
The Mondex Purse: Requirements and Retrenchments

Richard Banach

School of Computer Science,
University of Manchester,
Manchester M13 9PL, UK.
`banach@cs.man.ac.uk`

1 Introduction

Few human activities challenge our skill in distinguishing between the essential and the less important than the modelling of an aspect of the real world within a mathematically precise framework. Some such tasks are easier than others. For example, when Newtonian mechanics is used to model the behaviour of celestial objects (eg. the planets in our solar system), it is not hard to figure out what the crucial aspects of the situation are. In the solar system scenario we have several facts to fall back on: (1) the sun is so heavy that to an excellent approximation one can regard it as stationary; (2) the planets go round the sun, to an excellent approximation independently of each other; (3) if one wants greater accuracy, it is sufficient to calculate the perturbations attributable to Jupiter; (4) everything else can be ignored (unless one is interested in extreme accuracy).

It is almost the case that one can *define* physics as that portion of human inquiry in which the interactions between the agents involved distill into layers, well separated according to strength, and at each level the interactions between the relevant agents are sufficiently straightforward that analytical techniques yield good progress — *almost* that is, since the advent and increasing power of computer modelling now enables the predictions of analytical models (and, increasingly, stochastic ones) to be checked far beyond the limits imposed earlier by the capabilities of human calculation.

But such well organised domains constitute a fraction of the phenomena that we deal with in the real world. Very often, it is the case that systems are *complex*, consisting of large communities of agents, interconnected to each other in ways that do not permit the neglecting of all but a few key interactions. With such systems, the challenge of knowing what must be included in a model and what can be safely left out, can be a good deal more severe.

In many cases of complex systems, computers necessarily interact with the system in an essential way. Since computers are undoubtedly complex

themselves, they just add to the problem — with the digital system complexity just increasing the intrinsic external complexity. Thus the entire system needs to be modelled, analysed —ultimately verified and validated— if a significant degree of reliance is to be placed on it. And the greater the degree of reliance needed, the more pressing the need for dependable analysis. Ultimately this forces the uptake of rigorous, mathematically based, techniques, since history has shown that mathematically based techniques do, when properly used, yield the highest levels of predictability and consistency that human science and engineering have ever achieved.

Therefore, when confronting the problem of the dependable design of complex computer systems, we are faced with the dual problems of firstly deciding where we can safely draw the boundary between what is modelled and what is not modelled, and secondly of doing the modelling itself in an accurate and reliable way. The present paper is concerned with these two issues in the context of the Mondex Electronic Purse [SCW00]. Since the second issue (the formal modelling and verification) is by now an intensively studied area due to its prominence as the first major case study in the Verification Grand Challenge [JOW06, Woo06, WB07], we concentrate on the first of them, focusing on how retrenchment can bridge the gap between what is captured formally and what is left out, and showing how this can be connected to the formal work done for Mondex.

The remainder of the paper is as follows. In Section 2 we overview some aspects of model based refinement, the rigorous technique employed in the formal development of Mondex. In Section 3 we discuss retrenchment, designed to address some of the ‘cons’ that arise in the ‘pros and cons’ that emerge from model based refinement. In Section 4 we discuss the *Tower Pattern*, the technique that builds the technical bridge between the refinement models and the remaining aspects catered for using retrenchment. In Section 5 next, we overview the Mondex Purse, pointing to the extensive verification work done to date. Section 6 looks at generalisations that the non-atomic nature of the Mondex protocol has sparked. In Section 7, we overview the scope for using retrenchment in the context of Mondex, to handle issues ‘at the edges’ of the formal modelling, the so-called Mondex ‘retrenchment opportunities’. These, five in number, are examined in more detail in the five succeeding sections. Section 13 concludes.

2 Model based Refinement: Pros and Cons

In the field of digital systems, model based refinement is well known as the standard industrial strength technique for rigorously progressing abstract system designs towards implementations. The abstract designs are typically expressed in a modelling language permitting the maximum of expressivity, abstraction, mathematical rigour and succinctness, without concern for executability. The lower level models lean increasingly towards the actual

capabilities of real computing devices, and the algorithms that they must utilise. There are a number of specific formulations of model based refinement, which can differ as regards particular technical details, but which share the same overall strategy for establishing the correctness of an implementation: namely that for every run of the concrete system, there must be a run of the abstract system which maintains the desired notion of correct correspondence between them. Among the more well known techniques we can mention Z [Spi92, WD96, ISO02], B [Abr96, Sch01b, LH96] (and its contemporary offspring Event-B [Abr, Roda, Rodb]), VDM [Daw91, Jon90, FGL98], RAISE [RAI95, VGJM02] and ASM [BS03, Bör03, Sch01a, Sch05].

Besides being well established in the academic sphere, refinement has had notable successes on the industrial front in recent years. For Z we can cite not only the Mondex Purse itself [SCW98, SCW00], but also the Multos Operating System [SC00, Ste01], and numerous other projects not in the public domain. For B, we can cite the MÉTÉOR Project [BDJM00] and numerous other railway system projects in France and elsewhere. For ASM we can cite a number of language definitions and language abstract machine definitions [SSB00, BR94, GH93].

Despite these undoubted successes, practitioners have known for some time that when refinement is used as the sole means of progressing from an abstract model to a concrete one, then certain difficulties can plague the development process due to the exacting nature of typical refinement proof obligations. This is not a technical difficulty with refinement, rather it is a manifestation of a human inclination to view certain things as abstractions/concretisations of the same phenomenon, regardless of the fact that a given refinement formalism does not sustain such a view. Since the human notion of abstraction is inevitably imprecise, and the mathematical notion of abstraction pertaining to any specific refinement formalism is *de facto* extremely precise, some dislocation between the two is bound to occur sometimes.

Usually, if the scale of the problem is small, this dislocation can be overcome easily enough. Frequently it is sufficient to make some small adjustment to one or other of an abstract/concrete pair of models to bring them into line. However, when the problem size is large, such manipulations can become prohibitive; this may be on grounds of sheer cost, or on more prosaic grounds. For large problems, there are usually stakeholders other than the refinement specialists involved, who ‘own’ the models in question, and they may simply not agree to changes in the models as suggested by the refinement specialists, regardless of the latter’s protestations. Thus the human aspects of the development milieu become paramount. This is nothing more than a corollary of the fact that the construction of large systems is an engineering problem, and not purely a problem in formal system construction. The key desiderata in the two domains are just different.

A simple and commonly occurring example arises with natural number arithmetic. Implementable whole numbers are invariably finite. So arithmetic always generates within-bounds and out-of-bounds cases. If there are n dif-

ferent quantities in the model, then for a typical operation there will be one all-within-bounds case, and easily of the order of $2^n - 1$ or more out-of-bounds cases to consider. In the system specification, the syntactic descriptions of the latter can often swamp that of the single all-within-bounds case, which is the one of most interest. For such reasons it is often desirable to idealise the arithmetic and use unbounded naturals at the abstract level; these cause no exceptions. Unfortunately in the overwhelming majority of refinement formalisms there is no refinement from unbounded naturals to bounded ones that handles a sensible selection of the operations that are normally needed.

3 Retrenchment

Retrenchment [BP98, BP99, BPJS07, BJP08, Ret] was introduced in order to be able to address the difficulties caused by the situation discussed above, and many others in which the humans' idea of 'abstraction' is in conflict with what is permitted by some notion of model based refinement. It proceeds by inverting the usual trajectory from broad principles to proof obligations found in refinement. In refinement, the starting point is a notion of correct correspondence between abstract and concrete models, from which proof obligations are derived. In retrenchment by contrast, the proof obligations are manipulated so that they can encompass situations like the example above, and whatever broad principles can be derived therefrom, are sought. Certainly the typical guarantees offered by refinement are forfeit.

To illustrate retrenchment, we take a forward refinement operation proof obligation and turn it into a retrenchment proof obligation. For the former we take the following:

$$\begin{aligned} R(u, v) \wedge RIn_{Op}(i, j) \wedge \text{pre}(Op_A)(u, i) \wedge Op_C(v, j, v', p) \Rightarrow \\ (\exists u', o \bullet Op_A(u, i, u', o) \wedge R(u', v') \wedge ROut_{Op}(o, p)) \end{aligned} \quad (1)$$

In the above, u, v are (Abstract/Concrete) states (primed for after-states), i, j are (Abstract/Concrete) inputs, o, p are (Abstract/Concrete) outputs. R is the retrieve relation, while $RIn_{Op}, ROut_{Op}$ are input/output relations respectively. Especially when strengthened by suitable assumptions about $R, RIn_{Op}, ROut_{Op}$, the above is equivalent to typical operation POs found in the literature. To turn (1) into a retrenchment PO, we modify it to:

$$\begin{aligned} R(u, v) \wedge W_{Op}(i, j, u, v) \wedge Op_C(v, j, v', p) \Rightarrow \\ (\exists u', o \bullet Op_A(u, i, u', o) \wedge ((R(u', v') \wedge Op_C(o, p; u', v', i, j, u, v)) \vee \\ C_{Op}(u', v', o, p; i, j, u, v))) \end{aligned} \quad (2)$$

In (2), the earlier $RIn_{Op} \wedge \text{pre}(Op_A)$ has been generalised to the within relation $W_{Op}(i, j, u, v)$, which is an arbitrary relation in the before-values. The earlier $ROut_{Op}$ has been generalised to the retrenchment output relation

$O_{Op}(o, p; u', v', i, j, u, v)$, which now allows all the variables to occur. And, most importantly, there is a concedes relation $C_{Op}(u', v', o, p; i, j, u, v)$ to describe what happens if the retrieve relation cannot be re-established by a given pair of (Abstract/Concrete) steps.¹ From such a starting point, we derive what broad principles we can.

To the above we add a stipulation about intialisation which is identical to what is demanded in most forms of refinement:

$$Init_C(v') \Rightarrow \exists u' \bullet Init_A(u') \wedge R(u', v') \quad (3)$$

Formally, as regards retrenchment, we are done. The two conditions (3) and (2) express all that is needed to define retrenchment. This is in contrast to what one sees in a specific formulation of refinement, where there are typically a host more conditions that need to be satisfied in order to guarantee the detailed notion of correctness that is being demanded. See [BPJS07, Ban08] for a more detailed discussion.

The flexibility introduced into formal development by retrenchment lends itself to many uses. At one extreme, it can be restricted to the kind of situation outlined above, namely handling irritating restrictions forced onto the development by the finiteness or other limitations of implementable data types [BP98]. At the other extreme, it can be used to capture very general evolution of system definitions, as new considerations impact preliminary models on the route to the final system description [BP03]. How far along this scale of possibilities one happens to be, depends on one's perspective about a particular change in system description. The same change in the system might be viewed by one person as a system evolution, and thus as residing firmly in the requirements engineering arena, while for another person, it could be very much tied up with the road to an implementation, and thus be viewed as a development step; much can depend on whether the individual is focused on user needs, or technology capabilities. It is important to see that from a purely engineering vantage point, there is no hard and fast boundary between these two activities: requirements evolution can blend smoothly into development and implementation. This is in contrast to the formal refinement vantage point in which the boundary is clear: anything that cannot be captured by a refinement must be a requirements evolution step. Retrenchment can build a dialogue between these two perspectives.

4 The Tower Pattern

Thus far retrenchment has been introduced as a seemingly *ad hoc* modification of some refinement PO, and, as far as it goes, this is the case. It is not clear from this whether or not the two techniques can co-operate, and if so, how.

¹ The semicolons in O_{Op}, C_{Op} are purely cosmetic, separating the variables of 'most interest' from others, which are permitted, but less often needed.

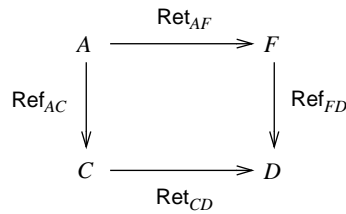


Fig. 1. The Tower Pattern

It is obvious that to maximise the benefits of both retrenchment and refinement for developers, it is vital that the two techniques work smoothly together in a well understood way. Consider an example. In the preliminary stages of design, one typically works with over-idealised modelling ingredients (such as unbounded natural numbers rather than finite ones, as mentioned above) in order to make the initial conceptualisation easy and perspicuous. (Data types without constraints usually have syntactically more compact axiom systems, which makes them easier to grasp, and to work with — hence the appeal of unbounded naturals etc.) Various concretisation steps can then be taken from the initial model, all still in the idealised sphere, and described by refinements. Subsequently one needs to take into account the finite nature of computing resources, and one must modify the preceding in the light of such constraints. This is usually best done via a retrenchment.

Formally one would do this as follows. The first, idealised part of the development, would be done via refinement, as illustrated in the downward vertical path from system model A to system model C in the left hand side of Fig. 1 say. This would be followed by a retrenchment from system model C to system model D in Fig. 1. As a sanity check, one would like to confirm that the development could actually have been done using a bona fide refinement. But this would entail going from system model A to some other system model first, since system model A will not be refinable to D in general. The needed system model, F , is also shown in Fig. 1. System model F is both a retrenchment of system model A , and an abstraction (i.e. inverse refinement) of system model D . So system model F is a solution to a ‘square completion’ problem, and system model F defines quite precisely the abstract system model of which system model D is an implementation.

On a more technically detailed level, the two paths round the square in Fig. 1 must commute, i.e. the composition of the A to C refinement with the C to D retrenchment must be equal to the composition of the A to F retrenchment with the F to D refinement. Such compositions of retrenchments with refinements (which in all cases yield retrenchments) are explored in [BJP08]. Moreover we want system model F to be the ‘best’ model to fulfill the commuting desideratum, i.e. we want system model F to be a canonical completion of the square partially defined by the $A \rightarrow C \rightarrow D$ L-shape. This

aspect gets technically intricate. In [Jes05], a whole host of similar square completion problems are investigated within a consistent framework.

Square completions enable the building of the Tower Pattern. This is essentially just a collection of such commuting squares, in which adjacent squares abut one another, sharing a side. In the obvious manner, one can form a grid pattern of such squares, appropriate to the development at hand. The theorems of [Jes05] enable towers to be built in many different ways. Essentially, what is needed is just one path of refinement and retrenchment development steps between diagonally opposite corners of the prospective tower, and the theorems do the rest. In all cases, given an overall rectangular grid, vertically distributed collections of horizontal retrenchment ‘rungs’ connect pairs of refinement development ‘columns’ that treat the same (or similar) set(s) of requirements at different (and refinement-theoretically incompatible) levels of detail — Fig. 1 is just the simplest such shape. The particularly appealing aspect of this approach is that the same tower constructions work regardless of the nature of the precise requirements issue(s) being captured by the constituent retrenchments. We noted above the capacity of retrenchment to capture a wide variety of requirements issues, so the tower gains extremely wide applicability thereby, as we confirm below.

5 The Mondex Purse and its Refinements

We focus now on the Mondex Purse [SCW00], mentioned above. This was an industrial scale development of an electronic purse smartcard application, enabling financial transactions to be performed without the use of any physical money. What this meant was that Mondex money was intended to be like cash in the real world, i.e. Mondex money was ingeniously designed to be non-forgable, so as to maintain the confidence of both its users in the field, and of the bank that ultimately underwrote the value contained in a Mondex purse. But, again like cash, Mondex money was not responsible *of itself*, for whether or not the transactions it participated in were honest or fraudulent, or were transacted by parties who genuinely were who they declared themselves to be to each other, etc.

Central to the certification of Mondex, which achieved an ITSEC level E6 rating [Dep91] (equivalent these days to Common Criteria EAL 7, [ISO05]), is a refinement that goes from an atomic operation for money transfer between two purses, to a distributed algorithm that fairly closely models the code for the money transfer protocol actually used (and that had previously been written).² Lately, this refinement, the essential ingredient of what is basically one of the very few detailed formal developments of a commercial product documented in the public domain, has been the subject of intense study using

² As often happened in the early days of system engineering, the implementation preceded the abstraction.

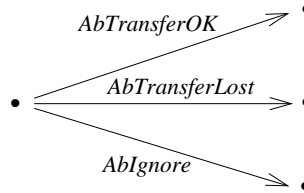


Fig. 2. The Mondex atomic protocol.

mechanical verification tools of various kinds, this activity constituting the first case study in the Verification Grand Challenge [JOW06, Woo06, WB07]. A detailed account of the various mechanised verifications attempted can be found in [JWe08], and further work is ongoing (see eg. [SB08]). The fact that refinement was so central a feature of the development was fully in line with our contention above, that mathematical techniques ultimately deliver the highest levels of dependability when properly used. The cited references, especially [SCW00] and [JWe08] thoroughly document this side of the development.

The Mondex development of [SCW00] consists of three system models: the *A*(bstract) model, the *B*(etween) model, and the *C*(oncrete) model, together with refinements from *A* to *B* and *B* to *C*; the latter two, when composed, yields a refinement from *A* to *C*.

The *A* model is an atomic model, in which atomic actions describe the permitted overall outcomes of playouts of the underlying detailed protocol. It is illustrated in Fig. 2 which shows the three possible events at this level: *AbTransferOK*, *AbTransferLost*, *AbIgnore*. *AbTransferOK* describes a transfer which succeeds, i.e. the agreed sum of money is safely transmitted from the sender (the *From* purse) to the recipient (the *To* purse). *AbTransferLost* describes a transfer in which the money is lost, but its definition insists that we *know* that the money is lost, permitting offline recovery later. Finally, *AbIgnore* skips.

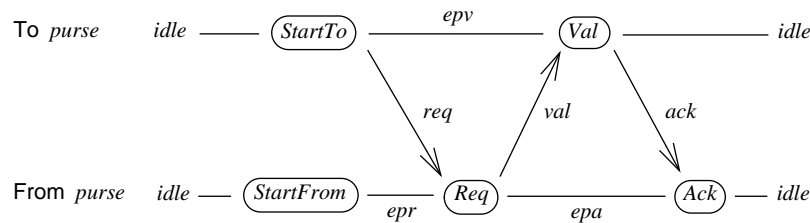


Fig. 3. The Mondex concrete protocol.

One absolute requirement that the *A* model embodies, is the prohibition of *failing but non-recoverable* transfers. Considering that real money is involved, this is as we would wish.

The *B* model captures the message passing protocol implemented in purses at a suitable level of abstraction. Fig. 3 shows how it works. The protocol starts its run involving a previously identified pair of purses (the *From* purse and the *To* purse) by calling the *StartFrom* and *StartTo* events in the respective purses, assuming that both purses are in an *idle* state. These two events prime both purses with the information needed to execute the protocol, including in particular, for each purse, the information directly pertaining to its counterpart. This puts both purses in a position to check the extent to which (to the best of the available local knowledge) the playout of a running protocol instance conforms to what is expected at that point. This reconciling of actual against expected is what that gives the Mondex protocol its recoverability properties in the face of protocol failure and interruption.

Once the *Start* events take place, the protocol proper commences. As part of *StartTo*, the *To* purse issues a (cryptographically protected) *req* (request) message to the *From* purse and enters the *epv* (expecting payment value) state. On receipt of the *req* message, the *From* purse, which has been in the *epr* (expecting payment request) state since its *StartFrom*, executes the *Req* (Request) event, decrements its balance appropriately, and sends the amount requested in a (cryptographically protected) *val* (value) message to the *To* purse, itself going into the *epa* (expecting payment acknowledgement) state. On arrival of the *val* message, the *To* purse, executes the *Val* (Value) event, increments its balance appropriately, becomes *idle* again, and sends a (cryptographically protected) *ack* (acknowledgement) message back to the *From* purse. When this finally arrives, the *From* purse, executes the *Ack* (Acknowledgement) event, and the protocol completes with both purses becoming *idle* once more. In addition to this just described ideal protocol run outline, are numerous cases corresponding to failed protocol runs. In reality, any of the messages we mentioned may get lost in transit, and any of the events that produce and/or consume them may fail to take place. Tying off the loose ends in the protocol generated thereby, is the *Abort* event, which unconditionally cleans up any partially completed protocol run, logging any information needed for recovery, and resets the relevant purse to the *idle* state. (*Abort* is called in each purse at the start of every protocol run, as a precaution in case the purse is still waiting for some previous transfer to complete.) The recoverability properties of Mondex are attributable to the rather subtle properties of the *Abort* event.

While the *B* model captures the essence of the protocol, its precise formulation requires the assumption of a number of properties of the detailed purse world, that are needed for the refinement proof from *A* to *B*, but which cannot be assumed to hold *a priori*. The job of the *C* model is to deal with this aspect. It is like the *B* model, but weakens the needed assumptions about the purse world to ones which can reasonably be assumed to hold in a practical

context, and shows that the additional assumptions are inductive properties of runs of the whole purse world model. In this way the refinement from the atomic to the concrete is established.

There is one unassailable fact about the situation we have outlined. The atomic model covers the entire protocol in a single event, whereas the protocol takes several steps to do so. However, refinement, at least as we described it earlier, makes a single abstract action correspond to a single concrete one. How then is the single abstract event to be connected to the several concrete ones of Mondex? The answer, unsurprisingly, is that one of the several concrete events must do the job of refining the abstract event, and the remaining concrete events must refine *skip* (or *AbIgnore* in Mondex-speak).

In [SCW00], the *Req* concrete event has the responsibility of refining the *AbTransferOK* and *AbTransferLost* events, and the remaining concrete events refine *AbIgnore*. Since *AbIgnore* does not change the state, spurious *AbIgnores* not corresponding to any distinct transactions of their own in the concrete world (but merely acting as technical helpers in the refinement of other transactions) do no harm.

Since the outcome of the protocol (in terms of success or failure) is still ambiguous at the *Req* stage, the refinement of the *A* model to the *B* model needs to be a backwards one. However, once this major change in abstraction level and granularity has been achieved, the remaining work, the inductive proof that the *B* model's additional properties are invariants of a more simply constituted *C* model, can be done with a forwards *B* to *C* refinement.

6 Mondex and Atomicity

Since [SCW00], with its essential use of backwards refinement, people wondered about the possibilities for a completely forwards refinement from *A* to *C*, and the first such treatment appeared in [BJPS07]. Since then, several related variations on the forwards refinement theme for Mondex have appeared, many being documented in [JWe08].

The essential idea goes as follows. Any time an expected message fails to arrive, the purse in question eventually *Aborts*. If the message is either *req* or *ack* then nothing is really amiss, the money is safe. Either it has not departed (in the *req* case) or it has arrived safely (in the *ack* case), and the *Abort* merely represents ignorance on the part of the other purse. However if the *val* message is lost, then that is serious, since it contained the money being transferred. In that case *both* purses *Abort* the transfer (the *To* purse because the *val* it is expecting never arrives, the *From* purse because the *ack* it is expecting never arrives), and this situation needs to be recovered eventually. It is clear that a somewhat subtle case analysis of *Abort* events is required to distinguish this case from others. However once this is done, one of the *Abort* events in the double-*Abort* scenario can refine the atomic *AbTransferLost*, and *AbTransferOK* can then be refined by the *Val* event.

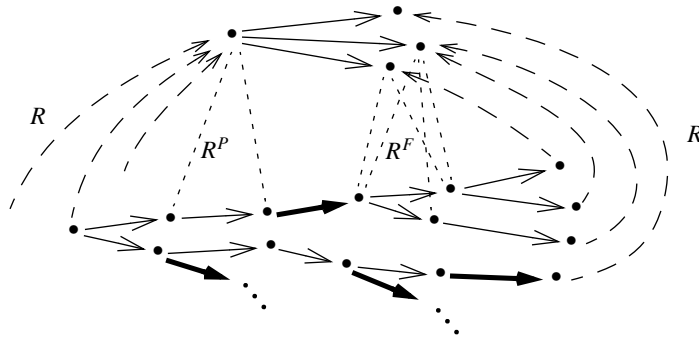


Fig. 4. Refining an isolated atomic protocol.

Since the non-determinism inherent in the *Abort* events is now resolved late, the whole can be arranged as a forwards refinement.

Consideration of these aspects of Mondex has generated interest in the general question of the possibilities for choosing forwards and backwards refinements in the context of ‘isolated protocols’ such as Mondex. A general treatment has appeared in in [BS07], extended in [BS08].

In fact the general picture is quite simple to describe. At the top of Fig. 4 we see an atomic action (having several distinct outcomes). Below it we see an isolated protocol, represented by its computation tree (we could use a DAG too). Relating the two is a functional retrieve relation R from the protocol initial and final states to the atomic initial and final states (R being undefined on non-initial/final states). This gives a ‘big diagram’ refinement of the atomic action to the protocol, the ‘big’ referring to the fact that the refinement does not enquire as to how the protocol internal states and activities relate to the atomic ones.

To derive a ‘small diagram’ refinement, in which the individual steps of the protocol are related to the atomic action too, we introduce a *synchronisation assignment* (SA). An SA is a choice among the steps in the concrete protocol computation tree, such that each maximal path through the tree contains exactly one SA step (the SA steps are shown bold in Fig. 4). With such a choice of SA, we can generically *calculate* the needed internal retrieve relations from the ingredients mentioned (represented in Fig. 4 by R^P and R^F , standing for ‘past’ and ‘future’ relative to the SA step). This is a big advantage since the *ad hoc* derivation of R^P and R^F is arduous for non-trivial protocols.

The simple picture in Fig. 4 conceals a small complication. In the real world, protocol instances do not necessarily have ‘clean starts’ and ‘clean ends’, as in Fig. 4. Specifically, an agent that is not needed till some way into the protocol, is at liberty to be engaged in unrelated activities up to the point of need. This makes the simple functional nature of R unrepresentative of reality, and adds to the already significant complexity of deriving retrieve

relations in a ‘monolithic’ manner. The formulation in [BS08] separates these concerns, bringing much needed clarity to the development via refinement of protocols and their instantiations within system runs.

7 Mondex Requirements, and Retrenchments

In Section 5, we presented the Mondex design as a *fait accompli*. In particular, we gave it as if the requirements embodied in the design broached no alternative. However, despite the impeccable nature of the refinement, so roundly vindicated in the studies in [JWe08], Mondex contained a number of requirements issues that could have been handled in more than one way, and which, actually, were dealt with in a way that one can view as less than ideal. Such issues give rise to a number of ‘retrenchment opportunities’, pertaining to gaps between the refinement models and the reality of the code. These constitute an excellent testbed for exercising the capabilities of the retrenchment framework in dealing with genuine requirements uncertainties in a realistic application. We summarise these below, treating them individually in subsequent sections.

Sequence Number: The integrity of the Mondex concrete protocol depends partly on a unique global transaction identifier to help protect against replay attacks. In reality, the global transaction identifier is modelled as a quadruple consisting of the two participating purses’ unique purse identifiers, together with the two purses’ individual sequence numbers for the transaction in progress. Sequence numbers occur in the Mondex *B* and *C* models where they are naturals. In reality they are bounded numbers.

Log Full: Transfers completing abnormally are logged by purses. The Mondex concrete model implements the abstract ‘lost value’ component as a rather complicated expression involving individual purses’ exception log entries. Some of these are held in the purses themselves, the remainder are held in a central archive. An individual purse needs to be assured that the data in its local log is stored in the archive before it can clear it from its own, highly constrained, log memory. Logs occur in the Mondex *B* and *C* models where they are unbounded sets. In reality they are finite.

Hash Function: Clearing a purse’s log after its contents are centrally archived is done via a message containing a `CLEAR` code. The purse log contents are assumed to be in total injective correspondence with the `CLEAR` codes, as that property is required in the proof of the security properties. In reality of course a cryptographic hash function is used, which is neither total, nor injective, but is informally argued to be ‘sufficiently injective’.

Balance Enquiry: Each purse has a balance enquiry operation. If this is invoked at a particular point in the middle of a concrete model value transfer, a discrepancy can occur between the abstract and concrete balances obtained, due to differences in where the nondeterminism is resolved in the two models.

This is handled formally by a modelling trick, using finalisation instead of the enquiry operation to observe the state.

DOS: In the concrete protocol, *StartFrom* and *StartTo* are independently called from the environment. This means that their parameters (including the transaction's identifying data, which includes purse sequence numbers) are unencrypted and, in particular, permit two kinds of denial of service attack on Mondex purses. During the original development, these were noted but judged not to be a serious threat for realistic use of purses.

8 The Sequence Number Retrenchment

The sequence numbers we mentioned are unbounded naturals in the *C* model, but obviously, their actual implementation in the Mondex code uses a finite number of bits, imposing an overall maximum size. The discrepancy between the two is relatively easy to address using retrenchment. We sketch the main ingredients of the development.

As stated earlier, the Mondex development proceeds from the *A* model to the *C* model, taking us down the left hand side of Fig. 1. To tackle the boundedness of the sequence numbers, we build a new model (the *D* model), in which the sequence numbers are bounded and appropriate actions are defined in case the bound is breached. We illustrate this by showing how the simplest Mondex operation that manipulates the sequence number, *CIncreasePurseOkay* (the *C* prefix being our convention for renaming apart entities in the various models for clarity) gets modified in the *D* model:

$$\begin{array}{c}
 \boxed{
 \begin{array}{l}
 CConPurseIncrease == \\
 CConPurse \setminus (CnextSeqNo) \\
 \\
 \hline
 CIncreasePurseOkay \text{ —————} \\
 \Delta CConPurse \\
 Cm?, Cm! : CMESSAGE \\
 \\
 \hline
 \exists CConPurseIncrease \\
 CnextSeqNo' \geq CnextSeqNo \\
 Cm! = \perp
 \end{array}
 }
 \end{array}
 \quad
 \begin{array}{c}
 \boxed{
 \begin{array}{l}
 DIncreasePurseOkay \text{ —————} \\
 \Delta DConPurse \\
 Dm?, Dm! : DMESSAGE \\
 \\
 \hline
 \exists DConPurseIncrease \\
 (DnextSeqNo < BIGNUM \Rightarrow \\
 \quad DnextSeqNo' \geq DnextSeqNo \wedge \\
 \quad Dm! = \perp) \\
 (DnextSeqNo = BIGNUM \Rightarrow \\
 \quad DnextSeqNo' = DnextSeqNo \wedge \\
 \quad Dm! = DpurseBlocked \ Dname)
 \end{array}
 }
 \end{array}$$

In the above Z schemas, *CConPurseIncrease* specifies the *C* model state *aside from* the actual sequence number, *CnextSeqNo*. This is for the benefit of the $\exists CConPurseIncrease$ schema inside *CIncreasePurseOkay* which specifies that this part of the state is *not* to change. The part that changes is *CnextSeqNo* itself, and the body of *CIncreasePurseOkay* says that it may increase. (In reality, this operation is an abstraction of several operations in the Mondex code that do various kinds of bookkeeping, some of which need to increase the sequence number and some of which don't.)

Beside *CIncreasePurseOkay* is its *D* model counterpart, *DIincreasePurseOkay*. Its specification is more elaborate, since the introductions of arbitrary constraints into an otherwise unrestricted freely generated data structure (the naturals in this case) is always syntactically more complicated. We see that *DIincreasePurseOkay* effectively splits into the below-limit ($DnextSeqNo < BIGNUM$) and at-limit ($DnextSeqNo = BIGNUM$) cases. The former is just a renaming of the *C* model behaviour, while the latter issues a new message type “*DpurseBlocked Dname*” to inform the environment about the state of affairs. *DIincreasePurseOkay* also relies on the *DConPurseIncrease* subset of its state in the same way that *CIncreasePurseOkay* relies on *CConPurseIncrease* (with *DConPurseIncrease* obtained from *CConPurseIncrease* by changing the C prefix on everything to a D prefix, so we do not quote it explicitly).

Although we cannot relate the *D* model to the *C* model, via a refinement which makes the sequence numbers in the two models correspond in a natural way (i.e. via equality), it is straightforward enough to do so using a retrenchment. Next, we give the ingredients needed (i.e. the retrieve, within, output and concedes relations for the two *IncreasePurseOkay* operations):

$\frac{R_{CD} \text{ —————}}{CConPurse; DConPurse}$ $CDConPurseIncreaseEquality$ <hr style="border: 0.5px solid black;"/> $CnextSeqNo = DnextSeqNo$	$\frac{O_{CD, IncreasePurseOkay} \text{ —————}}{Cm! : CMESSAGE}$ $Dm! : DMESSAGE$ <hr style="border: 0.5px solid black;"/> $Cm! = Dm!$
$\frac{W_{CD, IncreasePurseOkay} \text{ —————}}{CConPurse; DConPurse}$ $Cm? : CMESSAGE$ $Dm? : DMESSAGE$	$\frac{C_{CD, IncreasePurseOkay} \text{ —————}}{CConPurse'; DConPurse'}$ $CDConPurseIncreaseEquality'$ $Cm! : CMESSAGE$ $Dm! : DMESSAGE$ <hr style="border: 0.5px solid black;"/> $CnextSeqNo' \geq DnextSeqNo'$ $Cm! = \perp$ $Dm! = DpurseBlocked Dname$

In the above, we see that the retrieve relation R_{CD} asserts the equality of the *C* and *D* sequence numbers. (Via *CDConPurseIncreaseEquality* it also asserts the equality between corresponding but differently named (because of the C and D prefixes) unneeded parts of the state.) The within relation $W_{CD, IncreasePurseOkay}$, is trivial (defaulting to true) as there are no constraints needing to be imposed on the applicability of the two operations. The output relation $O_{CD, IncreasePurseOkay}$, appropriate for the below-limit case, just asserts the equality of the messages output. The concedes relation $C_{CD, IncreasePurseOkay}$, appropriate for the at-limit case, states the relationship between the two models' sequence numbers (i.e. an inequality), and cites the messages output by each.

This all gives a nice account of matters for an individual purse. However, the true Mondex state is a world of many purses, interacting pairwise with each

other during transactions (whose participating purse pairs are determined by the environment). So the above account needs to be elaborated with the machinery to handle this aspect.

The standard Z technique for constructing a system state out of a number of copies of an individual subsystem is promotion [WD96,DB01]. In promotion, an indexing function from a set of identifiers to the state of an individual subsystem is used to build the whole system, as one would expect. The more intricate details of promotion are concerned with carefully distinguishing the whole system from the subsystem, yet making the needed identifications correctly. The original Mondex design [SCW00] was built using promotion, and a refinement of the whole system founded on the refinement of an individual subsystem factors cleanly through promotion. This was the basis of the Mondex refinements. We do not review these details here, both for lack of space and because they do not add much to the essential story.

The strategy of factoring cleanly through promotion can also be applied to a promoted retrenchment. So we can build a whole system retrenchment from C to D analogously to the whole system refinement from B to C . The two can then be composed, and we have the ingredients for an application of the tower. Thus we obtain the E model (like the B model, not shown in Fig. 1), and a commuting square formed by the B , C , D , E models.

It now turns out that the E model is after all a backwards refinement of the A model. This is attributable primarily to three things: the clean and minimally invasive way that the D model was built out of the C model; the ‘always available, if only to do skip’ policy for all Mondex operations; and the fact that there are no sequence numbers at all in the A model. So we can complete the tower with what is in this instance an *ad hoc* construction of the F model, identical aside from renaming to the A model, and shown at the top right of Fig. 1.

One beneficial side-effect of the retrenchment treatment of this requirements issue, is the fact that it generates an object in the formal analysis (the concedes relation of the C to D retrenchment) around which we can focus a validation. Essentially, since our approach to the finite limit is to ‘do nothing’ once it is reached, which makes a purse unusable thereafter, the main point that validation has to establish is that the limit will in fact not be reached in a purse’s lifetime. To this aim, we can observe that the increments of the sequence number can be phrased as the ‘arrivals’ of a renewal process, a well studied kind of stochastic process [GD01, Res92, KT75]. Profiting from the known properties of renewal processes, we can then set the limit appropriately high.

Of course, this degree of formality is not indispensable in the case of such a simple thing as a large numerical limit. Adding a few more bits to the numerical representation is both cheap and vastly increases the capacity. But the availability of the formal route shows the way that more intricate cases might be handled. A more detailed treatment of the sequence number case study can be found in [BPJS05].

9 The Log Full Retrenchment

As regards the finiteness of the purse logs, this has many aspects that resemble the previous case — mathematically, the constraint in question is identical. However, the requirements issues surrounding the log full scenario are very different. To start with, sequence numbers are just numbers, so can be encoded in binary, giving a huge capacity for the price of a few bits via binary encoding. Log entries are different. They each have to be stored individually, making the encoding analogous to unary, and furthermore, the log entries themselves are non-trivial data structures, making the ‘bits’ of the unary encoding rather fat. Add to this the severely constrained memory available to smartcards in the era of the Mondex development (the mid-90s), and we face a completely different requirements challenge.

The architecture of the retrenchment treatment of this requirements challenge is the same as previously. It will be treated via the tower, testifying to the tower’s ability to handle a wide range of requirements issues. Thus we start in the same place, the C model. This time the relevant Mondex operation is $CAbortPurseOkay$, since it is during the $Abort$ operation that log entries are created. Below we see its C model schema together with the $CLogIfNecessary$ schema it uses. We see that $CAbortPurseOkay$ includes a $\exists CConPurseAbort$ schema, which prevents change in the part of the state not needed by $CAbortPurseOkay$, as in $CIncreasePurseOkay$ previously. We see that the purse’s $Cstatus$ becomes $Cidle$, and that the sequence number $CnextSeqNo$ is incremented.³ We also see that $CLogIfNecessary$ adds an entry $CpdAuth$ (consisting of the Authenticated payment details) to the C model exception log $CexLog$, but *only* if the purse status is either $Cepv$ or $Cepa$.⁴

$ \begin{array}{l} \overline{CAbortPurseOkay} \\ \Delta CConPurse \\ Cm?, Cm! : CMESSAGE \\ \hline \exists CConPurseAbort \\ CLogIfNecessary \\ CnextSeqNo' \geq CnextSeqNo \\ Cstatus' = Cidle \end{array} $	where	$ \begin{array}{l} \overline{CLogIfNecessary} \\ \Delta CConPurse \\ \hline CexLog' = CexLog \cup \\ \quad (\text{if } Cstatus \in \{Cepv, Cepa\} \\ \quad \text{then } \{CpdAuth\} \text{ else } \emptyset) \end{array} $
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

In the sequence number retrenchment, we were content to simply have the purse fail to work if the limit was reached, confident that it would never happen in practice. Here, we cannot take that line, since the limit in question is *bound* to be reached in practice. (In the context of the physical constraints of the day, the Mondex exception log had room for only about five entries.)

³ Thus we see that the previous requirements scenario impacts here too, but we will ignore this aspect for simplicity.

⁴ If the status is $Cepv$ then either the *req* or *val* message might be in flight. If the status is $Cepa$ then either the *val* or *ack* message might be in flight. This reflects the local ignorance of the global protocol state discussed earlier.

Thus, for the D model operation $DAbortPurseOkay$, we need a different strategy, amounting to a three way case split. If in a call of $DAbortPurseOkay$ the log size ($\# DexLog$) is less than $LOGMAX - 1$, then everything works as in the C model. If the log size is exactly $LOGMAX - 1$, then the log entry is created (if appropriate), but in addition a “Blocked” message is issued to warn the user to get the accumulated log entries transferred to the central archive and the purse log cleared. Aside from that, the purse status becomes $DexLogFull$, a newly introduced purse status value, that indicated that the log is now full, and that no further transactions may be attempted (since there is now nowhere to put *their* details, should they go wrong).⁵ A last case (log size $\geq LOGMAX$) makes $DAbortPurseOkay$ always available.

$\frac{DAbortPurseOkay \quad \Delta DConPurse \quad Dm?, Dm! : DMESSAGE}{\exists DConPurseAbort \quad DnextSeqNo' \geq DnextSeqNo \quad (\# DexLog < LOGMAX \Rightarrow DLogIfNecessary) \quad (\# DexLog \geq LOGMAX \Rightarrow DexLog' = DexLog) \quad (\# DexLog < LOGMAX - 1 \Rightarrow Dstatus' = Didle \wedge Dm! = \perp) \quad (\# DexLog \geq LOGMAX - 1 \Rightarrow Dstatus' = DexLogFull \wedge Dm! = \text{“Blocked.”})}$	where $\frac{DLogIfNecessary \quad \Delta DConPurse}{DexLog' = DexLog \cup \quad (\text{if } Dstatus \in \{Depv, Depa\} \text{ then } \{DpdAuth\} \text{ else } \emptyset)}$
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Having fixed the D model, we turn to the C to D retrenchment below. The retrieve relation R_{CD} is a bit more complicated than in the sequence number case since it has to cater for the new purse status value $DexLogFull$. Its $CDConPurseAbortEquality$ schema plays the same role as the corresponding schema in the sequence number case. The within relation $W_{CD,AbortPurseOkay}$, and output relation $O_{CD,AbortPurseOkay}$, are both as in the sequence number case. Only the concedes relation $C_{CD,AbortPurseOkay}$ has some new work to do: it asserts the inclusion between the C and D logs once the limit is reached; it describes the discrepancy between purse status values; it asserts the equality between the corresponding sequence numbers (in our formulation), and describes the different messages that get issued in the log full case.

Now the remainder of the construction follows the sequence number example, and form an instance of the tower. As before, we first incorporate the individual purse details into the system model via promotion — the refinements and retrenchment duly respect the promotion structure and factor through it neatly. So we build an E model according to the preceding template. As in the sequence number case, careful design means that the E model turns out to be a backwards refinement of the A model, so we can complete

⁵ The new status value prevents new transactions, since all the critical protocol steps we described above are guarded on the purse being in the appropriate state, whereby they are disabled by the purse’s being in $DexLogFull$.

the construction by creating an F model as a (renamed) copy of the A model. See [BJPS06a] for a more detailed account.

R_{CD} $CConPurse; DConPurse$ $CDConPurseAbortEquality$	$O_{CD,AbortPurseOkay}$ $Cm! : CMESSAGE$ $Dm! : DMESSAGE$
$(Cstatus = Dstatus \vee$ $Dstatus = DexLogFull)$ $CnextSeqNo = DnextSeqNo$ $CexLog = DexLog$	$Cm! = Dm!$
$W_{CD,AbortPurseOkay}$ $CConPurse; DConPurse$ $Cm? : CMESSAGE$ $Dm? : DMESSAGE$	$C_{CD,AbortPurseOkay}$ $CConPurse'; DConPurse'$ $CDConPurseAbortEquality'$ $Cm! : CMESSAGE$ $Dm! : DMESSAGE$
	$CexLog' \supseteq DexLog'$ $Cstatus' = Cidle$ $Dstatus' = DexLogFull$ $CnextSeqNo' = DnextSeqNo'$ $Cm! = \perp$ $Dm! = \text{"Blocked."}$

Recall that both the the sequence number case and the log case feature a ‘finite limit’ phenomenon. A complete lower level model will therefore contain both phenomena (as well as other things discussed below) and the description of each operation will therefore break up into (at least) four cases depending on whether the sequence number and/or purse log are still within bounds. This is already an example of the $2^n - 1$ case proliferation noted in Section 2, even though only very simple operations are required for the two data items. It becomes clear that relegating the concerns regarding these details to a lower level model, as permitted by the retrenchment approach, leaving earlier models to concentrate on core functionality, is very worthwhile.

10 The Hash Function Retrenchment

Once the purse log has filled up following successive calls of $DAbortPurseOkay$, freezing the purse, the purse user must take steps to get the log cleared and its contents archived. Aside from restoring the purse to a usable state, clearing the log is needed for recovering any funds lost in transit, as tracing what has happened to such funds depends on correlating the corresponding log entries from both the **From** and the **To** purses — this requires both set of log entries to have reached a common place, the central archive. Since the latter applies even to the C model, archiving is needed irrespective of whether the size of the archive is modelled accurately or not.

Purse log clearance starts by a process of sending the log contents to the central archive via a secure link in the bank. This aspect of log clearance

is outside the scope of the hash function retrenchment and is assumed to have terminated successfully. (If some failure occurs, then the following step too will fail, and the whole process can be repeated.) Given that the purse contents have been safely copied to the central archive, the local log can be cleared. In the C model, the operation that clears a purse log according to [SCW00] is the marvelously named *ClearExceptionLogPurseEafromOkay*. Let us abbreviate this to *CClearPurseOkay* in this paper. The Z schema is:

$$\begin{array}{l}
 \hline
 \textit{CClearPurseOkay} \\
 \hline
 \Delta \textit{CConPurse} \\
 \textit{Cm?}, \textit{Cm!} : \textit{CMESSAGE} \\
 \hline
 \textit{Cm?} = \textit{CexceptionLogClear}(\textit{Cimage} \ \textit{CexLog}) \\
 \exists \textit{CConPurseClear} \\
 \textit{Cstatus} = \textit{Cidle} \qquad \text{where} \\
 \textit{CexLog} \neq \emptyset \qquad \textit{Cimage} : \mathbb{P}_1 \ \textit{CpayDetails} \mapsto \textit{CLEAR} \\
 \textit{CexLog}' = \emptyset \\
 \textit{Cm!} = \perp \\
 \hline
 \end{array}$$

In this, there is the usual $\exists \textit{CConPurseClear}$ schema, protecting irrelevant parts of the purse state from change. Besides this, the operation is guarded on the *Cidle* status, and by the receipt of the *CexceptionLogClear(Cimage CexLog)* message. The message tag *CexceptionLogClear* identifies this as a clear log message, while its contents ensure that the log is cleared only if it is safe to do so. Thus *Cimage CexLog* is the application of a total injective function *Cimage* to the entire log *CexLog*, yielding an individual CLEAR code. Since the message arrives from the archive and *Cimage* is injective, the purse can check that the archive truly contains precisely the log contents that it ought to, since if it did not, it would have been unable to offer the correct CLEAR code in the message. Once the credentials of the message have been confirmed by the purse, *CexLog* can be set to empty.⁶

In reality of course, there is no total injective function *Cimage*, and a hash function is used instead. We can capture this in a D model, embarking on a tower building exercise as in previous sections. The relevant schema becomes:

$$\begin{array}{l}
 \hline
 \textit{DClearPurseOkay} \\
 \hline
 \Delta \textit{DConPurse} \\
 \textit{Dm?}, \textit{Dm!} : \textit{DMESSAGE} \\
 \hline
 \textit{Dm?} = \textit{DexceptionLogClear}(\textit{Dimage} \ \textit{DexLog}) \\
 \exists \textit{DConPurseClear} \\
 \textit{Dstatus} = \textit{Didle} \qquad \text{where} \\
 \textit{DexLog} \neq \emptyset \qquad \textit{Dimage} : \mathbb{P}_1 \ \textit{DpayDetails} \rightarrow \textit{CLEAR} \\
 \textit{DexLog}' = \emptyset \\
 \textit{Dm!} = \perp \\
 \hline
 \end{array}$$

⁶ In [SCW00] the message also contains the purse name, for additional confirmation. We omit this from the present account in order to simplify the technical details.

The tiny change in signature of *Dimage* compared with *Cimage* allows *Dimage* to be a hash, and in particular allows the cardinality of *CLEAR* to be significantly smaller than the number of all possible log contents, an important consideration given the physical limitations of smartcards.

With the *D* model in place, we turn to the *C* to *D* retrenchment, given by the following schemas:

$\frac{R_{CD}}{CConPurse; DConPurse \\ CDConPurseHashEquality}$	$\frac{O_{CD,Clear}}{\Delta CConPurse; \Delta DConPurse \\ Cm! : CMESSAGE \\ Dm! : DMESSAGE}$
$\frac{W_{CD,Clear}}{CConPurse; DConPurse \\ Cm? : CMESSAGE \\ Dm? : DMESSAGE}$	$CexLog \subseteq Carchive \\ DexLog \subseteq Darchive \\ CDAllValueAccountedPurse' \\ Cm! = Dm!$
$Carchive \cup CexLog = \\ Darchive \cup DexLog \\ Cm? = CexceptionLogClear(\\ Cimage CexLog) \\ Dm? = DexceptionLogClear(\\ Dimage DexLog)$	$\frac{C_{CD,Clear}}{\Delta CConPurse; \Delta DConPurse \\ CDConPurseHashEquality' \\ Cm! : CMESSAGE \\ Dm! : DMESSAGE}$ $\neg(CexLog \subseteq Carchive \wedge \\ DexLog \subseteq Darchive) \\ \neg CDAllValueAccountedPurse' \\ Cm! = Dm!$

In this, *CDConPurseHashEquality* takes care of unneeded state in the two models, and is the only responsibility of the retrieve relation. In this retrenchment, the within relation is key. This asserts that the union of the local purse log and the (appropriate portion of the global) archive is the same in the two models, so all is well in the before-states of the two operations (since the *C* model cannot go wrong).

The output and concedes relations cover the possibilities for the after-states. If nothing went amiss when the logs were cleared, then it must have been the case that (in the before-states) the local log was a subset of the archive in both models, and the equality asserted in the within relation reduces to equality of the archives, as should be the case, causing the purse-wise security property *CDAllValueAccountedPurse* (which says that the system knows where all the money is) to be maintained. This is expressed in the output relation.

However, if there was some mismatch between the archives in the two models (caused perhaps by some log/archive transmission error after all) and this got concealed by the non-injectivity of the *D* model hash, and remained undetected in the log/archive union in the two models, then it was *not* the case that (in the before-states) the local log was a subset of the archive in both models,

a fact that the log/archive union (needed for the correspondence with the abstract model) was able to hide while the log remained uncleared. Log clearance exposes the discrepancy, invalidating the *CDAllValueAccountedPurse* security property once the log entries vanish. This is expressed in the *concedes* relation.

Note that, not knowing the underlying details, prevents us from deducing which of these possibilities takes place in a given instance. Thus the hash retrenchment enables us to make sensible statements about the failing situation despite knowing only that an injection has been replaced with a hash, making the $O_{CD,Clear}/C_{CD,Clear}$ choice non-constructive. All of this persists through promotion, and connects with failure of the system-wide *CDAllValueAccounted* security property. Finally one can supplement this account with a validation based on the statistical likelihood that the concession might be made true, i.e. that a purse receives in error a clear-log message with just the right properties to make it believable. See [BJPS06b] for a fuller account.

11 The Balance Enquiry Retrenchment

The preceding retrenchment opportunities were all ‘localised’ in that each of the discrepancies discussed could be viewed as being rooted in a single operation at a time.⁷ This one, the balance enquiry quandary, is more complicated, necessitating the consideration of the protocol as a whole.

As noted before, in the Mondex abstract model, the atomic transfer operation captures failing transactions as well as successful ones. This is so that even failing protocol transactions are able to be refinements of *something* abstract, this in turn following from the high standards of accountability demanded of all things financial. Thus in the abstract model, the nondeterminism between success and failure is resolved as soon as the operation runs.

In the concrete model, protocol failure is typically detected late, so the nondeterminism between success and failure is resolved late too. On the other hand, since the protocol has more than one essential event, the system state departs from the concrete *idle* state early, and is non-*idle* for an extended period. Unavoidably then, if one synchronises the atomic action with some specific event in the concrete protocol, there will be a period (or indeed two such periods) during which the relationship between abstract and concrete system states is not as one normally expects. In the context of Fig. 4, the normal relationship is R , and the abnormal relationship, in the interior of the protocol, is given by R^P and R^F . If one now introduces read-only operations on the abstract and concrete states, then in the middle of the protocol, one can observe the abnormal relationships described by R^P and R^F .

⁷ For example, even though sequence numbers are incremented within several purse operations, each such operation could be individually retrenched, without needing to refer to the others, and it was sufficient to discuss the simplest one to illustrate the point.

In the Mondex context, a balance enquiry operation is such a read-only operation on the state, and the repercussions of observing a non-standard relationship between abstract and concrete balances is superficially disconcerting. Suppose following [SCW00], we synchronise the abstract atomic action with the concrete *Req* operation. Suppose a transfer gets under way, and immediately after the *Req* operation, while the money is still in flight, we do abstract and concrete balance enquiries on the *To* purse. Obviously, the concrete one returns the original *To* balance since the money hasn't arrived yet. However at the abstract level, the transaction has already completed, so the *To* balance is already higher at this point. In other words we see different balances, hardly reassuring from a financial perspective.

Of course nothing is amiss really. The correspondence between abstract and concrete is really just a fiction — at best, only the concrete system is *real*. However the abstract model provides a powerful means of verifying that the concrete model is as it should be, so the discrepancy deserves an explanation, and one is provided by turning the *A* to *B* refinement (where the problem resides, persisting into the *C* model) into a retrenchment. It is however a retrenchment of a rather minimal kind. The only non-trivial part of it is the output relation for the *ToBalanceEnquiry* event:

$$\begin{array}{l}
 \overline{O_{AB, ToBalanceEnquiry}} \\
 \text{AbWorld}' \\
 \text{BetweenWorld}' \\
 \text{Abal!} : \mathbb{N} \\
 \text{Bbal!} : \mathbb{N} \\
 \hline
 (\exists BFstatus', BFpdAuth' \bullet \\
 \quad (\neg(BTstatus' = Bepv \wedge BFstatus' = Bepa \wedge BFpdAuth' = BTpdAuth') \\
 \quad \wedge Abal! = Bbal!) \quad \vee \\
 \quad ((BTstatus' = Bepv \wedge BFstatus' = Bepa \wedge BFpdAuth' = BTpdAuth') \\
 \quad \wedge Abal! - Bbal! = ATbalance' - BTbalance' = BTpdAuth'))
 \end{array}$$

The above has been simplified a bit to remove some details connected with promotion, for simplicity. The *T*, *F*, *A*, *B*, variable prefixes indicate *To* and *From* variables in the *A* and *B* models. What $O_{AB, ToBalanceEnquiry}$ says is in fact extremely simple: *either* a special state of affairs holds (namely that the *From* and *To* purses are both involved in the same transaction ($BFpdAuth' = BTpdAuth'$)⁸ and are in the critical *Bepa* and *Bepv* states respectively), whereupon the abstract and concrete *To* balance enquiry outputs, *Abal!* and *Bbal!*, differ by exactly the amount of the transaction $BTpdAuth'$, *or* the special state of affairs does *not* hold, whereupon everything is as one would expect, and the outputs agree.⁹

⁸ Different transactions can overlap in time.

⁹ Note though, that the choice between these options is non-constructive from the *To* purse's point of view; it does not know exactly what the *From* purse is doing when the balance enquiry takes place.

This kind of retrenchment is so minimal, that it hardly deserves to be called a retrenchment at all. It is really just a slight enhancement of the A to B refinement to enable it to better capture specific phenomena associated with the loss of atomicity during refinement. In [BJHS07] (a paper devoted to exploring how loss of transactions' ACID properties can be accounted for using retrenchment), we call such retrenchments, *retrenchment enhanced refinements*, in view of their mild nature.

There is in fact more to the Mondex balance enquiry story. The fact that the A to B refinement in [SCW00] is *backwards* brings additional technical complexities which we do not have space to discuss. Suffice it to say that the resulting arrangement of models and refinements, was, in respect of balance enquiries, so counterintuitive, that balance enquiries were removed completely from the public account in [SCW00]. See [BJPS07] for more details.

Do matters improve if we avail ourselves of the freedom to choose a different synchronisation point, as discussed at the end of Section 5? Unfortunately not. If we synchronise the atomic action with the *Val* event rather than *Req* (as outlined in Section 6), then the above anomaly disappears, but another appears in its place. Synchronising with *Val*, if we do a balance enquiry just before the *Val*, but again while the money is in flight, and this time on the *From* purse, we will again see a discrepancy, since the money has already been subtracted from the concrete *From* balance, whereas the abstract operation has not started yet. The 'cure' for this anomaly follows very similar lines to the previous case, but switching the roles of *To* and *From* purses.

Of course one might argue that inserting a balance enquiry into the middle of a concrete transfer makes little sense, but purses cannot afford to rely on common sense.¹⁰ Purses assume their environment to be hostile, so that the purse protocol might be interrupted at any moment. In this context, the easiest way to ensure that the system's global security invariants are preserved (these concern the aggregated balance of funds in the entire community of purses), is to have each concrete operation refine some abstract one. This constitutes a major driver for doing the Mondex development as a 'small diagram' refinement, and in turn makes it essential to have an abstract option that *skips* (i.e. *AbIgnore* in Mondex), in order to have equal numbers of 'real' abstract and concrete transactions in a framework in which abstract transactions are single step, while concrete transactions are multi-step.

12 The Denial of Service Retrenchment

In the concrete protocol, the *StartFrom* and *StartTo* events are independently called from the (cryptographically unprotected) environment. This means that their parameters (including the transaction's identifying data, which includes

¹⁰ Consider also, a transaction carried out over a slow IP link, and an impatient recipient anxiously querying the purse to see if his money has arrived yet.

the purse identifiers and purse sequence numbers) are unencrypted. As discussed in [SGH⁺08], this permits two denial of service (DOS) attacks on purses along the following lines.

In the first DOS attack, by the above remarks, an attacker can obtain the purse identifier and purse sequence number of a genuine **From** purse. He uses this data to impersonate the **From** purse; let us call this the **FakeFrom** purse. The **FakeFrom** purse can now engage in a putative transfer with a genuine **To** purse. According to Fig. 3, the **To** purse will send a genuine *req* message to the **FakeFrom** purse. This, of course, will never be responded to, so the **To** purse will eventually have to *Abort*, and since it is in the *epv* state, it will post an entry to its log. A few such attacks later, and the **To** purse's small log will have filled up with these unnecessary entries, preventing any subsequent use of the **To** purse.

The second DOS attack builds upon the first. This time, the attacker impersonates a **To** purse; let us call this the **FakeTo** purse. Using a suitable *req* message harvested from an instance of the first DOS attack, the **FakeTo** purse asks a genuine **From** purse for money. The **From** purse, knowing no better, sends the money off in a *val* message. Since the *val* message is encrypted, it is of no use to the **FakeTo** purse, which thus cannot appropriate the cash.¹¹ However, the **From** purse has been deprived of the money since its balance has been decreased during the *Req* event which sent the *val* message. So this is a genuine inconvenience to the **From** purse. Eventually the **From** purse must *Abort*, and since it is in the *epa* state, it will post an entry to its log. By this means, the attacker can deprive the **From** purse of an arbitrary amount of money, and by repetition, fill its log, blocking it.

The *val* message is never decoded — neither by the **FakeTo** purse (because, being a fake, it is unable to do so), nor by the real **To** purse (because it is a victim too, and can only *Abort*). So, from the protocol's point of view, the *val* message is lost in flight, and in particular, the payment looks genuine. The problem lies in the fact that, being victims of attack, it is quite probable that the two purses' owners do not know that they were involved in a transaction, so that the normal willing co-operation between them to restore the missing funds (which relies on matching up the two log entries) may be impossible, and some non-standard procedure may need to be invoked.

At the time of the original Mondex development, the possibility of these attacks was in fact considered, and a judgement was made, that in the context of the limited capabilities of the smartcard technology of the day, the weakness of the protocol that they exhibit, though obviously undesirable in principle, was nevertheless permissible, since it did not cause the actual loss of money (in the very specific Mondex sense of 'loss' — nothing that we have described violates any of the Mondex security invariants). Furthermore, it was unlikely

¹¹ The possibility exists that the attacker, conscience-stricken, could give the *val*, and hence the money, to the genuine **To** purse, provided it has not yet *Aborted*; but we view this as unlikely.

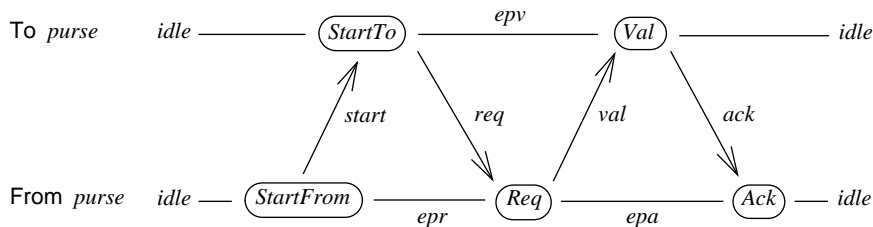


Fig. 5. The modified Mondex concrete protocol.

to be a problem in reality, as it would demand the concerted attack on a victim purse by one or more attackers and this would entail at least partial (if perhaps unknowing) collusion on the part of the victim, since a purse (unlike say an RFID tag) cannot participate in a protocol run ‘involuntarily’.

The technical solution to these problems is to modify the protocol slightly as explained in [SGH⁺08]. The modification introduces a new, encrypted, *start* message, sent by the *StartFrom* event to the *StartTo* event (these must now execute sequentially of course). This allows the exchange of sequence numbers, needed for protocol traceability and security, to become a challenge-response mechanism, performed in cryptographically protected secrecy between the two purses. Now, the the *To* purse cannot be prompted to send out the key *req* message without the authentic participation of the *From* purse. The modified protocol is shown in Fig. 5.

At the detailed level, the retrenchment between the original and modified protocol unproblematically documents the passage of information between state components and I/O variables, in the *StartTo* and *StartFrom* events of the two versions of the protocol. However, unlike the preceding cases of retrenchment, the real interest of this one is not in the alterations to the individual events, but in the impact on global behaviours of the protocol.¹² It is a prime example of a *coarse grained* retrenchment in which the notable change in the system is the effect of the co-operative activity of a number of events.

The fact that in both versions of the protocol the Mondex security invariants are preserved, makes the DOS scenarios hard to discern from the system state alone. In particular, the second scenario cannot be functionally distinguished from a normal transaction, and whether any suitable run of the protocol is an example of a DOS, or is just a normal transaction, must rest on the *interpretation* of the constituent events with respect to the surrounding context. The situation with the first DOS scenario is a bit better, since there is something specific to look for in a system run, namely: when there is a

¹² This is more the case here than in the Balance Enquiry example, where the manifestation of the problem was in the output of a single concrete event, even if the explanation for what was seen had wider connotations.

StartTo followed by an *Abort* of the *To* purse, and yet there is no corresponding *StartFrom* for the transaction, then we have a sequence of events that is open to interpretation as a DOS scenario. To get a grip on this theoretically though, we need to avail ourselves of a temporal notation that permits the counting of occurrences of events in a run.

There are yet more subtleties. The occurrence of a DOS scenario in the original protocol corresponds, in the modified protocol, to a normal transaction that terminates early. Both *Start* events are present, as are their corresponding *Aborts*. But this means that there is an increase in the *From* purse sequence number (due to the *StartFrom*) which is absent in the DOS scenario. This can be approached in a number of ways. Firstly, one can restrict to occurrences of the DOS scenario which just happen to include an occurrence of an additional Mondex event, *Increase*, which just increments the sequence number of the purse performing it. But these are not the only occurrences of the DOS scenario. So, secondly, one can simply remove the sequence numbers from the scrutiny of the retrieve relation, just checking them during events. Thirdly, one can admit a notion of simulation that occasionally ‘blinks’ and allows the tight relationship between simulator and simulatee to break down for a time. The latter hints at the wide range of unusual notions of simulation and observation that retrenchment allows when it is pushed to the limit.

13 Conclusions

In the preceding sections we started from the premise that the more complex the system that we need to understand or design, the more issues there will be regarding where to draw the line between what is modelled mathematically, and what is not. We then focused on the formal development of discrete transition systems via refinement, and reviewed how the widely available Mondex development has served as the springboard for a number of technical developments in this area. Some of these concern the refinement of atomic actions into protocols, while others concern how one could treat various requirements gaps in Mondex via retrenchment, such as the ‘retrenchment opportunities’ of the last five sections. These illustrate very well the nature of the ‘fuzzy boundary’ at the edge of the fully formalised domain.

One notable aspect of our treatment was that we treated each requirements issue as an independent entity. We made no attempt to integrate them into a single retrenchment from the published Mondex development to one including all the retrenchment opportunity issues, properly treated. From the nature of the individual treatments, we can surmise the quite significant descriptive blowup that would transpire should we have done so. So we can see that the way that the requirements boundary of the original formal Mondex development was drawn, was a reasonable treatment of the requirements, especially bearing in mind the state of the art at the time, and the commercial

costs and benefits of the modelling and proving exercise, something that is always a powerful driver in real world engineering.

Added to the above is the issue of mechanical verification for Mondex which has attracted so much interest and has been so extensively reported elsewhere. Although the developments reviewed in this paper have not been mechanically checked, there is no technical reason why they could not be. The Tower Pattern in particular —so useful for connecting a wide variety of requirements issues at the periphery of a formal development to the more mainstream requirements at its core, and for doing so in a more formal way— has been designed to allow straightforward mechanisation. This remains as interesting work for the future.

References

- [Abr] J.-R. Abrial. *Event-B*. Cambridge University Press, to be published.
- [Abr96] J.-R. Abrial. *The B-Book: Assigning Programs to Meanings*. Cambridge University Press, 1996.
- [Ban08] R. Banach. Model Based Refinement and the Design of Retrenchments. 2008. Submitted.
- [BDGK00] J.P. Bowen, S. Dunne, A. Galloway, and S. King, editors. *Proc. ZB2000: Formal Specification and Development in Z and B*, volume 1878 of *LNCS*, York, UK, August 2000. Springer.
- [BDJM00] P. Behm, P. Desforges, and Meynadier J-M. MÉTÉOR: An Industrial Success in Formal Development. In Bowen et al. [BDGK00], pages 374–393.
- [BJHS07] R. Banach, C. Jeske, A. Hall, and S. Stepney. Retrenchment and the Atomicity Pattern. In M. Hinchey and T. Margaria, editors, *Software Engineering and Formal Methods*, pages 37–46, London, UK, 2007. IEEE.
- [BJP08] R. Banach, C. Jeske, and M. Poppleton. Composition Mechanisms for Retrenchment. *J. Log. Alg. Prog.*, 75:209–229, 2008.
- [BJPS06a] R. Banach, C. Jeske, M. Poppleton, and S. Stepney. Retrenching the Purse: Finite Exception Logs, and Validating the Small. In M. Hinchey, editor, *Software Engineering Workshop 30*, pages 234–245, Layola College Graduate Center, Columbia, MD, 2006. IEEE.
- [BJPS06b] R. Banach, C. Jeske, M. Poppleton, and S. Stepney. Retrenching the Purse: Hashing Injective CLEAR Codes, and Security Properties. In B. Steffen, T. Margaria, and A. Philippou, editors, *Second International Symposium on Leveraging Applications of Formal Methods*, Paphos, Cyprus, 2006. IEEE.
- [BJPS07] R. Banach, C. Jeske, M. Poppleton, and S. Stepney. Retrenching the Purse: The Balance Enquiry Quandary, and Generalised and (1,1) Forward Refinements. *Fundamenta Informaticae*, 77:29–69, 2007.
- [Bör03] E. Börger. The ASM Refinement Method. *Formal Aspects of Computing*, 15:237–275, 2003.
- [BP98] R. Banach and M. Poppleton. Retrenchment: An Engineering Variation on Refinement. In D. Bert, editor, *2nd International B Conference*, volume 1393 of *LNCS*, pages 129–147, Montpellier, France, April 1998. Springer.

- [BP99] R. Banach and M. Poppleton. Sharp Retrenchment, Modulated Refinement and Simulation. *Formal Aspects of Computing*, 11:498–540, 1999.
- [BP03] R. Banach and M. Poppleton. Retrenching Partial Requirements into System Definitions: A Simple Feature Interaction Case Study. *Requirements Engineering Journal*, 8(2):266–288, 2003.
- [BPJS05] R. Banach, M. Poppleton, C. Jeske, and S. Stepney. Retrenching the Purse: Finite Sequence Numbers and the Tower Pattern. In *Formal Methods 2005*, volume 3582 of *LNCS*, pages 382–398. Springer, 2005.
- [BPJS07] R. Banach, M. Poppleton, C. Jeske, and S. Stepney. Engineering and Theoretical Underpinnings of Retrenchment. *Science of Computer Programming*, 67:301–329, 2007.
- [BR94] E. Börger and D. Rosenzweig. The WAM – Definition and Compiler Correctness. In C. Beierle and L. Plümer, editors, *Logic Programming: Formal Methods and Practical Applications*, pages 20–90. North-Holland, 1994.
- [BS03] E. Börger and R.F. Stärk. *Abstract State Machines. A Method for High Level System Design and Analysis*. Springer, 2003.
- [BS07] R. Banach and G. Schellhorn. On the Refinement of Atomic Actions. In Boiten, E. and Derrick, J. and Smith, G., editor, *Proceedings of REFINE 2007*, volume 201 of *ENTCS*, pages 3–30. Elsevier, 2007.
- [BS08] R. Banach and G. Schellhorn. Atomic Actions, and their Refinements to Isolated Protocols. *FAC*, 2008.
- [Daw91] J. Dawes. *The VDM-SL Reference Guide*. UCL Press/Pitman Publishing, London, 1991.
- [DB01] J. Derrick and E. Boiten. *Refinement in Z and Object-Z*. FACIT. Springer, 2001.
- [Dep91] Department of Trade and Industry. Information Technology Security Evaluation Criteria, 1991. <http://www.cesg.gov.uk/site/iacs/itsec/media/formal-docs/ltsec.pdf>.
- [FGL98] J. Fitzgerald and P. Gorm Larsen. *Modelling Systems: Practical Tools and Techniques for Software Development*. Cambridge University Press, 1998.
- [GD01] G. Grimmett and Stirzaker D. *Probability and Random Processes*. O.U.P., 3 edition, 2001.
- [GH93] Y. Gurevich and J. Huggins. The Semantics of the C Programming Language. In E. Börger, H. Kleine Büning, G. Jäger, S. Martini, and M. M. Richter, editors, *Computer Science Logic*, volume 702 of *LNCS*, pages 274–309. Springer, 1993.
- [ISO02] ISO/IEC 13568. *Information Technology – Z Formal Specification Notation – Syntax, Type System and Semantics: International Standard*, 2002. [http://www.iso.org/iso/en/ittf/PubliclyAvailableStandards/c021573_ISO_IEC_13568_2002\(E\).zip](http://www.iso.org/iso/en/ittf/PubliclyAvailableStandards/c021573_ISO_IEC_13568_2002(E).zip).
- [ISO05] ISO 15408, v. 3.0 rev. 2. *Common Criteria for Information Security Evaluation*, 2005.
- [Jes05] C. Jeske. *Algebraic Integration of Retrenchment and Refinement*. PhD thesis, University of Manchester, 2005.
- [Jon90] C.B. Jones. *Systematic Software Development using VDM*. Prentice-Hall, second edition, 1990.
- [JOW06] C.B. Jones, P. O’Hearne, and J. Woodcock. Verified Software: A Grand Challenge. *IEEE Computer*, 39(4):93–95, 2006.

- [JWe08] C. Jones and J. Woodcock (eds.). FAC Special Issue on the Mondex Verification. *Formal Aspects of Computing*, 20(1):1–139, January 2008.
- [KT75] S. Karlin and H.M. Taylor. *A First Course in Stochastic Processes*. Academic, 1975.
- [LH96] K. Lano and H. Haughton. *Specification in B: An Introduction Using the B-Toolkit*. Imperial College Press, 1996.
- [RAI95] RAISE Method Group. *The RAISE Method Manual*. Prentice Hall, 1995.
- [Res92] S.L. Resnick. *Adventures in Stochastic Processes*. Birkhauser, 1992.
- [Ret] Retrenchment Homepage. <http://www.cs.man.ac.uk/retrenchment>.
- [Roda] Rodin. European Project Rodin (Rigorous Open Development for Complex Systems) IST-511599 <http://rodin.cs.ncl.ac.uk/>.
- [Rodb] Rodin Platform. <http://sourceforge.net/projects/rodin-b-sharp/>.
- [SB08] G. Schellhorn and R. Banach. A Concept-Driven Construction of the Mondex Protocol using Three Refinements. In Boca, P. and Börger, E. and Bowen, J. and Butler, M., editor, *Proc. ABZ 2008*, volume 5238 of *LNCS*, pages 57–70. Springer, 2008.
- [SC00] S. Stepney and D. Cooper. Formal Methods for Industrial Products. In Bowen et al. [BDGK00], pages 374–393.
- [Sch01a] G. Schellhorn. Verification of ASM Refinements Using Generalized Forward Simulation. *JUCS*, 7:952–979, 2001.
- [Sch01b] S. Schneider. *The B-Method*. Palgrave Press, 2001.
- [Sch05] G. Schellhorn. ASM Refinement and Generalisations of Forward Simulation in Data Refinement: A Comparison. *Theoretical Computer Science*, 2005.
- [SCW98] S. Stepney, D. Cooper, and J. Woodcock. More Powerful Z Data Refinement: Pushing the State of the Art in Industrial Refinement. In J.P. Bowen, A. Fett, and M.G. Hinchey, editors, *11th International Conference of Z Users*, volume 1493 of *LNCS*, pages 284–307, Berlin, Germany, September 1998. Springer.
- [SCW00] S. Stepney, D. Cooper, and J. Woodcock. An Electronic Purse: Specification, Refinement and Proof. Technical Report PRG-126, Oxford University Computing Laboratory, 2000.
- [SGH⁺08] G. Schellhorn, H. Grandy, D. Haneberg, N. Moebius, and W. Reif. A Systematic Verification Approach for Mondex Electronic Purses using ASMs. In J.-R. Abrial and U. Glässer, editors, *Dagstuhl Seminar on Rigorous Methods for Software Construction and Analysis*, LNCS. Springer, 2008.
- [Spi92] J.M. Spivey. *The Z Notation: A Reference Manual*. Prentice-Hall, second edition, 1992.
- [SSB00] R.F. Stärk, J. Schmidt, and E. Börger. *Java and the Java Virtual Machine: Definition, Verification, Validation*. Springer, 2000.
- [Ste01] S. Stepney. New Horizons in Formal Methods. *The Computer Bulletin*, pages 24–26, 2001.
- [VGJM02] H.D. Van, C. George, T. Janowski, and R. Moore. *Specification Case Studies in RAISE*. FACIT, Springer, 2002.
- [WB07] J. Woodcock and R. Banach. The Verification Grand Challenge. *JUCS*, 13(5):661–668, May 2007.
- [WD96] J. Woodcock and J. Davies. *Using Z: Specification, Refinement and Proof*. Prentice-Hall, 1996.
- [Woo06] J. Woodcock. First Steps in the The Verified Software Grand Challenge. *IEEE Computer*, 39(10):57–64, 2006.

