two forms of trace inclusion are of interest. *Weak trace inclusion* merely states that for every concrete trace there is a simulating abstract one. *Strong trace inclusion* goes beyond that and states that if *Asteps* simulates *Csteps* and we extend *Csteps* to *Csteps* **;** C_{nxt}, then *Asteps* can be extended to *Asteps* **;** A_{nxt} which also simulates. With weak trace inclusion, we might have to abandon *Asteps* and find some unrelated $Asteps_{different}$ to recover simulation of *Csteps* **;** C_{nxt}. Given the crucial role of trace inclusion, it is perhaps surprising that in many cases, the POs derived for refinement based formalisms do not guarantee trace inclusion without further assumptions.

Composition. It is a given that large systems are built up out of smaller components, so the interaction of this aspect with the details of a development methodology are of some interest, at least for practical applications. Even more so than for notions of correctness, there is considerable variation among refinement techniques on how compositionality is handled — the small number of techniques we review in more detail below already exhibit quite a diversity of approaches to the issue.

3. Some Well-Known Refinement Formalisms

In this section, we review how the various elements of model based refinement methodologies outlined above are reflected in a number of specific and well-known formalisms. We look at Z, B, Event-B, ASM, VDM, RAISE, IO-automata and TLA+. For simplicity, brevity and relevance to retrenchment below, we stick to a forward simulation perspective throughout.

3.1. Z

Since Z itself [9] is simply a formal mathematical language, one cannot speak definitively of *the* Z refinement. We target our remarks on the formulations in [10-11].

Formal language: Z uses the well-known schema calculus, in which a schema consists of named and typed components which are constrained by a formula built up using the usual logical primitives. This is an all-purpose machinery; 'delta' schemas enable before-after relations that specify transitions to be defined; other schemas define retrieve relations, etc. The schema calculus itself enables schemas to be combined so as to express statements such as the POs of a given refinement theory.

Granularity and naming: Most of the refinement formulations in [10-11] stick to a (1,1) framework. Purely theoretical discussions often strengthen this to identity on 'indexes' (i.e. names) of operations at abstract and concrete levels, though there is no insistence on such a close tieup in [12-13].

Concrete-abstract fidelity: In the above context for Z refinement, the fidelity PO comes out as follows, which refers to the contract interpretation without I/O (while the behavioural interpretation drops the 'pre *AOp*'):

$$(\forall\, AState(u) \;;\; CState(v) \;;\; CState'(v') \bullet \text{pre}\ AOp(u) \wedge R(u,v) \wedge COp(v,v') \\ \Rightarrow (\exists\, AState'(u') \bullet AOp(u,u') \wedge R'(u',v')))$$

where $AState(u)$, $CState(v)$ are (abstract and concrete) state schemas in state variables u, v respectively (primes denote after-states), $AOp(u,u')$, $COp(v,v')$ are corresponding operations, $R(u,v)$ is the retrieve relation, and 'pre $AOp(u)$', the precondition, in fact denotes the domain of $AOp(u,u')$.

Notions of correctness: In Z, an induction on execution steps is used in the (1,1) framework to derive trace inclusion. To work smoothly, totality (on the state space) of the relations expressing operations is assumed. To cope with partial operations, a $\perp$ element is added to the state space, and *totalisations* of one kind or another, of the relations representing the operations, are applied. The consequences of totalisation (such as the correctness condition above), got by eliminating mention of the added parts from a standard forward simulation implication, constitute the POs of, and embody the notion of correctness for, the totalisation technique under consideration. These turn out to be the same for both contract and behavioural approaches, aside from the difference noted above.

Interpretation: The two main totalisations used, express the *contract* and the *behavioural* interpretations. In the former, an operation may be invoked at any time, and *the consequences of calling it outside its precondition are unpredictable* (within the limits of the model of the syntax being used), including $\perp$, nontermination. In the latter, $\perp$ is guaranteed outside the precondition (usually called the guard in this context, but still defined as the domain of the relevant partial relation), which is typically interpreted by saying the operation *will not execute* if the guard is false.

Trace inclusion: Trace inclusion has been cited as the underlying derivation technique for the POs, and since an inductive approach is used, it is strong trace inclusion. However, the 'fictitious' transitions of operations introduced by totalisation are treated on an equal footing to the original 'honest' ones, so many spurious traces, not corresponding to real world behaviour, can be generated. For instance a simulation of a concrete trace may hit a state (whether abstract or concrete) that is outside the 'natural' domain of the *next* partial operation. Then, in the contract interpretation, the trace can continue in a very unrestricted manner, despite the different way that one would view the constituent steps from a real world perspective. Things look a bit better in the behavioural interpretation, since such a trace is thereafter confined to $\perp$.

Composition: One prominent composition mechanism to be found in Z is *promotion*. In promotion, a component which is specified in a self-contained way is replicated via an indexing function to form a family inside a larger system; this interacts cleanly with refinement [10-11]. However, the schema calculus in general is not monotonic with respect to refinement without additional caveats [14].

3.2. B

The original B Method was described with great clarity in [15], and there are a number of textbook treatments e.g. [16-18].

Formal language: Original B was based on predicates for subsets of states, written in a conventional first order language, and on weakest precondition predicate transformers (wppts) for the operations. The use of predicate transformers obviates the need for explicitly adjoining $\perp$ elements to the state spaces.

Granularity and naming: Original B adheres to a strict (1,1) framework; 'strict' in the sense that tools for original B demand identical names for operations and their refinements. Abstract models of complex operations can be assembled out of smaller pieces using such mechanisms as INCLUDES, USES, SEES. However once the complete abstract model has been assembled, refinement proceeds monolithically towards code. The last step of refinement to code, is accomplished by a code generator which plugs together suitably designed modules that implement the lowest level B constructs.

Concrete-abstract fidelity: This is handled via the predicate transformers. Adapting the notation of [15] for ease of comparison with Z, the relevant PO can be written:

$$AInv(u) \wedge CInv(u,v) \wedge \text{trm}\, AOp(u) \\ \Rightarrow [\, COp(v,v') \,] \neg [\, AOp(u,u') \,] \neg CInv(u',v')$$

In this, $AInv(u)$ and trm $AOp(u)$ are the abstract invariant and termination condition (the latter being the predicate of the precondition), while $CInv(u,v)$ is the concrete invariant, which in original B, involves both abstract and concrete variables and thus acts also as a retrieve relation; all of these are predicates. $[\, AOp(u,u') \,]$ and $[\, COp(v,v') \,]$ are the wppts for the abstract and concrete operations, so the equation says that applying the concrete and doubly negated abstract wppts to the after-state retrieve relation yields a predicate (on the before-states) that is implied by the before-state quantities to the left of the implication.

Notions of correctness: In original B, precondition (trm) and guard (fis) are distinct concepts (unlike Z), albeit connected by the implication $\neg$ trm $\Rightarrow$ fis , due to the details of the axiomatic way that these two concepts are defined. Moreover, trm $\wedge \neg$ fis can hold for an operation, permitting *miracles*, a phenomenon absent from formalisms defined in a purely relational manner. In original B, trm is a conjunct of any operation's definition, so outside trm, nothing is assumed, and when interpreted relationally, it leads to something like a 'totalisation' (though different from the Z ones). During refinement, the precondition is weakened and the guard is strengthened, the former of which superficially sounds similar to Z, though it is again different technically.

Interpretation: The interpretation of operation steps for which trm and fis both hold is the conventional unproblematic one. Other steps fire the imagination. If trm is false the step aborts, i.e. it can start, but not complete normally; modelled relationally by an unconstrained outcome, a bit like contract Z. If fis is false the step does not start normally, but can complete; a miracle indeed, usually interpreted by saying that the step will not take place if fis is false.

Trace inclusion: In original B, trace inclusion is not addressed directly, but as a consequence of monotonicity. Refinement is monotonic across the B constructors, including sequential composition. This yields a notion of weak trace inclusion, since the trm and fis of a composition are an *output* of a composition calculation, not an *input*, and in particular, cannot be assumed to be the trm and fis of the first component, as one would want if one were extending a simulation by considering the next step. And even though the sufficient condition for fidelity, above, is a strengthening of the natural B refinement condition, it does not lead to an unproblematic strong trace inclusion, since in a relational model, we have the additional transitions generated by the 'totalisation', and miracles do not give rise to actual transitions.

Composition: In a real sense, the interaction of refinement and composition is not an issue in original B. The INCLUDES, USES, SEES mechanisms mentioned above are certainly composition mechanisms, but they act exclusively at the top level. Only the finally assembled complete abstract model is refined, which avoids the possibility of nonmonotonicity problems, such as those that arise for Z [14]. The IMPORTS mechanism allows the combination of independent developments.

3.3. Event-B

Event-B [19-21] emerged as a focusing of original B onto a subset that allows for both more convenient practical development, and also an avoidance of the more counterintuitive aspects of the original B formalism, such as miracles.

Formal language: Event-B is rooted in a traditional relational framework, derived by restricting original B operations (henceforth called events) to have a trm which is *true*, and controlling event availability purely via the guard, which is the domain of the event transition relation, as in Z. Distinguishing between guard and event in the syntax enables event transitions to be defined via

convenient notations (such as assignment) which are more widely defined than the desired guard. Formally, the more exotic possibilities afforded by predicate transformers are no longer needed.

Granularity and naming: Event-B relaxes the strict (1,1) conventions of original B. As in original B, the syntax of the refinement mechanism is embedded in the syntax of the refining machine, so an abstraction can be refined in more than one way, but not *vice versa*. However, a refining event now names its abstract event, so an abstract event can have several refinements within the same refining machine. New events in a refining machine are *implicitly* understood to refine an abstract *skip*, something which needed to be stated explicitly in original B, cluttering incremental development.

Concrete-abstract fidelity: The absence of the more exotic aspects of predicate transformers gives the Event-B fidelity PO a quite conventional appearance:

$$(\forall\, u, v, v' \bullet AInv(u) \wedge CInv(u,v) \wedge G_{CEv}(v) \wedge CEv(v,v') \\ \Rightarrow (\exists\, u' \bullet AEv(u,u') \wedge CInv(u',v')))$$

This says that assuming the abstract invariant and the concrete invariant (which is again a joint invariant i.e. retrieve relation) and the concrete guard for the before-states, and the concrete transition relation, yields the existence of an abstract event which re-establishes the joint invariant in the after-states.

Notions of correctness: The absence of preconditions distinct from guards simplifies matters considerably. The previous 'weakening of the precondition' during refinement of an operation, is now taken over by 'disjunction of concrete guard with guards of all new events is weaker than the abstract guard'. This is a quite different criterion, which nevertheless guarantees that if something can happen at the abstract level, a suitable thing is enabled at the concrete level. This is also combined with guard strengthening in the refinement of individual events, and a well foundedness property to prevent new events from being always enabled relative to old events. Totalisations are no longer present in any form, which has an impact on trace inclusion (see below).

Interpretation: The absence of preconditions distinct from guards simplifies interpretational matters considerably. There is a firm commitment to the idea that events which are not enabled do not execute, avoiding the need to engage with miracles and with spurious transitions generated by totalisation.

Trace inclusion: In the Event-B context, trace inclusion wins massively. Since for a refined event, the concrete guard implies the abstract one, the implication has the same orientation as the implication above, so the two work in harmony to enable any concrete step joined to an appropriate abstract before-state, to be unproblematically simulated, a phenomenon not present in formalisms mentioned earlier — simulated moreover, by a 'real' abstract event, not a fictitious one introduced via totalisation. New events do not disturb this, since they are by definition refinements of skip, which can always effortlessly simulate them. So we have genuine, uncluttered, strong trace inclusion.

Composition: Event-B takes a more pro-active approach to composition than original B. Event-B's top-down and incremental approach means that system models start out small and steadily get bigger. This allows composition to be instituted via *decomposition*. As a system model starts to get big, its events can be partitioned into subsystems, each of which contains *abstractions* of the events not present. These abstractions can capture how events in different subsystems need to interact, allowing for independent refinement, and avoiding the non-monotonicity problems mentioned earlier.

3.4. ASM

The Abstract State Machine approach developed in a desire to create an operationally based rigorous development framework at the highest level of abstraction possible. A definitive account is given in [8].

Formal language: Among all the methodologies we survey, ASM is the one that de-emphasises the formality of the language used for modelling the most — in a laudable desire to not dissuade users by forcing them to digest a large amount of technical detail at the outset. System states are general first order structures. These get updated by applying ASM rules, which modify the FO structures held in one or more *locations*. In a departure from the other formalisms reviewed, *all* rules with a true guard are applied simultaneously during an update, and therefore must be consistent.

Granularity and naming: The ASM approach tries as hard as it can to break the shackles of imposing, up front, any particular scheme of correspondence between abstract and concrete steps

during refinement. Nevertheless, there must be *some* correspondence between the two, or else we are totally adrift, so to underpin this, an 'equivalence' ≡ between the state spaces is postulated, which functions as a retrieve relation. It is demanded that the equivalence has to be periodically re-established, and in order to achieve this, a pair of simulating runs should be broken up into (*m*,*n*) diagrams of *m* abstract steps and *n* concrete ones (for arbitrary finite $m + n > 0$), without any preconceptions about how the (*m*,*n*) diagrams are to be arrived at, or which abstract or concrete steps might occur in an (*m*,*n*) diagram.

Concrete-abstract fidelity: In striving to be as unrestrictive as possible, ASM does not prescribe specific low level formats for establishing refinement. However, one technique, generalised forward simulation, established by Schellhorn [22] (see also [23]), has become identified as a *de facto* standard for ASM refinement. This demands that the (*m*,*n*) diagrams mentioned above are shown to be simulating by having a 'working' retrieve relation ≅, which implies the actual retrieve relation ≡. The ≅ relation is then used in implications of a similar form to those seen above, except that several abstract or concrete steps (or none) can be involved at a time. As many (*m*,*n*) diagram simulations as needed to guarantee coverage of all cases that arise must then be established.

Notions of correctness: It has already been mentioned that the retrieve relation ≡ is referred to as an equivalence. While almost all retrieve relations used in practice are in fact partial or total equivalences between the state spaces [24], knowing this for sure *a priori* has some useful consequences. It leads to a simple relationship between the guards of the run fragments in simulating (*m*,*n*) diagrams, subsuming guard strengthening, and eliminating many potential complications. Refinement is defined directly via a trace-inclusion-like criterion (periodic re-establishment of ≡), and for (0,*n*) and (*m*,0) diagrams, there is a well foundedness property to prevent permanent lack of progress in one or other system in a refinement. The analogue of 'precondition weakening' (though we emphasise that there is no separate notion of precondition in ASM) is subsumed by the notion of 'complete refinement' which demands that the abstract model refines the concrete one (as well as *vice versa*), thus ensuring that any time an abstract run is available, so is a suitable concrete one, yielding persistence of coverage of requirements down a refinement chain. Of course not all refinements need to be complete, permitting convenient underspecification at higher levels, in a similar manner to Event-B.

Interpretation: Since states and transitions are defined directly, there are no subtle issues of interpretation associated with them. Also, ASM rule firing is a hardwiring of the 'transitions which are not enabled do not execute' convention into the formalism.

Trace inclusion: The (*m*,*n*) diagram strategy of ASM modifies the notion of trace inclusion that one can sustain. The ASM (*m*,*n*) notion, at the heart of the ASM *correct refinement* criterion, can be viewed as a generalisation of the Event-B (1,1) strategy.

Composition: With the major focus being on identifying the ground model, and on its subsequent refinement (rather as in original B), the composition of independent refinements is not prominent in [8][23]. On the other hand, if ≡ *really is* an equivalence (or as we would need to have it between two state spaces which are different, a *regular* relation a.k.a. a *difunctional* relation [24]), there is a beneficial effect on any prospective composition of refinements. Many of the issues noted in [14] arise, because incompatible criteria about abstract sets (of states, say) which are unproblematic due to the abstract sets' disjointness, can become problematic due e.g. to precondition weakening when the sets' concrete retrieve images become non-disjoint via a non-regular retrieve relation. A regular retrieve relation does much to prevent this, facilitating composition of refinements.

3.5. VDM

VDM is among the oldest of the model based refinement methodologies, dating back to the early 70's [25-28]. Many of the notions that are now routinely found in model based methodologies had their earliest incarnations in VDM.

Formal language: VDM uses conventional discrete mathematical concepts for modelling state. It also uses a pre- and post- condition style of specification for state change, so that aspect is basically relational. One notable aspect of VDM is that it allows the definition of the *body* of an operation *Op* say, to be distinct from the definition of its pre- and post- conditions, pre-*Op* and post-*Op* respectively; this allows the definition of the body to contain imperative elements such as assignments. (*Specifications* refer to operations without a body.) Since a body must satisfy its specification, the

specification/body distinction means that VDM contains refinement in two different ways: between specs and their bodies, and between levels of abstraction (as usual). Another notable feature of VDM is the prominent use of a three valued logic for reasoning about partial functions and the like.

Granularity and naming: The literature on VDM typically features examples which contain operations which are *reified* (VDM-speak for 'data refined') to similarly named operations, e.g. *OP* is the reification of the abstract *OPa* [25]. So there is a (1,1) discipline in place with name almost-identity which is not formally prescribed.

Concrete-abstract fidelity: In VDM concrete-abstract fidelity is taken care of by the *Result Rule* ([25] Appendix E.3). Adjusting the notation for easier comparison with formalisms already discussed, this comes out as:

$$(\forall\, v, v' \bullet \text{pre-}AOp(R(v)) \wedge \text{post-}COp(v,v') \\ \Rightarrow \text{post-}AOp(R(v), R(v')))$$

In this, the retrieve *function R* is exactly that, a function, and the totality of *R*, and well-definedness of all the constituents of the implication are assumed. This allows the whole PO to be stated in terms of concrete variables alone. Aside from this, the condition is a simulation condition of a conventional kind. The fact that operation bodies are distinct from their pre- and post- conditions, means that the relationship between operation bodies and their pre- and post- conditions is captured in POs, amongst which, the relevant one has content amounting to the one quoted.

Notions of correctness: In VDM pre-*Op* has to *imply* the domain of *Op*, making it a guard in earlier terminology. Furthermore, preconditions are weakened during refinement, the consequences of which are similar to those in Z.

Interpretation: In the VDM standard [29], the VDM language is given a denotational semantics. Since this must give a meaning to all syntactically well-formed utterances of the language, $\perp$ is needed to cater for non-denoting utterances. This entails a kind of totalisation policy when interpreted in the domains of the denotational semantics, the consequences of which can be imagined from earlier discussions.

Trace inclusion: Without spelling it out directly, the preceding will have hinted that VDM runs into trouble regarding trace inclusion. Since the preconditions are weakened during refinement, strong trace inclusion (of *bona fide* traces, unpolluted by any spurious transitions generated by the totalisation policy) is impossible,[2] and a variant of the weak form (derived via the compositionality of the denotational semantics of VDM and affected by totalisation, as we have seen before) is the best that can be hoped for.

Composition: The composition of refinement developments is not discussed in the usual VDM literature. However, *Operation Decomposition* allows the body of an operation to be expressed as an algorithm using (specifications of) lower level operations — the algorithm has to be proved to be a correct implementation of the specification, and VDM provides a collection of Hoare-triple proof rules to accomplish this. The approach is very similar in spirit to the way that an original B IMPLEMENTATION can IMPORT lower level operations into an algorithm that constitutes the IMPLEMENTATION's body, and the designer has to prove that this algorithm correctly refines the system model that the IMPLEMENTATION claims to refine.

3.6. RAISE

Just as original B evolved into Event-B, so one could say that VDM prompted the development of RAISE [30-32]. Whereas the former process was one of specialisation, the latter was one of expansion and inclusion, in which the model oriented features of VDM were combined with algebraic features such as those in OBJ [33], with concurrency features such as those in CSP [34], with modularity features such as those in ML [35], with real time, and more.

---

[2] This is easy to see. Suppose a given concrete trace has been simulated, up to state *v* say, re-stablishing the retrieve relation with an abstract state *u*. Now *v* satisfies the precondition of the *next* concrete step by hypothesis. But *u* has been produced by simulating the *previous* concrete step. Since abstract preconditions are stronger than concrete ones, there is no guarantee that *u* satisfies the precondition of the *next* abstract step, as it must do to extend the simulation.

Formal language: Compared with all the formalisms we have looked at thus far, the RAISE formal language, RSL, is the most wide-spectrum and heavyweight, principally as a result of needing structuring mechanisms for all the various flavours of definition that it caters for simultaneously, a phenomenon amplified somewhat by the heavily universal-algebra-theoretic nature of most of its constructs. Nevertheless, in the state-centric part, states are specified via types and predicates, much as everywhere else. The richness of RAISE enables state change to be specified in a wide variety of ways, including imperative, axiomatic and algebraic.

Granularity and naming: The foundations of RAISE, based on universal algebra, imply that RAISE *implementation* (RAISE-speak for refinement) is heavily restricted. The refining system must have a signature that includes the signature of the refined one. In effect, this typically imposes a rather tightly drawn (1,1) discipline, in our terminology.

Concrete-abstract fidelity: Aside from signature inclusion, the other major plank of the RAISE implementation relation, is preservation of the 'properties' of the abstract system, where the properties constitute a precisely defined theory derivable from the system definition in a prescribed way. This is a wide-ranging and abstract definition of refinement, applicable across the whole of RAISE. If one focuses on the state-centric part, and works hard enough, a VDM-like fidelity condition emerges, which is even more restrictive than the original VDM one, due to the restrictions on signatures.

Notions of correctness: RAISE's notion of correctness is that the *environment* of the system (whose understanding of the system is via the system's abstract model) determines what is asked of the system and when. Thus any time an abstract operation is available, any refinement of it must also be available. This certainly preserves requirements coverage down through refinement levels.

Interpretation: With its insistence on viewing the abstract model as well defined and as paramount, the idea of asking what the concrete model might do in places where one was not supposed to look (i.e. in places which did not directly refine abstract behaviour) is not really in scope for RAISE. One can ask the question of course, and if one attempts to answer it in the context of a denotational semantics (say), one would detect phenomena similar to ones discussed already.

Trace inclusion: RAISE has trace inclusion, but the wrong way round (from our point of view). Thus, since a trace of the abstract system is a property of the abstract system derivable in a suitable way (i.e. provided one defines suitable observers of state transitions etc.), and the preservation of properties from abstract to concrete defines the notion of refinement in RAISE, an abstract trace will have a corresponding concrete one, but not necessarily vice versa. This is connected with the idea that the system's environment has the initiative about which system operations get invoked and when; and in particular, that the system itself never takes such initiative. This same idea holds in VDM (due to the details of the way that 'adequacy' and 'totality' restrictions work) and to varying degrees in other formalisms that take seriously the idea of 'weakening the precondition', but it finds its clearest expression in RAISE with its *ab initio* demands to preserve abstract properties.

Composition: With its universal algebra-theoretic foundations, the compositionality of RAISE developments is a top priority concern. The rather severe restrictions found in various parts of the RAISE approach ensure that this aspect works smoothly.

3.7. IO-Automata

IO-automata were introduced in [36] and are discussed in [37]. Textbook treatments are found in [38-39] where they constitute a convenient technique for discussing distributed algorithms.

Formal language: In the formal theory of IO-automata, states are elements of a state set, and state transitions involve changing the focus from the before-state to the after-state, in line with the automata theoretic flavour. In practical use, rather as in ASMs, the linguistic framework of IO-automata is not tightly constrained in order to improve the capacity to communicate. So notions like stacks and queues can appear directly in the straightforward imperative-language-like constructions used, instead of unravelling them to their basic mathematical ingredients. Thus IO-automata are rooted in a relational framework of a conventional kind.

Granularity and naming: The IO-automata notion of refinement is defined via trace inclusion, and established via (m,n) diagrams very like the ones of ASM, but in which $n = 1$ (i.e. always exactly one concrete step). Accordingly, there is no specific restriction on how names of abstract and concrete operations should correspond in refinement.

Concrete-abstract fidelity: As just noted, the basic refinement criterion is that a concrete step should be simulated by zero or more abstract ones. This generates a fidelity PO of a kind similar to the ASM one.

Notions of correctness: One detail not mentioned thus far about the IO-automata fidelity criterion, is that concrete steps need be simulable by abstract ones only when the concrete before-state is in the retrieve relation with a *reachable* abstract (before-)state. This is a variation on what is usually stated in a fidelity PO, and allows the abstract and concrete systems to behave in incompatible ways in the non-reachable portions of their transition systems, but does not endanger trace inclusion.

Interpretation: The automata theoretic perspective of IO-automata implies, as do all automata theoretic frameworks, an absence of problematic interpretational issues. The automata theoretic perspective is based on the direct depiction of the states and transitions that exist, without introducing intensional aspects regarding volition (whether on the part of the system or of the environment) and without the use of logical formulae whose interpretation one has to worry about when they evaluate to *false*.

Trace inclusion: More directly than in any other framework surveyed, IO-automata place trace inclusion at the heart of the definition of refinement ([37] discusses several detailed versions). Technically, the definition states a weak trace inclusion criterion. However, all practical means of establishing trace inclusion are focused on strong trace inclusion via the usual kind of inductive technique discussed above. It is clear that the restriction of the fidelity PO to *reachable* abstract before-states does not threaten strong trace inclusion (provided one starts things off using the usual initialisation PO).

Composition: The composition of IO-automata is discussed in the cited references. Consistent with the automata theoretic flavour, only *compatible* collections of IO-automata can be composed, compatibility amounting to a number of non-interference conditions on the components of an IO-automaton. These components amount to named operations in our terminology, and their names are 'free', i.e. they are implicitly regarded as residing in a universal name space, and so might clash. For compatible (appropriately non-clashing) collections, composition becomes a straightforward product construction. With this perspective on composition, the composition of independent refinements is not problematic.

3.8. TLA+

TLA+ [40] evolved as a more complete version of the earlier TLA [41]. The emphasis in TLA+ is much more on traces as a whole rather than individual steps considered in isolation. Accordingly, temporal properties (notably fairness properties) are much more to the fore.

Formal language: The trace descriptions in TLA+ are phrased in terms of states and state changes, as expected. States are defined using the usual machinery, and state changes are pairs of before- and after- states, so that particular aspect is basically relational. State changes are assembled into traces by concatenation of individual steps in the usual way.

Concrete-abstract fidelity: The fundamental notion of refinement in TLA+ (called implementation in TLA literature) is based on implication of the temporal logic formula specifying the abstract system by the temporal logic formula specifying the concrete system, all up to stuttering since well behaved TLA+ specs should be stuttering invariant. The focus on implication as the key idea, means that the free variables of the two systems must correspond, and that consequently, things like local variables must be appropriately quantified. To prove the implication in practice, one has to strip away these quantifications and construct refinement mappings [42], which play the role of our retrieve relations, but with the added temporal dimension. Since the concrete trace has to imply the abstract one, we effectively regain a conventional fidelity criterion, modulated somewhat by the stuttering invariance requirement.

Granularity and naming: As is clear from the preceding point, since what is demanded is trace implication up to stuttering, and the traces are constructed from formulae whose *subformulae* typically correspond to what we called operations or events above, there is no insistence in TLA+ that these subformulae (and more precisely their *names*) need correspond in any particular way. In practice though, in many examples, a straightforward (1,1) meta level correspondence is clearly visible.

Notions of correctness: With the focus on implication, there is no separate criterion that is aimed at policing requirements coverage from abstract down to concrete as in other formalisms, and consequently no conflicts with other desiderata arise.

Interpretation: With the focus on implication, no unusual issues concerning interpretation arise.

Trace inclusion: With the focus on implication of temporal logic formulae, the link to concrete-to-abstract trace inclusion is very strong. It has to be remembered though that operation names do not figure in the notion of trace relevant to TLA+ and that TLA+ traces are stuttering invariant.

Composition: An easy consequence of the focus on implication, with its insistence that local variables must be quantified, is the fact that independently derived refinements (presumed to refer to distinct free variables), can be composed, the quantification of local variables acting in a manner very like the equivalences in ASM refinement, in preventing clashes.

4. Semantic Variety

The preceding subsections briefly overviewed a number of well-known model oriented refinement paradigms. Even from this cursory look, it is easy to be struck by how so many of the detailed issues we have highlighted can be seen to be mere *design decisions* about one or other aspect of the methodology in question. In this sense, it is often the case that one has a *choice* about how some particular issue in some particular formalism, might be handled. The fact that we note that there can be a choice is not intended to imply that the choice is made on a whim. In practice the choices are typically governed by the higher level concerns that drive the design of the refinement notion.

In truth, the degree to which such choice arises is significantly controlled by the extent to which the refinement notion is derived in a 'monolithic way'. Thus in RAISE, preservation of abstract properties determines a host of other details uniquely; in TLA+ and IO-automata, implication and trace inclusion have a similar, though a not quite as far-reaching, effect. On the other hand, the formalisms treated earlier in Section 3 could more easily be assembled out of smaller pieces, allowing greater scope for replacing a design decision about some aspect with an alternative. We mention a few such potential design realignments for purposes of illustration.

- Regarding Z, one could easily imagine its notion(s) of correctness being substituted by the ones from Event-B or ASM. Its notion of trace inclusion would then be replaced by one not requiring the use of 'fictitious' transitions generated by totalisation. Further options include the VDM or IO-automata notions of correctness. In fact, alternatives for what we have here called notions of correctness for Z, have been explored in [43] and related publications.
- For B, one could easily imagine adding $\perp$ elements to state spaces etc. in order to obtain a different relational semantics, with fewer 'fictitious' transitions.
- For Event-B and ASM one could imagine bringing in some aspects of the Z modelling, though it appears that little would be gained by doing so. Alternative correctness ideas from IO-automata or TLA+ could easily be contemplated.
- It would not be hard to replace VDM's notion of correctness by Z's, B's, Event-B's, ASM's or IO-automata's, thus altering the balance between abstract and concrete models. In fact, the constraints of the VDM functional retrieve relation framework were relaxed in Nipkow's modified fidelity rule [44], making it much like the Event-B rule.
- One could take RAISE's policy on property preservation as the fundamental criterion, and by formulating suitable notions of 'property' in the other formalisms, rebuild their notions of refinement in that light. Given RAISE's monolithic approach of doing everything via properties, there is less scope for low level detailed 'fiddling', but if one discarded the monolithic property approach, then the semantic framework of RAISE could easily host a wide variety of alternative views.
- One could take the IO-automata reachability criterion on the abstract states that must simulate, and transplant it painlessly to other formalisms. Equally one could import various different criteria on correctness and fidelity from other formalisms into IO-automata without difficulty.
- One could take TLA+'s policy on implication of temporal logic formulae as the fundamental criterion, and re-interpret it in a variety of other formalisms.

Of course such ideas are not new. In many instances, alternatives of one kind or another have been investigated, whether in the normal research literature or as student research projects. Although we do

not attempt to catalogue all the variations that have been considered over the years, two, both connected with B, are worth noting for their practical impact.

One is found in the context of ProB [45], a model checker and animator for the B-Method first implemented for original B. There, the original B preconditions are re-interpreted as (i.e. given the semantics of) additional guards. Such a move is needed to ensure that the theoretical constructions at the heart of model checking remain valid.

The other is found in the context of the Météor Project [46], where the semantics of original B was modified to explicitly check well-definedness conditions for applications of (partial) functions, using techniques going back to Owe [47]. This was in recognition of the need to be more careful about domains of partial functions and operations in the context of a safety-critical application. In Event-B, a more recent development, such checks are built in *ab initio*, and Event-B's semantics fits model checking needs much better too.

What the above variety, and the associated remarks, make clear, is that there is no unique consensus about what constitutes model based refinement. Different setups are created with different kinds of requirements scenarios either explicitly or implicitly in mind, and the resulting formalisms are promoted as being relatively general purpose, a view that holds up as long as a requirements scenario that does not differ too markedly from the initial ones is not encountered. If a sufficiently different requirements scenario *is* encountered, then a common reaction (at least in the research literature) is to invent a new model based refinement paradigm, better suited to the scenario at hand. In more industrial environments, the reaction may well be to 'make do' as best one can, or to quietly decide not to bother with formal techniques in future.

In the light of this variety, one thing that we can do, is to look for areas of commonality between the different formulations. We can take the view that the more commonly a particular feature or design decision occurs amongst different frameworks, the more it contributes to the 'essence' of model based techniques. Conversely if a feature is only rarely found, it is easier to justify trading it for an alternative. Roughly speaking, it is the *intersection* rather than the *union* of features among different frameworks that we focus on.

To this aim, one cannot help notice that the area most free from excessive variation has been what we call the 'concrete-abstract fidelity' area. This indicates a strong consensus among approaches that simulation (in one form or another) is *the* key criterion that techniques must establish.

One cannot help further notice that the places where the greatest semantic variety is to be found, occur in the 'notions of correctness' and 'interpretation' areas. One can attribute a lot of this variation, to different views on the extent to which the abstract model should dictate what the concrete model does. In other words, does the abstract system demand that the concrete one furnishes some (suitable) counterpart to *every* behaviour it is capable of, or does the abstract system merely provide a container beyond which the concrete one is not allowed to stray? Different views on what refinement is for, heavily influence the preference for one or the other perspective.

On this point, one can argue both ways. Demanding that the abstract system demands concrete compliance, ensures requirements coverage (suitably interpreted), but clutters the formalism with (possibly many) additional POs; ones, moreover, that can have a detrimental effect on trace inclusion unless one is particularly careful. Not demanding that the concrete system cover every move of the abstract one frees the formalism from many extra POs, and is helpful for trace inclusion, but potentially lets through the possibility of explicitly empty (including syntactically contradictory) implementations, resting on the 'don't care' discharge of the implication at the heart of any fidelity PO, when its hypothesis is false. One could counter this by saying that no one would entertain an explicitly empty implementation, but then, inadvertently ending up with one that was semantically empty, by virtue of hiding a contradiction within the labyrinth of a large syntactic description, can happen much more easily. Also, the more developers rely on tools, the more likely they are to become personally inattentive, slipping easily into the mode of believing all is well just because a tool has said 'OK', even if it was on the basis of an unsatisfiable hypothesis. Matters are made more complicated in the latter case by virtue of the fact that a code generator working on such an implementation definition (i.e. one that was contradictory in this way) would not be sensitive to its semantic emptiness, and, blissfully unaware, could generate a lot of code, large amounts of which would actually be entirely useful.

It becomes clear why there is so much variation in views around the issue of notions of correctness. By contrast, other issues from Section 2, such as 'formal language', 'granularity and naming' and

'trace inclusion', 'composition', can be seen as either setting the stage for creating a given framework, or as derivable consequences of the design decisions taken.

5. Retrenchment

The previous sections dissected what goes into a typical model based refinement framework. We saw that each such notion examined, depended on a number of design decisions about various aspects of the development process. The trouble with this is that, once these design decisions are made, formalisation casts them in stone to a large degree. At some later point, along comes a case study for which the previously made decisions turn out to be less than optimal. Then, a struggle can ensue to make the problem fit the technique. The result can be successful, or not, to varying degrees.

Retrenchment was invented as a response to the phenomenon that a given refinement technique (or even refinement techniques in general) do(es) not always fit all application scenarios in which the rigour characteristic of such techniques is desirable, and that consequently, some means of reconciling those aspects of the application that did not fit well into the selected refinement technology with those that did, was desirable. And the greater the rigour that could be brought to bear on this issue the better. Although retrenchment arose in the context of the B-Method [48], it seemed evident, even at the outset, that one should design similar notions for other approaches too. With the accumulated experience of the intervening years, we can now set out how to do this.

Notions of refinement are intended to give *a priori guarantees* that the more concrete model conforms in a certain way to the abstract one. Often it is glibly stated that 'the concrete model preserves all the properties of the abstract model' but this is palpably false unless one makes the technical terms in the remark, such as 'all' and 'properties', quite precise, and either proves a suitable theorem, or takes the ensuing statement as the definition of refinement and derives the consequences (as in the RAISE approach). Security properties usually provide the quickest route to demolishing such too-loosely-phrased 'property preservation' statements [49].

By contrast, the role of retrenchment is not so much to propose *a priori properties* —recognising that it would be almost impossible to predetermine every possible way in which the formulation of a refinement notion might prove inconvenient in practice— and much more to provide a framework within which such incompatibilities might be described in general terms, and linked in a formal way to what the refinement notion *can* accomplish. Often it will be true that particular, much stronger, properties might follow in a specific application area, but it is better (from the real world developer's perspective) to have these emerge as a strengthening of the general theory of retrenchment, than to develop a special purpose theory whose remit, in practice, might be confined to just that case — bringing into being therefore as it would, the necessity of developing an alternative theory for every different problem.

5.1. Key Features of Retrenchment Designs

Above, we saw that what the overwhelming majority of model based refinement notions had in common, was the concrete-abstract fidelity criterion. Given that what we seek in retrenchment is the maximum flexibility of expressiveness, and yet we have to keep *something* relatively fixed (else it is difficult to speak of a single unifying notion), we make *the concrete-abstract fidelity criterion* the predominant focus of retrenchment designs.

Equally, we saw above that there was a wide variety of views in what we referred to as the 'notions of correctness' area, which could be viewed as arising by adopting a corresponding variety of subsidiary design decisions, and adding their consequences to the fidelity PO. Since the consequences of adopting different design decisions in this area are often incompatible with each other, we determine that retrenchment should *not* be prescriptive in this area, in order to maximise the applicability of general retrenchment principles to a wide variety of domains, with different domains exhibiting different perspectives on how the theory connects with reality.[3]

[3] In the early days of retrenchment, when [48] and similar papers were written, it was presumed that retrenchment ought to make suitable demands regarding notions of correctness and the like. It took a considerable time, conclusively confirmed by experience with the *Tower Pattern* (see Section 5.2), to convince us that such a position, while tenable, was nevertheless ill advised. One might conjecture that

Thus, retrenchment should align with refinement where there is a high degree of commonality of views, in order to enhance interworking, and should desist from committing itself where there is a high degree of diversity of views, acting in the latter case, as a counterbalance to the excessive specificity that would otherwise arise.

A further guiding principle that we would like to see embodied in retrenchment, is that its fidelity PO should reduce to (the appropriate version of) the refinement fidelity PO under suitable circumstances, i.e. when the requirements issues that it is designed to cover trivialise.

The means by which retrenchment achieves all these things is to modify the fidelity PO of refinement, to gain greater expressivity. As we have seen, refinement fidelity POs, across all the formalisms reviewed, come down to an implicative structure, based on re-establishing a retrieve relation. Accordingly, retrenchment modifies this by introducing a number of additional relations into the fidelity PO: a within relation *W* that strengthens the hypotheses of the implication, to restrict the scope of the statement, where needed; an output relation *O* strengthening 'good' outcomes in the conclusion by permitting the incorporation of detail not expressible within the constraints of the refinement formalism; and a concedes relation *C* in the conclusion, occurring disjunctively, to allow the description of behaviours that violate the retrieve relation. In 'formalism-independent' terms, the generic fidelity PO that emerges has the appearance of the TLA+ case below, assuming one ignores any TLA+-specific connotations.

Since the justification for this general approach has been extensively discussed elsewhere in an abstract setting (see e.g. [6-7]), and its utility has been well borne out in case studies, notably those concerning the Mondex Purse (see e.g. [50-53]}) we do not repeat all that here, concentrating instead on checking how the general ideas fit with the specific formalisms we have looked at. We note though, that letting *W* and *O* to default to *true* and *C* default to *false* does indeed yield the desired continuity with refinement notions in the I/O-free case.[4]

5.2. Tower Compatibility

Thus far, we have concluded that retrenchment should concentrate on a modified fidelity PO, and largely disregard 'notions of correctness' criteria, supporting this stance using predominantly heuristic evidence. This position is considerably strengthened when we consider refinement/retrenchment interworking.

Besides the retrenchment fidelity PO defaulting to the refinement one when *W*, *O*, *C*, suitably trivialise, we need to consider how retrenchment and refinement co-operate during the course of a system development in which each participates non-trivially. The key criterion here is *Tower Compatibility*, and it refers to the conditions that must hold in order to avoid the interpretational pathologies of some of the refinement approaches, and in order that the basic constructions of the *Tower Pattern* can be established.

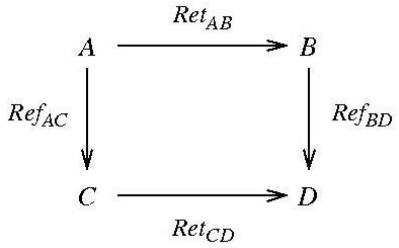
The *Tower Pattern* itself is built out of 'commuting' squares of refinements and retrenchments, such as occur in the figure below. In the figure, refinements are vertical arrows (Ref_{AC} and Ref_{BD}, going from the abstract system to the concrete one), and the retrenchments are horizontal arrows (Ret_{AB} and Ret_{CD}, also going from the abstract system to the concrete one). The 'commuting' nature of the diagram expresses the compatibility between the two notions that we seek. We said 'commuting' in inverted commas because the precise sense in which commutativity is intended, and the extent to which it is actually achieved (in a given collection of theoretical results) varies, depending on the precise theoretical direction taken. In [54], a suite of 'square completion' results was proved that interpreted commutativity as equality 'on the nose' of the two retrenchments resulting from composing the retrenchment/refinement pairs going round the square in two ways from system *A* to system *D*. Although superficially appealing, the technical details of pursuing this approach turned out to be eye-wateringly complicated. More recently, the question has been revisited in [55]. There, the commutativity was interpreted as *compatibility* of the two retrenchments resulting from composing the

---

just as the most problematic errors in system design arise from missing requirements, so the subtlest problems in designing system design methodologies can be attributed to determining what requirements *ought to be* missing.

[4] Where there is I/O, *W* and *O* default to the 'natural' correspondences between the input spaces and output spaces respectively.

retrenchment/refinement pairs going round the square, in the sense that each of the composed pairs expressed a part of a larger retrenchment from *A* to *D*. The latter could in turn be calculated from the two composed the retrenchment/refinement pairs using a form of composition of retrenchments called fusion composition (see [55] for details). The technical complexity of constructing the square completions using this approach (and of associated technical issues whose details need not concern us here) was dramatically reduced.



Both investigations showed that a simple criterion facilitated retrenchment/refinement interworking. Stated informally and in generic terms, it amounts to the statement that the within relation must be stronger than any criterion that contributes to the definition (in terms of before-states and inputs) of 'well behaved' transitions in either the abstract or concrete system. This is the *Tower Compatibility* PO. A 'formalism-independent' statement of it resembles the TLA+ case below, assuming one ignores the TLA+-specific connotations. However, it is to be noted that since it constitutes the interface between retrenchment notions and the rather varied 'correctness' notions of a variety of refinement formalisms, there is more variation among the specific incarnations of the *Tower Compatibility* PO than among those of the fidelity PO, as we see below.

To elaborate a little further, in the formalisms we surveyed briefly above, we often encountered entities such as preconditions, termination conditions, guards, etc. They were all expressible using predicates in the before-states (and inputs). Moreover, different refinement notions contained different numbers of these entities, and their precise behaviour in the passage from abstract to concrete system also varied considerably. The *Tower Compatibility* criterion says that no matter what the number or demanded behaviour in relation to the refinement notion under consideration is of these entities, the within relation for the associated notion of retrenchment, must be stronger than every such entity. Further illustration of the concept is best done in the context of some specific notions of retrenchment, to which we turn next.

6. Retrenchment Designs for Various Refinement Notions

Thus, we have come to the conclusion that retrenchment should comprise a fidelity PO compatible with the one used for refinement, and a compatibility condition for tower interworking. To this we add an initialisation PO like the standard refinement one. We now review how this works out for the refinement notions examined above. Further general background can be found in [6-7].

6.1. Retrenchment for Z

The Z version of retrenchment has been extensively exercised in the context of the Mondex retrenchment case studies [50-53]. In Z, retrenchment amounts to the initialization PO:[5]

$$(\forall\ CState'(v') \bullet CInit'(v') \Rightarrow (\exists\ AState'(u') \bullet AInit'(u') \wedge R'(u',v')))$$

and the fidelity PO:

[5] Since the initialisation PO is defined to be identical to that for (the relevant notion of) refinement, a policy persists unchanged through all the formalisms we cover, we will not mention initialisation further below.

$$(\forall\ AState(u)\ ;\ CState(v)\ ;\ CState'(v') \bullet R(u,v) \wedge W_{Op}(u,v) \wedge COp(v,v')$$
$$\Rightarrow (\exists\ AState'(u') \bullet AOp(u,u') \wedge ((R'(u',v') \wedge O_{Op}(u',v',u,v)) \vee C_{Op}(u',v',u,v))))$$

Added to the above is the tower compatibility condition, which comes out as:

$$(\forall\ AState(u)\ ;\ CState(v) \bullet R(u,v) \wedge W_{Op}(u,v) \Rightarrow \text{pre}\ AOp(u) \wedge \text{pre}\ COp(v))$$

with evident addition of (input state schemas and) input variables to $W_{Op}(u,v)$ when I/O is present.

6.2. Retrenchment for B

For original B, the fidelity PO is an adaptation of the relevant refinement condition:

$$AInv(u) \wedge R(u,v) \wedge CInv(v) \wedge W_{Op}(u,v)$$
$$\Rightarrow [\ COp(v,v')\ ]\ \neg\ [\ AOp(u,u')\ ]\ \neg\ ((R(u',v') \wedge O_{Op}(u',v'\ \ldots)) \vee C_{Op}(u',v'\ \ldots))$$

However, compared with the refinement version, the above conceals a few subtleties. First of all, we have separated the two roles of $CInv(v)$: namely to act as the intrinsic invariant of the concrete system, and to act as the retrieve relation between abstract and concrete. The former role is still called $CInv(v)$ while the latter is covered by new predicate $R(u,v)$ explicitly. Secondly, predicate transformers transform after-values of state variables into before-values. To enable relationships involving both to be expressed (as in $O_{Op}(u',v'\ \ldots)$ and $C_{Op}(u',v'\ \ldots)$), one must use auxiliary variables, introducing defining occurrences for them in $W_{Op}(u,v)$ for the before-values, and reference occurrences in $O_{Op}(u',v'\ \ldots)$ and $C_{Op}(u',v'\ \ldots)$, a standard B trick, indicated by the ellipsis. Thirdly, one must use similar devices to model I/O: variables that are referenced but not updated for input, and variables that are updated but not referenced for output.

Added to this is the tower compatibility condition (with the obvious signature embellishment when I/O is present), which is more complex than for other formalisms, since preconditions and guards are (at least partly) independent in original B:

$$AInv(u) \wedge R(u,v) \wedge CInv(v) \wedge W_{Op}(u,v)$$
$$\Rightarrow \text{trm}\ AOp(u) \wedge \text{trm}\ COp(v) \wedge \text{grd}\ AOp(u) \wedge \text{grd}\ AOp(v)$$

6.3. Retrenchment for Event-B

Retrenchment for Event-B has been treated in fair detail in [56-57]. Here we just summarise the essentials for comparison with other formalisms. Compared to original B, retrenchment for Event-B is considerably simpler conceptually, due to the possibility of referring to both before- and after-variables in the same expression, without recourse to 'back-door' techniques such as auxiliary variables. The fidelity PO is straightforwardly relational:

$$(\forall\ u, v, v' \bullet AInv(u) \wedge R(u,v) \wedge CInv(v) \wedge W_{Ev}(u,v) \wedge G_{CEv}(v) \wedge CEv(v,v')$$
$$\Rightarrow (\exists\ u' \bullet AEv(u,u') \wedge ((R(u',v') \wedge O_{Ev}(u',v',u,v)) \vee C_{Ev}(u',v',u,v))))$$

In this we again separated retrieving properties from concrete invariant properties. Since Event-B encourages seeing I/O variables as being on the same footing as state variables, i.e. there is no specific category of output variables, the presence of O_{Ev} becomes almost dispensible (and in fact [56] develops the O_{Ev}-free version of the theory in detail). However, if one wants the PO to *guarantee* some strengthening of the retrieve relation via the PO, then recourse to O_{Ev} is needed. If all that is desired is a convenient container into which one can put some provable facts, then C_{Ev} by itself will do (the issue is discussed further in [6]). The tower compatibility condition is:

$$AInv(u) \wedge R(u,v) \wedge CInv(v) \wedge W_{Op}(u,v)$$
$$\Rightarrow \text{grd}\ AOp(u) \wedge \text{grd}\ AOp(v)$$

6.4. Retrenchment for ASM

For ASM, the main difference compared with preceding formalisms is brought about due to the fact that we deal with (m,n) diagrams, rather than (1,1) diagrams. This introduces additional quantifications not present elsewhere. Let *CFrags* be the set of concrete execution fragments that we have previously determined will permit a covering of all concrete execution sequences of interest (and thus whose simulation will guarantee strong trace inclusion). We write $v::vs::v' \in CFrags$ to denote an element of *CFrags* starting with v, ending with v', and with intervening state sequence vs. Then the ASM fidelity PO for retrenchment becomes:

$$(\forall\, u, v, vs, v' \bullet v::vs::v' \in CFrags \wedge u \equiv v \wedge W_{AOps,COps}(u,v) \wedge COps(v::vs::v')$$
$$\Rightarrow (\exists\, us, u' \bullet AOps(u::us::u') \wedge ((u' \equiv v' \wedge O_{AOps,COps}(u',v',u,v)) \vee C_{AOps,COps}(u',v',u,v))))$$

In this, $\equiv$ is the ASM retrieve relation equivalence between abstract and concrete state spaces discussed earlier, and $COps(v::vs::v')$ is a concrete fragment that we need to simulate, where $v::vs::v' \in CFrags$ is the sequence of states in the concrete part of the (m,n) diagram. Similar notations hold at the abstract level. Since us and u' are existentially quantified inside the outer quantification, both the length of $u::vs::u'$ and the abstract operations involved, depend in detail on the concrete fragment chosen. The within, output and concedes relations, now depend on both sequences of operations *AOps* and *COps*, since there is no predefined naming convention in force between abstract and concrete names. Moreover, the output and concedes relations have been shown as depending only on the extreme abstract and concrete states; a more detailed dependence on the internal states can obviously be demanded, especially since, in practice, the given relation would be proved by examining corresponding concrete and abstract behaviours in detail. Again, I/O can be added unproblematically.

Finally, the tower compatibility condition becomes:

$$u \equiv v \wedge W_{AOps,AOps}(u,v) \Rightarrow \text{dom}\, AOps(u) \wedge \text{dom}\, COps(v)$$

6.5. Retrenchment for VDM

For VDM, there is little change compared with the relational worlds of Z and Event-B, except for the stronger assumptions in force regarding the retrieve function, namely: totality, adequacy, and (of course) the fact that it is required to be an actual function. Building all this, as appropriate, into the fidelity PO, we get:

$$(\forall\, u, v, v' \bullet u = R(v) \wedge W_{Op}(u,v) \wedge \text{post-}COp(v,v')$$
$$\Rightarrow (\exists\, u' \bullet \text{post-}AOp(R(v), u') \wedge ((u' = R(v') \wedge O_{Op}(u',v',u,v)) \vee C_{Op}(u',v',u,v))))$$

and tower compatibility reduces to:

$$u = R(v) \wedge W_{Op}(u,v) \Rightarrow \text{pre-}AOps(u) \wedge \text{pre-}COps(v)$$

The consequences of introducing explicit I/O into these relations are easy to imagine.

6.6. Retrenchment for RAISE

As we said earlier, RAISE is a very broadly based formalism. On the basis that appropriate observers are formulated to enable transitions to be described and reasoned about, the refinement criterion for the model-based part is very VDM-like. Accordingly, the retrenchment notion that will be appropriate will be as for VDM, so we do not quote it again.

6.7. Retrenchment for IO-Automata

We noted above that the IO-automata notion of refinement reduced to a special case of ASM refinement. Therefore the corresponding retrenchment notion will specialise the ASM notion too.

Bearing in mind that there is always exactly one concrete step $COp(v,v')$ involved, the fidelity PO becomes:

$$(\forall\, u, v, v' \bullet R(u,v) \wedge W_{AOps,COp}(u,v) \wedge COp(v,v') \\ \Rightarrow (\exists\, us, u' \bullet AOps(u::us::u') \wedge ((R(u',v') \wedge O_{AOps,COp}(u',v',u,v)) \vee C_{AOps,COp}(u',v',u,v))))$$

while tower compatibility becomes a slight modification of the ASM version:

$$R(u,v) \wedge W_{AOps,COp}(u,v) \Rightarrow \mathrm{dom}\, AOps(u) \wedge \mathrm{dom}\, COp(v)$$

6.8. Retrenchment for TLA+

We noted above that the TLA+ refinement notion was based on implication of temporal logic formulae, a requirement that demands the inclusion of concrete observable behaviour within the observable behaviour permitted for the abstract system. This is a quite pure notion of 'black box' refinement. Retrenchment however, is chiefly concerned with expressing, in a 'glass box' manner, refinement-theoretic incompatibilities that arise during development in real world scenarios [6]. This makes insisting on implication between TLA+ formulae, into an inappropriate criterion on which to base a corresponding definition of retrenchment, without making it unduly restrictive. Accordingly, the most appropriate notion of retrenchment for TLA+ utilises the pure transition system formulation given in [6], for which the fidelity PO is:

$$(\forall\, u, v, v' \bullet R(u,v) \wedge W_{Op}(u,v) \wedge COp(v,v') \\ \Rightarrow (\exists\, u' \bullet AOp(u,u') \wedge ((R(u',v') \wedge O_{Op}(u',v',u,v)) \vee C_{Op}(u',v',u,v))))$$

To this we add the tower compatibility PO:

$$R(u,v) \wedge W_{Op}(u,v) \Rightarrow \mathrm{dom}\, AOp(u) \wedge \mathrm{dom}\, COp(v)$$

We observe that both of these work at the level of individual steps, so any interaction with stuttering must be taken care of at a level meta- to the one figuring in the POs above.

Of course, thinking about stuttering unavoidably warrants thinking about execution sequences, and about the fact that our POs do not offer any guarantees that any kind of inductive correspondence can be constructed between concrete and abstract executions without additional assumptions. Such questions concern the mapping of system properties under retrenchment, a topic outside the scope of this paper.

7. Discussion and Conclusions

In this paper, we have examined some key features of a representative number of well-known refinement methodologies, and commented on their similarities and differences. We noted that many features were not especially specific to the methodologies in which they were found, and that we could just as easily transplant them elsewhere — the one notable exception to this perhaps being RAISE, with its monolithic derivation of everything from the idea of formal property preservation.

We took the evidence accumulated thereby, and used it to support the way that retrenchment is formulated in general, and showed how this policy could be made concrete in the refinement methodologies previously examined. This general analysis of retrenchment, and its unified detailing in the specific methodologies mentioned, constitute the main contributions of this paper. The high degree of similarity among the various specific notions, upholds a view that the essence of retrenchment has been correctly identified. Additional supporting leverage for the way that retrenchment has been formulated came from the algebraic theory of the *Tower Pattern*, which supplied crucial extra insight relevant to the 'notions of correctness' area, the place where the widest diversity of detailed views is typically found.

The clean interaction between refinement and retrenchment is vital for practical working in general, and for support by tools in particular. The Frog tool [58-59] is an experimental tool whose design

incorporates the flexibility needed to smoothly integrate the POs of refinement and retrenchment in the manner suggested, and so, shows the path that more industrial strength tools could follow. Such a tools strategy is also in harmony with the call for an Evidential Tool Bus [60]. This is a tools framework which would allow different tools, focusing on different aspects of verification, to communicate their findings to other tools, and to include other tools' results in their own deliberations. Moreover, the tools strategy being advocated is also in harmony with the currently active Verification Grand Challenge [2-4], whose remit is not only to put large scale verification case studies into the public domain, so as to promote the uptake of formal techniques in general, but also to aid tool integration, so that industrial scale formal development can be done with increasing cost effectiveness.

References

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

[2] Jones, C., O'Hearne, P., Woodcock, J.: Verified Software: A Grand Challenge. IEEE Computer 39(4), (2006) 93–95.

[3] Woodcock, J.: First Steps in the The Verified Software Grand Challenge. IEEE Computer 39(10), (2006) 57–64.

[4] Woodcock, J., Banach, R.: The Verification Grand Challenge. JUCS 13(5), (2007) 661–668.

[5] Banach, R.: Model Based Refinement and the Tools of Tomorrow. In ABZ 2008: Volume 5238 of LNCS., Springer (2008) 42–56.

[6] Banach, R., Poppleton,M., Jeske, C., Stepney, S.: Engineering and Theoretical Underpinnings of Retrenchment. Sci. Comp. Prog. 67 (2007) 301–329.

[7] Banach, R., Jeske, C., Poppleton, M.: Composition Mechanisms for Retrenchment. J. Log. Alg. Prog. 75, (2008) 209–229.

[8] Börger, E., Stärk, R.: Abstract State Machines. A Method for High Level System Design and Analysis. Springer (2003).

[9] ISO/IEC 13568: Information Technology — Z Formal Specification Notation — Syntax, Type System and Semantics: International Standard. (2002) http://www.iso.org/iso/en/ittf/PubliclyAvailable Standards/c021573_ISO_IEC_13568 2002(E).zip .

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

[11] Derrick, J., Boiten, E.: Refinement in Z and Object-Z. FACIT. Springer (2001).

[12] Spivey, J.: The Z Notation: A Reference Manual. Second ed. Prentice-Hall International (1992).

[13] Cooper, D., Stepney, S., Woodcock, J.: Derivation of Z Refinement Proof Rules. Technical Report YCS-2002-347, University of York (2002).

[14] Groves, L.: Practical Data Refinement for the Z Schema Calculus. In: ZB 2005: Formal Specification and Development in Z and B. Volume 3455 of LNCS., Springer (2005) 393–413.

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

[16] Lano, K., Haughton, H.: Specification in B. Imperial College Press (1996).

[17] Habrias, H.: Specification Formelle avec B. Hermes Sciences Publications (2001).

[18] Schneider, S.: The B-Method. Palgrave (2001).

[19] Abrial, J.R.: Modeling in Event-B: System and Software Engineering. Cambridge University Press (2010).

[20] Rodin. European Project Rodin (Rigorous Open Development for Complex Systems) IST-511599 http://rodin.cs.ncl.ac.uk/ .

[21] The Rodin Platform. http://sourceforge.net/projects/rodin-b-sharp/ .

[22] Schellhorn, G.: Verification of ASM Refinements Using Generalised Forward Simulation. J.UCS 7(11) (2001) 952–979.

[23] Börger, E.: The ASM Refinement Method. Form. Asp. Comp. 15 (2003) 237–257.

[24] Banach, R.: On Regularity in Software Design. Sci. Comp. Prog. 24 (1995) 221–248.

[25] Jones, C.: Systematic Software Development Using VDM. Prentice-Hall (1990) Second edition.

[26] Woodman, M., Heal, B.: Introduction to VDM. McGraw-Hill (1993).

[27] Fitzgerald, J., Gorm Larsen, P.: Modelling Systems: Practical Tools and Techniques for Software Development. Cambridge University Press (1998)
[28] Dawes, J.: The VDM-SL Reference Guide. UCL Press/Pitman Publishing, London (1991).
[29] ISO/IEC 13817-1:1996 Amended by INCITS/ISO/IEC 13817-1-1996: Vienna Development Method – Specification Language – Part 1: Base language. (1996).
[30] The RAISE Language Group: The RAISE Specification Language. The BCS Practitioners Series. Prentice-Hall (1992)
[31] RAISE Method Group: The RAISE Method Manual. Prentice Hall (1995).
[32] Bjorner, D.: Software Engineering. Springer (2006) (Vols. 1-3).
[33] Futatsugi, K., Goguen, J., Jounnaud, J.P., Meseguer, J.: Principles of OBJ-2. In: Formal Methods, A.C.M. (1985) 52–66
[34] Hoare, C.A.R.: Communicating Sequential Processes. Prentice-Hall (1985).
[35] Paulson, L.C.: ML for the Working Programmer. C.U.P. (1996) 2nd. Edition.
[36] Lynch, N., Tutle, M.: Hierarchical Correctness Proofs for Distributed Algorithms. In: A.C.M. Symposium on Principles of Distributed Computing, A.C.M. (1987) 137–151. Also M.I.T. Tech. Report MIT/LCS/TR-387.
[37] Lynch, N., Vaandrager, F.: Forward and Backward Simulations—Part I: Untimed Systems. Inf. and Comp. 121(2) (1995) 214–233 Also M.I.T. Tech. Memo MIT/LCS/TM-486.b.
[38] Lynch, N., Merritt, M., Weihl, W., Fekete, A.: Atomic Transactions. Morgan Kaufmann (1994).
[39] Lynch, N.: Distributed Algorithms. Morgan Kaufmann (1996).
[40] Lamport, L.: Specifying Systems, the TLA+ Language and Tools for Hardware and Software Engineers. Addison-Wesley (2002).
[41] Lamport, L.: The Temporal Logic of Actions. A.C.M. Trans. Prog. Lang. Sys. 16(3) (1994) 872–923.
[42] Abadi, M., Lamport, L.: The Existence of Refinement Mappings. Theoretical Computer Science 82 (1991) 253–284.
[43] Derrick, J., Boiten, E.: Relational Concurrent Refinement. Form. Asp. Comp. 15 (2003) 182–214.
[44] Nipkow, T.: Non-deterministic Data Types: Models and Implementations. Acta Inf. 22(6) (1986) 629–661.
[45] Leuschel, M., Butler, M.: ProB: A Model Checker for B. In Araki, K., Gnesi, S., Mandrioli, D., eds.: Formal Methods 2003. Volume 2805 of LNCS., Springer (2003) 855–874.
[46] Behm, P., Benoit, P., Faivre, A., Meynadier, J.M.: Météor: A Successful Application of B in a Large Project. In Wing, J. and Woodcock, J. and Davies. J, ed.: Formal Methods 1999. Volumes 1708, 1709 of LNCS., Springer (1999) 369–387.
[47] Owe, O.: Partial Logics Reconsidered: A Conservative Approach. Form. Asp. Comp. 3 (1993) 1–16.
[48] Banach, R., Poppleton, M.: Retrenchment: An Engineering Variation on Refinement. In Bert, D., ed.: 2nd International B Conference. Volume 1393 of LNCS., Springer (1998) 129–147.
[49] Jacob, J.L.: Basic Theorems about Security. J. Computer Security 1 (1992) 385–411.
[50] Banach, R., Poppleton, M., Jeske, C., Stepney, S.: Retrenching the Purse: Finite Sequence Numbers, and the Tower Pattern. In Fitzgerald, J., Hayes, I., Tarlecki, A., eds.: Formal Methods 2006. LNCS, Springer (2005) 382–398.
[51] Banach, R., Jeske, C., Poppleton, M., Stepney, S.: Retrenching the Purse: Finite Exception Logs, and Validating the Small. In Hinchey, M., ed.: IEEE/NASA Software Engineering Workshop 30-06. (2005) 234–245.
[52] Banach, R., Jeske, C., Poppleton, M., Stepney, S.: Retrenching the Purse: Hashing Injective CLEAR Codes, and Security Properties. In: IEEE ISOLA 2006 (2006) 82–90.
[53] Banach, R., Poppleton, M., Jeske, C., Stepney, S.: Retrenching the Purse: The Balance Enquiry Quandary, and Generalised and (1,1) Forward Refinements. Fundamenta Informaticae. 77 (2007) 29–69.
[54] Jeske, C.: Algebraic Integration of Retrenchment and Refinement. PhD thesis, University of Manchester (2005).
[55] Banach, R., Jeske, C.: Retrenchment and Refinement Interworking: the Tower Theorems. Mathematical Structures in Computer Science. **25** (2015), 135–202.

[56] Banach, R.: UseCase-wise Development: Retrenchment for Event-B. In ABZ 2008: Volume 5238 of LNCS., Springer (2008) 167–180.

[57] Banach, R.: Retrenchment for Event-B: UseCase-wise Development and Rodin Integration. Form. Asp. Comp., 23 (2011) 113-131.

[58] Fraser, S., Banach, R.: Configurable Proof Obligations in the Frog Toolkit. In: 5th IEEE International Conference on Software Engineering and Formal Methods. IEEE Computer Society Press, IEEE (2007) 361–370.

[59] Fraser, S.: Mechanized Support for Retrenchment. PhD thesis, School of Computer Science, University of Manchester (2008).

[60] Rushby, J.: Harnessing Disruptive Innovation in Formal Verification. In: 4th IEEE International Conference on Software Engineering and Formal Methods. IEEE Computer Society Press, IEEE (2006) 21–28.