Reconciling Retrenchments and Refinements II

Czeslaw Jeske, Richard Banach

Computer Science Dept., Manchester University, Manchester, M13 9PL, U.K.
{jeske,banach}@cs.man.ac.uk

Abstract

The drawbacks of using refinement alone in the construction of specifications from simple abstract models is used as the spur for the introduction of retrenchment, a method based on the main ideas of refinement, but one which is more liberal in character. The basics of the retrenchment mechanism are reviewed in preparation for exploring its integration with refinement.

The particular aspect of integration investigated in this paper continues on from earlier work which examined the pushout-like problem of completing a square [13] *and examines the complementary pullback-like problem of completing a square. More specifically, given a model which is both a refinement of a model* Ret *and a retrenchment of a model* Ref*, the problem of finding a model for which there is both a retrenchment to* Ret *and a refinement to* Ref *is examined. A construction is given that solves the problem in a universal manner, in that it is the most abstract reconciliation of* Ret *and* Ref.

1. Introduction

Retrenchment, first put forward in [3], is a technique based on refinement that provides a more flexible specification constructor than its precursor. This greater flexibility permits us to formally describe steps in the development of a specification which cannot be captured by refinement. Consult [7] for an introduction to retrenchment and also see [4, 5, 6, 8].

In this paper we focus on the approach to developing model-based specifications found in refinement methods like B, VDM and Z [1, 14, 24]. In such methods a development step involves incorporating more detail/information into a specification and showing that the result still preserves the requirements expressed in its predecessor. This is done by discharging a number of formal proofs known as proof obligations (POs). Thus a complete specification can be built up gradually, and refinement provides the assurance that properties introduced at each stage are preserved through the rest of the development.

Unfortunately, when we try to apply refinement to real-world applications the process often runs into difficulties. For systems whose most abstract specifications are cast in the language of continuous mathematics, the transformation into the discrete specifications necessary for subsequent implementation are beyond the grasp of refinement. The strong nature of the refinement POs is just too restrictive.

Even when we consider developing specifications for systems whose models fall entirely into the discrete domain, refinement is not always in the clear. To manage the complexity of such systems we begin with models which express the main properties of the system. Such abstraction makes the models not only more approachable but also more amenable to formal analysis. Only later do we introduce lower level detail, like that dealing with the finiteness of data structures, a property present in any concrete implementation. However, the need to establish refinement relationships between the models involved often forces the consideration of these concrete properties much earlier in the design process than desired.

Difficulties can also arise when we try to apply refinement to already established development routes. The hierarchy of models in such a development is governed by engineering concerns and it may be the case that the relationship between some models is not expressible using refinement. The adoption of refinement would therefore entail an overhaul of the established engineering procedure. A situation eclipsing any perceived benefits of the rigour introduced by refinement.

Retrenchment was designed to address the difficulties outlined above. Being more liberal than refinement it is able to relate models which cannot be connected by the refinement mechanism. As is well known, a refinement step between levels of abstraction permits the weakening of the precondition and strengthening of the postcondition. In contrast, a retrenchment step permits the opposite: strengthening of the precondition and weakening of the postcondition. It is this that gives rise to the more liberal nature of the retrenchment POs. Additionally, retrenchment allows state and I/O entities to move from one to the other across a development step. In refinement the relationship between two levels, commonly referred to as abstract and concrete, is expressed by a retrieve or abstraction relation which links corresponding states. In retrenchment there are two extra relations called within and concedes. The former expresses the precondition strengthening, and the latter expresses the postcondition weakening. In particular, non-refinement-like behaviour can be accommodated within the framework via the weakened postcondition.

We stress that our intention is not to replace refinement with retrenchment, but in fact to extend the family of formal techniques available to the developer. We think it is particularly important that the two techniques work smoothly together. Given this aim, a significant branch of research is concerned with investigating the integration of retrenchment and refinement. In practice this amounts to solving a number of algebraic problems.

In [2, 12] we investigated the canonical factorisation of an arbitrary retrenchment into a refinement and an abstraction-preserving retrenchment. In [13] we explored the pushout-like problem of completing a square.

Continuing the categorical theme of [13], this paper considers another way of completing the square, by exploring a pullback-like arrangement. Thus suppose we have a system *Conc* which is both a refinement of a system *Ret* and a retrenchment of a system *Ref*. We want to find a system *Univ* which is *both* retrenchable to *Ret* and refinable to *Ref*. We seek a canonical reconciliation by requiring that for any other system *Xtra* which also completes the square, there is a refinement from *Univ* to *Xtra*. Thus *Univ* is the most abstract completion possible. A summary of the problem is shown in Figure 1.

The above construction lets us to automatically lift *Ref* to the level of abstraction of *Ret*, thus allowing *Ref* to be considered from a different perspective; and considering a problem under a different light is *always* advantageous. The construction is also of algebraic interest, as it complements the result given in [13] and is one more part towards a complete algebraic theory of the integration of retrenchment and refinement. A further benefit of the reconciliation, is that it helps to absorb retrenchment, a new technique, into the fold of trusted formal tools. Reconciling retrenchment and refinement reassures practitioners that in using retrenchment, they do not risk fracturing the development process into incompatible and irreconcilable paths.

The rest of the paper is structured as follows. In Section 2 we present a simple example which gives a flavour of the kinds of problems that may occur when we try to use refinement to capture all desirable development steps. Section 3 introduces retrenchment, which is designed to address such problems. Section 4 presents I/O-filtered refinements, the form of refinement used in the reconciliation. In Section 5 we give the details of the construction achieving the reconciliation. Section 6 examines the result in the context of the running example. Section 7 concludes.

Notation. In the sequel we will view systems mainly from a set theoretic and relational viewpoint, which we discuss using a logical meta-notation. Thus a predicate *is* just a notation for a set etc.

2. Problems with Refinement.

Let us suppose we are developing a large software system. Separating concerns, we begin by producing abstract models that describe the basic functional requirements. Implementational constraints, like the finiteness of data structures, are ignored till later. We use refinement to show that properties are preserved between successive models. Given the size of the system under development the resulting functional model is already a large document. Imagine that the many elements introduced into the specification by this stage include a sequence of NATs, an operation, *Add*, which adds numbers to the sequence, and an operation, *Rem*, which removes numbers from it. By formal analysis we satisfy ourselves that the model so far, captures the key behaviour of the system. We therefore start to focus on lower level details like implementational constraints. Hence, at some subsequent stage we shall come to write a development step in which we address the finiteness of the sequence of NATs. Let us see how refinement fares when faced with such a step.

We begin by setting up a suitable framework. Let *Ref* be the model we wish to develop further by limiting the size of the sequence, and *Conc* be the model in which the constraint has been applied. Each model will be described by a state space and a set of operation names. Individual operations will be defined by a transition or step relation, and each will have its own input and output spaces.

For *Ref* let the state space be W with typical element w. $\mathsf{Ops}_{\mathrm{F}}$ will be the set of operation names with Op_{F} designating a typical operation. For each Op the input and output spaces will be I_{Op} and O_{Op} with i and o representing typical elements respectively. We dispense with the subscripts for i and o as the operation in question will be clear from the context. A typical Op_{F} transition will be depicted by w -(i, Op_{T}, o)-› w', where w and w' are the before- and after-states, and i and o are the input and output. The set of such transitions form the step relation $stp_{Op_{\mathrm{F}}}(w, i, w', o)$. $Init_{\mathrm{F}}(w')$ is the initialisation operation which sets the state to an initial value w'.

Conc has a similar setup. The operation names are Op_{C}

$\in \mathsf{Ops}_C$ with $\mathsf{Ops}_F = \mathsf{Ops}_C$, and states are $t \in \mathsf{T}$. In classical refinement the I/O signature of operations is not allowed to change across a development step. Thus input and output entities are the ones defined above. Transitions are $t \text{ -}(i, Op_C, o)\text{-›} t'$, and these are members of the step relation $stp_{Op_C}(t, i, t', o)$. The initialisation operation is $Init_C(t')$.

To define a refinement relationship between adjacent models in a development step we need to also specify a retrieve relation whose job is to link corresponding states between models. We denote this relation by $G(w, t)$. Then, for *Conc* to refine *Ref*, initialisation PO (Init PO) (2.1) must hold, and for each *Op*, operation PO (Op PO) (2.2) must hold.

$$Init_C(t') \ \Rightarrow\ (\exists\, w' \bullet Init_F(w') \wedge G(w', t'))\ . \tag{2.1}$$

$$G(w, t) \wedge stp_{Op_C}(t, i, t', o) \ \Rightarrow \\ (\exists\, w', q \bullet stp_{Op_F}(w, i, w', o) \wedge G(w', t'))\ . \tag{2.2}$$

These POs require that every Op_C-step has a corresponding Op_F-step and arise from the basic notion of substitutivity of concrete for abstract model on which refinement is based. Here, we have given the forward simulation formulation of refinement, since almost all applications of refinement are of this kind. We will also work in a partial correctness setting. For a comprehensive discussion of both forward and backward simulation, total and partial correctness, and the notion of substitutivity see [9, 10, 26].

To keep the example focused on the problem in hand, we will assume that the state of each system is just the sequence of NATs and ignore any other components that make up the state. We will also ignore all operations other than *Add* and *Rem*. Since these are the only ones that operate on the sequence, this is an acceptable simplification.

We define the state space W of *Ref* as the set of injective sequences of NATs, i.e. $\mathsf{W} = iseq(\mathsf{NAT})$. $\mathsf{I}_{Add} = \mathsf{O}_{Rem} = \mathsf{NAT}$ and $\mathsf{O}_{Add} = \mathsf{I}_{Rem} = \varnothing$. Transitions for the two operations Add_F and Rem_F have the form

$$w \text{ -}(i, Add_F)\text{-›}\ w \mathbin{^\wedge} \langle i \rangle, \text{ where } i \notin ran(w) \\ w \text{ -}(i, Add_F)\text{-›}\ w, \text{ where } i \in ran(w) \tag{2.3}$$

and

$$\langle o \rangle \mathbin{^\wedge} w \text{ -}(Rem_F, o)\text{-›}\ w\ . \tag{2.4}$$

The state and I/O spaces for *Conc* are the same as for *Ref*. In *Conc* we want to limit the size of the sequence. We will allow a maximum length of 10. We can achieve this by defining the transitions for Add_C to be

$$t \text{ -}(i, Add_C)\text{-›}\ t \mathbin{^\wedge} \langle i \rangle, \text{ where } i \notin ran(t) \wedge \\ len(t) \le 9 \\ t \text{ -}(i, Add_C)\text{-›}\ t, \text{ where } i \in ran(t) \tag{2.5}$$

and leave *Rem* unchanged, giving

$$\langle o \rangle \mathbin{^\wedge} t \text{ -}(Rem_C, o)\text{-›}\ t\ . \tag{2.6}$$

If we now define the retrieve relation $G(w, t)$ to be an identity, then the above is a refinement of *Ref* since Op PO (2.2) holds. However, our specification is incomplete because we have not specified what happens when the sequence is full and we try to add another element. One possibility is to do nothing, that is just *skip*. Thus we could try and extend the definition by adding the transitions

$$t \text{ -}(i, Add_C)\text{-›}\ t \mathbin{^\wedge} \langle i \rangle, \text{ where } i \notin ran(t) \wedge \\ len(t) = 10\ . \tag{2.7}$$

Unfortunately this amendment means we no longer have a refinement because an element not already in the sequence will still be added in the abstract model, and so the retrieve relation will fail. This should be apparent seeing that beforehand $len(w) = len(t) = 10$, yet after, $len(w) = 11$ but $len(t) = 10$. Perhaps the problem could be fixed by modifying G. One possibility is $G(w, t) = (first(w, 10) = first(t, 10))$, where $first(s, l)$ returns a sequence made up of the first l elements of s, or all of s if $len(s) < l$. But now of course when $len(w) > 10$ and $len(t) = 10$, if *Rem* is used to remove an element, the retrieve relation will no longer hold. So we are running into difficulties trying to establish a refinement relationship for our desired step.

Let us now consider Rem_C. Notice (2.6) does not specify what happens if we try to remove a number when the sequence is empty. The sensible thing to do in such a situation is to signal an error. Accordingly we could extend the definition with

$$\langle\,\rangle \text{ -}(Rem_C, \mathsf{EMPTY})\text{-›} \langle\,\rangle\ . \tag{2.8}$$

However this fails on two counts. First, this cannot be a refinement because there is no corresponding abstract step. Second, a change in I/O signature is not allowed. In fact whatever we choose to do, we will not get a refinement, simply because there is no corresponding abstract step.

It is fair to say that all the above problems can be avoided by dealing with the issue of finiteness in the abstract model. However, recall that our example just focuses on one small part of a much larger system. Given an industrial scale project, it is not difficult to appreciate that lifting all the necessary low level detail to the abstract level in order to get a refinement, will significantly bloat an already sizeable document. The result is a model more difficult to formally analyse and harder to penetrate for developers trying to comprehend the behaviour of the described system. Retrenchment, introduced in the next section, plays a more subordinate role than refinement and gives the developer more control in the way a specification is constructed.

Many situations involving a finite computable subdomain of a mathematically ideal and infinite one, follow the pattern described above, and a number of approaches described in the literature, have been designed to address the problem. One technique is Neilson's thesis [17], which describes the concept of acceptably inadequate refinements. These tackle the problem by observing that the infinite ideal domains usually arise as well behaved limits of corresponding finite ones, and thus refinement in the idealised case can be understood as the limit of a finite version. A second technique is presented in [18, 19], in which Owe proposes a logical approach based on a careful analysis of the effects of ill-definedness on a programming logic.

Two further proposals are Liu's evolution [15] and Smith's realisation [22, 23]. These like retrenchment are a weakening of refinement. Evolution relates models by requiring that the pre- and post-conditions of the abstract specification in a development step be semantically equivalent to subformulae of those at the more concrete level. Realisation relates models by substituting subterms when moving from the abstract to the concrete specification.

A separate issue raised in the example is the desirability of changing I/O signatures across a development step. Works on this question include [10, 11, 16, 25].

3. Retrenchment

We begin by modifying the framework introduced in the previous section. For *Ref* each Op_F now has inputs $k \in \mathsf{K}_{Op_F}$ and outputs $q \in \mathsf{Q}_{Op_F}$. Transitions are therefore w -(k, Op_F, q)-› w', and are members of the step relation $stp_{Op_F}(w, k, w', q)$. For *Conc* each Op_C has inputs $h \in \mathsf{H}_{Op_C}$ and outputs $s \in \mathsf{S}_{Op_C}$. Transitions are t -(h, Op_F, s)-› t' and the step relation is $stp_{Op_C}(t, h, t', s)$. In retrenchment it is assumed that there is a distinct Op_C corresponding to each distinct Op_F, but not necessarily vice versa, so the concrete level may contain additional operations. For convenience we assume this correspondence is $\mathsf{Ops}_F \subseteq \mathsf{Ops}_C$.

The relationship between abstract and concrete state spaces is given, as before, by a retrieve relation, which we now write as $H(w, t)$. The Init PO is the same as found in refinement, and in the present context has the form

$$Init_C(t') \;\Rightarrow\; (\exists\, w' \bullet Init_F(w') \wedge H(w', t'))\ . \tag{3.1}$$

Retrenchment differs from refinement in that H alone is not enough to fix the relationship between the two levels. We also have for all Op in Ops_F, the within relation $Q_{Op}(k, h, w, t)$ and concedes relation $D_{Op}(w', t', q, s; k, h, w, t)$. The punctuation in D_{Op} is intended to emphasize that this relation is mainly concerned with after-values, but may refer to the before-values if required. The three relations H, Q_{Op} and D_{Op}, are combined into the retrenchment Op PO for the Ops_F operations which says that for each such Op

$$\begin{aligned} & H(w, t) \wedge Q_{Op}(k, h, w, t) \wedge stp_{Op_C}(t, h, t', s) \;\Rightarrow \\ & \quad (\exists\, w', q \bullet stp_{Op_F}(w, k, w', q) \wedge \\ & \quad (H(w', t') \vee D_{Op}(w', t', q, s; k, h, w, t))) \end{aligned} \tag{3.2}$$

This means the following. We assert the consequent of the implication, but only provided both H and Q hold. This enables us to restrict via the within relation Q, the applicability of the relationship between the abstract and concrete systems; permitting the bringing together of models that would otherwise fail to support a refinement. The consequent itself asserts that for every concrete step, there is an abstract step that either re-establishes the retrieve relation H, or failing that, satisfies the concedes relation D. Again the additional flexibility allowed by D permits us to relate models that would not otherwise be capable of being formally related.

Thus the within relation strengthens the retrieve relation in before-states, and most importantly, the concedes relation weakens the retrieve relation in after-states. Beyond the ability to restrict the relationship between abstract and concrete levels, the within relation captures any non-trivial relationship between inputs and before-states. Likewise the concedes relation captures non-refinement-like properties, and non-trivial relationships between outputs and after-states[1] (and also before-entities if appropriate).

We now reconsider our earlier example as a retrenchment. We define $\mathsf{Ops}_F = \mathsf{Ops}_C = \{Init, Add, Rem\}$, $\mathsf{W} = \mathsf{T} = iseq(\mathsf{NAT})$. $\mathsf{K}_{Add_F} = \mathsf{H}_{Add_C} = \mathsf{NAT}$ and $\mathsf{Q}_{Add_F} = \mathsf{S}_{Add_C} = \varnothing$. $\mathsf{K}_{Rem_F} = \mathsf{H}_{Rem_C} = \varnothing$, $\mathsf{Q}_{Rem_F} = \mathsf{NAT}$ and $\mathsf{S}_{Rem_C} = \mathsf{NAT} \cup \{\mathsf{EMPTY}\}$. Add_C is given by (2.5) and (2.7), so the operation does a *skip* when the sequence is full. Rem_C is given by (2.6) and (2.8), and so an error is output when the sequence is empty.

To establish a retrenchment between *Ref* and *Conc* we need to specify the retrieve relation and also the within and concedes relations for *Add* and *Rem*. Hence,

$$\begin{aligned} & H(w, t) = (w = t \wedge len(w) \le 10)\ , \\ & Q_{Add}(k, h, w, t) \;=\; (k = h)\ , \\ & D_{Add}(w', t'; k, h, w, t) \;= \\ & \quad (len(w) = 10 \wedge k \notin ran(w) \wedge w' = w \mathbin{\char`^} {<}\, k\, {>} \wedge \\ & \quad t' = w)\ , \\ & Q_{Rem}(w, t) \;=\; (len(t) \ne 0)\ , \\ & D_{Rem}(w', t'\ q, s; w, t) \;=\; \mathit{false}\ . \end{aligned} \tag{3.3}$$

Notice how concession D_{Add} allows *Conc* to exhibit different behaviour when the sequence is full. *Ref* adds an-

1. Relations between outputs and states ought to hold universally, and not just when H fails. Sharp retrenchment and output retrenchment addresses this, establishing in the consequent of the PO, in the former case $((H \vee D) \wedge V)$, and in the latter $((H \wedge O) \vee D)$ where V, the nevertheless relation, or O the output relation allows extra conjunctive properties to be expressed. See [6] and [8].

other element to the sequence but *Conc* preforms a *skip*. So although the retrieve relation does not hold for the after-sates, D_{Add} does, and thus Op PO (3.2) is satisfied. For *Rem* we use the within clause Q_{Rem} to exclude the ill-behaved cases from consideration. When $len(t) = 0$, Q_{Rem} is false and so (3.2) holds trivially.

We concede that the simplicity of our example does not do much to champion the case for retrenchment. However, limited space and a wish not to obfuscate the construction presented in this paper are points in its favour. But more importantly, is the fact that it is characteristic of the kinds of situations retrenchment was designed to address, for retrenchment allows developers to delay the introduction of detail in circumstances where refinement does not. More convincing problems in support of retrenchment can be found in [20], which looks at the specification of a program for dose calculation in radiotherapy, and in [21], which presents a control system case study.

4. I/O-Filtered Refinements

We now make precise the notion of refinement we need so that our subsequent results go through. Since I/O signatures can be changed during retrenchment, it is useful to absorb this capability into refinement. We assume we have an abstract system *Ret* and a concrete system *Conc*.

The notation set up already will do for *Conc*. For *Ret*, with operation names $\mathsf{Ops}_{\mathrm{T}}$, where $\mathsf{Ops}_{\mathrm{T}} = \mathsf{Ops}_{\mathrm{C}}$, the state space will be V with elements v. The input and output spaces for Op_{T} are given by $j \in \mathsf{J}_{Op_{\mathrm{T}}}$ and $p \in \mathsf{P}_{Op_{\mathrm{T}}}$. The abstract and concrete state spaces are related by a retrieve relation $K(v, t)$. For an I/O-filtered refinement, we further have for each $Op \in \mathsf{Ops}_{\mathrm{T}}$, a within relation $R_{Op}(j, h)$, and a nevertheless relation $V_{Op}(p, s)$. This assembly of components is required to verify the following POs.

Firstly there is the Init PO. This is essentially the same as (3.1), and so using the relations above has the form

$$Init_{\mathrm{C}}(t') \;\Rightarrow\; (\exists\, v' \bullet Init_{\mathrm{T}}(v') \wedge K(v', t')) \;. \tag{4.1}$$

Secondly there is the Op PO, which for a typical *Op* says

$$\begin{aligned} & K(v, t) \wedge R_{Op}(j, h) \wedge stp_{Op_{\mathrm{C}}}(t, h, t', s) \;\Rightarrow \\ & \quad (\exists\, v', p \bullet stp_{Op_{\mathrm{T}}}(v, j, v', p) \wedge \\ & \quad K(v', t') \wedge V_{Op}(p, s)) \;. \end{aligned} \tag{4.2}$$

Note that V_{Op} enters the consequent of (4.2) conjunctively, in contrast to the retrenchment case.

We return to our running example and introduce a simple refinement. The states of *Ret* are sets of NATs, so $\mathsf{V} = \mathbb{P}(\mathsf{NAT})$. $\mathsf{J}_{Add_{\mathrm{T}}} = \mathsf{H}_{Add_{\mathrm{C}}}$, $\mathsf{P}_{Add_{\mathrm{T}}} = \mathsf{S}_{Add_{\mathrm{C}}}$, $\mathsf{J}_{Rem_{\mathrm{T}}} = \mathsf{H}_{Rem_{\mathrm{C}}}$ and $\mathsf{P}_{Rem_{\mathrm{T}}} = \mathsf{S}_{Rem_{\mathrm{C}}}$.

The retrieve relation is $K(v, t) = (v = rng(t))$ and thus associates each set with all its possible serialisations. Hence, for example, $v = \{1, 2, 3\}$ corresponds to $t = \langle 1, 2, 3\rangle$, or $v = \langle 2, 1, 3\rangle$, or to any of four other possibilities. The within and nevertheless relations are all the obvious identities.

$$\begin{tikzcd}
\mathit{Xtra} \arrow[rrd, "H^{\sim}\ Q^{\sim}\ D^{\sim}"] \arrow[rd, dashed, "K^{\circ}\ R^{\circ}\ V^{\circ}"'] \arrow[rdd, "K^{\sim}\ R^{\sim}\ V^{\sim}"'] & \subseteq & \\
\subseteq & \mathit{Univ} \arrow[r, "H^{\bullet}\ Q^{\bullet}\ D^{\bullet}"] \arrow[d, "K^{\bullet}\ R^{\bullet}\ V^{\bullet}"'] & \mathit{Ret} \arrow[d, "K\ R\ V"] \\
& \mathit{Ref} \arrow[r, "H\ Q\ D"] & \mathit{Conc}
\end{tikzcd}$$

All arrows labelled with a *H Q D* are retrenchments.
All arrows labelled with a *K R V* are refinements.

Figure 1.

It remains to define the operations. These exhibit the same behaviour as their *Conc* counterparts. Therefore Add_{T} is given by

$$\begin{aligned} & v \text{ -}(j, Add_{\mathrm{T}})\text{-}\!\!\rightarrow v \cup \{j\}, \text{ where } |v| < 10 \\ & v \text{ -}(j, Add_{\mathrm{T}})\text{-}\!\!\rightarrow v, \text{ where } |v| = 10 \;, \end{aligned} \tag{4.3}$$

and Rem_{T} is given by

$$\begin{aligned} & v \uplus \{p\} \text{ -}(Rem_{\mathrm{T}}, p)\text{-}\!\!\rightarrow v \\ & \{\ \} \text{ -}(Rem_{\mathrm{T}}, \mathsf{EMPTY})\text{-}\!\!\rightarrow \{\ \} \;, \end{aligned} \tag{4.4}$$

where $\uplus$ represents disjoint union.

With the self-evident initialisation, it is clear that (4.2), suitably instantiated, holds for this arrangement.

5. The Reconciliation

In this section we take the retrenchment from *Ref* to *Conc* and the I/O-filtered refinement from *Ret* to *Conc*, and build a new universal system *Univ*, from which there is both a retrenchment to *Ret* and a refinement to *Ref*; see Fig. 1. Due to a shortage of space we present all the material in this section without proof [2]. First let

$$\begin{aligned} & HD(w, t) = H(w, t) \vee \\ & \quad \bigvee_{Op} (\exists\, \underline{\underline{q}}, \underline{\underline{s}}, \underline{\underline{k}}, \underline{\underline{h}}, \underline{\underline{w}}, \underline{\underline{t}} \bullet D_{Op}(w, t, \underline{\underline{q}}, \underline{\underline{s}}; \underline{\underline{k}}, \underline{\underline{h}}, \underline{\underline{w}}, \underline{\underline{t}})) \end{aligned} \tag{5.1}$$

2. Proofs available online in `rec.ret.ref.2.prf.ps.gz` at `http://www.cs.man.ac.uk/~banach/some.pubs/` .

$$QI_{Op}(k, h) = (\exists\ \underline{\underline{w}}, \underline{\underline{t}} \bullet Q_{Op}(k, h, \underline{\underline{w}}, \underline{\underline{t}})) \tag{5.2}$$

$$\begin{aligned} &DO_{Op}(q, s) = \\ &\quad (\exists\ \underline{\underline{w}}', \underline{\underline{t}}', \underline{\underline{k}}, \underline{\underline{h}}, \underline{\underline{w}}, \underline{\underline{t}} \bullet D_{Op}(\underline{\underline{w}}', \underline{\underline{t}}', q, s; \underline{\underline{k}}, \underline{\underline{h}}, \underline{\underline{w}}, \underline{\underline{t}})) \end{aligned} \tag{5.3}$$

Given these, we now introduce the following equivalence relations.

$$\sim_{\mathrm{V}} = ((K;HD^{\mathrm{T}});(K;HD^{\mathrm{T}})^{\mathrm{T}})^* \tag{5.4}$$

$$\sim_{\mathrm{W}} = ((HD;K^{\mathrm{T}});(HD;K^{\mathrm{T}})^{\mathrm{T}})^* \tag{5.5}$$

$$\sim_{\mathrm{J}_{Op}} = ((R_{Op};QI_{Op}^{\mathrm{T}});(R_{Op};QI_{Op}^{\mathrm{T}})^{\mathrm{T}})^* \tag{5.6}$$

$$\sim_{\mathrm{K}_{Op}} = ((QI_{Op};R_{Op}^{\mathrm{T}});(QI_{Op};R_{Op}^{\mathrm{T}})^{\mathrm{T}})^* \tag{5.7}$$

$$\sim_{\mathrm{P}_{Op}} = ((V_{Op};DO_{Op}^{\mathrm{T}});(V_{Op};DO_{Op}^{\mathrm{T}})^{\mathrm{T}})^* \tag{5.8}$$

$$\sim_{\mathrm{Q}_{Op}} = ((DO_{Op};V_{Op}^{\mathrm{T}});(DO_{Op};V_{Op}^{\mathrm{T}})^{\mathrm{T}})^* \tag{5.9}$$

The operation names set of *Univ* is $\mathsf{Ops}_{\mathrm{U}}$ with elements Op_{U}. The state space is U with elements u, inputs are $i \in \mathsf{I}_{Op}$, outputs $o \in \mathsf{O}_{Op}$. These are all constructed from *Ret* and *Ref* as follows. Let $\mathsf{Ops}_{\mathrm{U}} = \mathsf{Ops}_{\mathrm{F}}$. The state space U is $\mathsf{V}/\sim_{\mathrm{V}} \times \mathsf{W}/\sim_{\mathrm{W}}$. Similarly the input and output spaces for each Op_{U} are $\mathsf{I}_{Op} = \mathsf{J}_{Op}/\sim_{\mathrm{J}_{Op}} \times \mathsf{K}_{Op}/\sim_{\mathrm{K}_{Op}}$ and $\mathsf{O}_{Op} = \mathsf{P}_{Op}/\sim_{\mathrm{P}_{Op}} \times \mathsf{Q}_{Op}/\sim_{\mathrm{Q}_{Op}}$.

Now for some more definitions.

$$\begin{aligned} &KH(\underline{v}, [w]) = \\ &\quad \big(\ \forall\ t \bullet K(\underline{v}, t) \Rightarrow (\exists\ \underline{w} \bullet \underline{w} \in [w] \wedge H(\underline{w}, t))\ \big) \end{aligned} \tag{5.10}$$

$$\begin{aligned} &HK([v], [w]) = \\ &\quad \big(\ \forall\ \underline{w}, t \bullet \underline{w} \in [w] \wedge H(\underline{w}, t) \Rightarrow \\ &\qquad (\exists\ \underline{v} \bullet \underline{v} \in [v] \wedge K(\underline{v}, t) \wedge KH(\underline{v}, [w]))\ \big) \end{aligned} \tag{5.11}$$

$$\begin{aligned} &KD_{Op}(\underline{v}, [w]) = \\ &\quad \big(\ \forall\ t \bullet K(\underline{v}, t) \Rightarrow \\ &\qquad (\exists\ \underline{w} \bullet \underline{w} \in [w] \wedge \\ &\qquad\quad (\exists\ \underline{\underline{q}}, \underline{\underline{s}}, \underline{\underline{k}}, \underline{\underline{h}}, \underline{\underline{w}}, \underline{\underline{t}} \bullet D_{Op}(\underline{w}, t, \underline{\underline{q}}, \underline{\underline{s}}; \underline{\underline{k}}, \underline{\underline{h}}, \underline{\underline{w}}, \underline{\underline{t}})))\ \big) \end{aligned} \tag{5.12}$$

$$\begin{aligned} &DK_{Op}([v], [w]) = \\ &\quad \big(\ \forall\ \underline{w}, t \bullet \underline{w} \in [w] \wedge \\ &\qquad (\exists\ \underline{\underline{q}}, \underline{\underline{s}}, \underline{\underline{k}}, \underline{\underline{h}}, \underline{\underline{w}}, \underline{\underline{t}} \bullet D_{Op}(\underline{w}, t, \underline{\underline{q}}, \underline{\underline{s}}; \underline{\underline{k}}, \underline{\underline{h}}, \underline{\underline{w}}, \underline{\underline{t}})) \Rightarrow \\ &\qquad\quad (\exists\ \underline{v} \bullet \underline{v} \in [v] \wedge K(\underline{v}, t) \wedge KD_{Op}(\underline{v}, [w]))\ \big) \end{aligned} \tag{5.13}$$

$$\begin{aligned} &RQ_{Op}(\underline{j}, \underline{v}, [k], [w]) = \\ &\quad \big(\ \forall\ h, t \bullet R_{Op}(\underline{j}, h) \wedge K(\underline{v}, t) \Rightarrow \\ &\qquad (\exists\ \underline{k}, \underline{w} \bullet \underline{k} \in [k] \wedge \underline{w} \in [w] \wedge H(\underline{w}, t) \wedge \\ &\qquad\quad Q_{Op}(\underline{k}, h, \underline{w}, t))\ \big) \end{aligned} \tag{5.14}$$

$$\begin{aligned} &QR_{Op}([j], [k]) = \\ &\quad \big(\ \forall\ h, t, \underline{k}, \underline{w}, v, w \bullet \underline{k} \in [k] \wedge H(\underline{w}, t) \wedge \\ &\qquad Q_{Op}(\underline{k}, h, \underline{w}, t) \wedge K^{\bullet}(([v], [w]), \underline{w}) \Rightarrow \\ &\qquad\quad (\exists\ \underline{j}, \underline{v} \bullet \underline{j} \in [j] \wedge \underline{v} \in [v] \wedge R_{Op}(\underline{j}, h) \wedge \\ &\qquad\qquad K(\underline{v}, t) \wedge RQ_{Op}(\underline{j}, \underline{v}, [k], [w]))\ \big) \end{aligned} \tag{5.15}$$

$$\begin{aligned} &VD_{Op}(\underline{v}', \underline{p}, \underline{j}, \underline{v}, [w'], [q], [k], [w]) = \\ &\quad \big(\ \forall\ t', s, h, t \bullet K(\underline{v}', t') \wedge V_{Op}(\underline{p}, s) \wedge R_{Op}(\underline{j}, h) \wedge \\ &\qquad K(\underline{v}, t) \Rightarrow \\ &\qquad\quad (\exists\ \underline{w}', \underline{q}, \underline{k}, \underline{w} \bullet \underline{w}' \in [w'] \wedge \underline{q} \in [q] \wedge \underline{k} \in [k] \wedge \\ &\qquad\qquad \underline{w} \in [w] \wedge H(\underline{w}, t) \wedge Q_{Op}(\underline{k}, h, \underline{w}, t) \wedge \\ &\qquad\qquad D_{Op}(\underline{w}', t', \underline{q}, s; \underline{k}, h, \underline{w}, t))\ \big) \end{aligned} \tag{5.16}$$

$$\begin{aligned} &DV_{Op}([p], [q]) = \\ &\quad (\forall\ t', s, h, t, \underline{w}', \underline{q}, \underline{k}, \underline{w}, v', w', j, k, v, w \bullet \\ &\qquad \underline{q} \in [q] \wedge H(\underline{w}, t) \wedge Q_{Op}(\underline{k}, h, \underline{w}, t) \wedge \\ &\qquad D_{Op}(\underline{w}', t', \underline{q}, s; \underline{k}, h, \underline{w}, t) \wedge K^{\bullet}(([v'], [w']), \underline{w}') \wedge \\ &\qquad R^{\bullet}{}_{Op}(([j], [k]), \underline{k}) \wedge K^{\bullet}(([v], [w]), \underline{w}) \Rightarrow \\ &\qquad\quad (\exists\ \underline{v}', \underline{p}, \underline{j}, \underline{v} \bullet \underline{v}' \in [v'] \wedge \underline{p} \in [p] \wedge \underline{j} \in [j] \wedge \\ &\qquad\qquad \underline{v} \in [v] \wedge K(\underline{v}', t') \wedge V_{Op}(\underline{p}, s) \wedge \\ &\qquad\qquad R_{Op}(\underline{j}, h) \wedge K(\underline{v}, t) \wedge \\ &\qquad\qquad VD_{Op}(\underline{v}', \underline{p}, \underline{j}, \underline{v}, [w'], [q], [k], [w]))\ \big) \end{aligned} \tag{5.17}$$

We can now define the component relations for the retrenchment from *Univ* to *Ret* and the refinement from *Univ* to *Ref*; see Fig. 1 again.

$$\begin{aligned} &K^{\bullet}(([v], [w]), \underline{w}) = \\ &\quad \underline{w} \in [w] \wedge HK([v], [w]) \wedge \bigwedge_{Op} DK_{Op}([v], [w]) \end{aligned} \tag{5.18}$$

$$\begin{aligned} &H^{\bullet}(([v], [w]), \underline{v}) = \\ &\quad \underline{v} \in [v] \wedge (\exists\ t \bullet K(\underline{v}, t)) \wedge KH(\underline{v}, [w]) \wedge \\ &\quad HK([v], [w]) \wedge \bigwedge_{Op} DK_{Op}([v], [w]) \end{aligned} \tag{5.19}$$

$$\begin{aligned} &R^{\bullet}{}_{Op}(([j], [k]), \underline{k}) = \\ &\quad \underline{k} \in [k] \wedge QR_{Op}([j], [k]) \end{aligned} \tag{5.20}$$

$$\begin{aligned} &Q^{\bullet}{}_{Op}(([j], [k]), \underline{j}, ([v], [w]), \underline{v}) = \\ &\quad \underline{j} \in [j] \wedge \underline{v} \in [v] \wedge (\exists\ h, t \bullet R_{Op}(\underline{j}, h) \wedge K(\underline{v}, t)) \wedge \\ &\quad RQ_{Op}(\underline{j}, \underline{v}, [k], [w]) \wedge QR_{Op}([j], [k]) \end{aligned} \tag{5.21}$$

$$\begin{aligned} &V^{\bullet}{}_{Op}(([p], [q]), \underline{q}) = \\ &\quad \underline{q} \in [q] \wedge DV_{Op}([p], [q]) \end{aligned} \tag{5.22}$$

$$\begin{aligned} &D^{\bullet}{}_{Op}(([v'], [w']), \underline{v}', ([p], [q]), \underline{p}; ([j], [k]), \underline{j}, ([v], [w]), \underline{v}) = \\ &\quad \underline{v}' \in [v'] \wedge \underline{p} \in [p] \wedge \underline{j} \in [j] \wedge \underline{v} \in [v] \wedge \end{aligned}$$

$$\begin{aligned}&(\exists\, t', s, h, t \bullet K(\underline{v}', t') \wedge V_{Op}(\underline{p}, s) \wedge R_{Op}(\underline{j}, h) \wedge \\ &\quad K(\underline{v}, t)) \wedge \\ &VD_{Op}(\underline{v}', \underline{p}, \underline{j}, \underline{v}, [w'], [q], [k], [w]) \wedge DV_{Op}([p], [q]) \wedge \\ &HK([v'], [w']) \wedge \bigwedge_{Op} DK_{Op}([v'], [w'])\end{aligned} \tag{5.23}$$

With these definitions Fig. 1 commutes in the following sense. Firstly $K^\bullet(u, \underline{w});H(\underline{w}, t) = H^\bullet(u, \underline{v});K(\underline{v}, t)$. We write this for short as

$$K^\bullet;H \equiv H^\bullet;K \equiv G\,. \tag{5.24}$$

Secondly $\mathbf{(}K^\bullet(u, \underline{w}) \wedge R^\bullet{}_{Op}(i, \underline{k})\mathbf{);(}H(\underline{w}, t) \wedge Q_{Op}(\underline{k}, h, \underline{w}, t)\mathbf{)}$ $= \mathbf{(}H^\bullet(u, \underline{v}) \wedge Q^\bullet{}_{Op}(i, \underline{j}, u, \underline{v})\mathbf{);(}K(\underline{v}, t) \wedge R_{Op}(\underline{j}, h)\mathbf{)}$, or,

$$\begin{aligned}&(K^\bullet \wedge R^\bullet{}_{Op});(H \wedge Q_{Op}) \equiv (H^\bullet \wedge Q^\bullet{}_{Op});(K \wedge R_{Op}) \\ &\equiv P_{Op}\,.\end{aligned} \tag{5.25}$$

Thirdly, quoting the shorter form only, and using a prime to indicate after-states are being referred to we have,

$$\begin{aligned}&(K^{\bullet\prime} \wedge V^\bullet \wedge R^\bullet \wedge K^\bullet);(H \wedge Q_{Op} \wedge D_{Op}) \equiv \\ &(H^\bullet \wedge Q^\bullet{}_{Op} \wedge D^\bullet{}_{Op});(K' \wedge V \wedge R \wedge K) \equiv C_{Op}\,.\end{aligned} \tag{5.26}$$

The initialisation and step relations of *Ref* and *Ret* must satisfy conditions (5.27) to (5.30) below.

$$Init_{\mathrm{T}}(\underline{v}') \Rightarrow (\exists\, v', w' \bullet H^\bullet(([v'], [w']), \underline{v}')) \tag{5.27}$$

$$\begin{aligned}&H^\bullet(([v], [w]), \underline{v}) \wedge Q^\bullet{}_{Op}(([j], [k]), \underline{j}, ([v], [w]), \underline{v}) \wedge \\ &\quad stp_{Op_{\mathrm{T}}}(\underline{v}, \underline{j}, \underline{v}', \underline{p}) \Rightarrow \\ &\qquad (\exists\, v', w', p, q \bullet H^\bullet(([v'], [w']), \underline{v}') \vee \\ &\qquad\quad D^\bullet{}_{Op}(([v'], [w']), \underline{v}', ([p], [q]), \underline{p}; \\ &\qquad\qquad ([j], [k]), \underline{j}, ([v], [w]), \underline{v}))\end{aligned} \tag{5.28}$$

$$Init_{\mathrm{F}}(\underline{w}') \Rightarrow (\exists\, v', w' \bullet K^\bullet(([v'], [w']), \underline{w}')) \tag{5.29}$$

$$\begin{aligned}&K^\bullet(([v], [w]), \underline{w}) \wedge R^\bullet{}_{Op}(([j], [k]), \underline{k}) \wedge \\ &\quad stp_{Op_{\mathrm{F}}}(\underline{w}, \underline{k}, \underline{w}', \underline{q}) \Rightarrow \\ &\qquad (\exists\, v', w', p, q \bullet K^\bullet(([v'], [w']), \underline{w}') \wedge \\ &\qquad\quad V^\bullet{}_{Op}(([p], [q]), \underline{q}))\end{aligned} \tag{5.30}$$

Now we can give the transitions of *Univ*. For each Op_{U} a typical transition is u -(i, Op_{U}, o)-› u' or, more explicitly,

$$([v], [w])\ \text{-}(([j], [k]), Op_{\mathrm{U}}, ([p], [q]))\text{-›}\ ([v'], [w']) \tag{5.31}$$

iff $[v]$, $[w]$, $[j]$, $[k]$, $[p]$, $[q]$, $[v']$, $[w']$ satisfy

$$\begin{aligned}&\mathbf{(}\ \exists\, \underline{w}, \underline{k}, \underline{w}', \underline{q} \bullet K^\bullet(([v], [w]), \underline{w}) \wedge R^\bullet{}_{Op}(([j], [k]), \underline{k}) \wedge \\ &\quad stp_{Op_{\mathrm{F}}}(\underline{w}, \underline{k}, \underline{w}', \underline{q}) \wedge K^\bullet(([v'], [w']), \underline{w}') \wedge \\ &\quad V^\bullet{}_{Op}(([p], [q]), \underline{q})\ \mathbf{)} && \text{(a)} \\ &\vee \\ &\mathbf{(}\ \exists\, \underline{v}, \underline{j}, \underline{v}', \underline{p} \bullet H^\bullet(([v], [w]), \underline{v}) \wedge \\ &\quad Q^\bullet{}_{Op}(([j], [k]), \underline{j}, ([v], [w]), \underline{v}) \wedge stp_{Op_{\mathrm{T}}}(\underline{v}, \underline{j}, \underline{v}', \underline{p}) \wedge \\ &\quad (H^\bullet(([v'], [w']), \underline{v}') \vee \\ &\qquad D^\bullet{}_{Op}(([v'], [w']), \underline{v}', ([p], [q]), \underline{p}; \\ &\qquad\quad ([j], [k]), \underline{j}, ([v], [w]), \underline{v}))\ \mathbf{)} && \text{(b)}\end{aligned} \tag{5.32}$$

The initialization predicate $Init_{\mathrm{U}}(u')$ sets u' to any value $([v'], [w'])$ for which

$$\begin{aligned}&\mathbf{(}\ \exists\, \underline{w}' \bullet Init_{\mathrm{F}}(\underline{w}') \wedge K^\bullet(([v'], [w']), \underline{w}')\ \mathbf{)} \vee \\ &\mathbf{(}\ \exists\, \underline{v}' \bullet Init_{\mathrm{T}}(\underline{v}') \wedge H^\bullet(([v'], [w']), \underline{v}')\ \mathbf{)}\end{aligned} \tag{5.33}$$

is true. This completes the definition of *Univ*.

The components introduced above define a retrenchment from *Univ* to *Ret* because they satisfy POs (5.34) and (5.35), and a refinement from *Univ* to *Ref* because they satisfy POs (5.36) and (5.37).

$$Init_{\mathrm{T}}(\underline{v}') \Rightarrow (\exists\, u' \bullet Init_{\mathrm{U}}(u') \wedge H^\bullet(u', \underline{v}')) \tag{5.34}$$

$$\begin{aligned}&H^\bullet(u, \underline{v}) \wedge Q^\bullet{}_{Op}(i, \underline{j}, u, \underline{v}) \wedge stp_{Op_{\mathrm{T}}}(\underline{v}, \underline{j}, \underline{v}', \underline{p}) \Rightarrow \\ &\quad (\exists\, u', o \bullet stp_{Op_{\mathrm{U}}}(u, i, u', o) \wedge \\ &\qquad (H^\bullet(u', \underline{v}') \vee D^\bullet{}_{Op}(u', \underline{v}', o, \underline{p}; i, \underline{j}, u, \underline{v})))\end{aligned} \tag{5.35}$$

$$Init_{\mathrm{F}}(\underline{w}') \Rightarrow (\exists\, u' \bullet Init_{\mathrm{U}}(u') \wedge K^\bullet(u', \underline{w}')) \tag{5.36}$$

$$\begin{aligned}&K^\bullet(u, \underline{w}) \wedge R^\bullet{}_{Op}(i, \underline{k}) \wedge stp_{Op_{\mathrm{F}}}(\underline{w}, \underline{k}, \underline{w}', \underline{q}) \Rightarrow \\ &\quad (\exists\, u', o \bullet stp_{Op_{\mathrm{U}}}(u, i, u', o) \wedge \\ &\quad K^\bullet(u', \underline{w}') \wedge V^\bullet{}_{Op}(o, \underline{q}))\end{aligned} \tag{5.37}$$

Finally, the composition of the *Univ* to *Ret* retrenchment and the *Ret* to *Conc* refinement on the one hand, or the *Univ* to *Ref* refinement and the *Ref* to *Conc* retrenchment on the other, yield a retrenchment from *Univ* to *Conc* because they satisfy POs (5.38) and (5.39) below. What is more, (5.24) to (5.26) show they both give the *same* retrenchment, with retrieves, within, and concedes relations given respectively by G, P_{Op} and C_{Op}.

$$Init_{\mathrm{C}}(t') \Rightarrow (\exists\, u' \bullet Init_{\mathrm{U}}(u') \wedge G(u', t')) \tag{5.38}$$

$$\begin{aligned}&G(u, t) \wedge P_{Op}(i, h, u, t) \wedge stp_{Op_{\mathrm{C}}}(t, h, t', s) \Rightarrow \\ &\quad (\exists\, u', o \bullet stp_{Op_{\mathrm{U}}}(u, i, u', o) \wedge \\ &\qquad (G(u', t') \vee C_{Op}(u', t', o, s; i, h, u, t)))\end{aligned} \tag{5.39}$$

Note also that *Univ* possesses the following properties.

$$\begin{aligned}&Init_{\mathrm{U}}(u') \ \Rightarrow \\ &\quad (\ \mathbf{(}\exists\, \underline{w}' \bullet Init_{\mathrm{F}}(\underline{w}') \wedge K^\bullet(u', \underline{w}')\mathbf{)} \vee \\ &\qquad (\exists\, \underline{v}' \bullet Init_{\mathrm{T}}(\underline{v}') \wedge H^\bullet(u', \underline{v}'))\ )\end{aligned} \tag{U1}$$

$$\begin{aligned}&stp_{Op_{\mathrm{U}}}(u, i, u', o) \ \Rightarrow \mathbf{(} \\ &\quad \mathbf{(}\exists\, \underline{w}, \underline{k}, \underline{w}', \underline{q} \bullet K^\bullet(u, \underline{w}) \wedge R^\bullet{}_{Op}(i, \underline{k}) \wedge \\ &\qquad stp_{Op_{\mathrm{F}}}(\underline{w}, \underline{k}, \underline{w}', \underline{q}) \wedge K^\bullet(u', \underline{w}') \wedge V^\bullet{}_{Op}(o, \underline{q})\mathbf{)} \\ &\quad \vee \\ &\quad (\exists\, \underline{v}, \underline{j}, \underline{v}', \underline{p} \bullet H^\bullet(u, \underline{v}) \wedge Q^\bullet{}_{Op}(i, \underline{j}, u, \underline{v}) \wedge\end{aligned}$$

$$\begin{aligned}&\qquad \mathit{stp}_{Op_{\mathrm{T}}}(\underline{v}, \underline{j}, \underline{v}', \underline{p}) \wedge \\ &\qquad (H^{\bullet}(u', \underline{v}') \vee D^{\bullet}{}_{Op}(u', \underline{v}', o, \underline{p}; i, \underline{j}, u, \underline{v})))\ \big)\end{aligned} \tag{U2}$$

$$K^{\bullet}(u', w') \wedge K^{\bullet}(u', \underline{w}') \ \Rightarrow\ w' \sim \underline{w}' \tag{U3}$$

$$\begin{aligned}&\mathbf{(}H^{\bullet}(u', v') \vee D^{\bullet}{}_{Op}(u', v', ...)\mathbf{)} \wedge \\ &\quad \mathbf{(}H^{\bullet}(u', \underline{v}') \vee D^{\bullet}{}_{Op}(u', \underline{v}', ...)\mathbf{)} \ \Rightarrow\ v' \sim \underline{v}'\end{aligned} \tag{U4}$$

$$V^{\bullet}{}_{Op}(o, q) \wedge V^{\bullet}{}_{Op}(o, \underline{q}) \ \Rightarrow\ q \sim \underline{q} \tag{U5}$$

$$D^{\bullet}{}_{Op}(..., o, p; ...) \wedge D^{\bullet}{}_{Op}(..., o, \underline{p}; ...) \ \Rightarrow\ p \sim \underline{p} \tag{U6}$$

$$H^{\bullet}(u', \underline{v}') \ \Rightarrow\ (\exists\, \underline{w} \bullet K^{\bullet}(u', \underline{w}')) \tag{U7}$$

$$H^{\bullet}(u', \underline{v}') \wedge K^{\bullet}(u', \underline{w}') \ \Rightarrow\ H^{\bullet}(([\underline{v}'], [\underline{w}']), \underline{v}') \tag{U8}$$

$$\begin{aligned}&H^{\bullet}(u, \underline{v}) \wedge Q^{\bullet}{}_{Op}(i, \underline{j}, u, \underline{v}) \wedge \\ &\quad D^{\bullet}{}_{Op}(u', \underline{v}', o, \underline{p}; i, \underline{j}, u, \underline{v}) \ \Rightarrow \\ &\qquad (\exists\, \underline{w}', \underline{q}, \underline{k}, \underline{w} \bullet K^{\bullet}(u', \underline{w}') \wedge V^{\bullet}{}_{Op}(o, \underline{q}) \wedge \\ &\qquad\quad R^{\bullet}{}_{Op}(i, \underline{k}) \wedge K^{\bullet}(u, \underline{w}))\end{aligned} \tag{U9}$$

$$\begin{aligned}&K^{\bullet}(u', \underline{w}') \wedge V^{\bullet}{}_{Op}(o, \underline{q}) \wedge R^{\bullet}{}_{Op}(i, \underline{k}) \wedge K^{\bullet}(u, \underline{w}) \wedge \\ &\quad H^{\bullet}(u, \underline{v}) \wedge Q^{\bullet}{}_{Op}(i, \underline{j}, u, \underline{v}) \wedge \\ &\quad D^{\bullet}{}_{Op}(u', \underline{v}', o, \underline{p}; i, \underline{j}, u, \underline{v}) \ \Rightarrow \\ &\qquad H^{\bullet}(([\underline{v}], [\underline{w}]), \underline{v}) \wedge Q^{\bullet}{}_{Op}(([\underline{j}], [\underline{k}]), \underline{j}, ([\underline{v}], [\underline{w}]), \underline{v}) \wedge \\ &\qquad\quad D^{\bullet}{}_{Op}(([\underline{v}'], [\underline{w}']), \underline{v}', ([\underline{p}], [\underline{q}]), \underline{p}; \\ &\qquad\qquad ([\underline{j}], [\underline{k}]), \underline{j}, ([\underline{v}], [\underline{w}]), \underline{v}))\end{aligned} \tag{U10}$$

$$V^{\bullet}{}_{Op}(o, q) \ \Rightarrow\ (\exists\, p \bullet V^{\bullet}{}_{Op}(([p], [q]), q)) \tag{U11}$$

The data given so far satisfies part (1) of the theorem below.

Theorem 5.1 Let there be a retrenchment from *Ref* to *Conc*, and a refinement from *Ret* to *Conc* (as shown in Fig. 1), which satisfy conditions (5.27) to (5.30). Then the following hold.

(1) There is a universal system *Univ* for which there is a retrenchment from *Univ* to *Ret* and an I/O-filtered refinement from *Univ* to *Ref* whose compositions with the original refinement and retrenchment respectively are equal as retrenchments from *Univ* to *Conc*, and which satisfies (U1) to (U11).

(2) Whenever there is a system *Xtra* and a retrenchment from *Xtra* to *Ret* and an I/O-filtered refinement from *Xtra* to *Ref* whose compositions with the original refinement and retrenchment respectively are equal as retrenchments from *Xtra* to *Conc*, and which satisfies (X1) to (X11) below, then there is an I/O-filtered refinement from *Univ* to *Xtra* such that $K^{\circ};H^{\sim} \Rightarrow H^{\bullet}$, $(K^{\circ} \wedge R^{\circ});(H^{\sim} \wedge Q^{\sim}) \Rightarrow (H^{\bullet} \wedge Q^{\bullet})$, $(K^{\circ\prime} \wedge V^{\circ} \wedge R^{\circ} \wedge K^{\circ});(H^{\sim} \wedge Q^{\sim} \wedge D^{\sim}) \Rightarrow (H^{\bullet} \wedge Q^{\bullet} \wedge D^{\bullet})$, and such that $K^{\circ};K^{\sim} \Rightarrow K^{\bullet}$, $R^{\circ};R^{\sim} \Rightarrow R^{\bullet}$, $V^{\circ};V^{\sim} \Rightarrow V^{\bullet}$.

(3) Whenever a system *Univ** has properties (1) and (2) above of *Univ*, then *Univ* and *Univ** are mutually interrefinable.

Part (2) of Theorem 5.1 is concerned with the refinement from *Univ* to *Xtra*. Suppose there is an I/O-filtered refinement from *Xtra* to *Ref* given by retrieve relation $K^{\sim}$, within relation $R^{\sim}$, and nevertheless relation $V^{\sim}$; and a retrenchment from *Xtra* to *Ret* given by retrieve relation $H^{\sim}$, within relation $Q^{\sim}$, and concedes relation $D^{\sim}$. Let the state, input and output spaces of *Xtra* be given by $u^{\sim} \in \mathsf{U}^{\sim}$, $i^{\sim} \in \mathsf{I}^{\sim}$, $o^{\sim} \in \mathsf{O}^{\sim}$ and let the initialisation and step predicates for *Xtra* be $\mathit{Init}_{\mathrm{X}}$ and $\mathit{stp}_{Op_{\mathrm{X}}}$. Finally let *Xtra* have properties (X1) to (X11) below.

$$\begin{aligned}&\mathit{Init}_{\mathrm{X}}(u^{\sim\prime}) \ \Rightarrow \\ &\quad \big(\ \mathbf{(}\exists\, \underline{w}' \bullet \mathit{Init}_{\mathrm{F}}(\underline{w}') \wedge K^{\sim}(u^{\sim\prime}, \underline{w}')\mathbf{)} \vee \\ &\qquad (\exists\, \underline{v}' \bullet \mathit{Init}_{\mathrm{T}}(\underline{v}') \wedge H^{\sim}(u^{\sim\prime}, \underline{v}'))\ \big)\end{aligned} \tag{X1}$$

$$\begin{aligned}&\mathit{stp}_{Op_{\mathrm{X}}}(u^{\sim}, i^{\sim}, u^{\sim\prime}, o^{\sim}) \ \Rightarrow \\ &\quad \big(\ \mathbf{(}\exists\, \underline{w}, \underline{k}, \underline{w}', \underline{q} \bullet K^{\sim}(u^{\sim}, \underline{w}) \wedge R^{\sim}{}_{Op}(i^{\sim}, \underline{k}) \wedge \\ &\qquad \mathit{stp}_{Op_{\mathrm{F}}}(\underline{w}, \underline{k}, \underline{w}', \underline{q}) \wedge K^{\sim}(u^{\sim\prime}, \underline{w}') \wedge V^{\sim}{}_{Op}(o^{\sim}, \underline{q})\mathbf{)} \\ &\quad \vee \\ &\quad\ (\exists\, \underline{v}, \underline{j}, \underline{v}', \underline{p} \bullet H^{\sim}(u^{\sim}, \underline{v}) \wedge Q^{\sim}{}_{Op}(i^{\sim}, \underline{j}, u^{\sim}, \underline{v}) \wedge \\ &\qquad \mathit{stp}_{Op_{\mathrm{T}}}(\underline{v}, \underline{j}, \underline{v}', \underline{p}) \wedge \\ &\qquad (H^{\sim}(u^{\sim\prime}, \underline{v}') \vee D^{\sim}{}_{Op}(u^{\sim\prime}, \underline{v}', o^{\sim}, \underline{p}; i^{\sim}, \underline{j}, u^{\sim}, \underline{v})))\ \big)\end{aligned} \tag{X2}$$

$$K^{\sim}(u^{\sim\prime}, w') \wedge K^{\sim}(u^{\sim\prime}, \underline{w}') \ \Rightarrow\ w' \sim \underline{w}' \tag{X3}$$

$$\begin{aligned}&\mathbf{(}H^{\sim}(u^{\sim\prime}, v') \vee D^{\sim}{}_{Op}(u^{\sim\prime}, v', ...)\mathbf{)} \wedge \\ &\mathbf{(}H^{\sim}(u^{\sim\prime}, \underline{v}') \vee D^{\sim}{}_{Op}(u^{\sim\prime}, \underline{v}', ...)\mathbf{)} \ \Rightarrow\ v' \sim \underline{v}'\end{aligned} \tag{X4}$$

$$V^{\sim}{}_{Op}(o^{\sim}, q) \wedge V^{\sim}{}_{Op}(o^{\sim}, \underline{q}) \ \Rightarrow\ q \sim \underline{q} \tag{X5}$$

$$D^{\sim}{}_{Op}(..., o^{\sim}, p; ...) \wedge D^{\sim}{}_{Op}(..., o^{\sim}, \underline{p}; ...) \ \Rightarrow\ p \sim \underline{p} \tag{X6}$$

$$H^{\sim}(u^{\sim\prime}, \underline{v}') \ \Rightarrow\ (\exists\, \underline{w}' \bullet K^{\sim}(u^{\sim\prime}, \underline{w}')) \tag{X7}$$

$$H^{\sim}(u^{\sim\prime}, \underline{v}') \wedge K^{\sim}(u^{\sim\prime}, \underline{w}') \ \Rightarrow\ H^{\bullet}(([\underline{v}'], [\underline{w}']), \underline{v}') \tag{X8}$$

$$\begin{aligned}&H^{\sim}(u^{\sim}, \underline{v}) \wedge Q^{\sim}{}_{Op}(i^{\sim}, \underline{j}, u^{\sim}, \underline{v}) \wedge \\ &D^{\sim}{}_{Op}(u^{\sim\prime}, \underline{v}', o^{\sim}, \underline{p}; i^{\sim}, \underline{j}, u^{\sim}, \underline{v}) \ \Rightarrow \\ &\quad (\exists\, \underline{w}', \underline{q}, \underline{k}, \underline{w} \bullet K^{\sim}(u^{\sim\prime}, \underline{w}') \wedge V^{\sim}{}_{Op}(o^{\sim}, \underline{q}) \wedge \\ &\qquad R^{\sim}{}_{Op}(i^{\sim}, \underline{k}) \wedge K^{\sim}(u^{\sim}, \underline{w}))\end{aligned} \tag{X9}$$

$$\begin{aligned}&K^{\sim}(u^{\sim\prime}, \underline{w}') \wedge V^{\sim}{}_{Op}(o^{\sim}, \underline{q}) \wedge R^{\sim}{}_{Op}(i^{\sim}, \underline{k}) \wedge K^{\sim}(u^{\sim}, \underline{w}) \wedge \\ &H^{\sim}(u^{\sim}, \underline{v}) \wedge Q^{\sim}{}_{Op}(i^{\sim}, \underline{j}, u^{\sim}, \underline{v}) \wedge \\ &D^{\sim}{}_{Op}(u^{\sim\prime}, \underline{v}', o^{\sim}, \underline{p}; i^{\sim}, \underline{j}, u^{\sim}, \underline{v}) \ \Rightarrow \\ &\quad H^{\bullet}(([\underline{v}], [\underline{w}]), \underline{v}) \wedge Q^{\bullet}{}_{Op}(([\underline{j}], [\underline{k}]), \underline{j}, ([\underline{v}], [\underline{w}]), \underline{v}) \wedge \\ &\quad D^{\bullet}{}_{Op}(([\underline{v}'], [\underline{w}']), \underline{v}', ([\underline{p}], [\underline{q}]), \underline{p}; \\ &\qquad ([\underline{j}], [\underline{k}]), \underline{j}, ([\underline{v}], [\underline{w}]), \underline{v})\end{aligned} \tag{X10}$$

$$V^{\sim}{}_{Op}(o^{\sim}, q) \ \Rightarrow\ (\exists\, p \bullet V^{\bullet}{}_{Op}(([p], [q]), q)) \tag{X11}$$

Notice properties (U1) to (U11) correspond to properties (X1) to (X11). Hence *Univ* and *Xtra* belong to the same class of systems that complete the square.

We now define relations K°, $R^{\circ}{}_{Op}$, and $V^{\circ}{}_{Op}$.

$$\begin{aligned}
&K^{\circ}(u, u^{\sim}) \;=\; K^{\circ}(([v], [w]), u^{\sim}) \;= \\
&\quad \boldsymbol{(}\forall\, \underline{w} \bullet K^{\sim}(u^{\sim}, \underline{w}) \Rightarrow K^{\bullet}(([v], [w]), \underline{w})\boldsymbol{)} \;\wedge \\
&\quad \boldsymbol{(}\forall\, \underline{v} \bullet H^{\sim}(u^{\sim}, \underline{v}) \Rightarrow H^{\bullet}(([v], [w]), \underline{v})\boldsymbol{)} \;\wedge \\
&\quad \boldsymbol{(}\forall\, \underline{v}, \underline{\underline{o}}^{\sim}, \underline{\underline{p}}, \underline{\underline{i}}^{\sim}, \underline{\underline{j}}, \underline{\underline{u}}^{\sim}, \underline{\underline{v}} \bullet H^{\sim}(\underline{\underline{u}}^{\sim}, \underline{\underline{v}}) \wedge Q^{\sim}{}_{Op}(\underline{\underline{i}}^{\sim}, \underline{\underline{j}}, \underline{\underline{u}}^{\sim}, \underline{\underline{v}}) \wedge \\
&\qquad D^{\sim}{}_{Op}(u^{\sim}, \underline{v}, \underline{\underline{o}}^{\sim}, \underline{\underline{p}}; \underline{\underline{i}}^{\sim}, \underline{\underline{j}}, \underline{\underline{u}}^{\sim}, \underline{\underline{v}}) \Rightarrow \\
&\qquad\quad (\exists\, \underline{\underline{o}}, \underline{\underline{i}}, \underline{\underline{u}} \bullet H^{\bullet}(\underline{\underline{u}}, \underline{\underline{v}}) \wedge Q^{\bullet}{}_{Op}(\underline{\underline{i}}, \underline{\underline{j}}, \underline{\underline{u}}, \underline{\underline{v}}) \wedge \\
&\qquad\qquad D^{\bullet}{}_{Op}(([v], [w]), \underline{v}, \underline{\underline{o}}, \underline{\underline{p}}; \underline{\underline{i}}, \underline{\underline{j}}, \underline{\underline{u}}, \underline{\underline{v}}))\boldsymbol{)}
\end{aligned} \tag{5.40}$$

$$\begin{aligned}
&R^{\circ}{}_{Op}(i, i^{\sim}) \;=\; R^{\circ}{}_{Op}(([j], [k]), i^{\sim}) \;= \\
&\quad \boldsymbol{(}\forall\, \underline{k} \bullet R^{\sim}{}_{Op}(i^{\sim}, \underline{k}) \Rightarrow R^{\bullet}{}_{Op}(([j], [k]), \underline{k})\boldsymbol{)} \;\wedge \\
&\quad \boldsymbol{(}\forall\, \underline{j}, \underline{v}, u, u^{\sim} \bullet K^{\circ}(u, u^{\sim}) \Rightarrow \\
&\qquad (\, H^{\sim}(u^{\sim}, \underline{v}) \wedge Q^{\sim}{}_{Op}(i^{\sim}, \underline{j}, u^{\sim}, \underline{v}) \Leftrightarrow \\
&\qquad\quad H^{\bullet}(u, \underline{v}) \wedge Q^{\bullet}{}_{Op}(([j], [k]), \underline{j}, u, \underline{v})\;)\;\boldsymbol{)}
\end{aligned} \tag{5.41}$$

$$\begin{aligned}
&V^{\circ}{}_{Op}(o, o^{\sim}) \;=\; V^{\circ}{}_{Op}(([p], [q]), o^{\sim}) \;= \\
&\quad \boldsymbol{(}\forall\, \underline{q} \bullet V^{\sim}{}_{Op}(o^{\sim}, \underline{q}) \Rightarrow V^{\bullet}{}_{Op}(([p], [q]), \underline{q})\boldsymbol{)} \;\wedge \\
&\quad \boldsymbol{(}\forall\, \underline{v}', \underline{p}, \underline{j}, \underline{v}, u', u^{\sim\prime}, i, i^{\sim}, u, u^{\sim} \bullet \\
&\qquad H^{\sim}(u^{\sim}, \underline{v}) \wedge Q^{\sim}{}_{Op}(i^{\sim}, \underline{j}, u^{\sim}, \underline{v}) \wedge \\
&\qquad D^{\sim}{}_{Op}(u^{\sim\prime}, \underline{v}', o^{\sim}, \underline{p}; i^{\sim}, \underline{j}, u^{\sim}, \underline{v}) \wedge \\
&\qquad K^{\circ}(u', u^{\sim\prime}) \wedge R^{\circ}{}_{Op}(i, i^{\sim}) \wedge K^{\circ}(u, u^{\sim}) \Rightarrow \\
&\qquad\quad H^{\bullet}(u, \underline{v}) \wedge Q^{\bullet}{}_{Op}(i, \underline{j}, u, \underline{v}) \wedge \\
&\qquad\quad D^{\bullet}{}_{Op}(u', \underline{v}', ([p], [q]), \underline{p}; i, \underline{j}, u, \underline{v}))\boldsymbol{)}
\end{aligned} \tag{5.42}$$

The above satisfy the necessary inclusions, and are the retrieve, within and nevertheless relations of an I/O-filtered refinement from *Univ* to *Xtra* because POs (5.43) and (5.44) hold. This completes part (2).

$$Init_{X}(u^{\sim\prime}) \Rightarrow (\exists\, u' \bullet Init_{U}(u') \wedge K^{\circ}(u', u^{\sim\prime})) \tag{5.43}$$

$$\begin{aligned}
&K^{\circ}(u, u^{\sim}) \wedge R^{\circ}{}_{Op}(i, i^{\sim}) \wedge stp_{Op_{X}}(u^{\sim}, i^{\sim}, u^{\sim\prime}, o^{\sim}) \Rightarrow \\
&\quad (\exists\, u', o \bullet stp_{Op_{U}}(u, i, u', o) \wedge \\
&\quad K^{\circ}(u', u^{\sim\prime}) \wedge V^{\circ}{}_{Op}(o, o^{\sim}))
\end{aligned} \tag{5.44}$$

Part (3) follows readily by observing that for a system *Univ** having the same properties as *Univ*, there will be an I/O-filtered refinement from *Univ* to *Univ** and an I/O-filtered refinement from *Univ** to *Univ*. This completes the reconciliation.

In this section we have shown how to take a retrenchment and a refinement to a common system and construct a canonical system that reconciles the two, giving the most abstract construction possible within a class of systems.

6. An Example

We return to our example and use it to illustrate the construction of *Univ*. First observe that conditions (5.27) to (5.30) hold. The states of *Univ* are pairs of equivalence classes $u = ([v], [w])$. So ([<2, 3, 88>], [{1, 9, 43454}]) is a valid state. It is instructive to work out the elements of the equivalence classes $[v]$ and $[w]$.

We begin with $[w]$. As we shall see there are three possibilities for a typical class. Consider w = <1, 2>, a state of *Ref*. This is linked by H to t = <1, 2> in *Conc*. Similarly H links w = <2, 1> and t = <2, 1>. Now both t's are linked to {1, 2} by K. Therefore, by (5.5), one class is $[w]$ = {<1, 2>, <2, 1>}. The same applies to any w with length 9 or less. Thus one possibility for $[w]$ is a set of sequences which are serialisations of X, where $\mathrm{X} \in \mathbb{P}(\mathsf{NAT}) \wedge |\mathrm{X}| \leq 9$.

Now take w = <1..10>. This is linked to t = <1..10> by H, which in turn is linked to v = {1..10} by K. In addition each <1..10, x> in *Ref*, where $x \in \mathsf{NAT} \wedge x > 10$, is linked to t = <1..10> by D_{Add}. The same arrangement applies for any sequence w of length 10. So by (5.5) another possibility for $[w]$ is a set of sequences which are serialisations of X and Y, where $\mathrm{X} \in \mathbb{P}(\mathsf{NAT}) \wedge |\mathrm{X}| = 10$, $\mathrm{Y} \in \mathbb{P}(\mathsf{NAT}) \wedge |\mathrm{Y}| = 11$ and $\mathrm{X} \subseteq \mathrm{Y}$. The final possibility for $[w]$ is a singleton whose member is a sequence of length 12 or more.

For $[v]$ the situation is much simpler. Each class is a singleton which is an element of $\mathbb{P}(\mathsf{NAT})$. For the I/O spaces the input and output classes are singleton sets whose element is a NAT, plus, for $\mathsf{P}_{Rem}/\sim_{\mathsf{P}_{Rem}}$, the class [EMPTY].

Let us now examine a typical non-boundary step for Add_{U}. Its structure can be deduced by inspecting (5.32). Well, either (5.32a) or (5.32b) must hold. Suppose (5.32b) holds for $stp_{Add_{T}}(\underline{v}, \underline{j}, \underline{v}')$. Then because the classes partitioning V and J are singletons, $u = ([\underline{v}], \text{-})$, $i = ([\underline{j}], \text{-})$ and $u' = ([\underline{v}'], \text{-})$. Furthermore, when $H^{\bullet}$, $Q^{\bullet}$ and $H^{\bullet\prime} \vee D^{\bullet}$ hold, then so do the $K^{\bullet}$, $R^{\bullet}$ and $K^{\bullet\prime}$ of (5.32a), for all valid $\underline{w}$, $\underline{k}$ and $\underline{w}'$. Notice also that $H^{\bullet}(([\underline{v}], [w]), \underline{v})$ pairs together classes $[\underline{v}]$ and $[w]$ such that members of $[w]$ are serialisations of $\underline{v}$. In the same way $Q^{\bullet}$ pairs $[\underline{j}]$ and $[k]$, and $H^{\bullet\prime} \vee D^{\bullet}$ pairs $[\underline{v}']$ and $[w']$. Thus it follows that (6.1) below is a suitable instance.

$$\begin{aligned}
&([<1,2>], [\{1,2\}]) \text{ -}(([3], [3]), Add_{U})\text{-›} \\
&\quad ([<1,2,3>], [\{1,2,3\}])\,.
\end{aligned} \tag{6.1}$$

When only (5.32a) holds we get junk transitions which consist of Add_{F} steps which have no corresponding Add_{T} step. One such Add_{F} step is <1..15> -(16, Add_{F})-› <1..16>. We will say nothing more about junk transitions.

Using the same analysis, a boundary case is

$$\begin{aligned}
&([<1..10>], [\{1..10\}]) \text{ -}(([11], [11]), Add_{U})\text{-›} \\
&\quad ([<1..11>], [\{1..10\}])\,.
\end{aligned} \tag{6.2}$$

So the equivalence classes bunch together all *Ref* and *Ret* steps which correspond. For instance, from (6.2), {1..10} -(11, Add_{T})-› {1..10} matches both <1..10> -(11, Add_{F})-› <1..11> and <10..1> -(11, Add_{F})-› <10..1,11>, amongst others. Thus we interpret *Univ* as being a system which essentially enjoys the behaviour of *Ref* but with data at the level of abstraction of *Ret*.

For Rem_{U} a typical non-boundary step is shown in (6.3).

$$([<1,2>], [\{1,2\}]) \xrightarrow{(Rem_U, ([1], [1]))} ([<2>], [\{2\}]) \tag{6.3}$$

Note that it is not possible to find values that satisfy (5.32) for a step which describes the removal of an element when there are none left. This is entirely consistent with our interpretation of *Univ*: *Ref* does not have a transition in this case that can be lifted to the level of *Ret*.

7. Conclusions

We have shown how a retrenchment and a refinement to a common system can be reconciled by a system universal within a class of similar reconciliations.

We sought the most general result, by trying to restrict the relations of the given retrenchment and refinement as little as possible, thus admitting the greatest number of situations. Work is ongoing to produce a mathematically more elegant solution, by enforcing a greater degree of regularity on the component relations, which, although not as general as the current result, should still be applicable to a large number of realistic problems.

The pushout-like construction of this paper is one result in the full algebraic integration of retrenchment and refinement. Other results appear in the already cited works [2, 12, 13], and yet others are work in progress. The complete suite of results will ensure that retrenchment adds a truly fresh dimension to possible development routes, complementing refinement in situations where it struggles to give a convincing account, rather than being a stand-alone technique that fragments development routes into incompatible branches. Ultimately the developer gains by having a richer integrated palette of tools with which to organise the building of systems.

References

1. Abrial J. R. *The B-Book: Assigning Programs to Meanings.* Cambridge University Press, 1996.
2. Banach R. Maximally Abstract Retrenchments. In *Proc. IEEE ICFEM-00*, 133-142, 2000.
3. Banach R., Poppleton M. Retrenchment: An Engineering Variation on Refinement. In *Proc. B-98*, Bert (ed.), Springer, LNCS **1393**, 129-147, 1998. See also Technical Report UMCS-99-3-2, `http://www.cs.man.ac.uk/cstechrep`
4. Banach R., Poppleton M. Retrenchment and Punctured Simulation. In *Proc. IFM-99*, Taguchi, Galloway (eds.), Springer, 457-476, 1999.
5. Banach R., Poppleton M. Retrenchment, Refinement and Simulation. In *Proc. ZB-00*, Bowen, Dunne, Galloway, King (eds.), LNCS **1878**, 304-323, Springer (2000).
6. Banach R., Poppleton M. Sharp Retrenchment, Modulated Refinement and Simulation. *Form. Asp. Comp.* **11**, 498-540, 1999.
7. Banach R., Poppleton M. Engineering and Theoretical Underpinnings of Retrenchment. Submitted. Available online as `http://www.cs.man.ac.uk/~banach/some.pubs/Retrench.Underpin.ps.gz`.
8. Banach R., Jeske C. Output Retrenchments, Defaults, Stronger Compositions, Feature Engineering. Submitted. Available online as `http://www.cs.man.ac.uk/~banach/some.pubs/Retrench.Def.Out.ps.gz`.
9. de Roever W.-P., Engelhardt K. *Data Refinement: Model-Oriented Proof Methods and their Comparison.* C.U.P., 1998.
10. Derrick J., Boiten E. *Refinement in Z and Object-Z: Foundations and Advanced Applications.* Springer (2001).
11. Hayes I. J., Sanders J. W. Specification by Interface Separation. *Form. Asp. Comp.* **7**, 430-439, 1995.
12. Jeske C., Banach R. Minimally and Maximally Abstract Retrenchments. In *Proc. IFM-02*, LNCS **2335**, 380-399, Springer (2002).
13. Jeske C., Banach R. Reconciling Retrenchment and Refinements I. Submitted.
14. Jones C. B. *Systematic Software Development Using VDM*, 2nd ed. Prentice Hall, 1990.
15. Liu S. Evolution: A More Practical Approach than Refinement for Software Development. In *Proc. ICECCS-97*, 142-151, IEEE, 1997.
16. Mikhajlova A, Sekerinski E. Class Refinement and Interface Refinement in Object-Oriented Programs. In *Proc. FME-97*, Fitzgerald, Jones, Lucas (eds.), LNCS **1313**, 82-101, Springer, 1997.
17. Neilson D. S. *From Z to C: Illustration of a Rigorous Development Method.* PhD Thesis, Oxford University Programming Research Group, Monograph PRG-101, 1990.
18. Owe O. An Approach to Program Reasoning Based on a First Order Logic for Partial Functions. University of Oslo Institute of Informatics Research Report No. 89, 1985.
19. Owe O. Partial Logics Reconsidered: A Conservative Approach. *Form. Asp. Comp.* **3**, 1-16, 1993.
20. Poppleton M., Banach R. Retrenchment: Extending Refinement for Continuous and Control Systems. In *Proc. IWFM'00*, Springer Electronic Workshop in Computer Science Series, `http://ewic.org.uk/ewic`. Springer, 2000.
21. Poppleton, M. and Banach, R. Controlling Control Systems: An Application of Evolving Retrenchment. In *Proc. ZB2002: Formal Specification and Development in Z and B*, Springer LNCS, 2002.
22. Smith G. Stepwise Development from Ideal Specifications. In Edwards (ed.), *Aus. Comp. Sci. Conf.* **22**(1), 227-233. IEEE Computer Society, 2000.
23. Smith G., Fidge C. Incremental Development of Real-Time Requirements: The Light Control Case Study. *JUCS* **6**, 704-730, 2000.
24. Spivey J. M. *The Z Notation: A Reference Manual*, 2nd ed. Prentice Hall, 1993.
25. Stepney S., Cooper D., Woodcock J. More Powerful Z Data Refinement: Pushing the State of the Art in Industrial Refinement. In *Proc. ZUM-98*, Bowen, Fett, Hinchey (eds.), LNCS **1493**, 284-307, Springer, 1998.
26. Woodcock J., Davies J. *Using Z, Specification, Refinement and Proof.* Prentice Hall, 1996.