Reconciling Retrenchments and Refinements I

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

Abstract. Some of the problems 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 presented. The paper then examines one particular way in which a retrenchment and a refinement step can be combined by exploring the pushout-like problem of completing a square. Thus given both a retrenchment of an abstract model, and a refinement of the same model, the problem of finding a model that is both a refinement of the retrenchment and a retrenchment of the refinement is examined. The construction given solves the problem in a universal manner, in that it is the most abstract reconciliation of the initial retrenchment and refinement.
Keywords: Refinement, Retrenchment, Integration, Formal Methods.

1 Introduction

Retrenchment was introduced in [2] in order to overcome the drawbacks of relying on refinement alone as the only way of going from an abstract to a concrete model of a system in a completely formal manner. Subsequently the technique was developed in [3, 4, 5, 7]. A broad reappraisal of the issues appears in [6]. Refinement, particularly in formulations that emphasise total correctness, imposes stringent constraints on the relationship that can hold between models, and this can restrict the applicability of the technique quite severely. This is most keenly felt in situations where the conception of the system starts with physical considerations, described using conventional applied mathematics, rather than the discrete systems ubiquitous in presentations of refinement. In such circumstances it is commonly found that all except the last few steps of the development process have to be performed informally — because the proof obligations (POs) of refinement are so demanding that adjacent pairs of models higher up the development hierarchy are simply unable to satisfy them. Thus we lose all the benefits of full formalism for the majority of the development.

In contrast to refinement, a retrenchment step strengthens the precondition and weakens the postcondition. Retrenchment also permits the mixing of state and I/O data between levels. This is achieved by having two extra predicates per retrenched operation, the within and concedes clauses. The former expresses the precondition strengthening, and the latter expresses the postcondition weakening. The key feature of retrenchment is that non-refinement-like behaviour can be accommodated via the weakened postcondition. This permits inconvenient low level detail of the true system from interfering with an idealised model at a high level of abstraction, giving hopefully cleaner, more understandable, earlier formalizable development routes.

As we are free to postpone the introduction of low level detail in retrenchment, it follows that retrenchment permits the gradual incorporation of requirements into a final specification. Thus retrenchment can be seen as providing a flexible *specification constructor* that we can combine with other techniques to enrich the palette we have for building specifications out of smaller simpler pieces. The principal specifi-

cation constructors include coproduct and similar categorical constructions, parameterisation mechanisms, and refinement itself. Since retrenchment is not equivalent to any of these, it properly enlarges the collection of available development routes: a more concrete model that was not derivable from one or more desired abstract precursors, may become so when retrenchment is added to the armoury. This is a positive thing as the relative value of different development routes ought to be judged on domain-specific engineering grounds, rather than being hamstrung by the limitations of available specification construction mechanisms. This is particularly the case in areas of engineering where there are already well accepted development routes for the systems of interest. Formal techniques should support and strengthen these instead of advocating their disruption.

Having suggested that retrenchment ought to coexist with other specification constructors, the interplay between these techniques is clearly of interest and boils down to a family of algebraic problems. The interaction with refinement is a key issue, and in [1, 11] we have explored canonical factorisations of an arbitrary retrenchment into a refinement and a 'retrenchment which preserves the level of abstraction'. In this paper we focus on the pushout-like problem of completing a square. Thus suppose we have a system *Abs* which is refined to a system *Ref*, and also retrenched to a system *Ret*. We want to find a system *Univ* such that *Univ* is *simultaneously* a retrenchment of *Ref* and a refinement of *Ret*. Moreover we seek to characterise the construction in a suitably universal manner, viz. that any other system *Xtra* which achieves the same reconciliation must be refinable from *Univ*. Thus *Univ* is the most abstract possible completion of the square. See the diagram in Fig. 1.

The interest in the reconciliation is not only algebraic; it also has a tangible software engineering payoff. Suppose a formal development via refinement of some software already exists, and later, or perhaps even during the latter stages of the development, a change to the original specification is adopted. Let this change be expressed as an alteration of the top level model. Frequently the nature of the alteration makes the new top level model non-derivable from its predecessor using previously available techniques. However the more flexible nature of retrenchment increases the probability that the new system arises as a retrenchment of its predecessor. Assuming this to be the case, then the reconciliation of retrenchment and refinement described here becomes an important tool for avoiding a complete reworking of the original development.

A further benefit of the reconciliation is that it helps to assimilate retrenchment, a relatively recent and unfamiliar technique, into the fold of acceptable and trusted formal development 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 this paper is as follows. Section 2 presents a simple example which shows one way in which refinement can obstruct the natural development of a specification. Section 3 introduces the basics of the retrenchment mechanism. Section 4 presents I/O-filtered refinements, the kind of refinement used in the reconciliation. Section 5 presents the reconciliation itself. Section 6 examines the structure of the universal system achieving the reconciliation. Section 7 concludes.

Notation. 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 Some Drawbacks of Refinement

Suppose that during the development of a specification for an industrial scale project, we have produced a model that includes the following elements. A set of NATs, which is a component of the state, an operation, *Add*, which adds numbers to the set, and an operation, *Rem*, which removes numbers from it. At each step in the development we have used refinement to provide assurance that already specified behaviour has been preserved. Having satisfied ourselves that the model thus far expresses the key abstract properties of the system, we now turn to consider lower level implementational constraints. Whatever data structure we choose to use to implement the set in the final code, it will be finite. Thus the complete specification, which we call the contracted model, has to deal with this aspect. Some subsequent development step must therefore constrain the maximum size of the set. Let ***Ret*** be the system model in which the constraint has been applied and ***Abs*** be the model just before. We shall describe each of these models by a state space and a set of operation names. Individual operations will be defined by a transition or step relation which will have its own input and output spaces.

For ***Abs*** let the state space be U with typical element u. Ops_A will be the set of operation names with Op_A 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_A transition will be depicted by u -(i, Op_A, o)-› u', where u and u' are the before- and after-states, and i and o are the input and output. The set of such transitions forms the step relation $stp_{Op_\mathrm{A}}(u, i, u', o)$. $Init_\mathrm{A}(u')$ is the initialisation operation which sets the state to an initial value u'.

Ret has a similar setup. The operation names are $Op_\mathrm{T} \in \mathsf{Ops}_\mathrm{T}$ with $\mathsf{Ops}_\mathrm{A} = \mathsf{Ops}_\mathrm{T}$, and states are $v \in \mathsf{V}$. In classical refinement the I/O signature of an operation does not change across a development step. Thus input and output entities remain the same. Transitions are therefore v -(i, Op_T, o)-› v', and these are members of the step relation $stp_{Op_\mathrm{C}}(v, i, v', o)$. The initialisation operation is $Init_\mathrm{T}(v')$.

To show ***Ret*** is a refinement of ***Abs*** we must first define the retrieve relation $G(u, v)$, which relates analogous states between the two models. Then, for ***Ret*** to be a refinement of ***Abs***, two conditions, (2.1) and (2.2), must hold.

$$Init_\mathrm{T}(v') \;\Rightarrow\; (\exists\, u' \bullet Init_\mathrm{A}(u') \wedge G(u', v')) \,. \tag{2.1}$$

$$G(u, v) \wedge stp_{Op_\mathrm{T}}(v, i, v', o) \;\Rightarrow\; (\exists\, u' \bullet stp_{Op_\mathrm{A}}(u, i, u', o) \wedge G(u', v')) \,. \tag{2.2}$$

The former, known as the initialisation PO (Init PO), requires that the two systems start from the same place. The latter, known as the operation PO (Op PO), requires that every Op_F-step has a corresponding Op_A-step. These POs arise from the basic notion of substitutivity on which refinement is based. We have given the forward simulation POs, since most applications of refinement are of this kind. We will also work in a partial correctness setting. For a discussion of both forward and backward simulation, total and partial correctness, and substitutivity see [8, 9].

To keep the example focused, we will assume that the state of each system is just sets of NATs and ignore any other components. We will also disregard all operations

other than *Add* and *Rem*. Thus $\mathsf{U} = \mathcal{P}(\mathsf{NAT})$. Furthermore, $\mathsf{I}_{Add} = \mathsf{O}_{Rem} = \mathsf{NAT}$ and $\mathsf{O}_{Add} = \mathsf{I}_{Rem} = \varnothing$. Transitions for the two operations Add_{A} and Rem_{A} have the form

$$u \text{ -}(i, Add_{\mathrm{A}})\text{-}\!\!> u \cup \{i\} \tag{2.3}$$

and

$$\{o\} \uplus u \text{ -}(Rem_{\mathrm{A}}, o)\text{-}\!\!> u \ , \tag{2.4}$$

where $\uplus$ represents disjoint union.

In ***Ret*** we want to limit the cardinality of the set. We restrict it to a maximum of 10. We can achieve this by defining the transitions for Add_{T} to be

$$v \text{ -}(i, Add_{\mathrm{T}})\text{-}\!\!> v \cup \{i\}, \quad |v| \leq 9 \tag{2.5}$$

and leave Rem_{A} unchanged, giving

$$\{o\} \uplus v \text{ -}(Rem_{\mathrm{T}}, o)\text{-}\!\!> v \ . \tag{2.6}$$

If we now define the retrieve relation $G(u, v)$ to be an identity, then the above is a refinement of ***Abs*** since, given the obvious initialisation, PO (2.2) holds. However, our specification is incomplete because we have not said what happens when the set 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

$$v \text{ -}(i, Add_{\mathrm{T}})\text{-}\!\!> v, \quad |v| = 10 \tag{2.7}$$

Unfortunately this amendment means we no longer have a refinement because an element not already in the set will still be added in ***Abs***, and so (2.2) will fail. This should be apparent seeing that beforehand $|u| = |v| = 10$, yet after, $|u| = 11$ but $|v| = 10$. Perhaps the problem could be fixed by modifying G. One possibility is

$$\begin{aligned} &G(u, v) = (u = v), \ |v| \leq 9 \\ &G(u, v) = (v \subseteq u), \ |v| = 10 \ . \end{aligned} \tag{2.8}$$

But now of course when $|u| > 10$ and $|v| = 10$, if *Rem* is used to remove an element, the retrieve relation will no longer hold for the after-states. So using refinement, we are already experiencing difficulty in capturing a reasonable development step.

Let us now consider Rem_{T}. Notice (2.6) does not specify what happens if we try to remove a number when the set is empty. The sensible thing to do here is to signal an error. Accordingly we could extend the definition with

$$\{\ \} \text{ -}(Rem_{\mathrm{T}}, \mathsf{EMPTY})\text{-}\!\!> \{\ \} \ . \tag{2.9}$$

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. So once again, refinement is obstructing the development of the specification in the way we would like.

Of course in this example we can easily overcome our difficulties by addressing the finiteness issue in ***Abs***. However, it is not hard to imagine that beyond our narrow focus, in any large project such as this, there will be many other problems which can

prevent there being a refinement between certain models in the development chain. The amount of detail that would need to be brought up to higher levels in order to get a refinement would defeat our initial intent. That is, to build the system in a gradual manner, concentrating first on the central abstract properties and introducing lower level detail at a later stage. For industrial scale specifications such an approach is clearly beneficial. Not only does it help to manage the complexity of the task, but the resulting abstract models will be both easier to approach from cold and to formally analyse. Retrenchment, introduced in the next section, aids designers by providing a more flexible mechanism, which enables them to formally describe steps in their chosen development route which cannot be described by refinement. This is particularly so for projects whose initial specifications are written using applied mathematics.

Many situations involving a finite computable subdomain of a mathematically ideal and infinite one, follow the pattern described above, and a number of approaches have been designed to address the problem. One technique is Neilson's thesis [14], 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 [15, 16], in which Owe proposes a logical approach based on a careful analysis of the effects of ill-definedness on a programming logic.

Other work which has a similar goal to our own includes Liu's evolution [12] and Smith's realisation [19]. These, like retrenchment, are a modification of refinement. A separate issue hinted at in the example is the desirability of changing I/O signatures during the course of a development step. The desire for this ranges from wanting to make the key aspects of a specification clearer with a different I/O signature, to simply wanting to incorporate change of I/O signature during refinement. The literature contains a number of works on this question, for example [9, 10, 13, 20].

3 Retrenchment

We begin by modifying the framework introduced in the previous section. For ***Ret*** each Op_{T} now has inputs $j \in \mathsf{J}_{Op_{\mathrm{T}}}$ and outputs $p \in \mathsf{P}_{Op_{\mathrm{T}}}$. Transitions are therefore v -(j, Op_{T}, p)-› v' and the step relation is $stp_{Op_{\mathrm{T}}}(v, j, v', p)$. Unlike in standard refinement, in retrenchment ***Ret*** is permitted to have additional operations, so $\mathsf{Ops}_{\mathrm{A}} \subseteq \mathsf{Ops}_{\mathrm{T}}$. As before, corresponding ***Abs*** and ***Ret*** states will be related by the retrieve relation, which we now write as $H(u, v)$. In addition, for each $Op_{\mathrm{A}} \in \mathsf{Ops}_{\mathrm{A}}$, there is a within relation $Q_{Op}(i, j, u, v)$ and a concedes relation $D_{Op}(u', v', o, p; i, j, u, v)$.

The retrieve, within and concedes relations combine to give the POs for retrenchment. The Init PO is the same as for refinement, and in the present context is

$$Init_{\mathrm{T}}(v') \;\Rightarrow\; (\exists\, u' \bullet Init_{\mathrm{A}}(u') \wedge H(u', v')) \,. \tag{3.1}$$

Each pair of matching operations must also satisfy the Op PO,

$$\begin{aligned} &H(u, v) \wedge Q_{Op}(i, j, u, v) \wedge stp_{Op_{\mathrm{T}}}(v, j, v', p) \;\Rightarrow \\ &\quad (\exists\, u', o \bullet stp_{Op_{\mathrm{A}}}(u, i, u', o) \wedge (H(u', v') \vee D_{Op}(u', v', o, p; i, j, u, v))) \,. \end{aligned} \tag{3.2}$$

The within relation is concerned with both before-states and input values and so enables a change in input signature as well as the movement of components between input and state between levels. What is more, appearing as a conjunct in the anteced-

ent, the within relation strengthens the retrieve relation in before-states and so can be used to restrict the relationship between adjoining levels of abstraction. The concedes relation is mainly concerned with after-states and output values, but may involve before-states and inputs for greater flexibility[1]; the syntax of D_{Op} is intended to stress this. Thus, in particular, the concedes relation allows change of output signature and movement of output and state information. But most importantly, it admits weakening of the retrieve relation in after-states by being a disjunct to it, and it is especially this aspect that allows the expression of non-refinement like behaviour. The added flexibility provided by the within and concedes relations thus introduces the possibility of relating models which cannot be brought together by refinement.

We now reconsider our running example as a retrenchment. Predictably $\mathsf{Ops}_\mathrm{A} = \mathsf{Ops}_\mathrm{T} = \{Init, Add, Rem\}$ and $\mathsf{U} = \mathsf{V} = \mathbb{P}(\mathsf{NAT})$. $\mathsf{I}_{Add_\mathrm{A}} = \mathsf{J}_{Add_\mathrm{T}} = \mathsf{NAT}$ and $\mathsf{O}_{Add_\mathrm{A}} = \mathsf{P}_{Add_\mathrm{T}} = \varnothing$. $\mathsf{I}_{Rem_\mathrm{T}} = \mathsf{J}_{Rem_\mathrm{T}} = \varnothing$, $\mathsf{O}_{Rem_\mathrm{A}} = \mathsf{NAT}$ and $\mathsf{P}_{Rem_\mathrm{T}} = \mathsf{NAT} \cup \{\mathsf{EMPTY}\}$. Add_T is given by (2.5) and (2.7), so the operation does a *skip* when the set is full. Rem_T is given by (2.6) and (2.9), and so an error is produced when the set is empty.

To establish a retrenchment between *Abs* and *Ret* we need to specify the retrieve relation and also the within and concedes relations for *Add* and *Rem*. Let

$$
\begin{aligned}
&H(u, v) \;=\; (u = v, |u| \leq 10)\,,\\
&Q_{Add}(i, j, u, v) \;=\; (i = j)\,,\\
&D_{Add}(u', v'; i, j, u, v) \;=\; (|u| = 10 \wedge i \notin u \wedge u' = u \cup \{i\} \wedge v' = u)\,,\\
&Q_{Rem}(u, v) \;=\; (|v| \neq 0)\,,\\
&D_{Rem}(w', t'\, o, p; w, t) \;=\; \mathit{false}\,.
\end{aligned}
\tag{3.3}
$$

Notice how concession D_{Add} allows *Ret* to exhibit different behaviour when the set is full. *Abs* adds another element but *Ret* performs a *skip*. So although the retrieve relation does not hold for the after-sates, D_{Add} does, and thus PO (3.2) is satisfied. For *Rem* the within clause Q_{Rem} excludes the ill-behaved cases from consideration. When $|v| = 0$, Q_{Rem} is false and so (3.2) holds trivially.

4 I/O-Filtered Refinements

To facilitate the integration of refinement with retrenchment, in this section we introduce I/O-filtered refinements whose POs correspond in structure more closely to those in retrenchment. We assume we are working with *Abs* and a system *Ref*.

The notation set up already for *Abs* will do. For *Ref*, with operation names Ops_F, where $\mathsf{Ops}_\mathrm{F} = \mathsf{Ops}_\mathrm{A}$, the state space will be W with elements w. The input and output spaces for Op_F are given by $k \in \mathsf{K}_{Op_\mathrm{F}}$ and $q \in \mathsf{Q}_{Op_\mathrm{F}}$. The state spaces are related by a retrieve relation $K(u, w)$. For an I/O-filtered refinement, we further have for each $Op \in \mathsf{Ops}_\mathrm{F}$, a within relation $R_{Op}(i, k)$, and a nevertheless relation $V_{Op}(o, q)$. We also have two POs. First the Init PO,

$$
Init_\mathrm{F}(w') \;\Rightarrow\; (\exists\, u' \bullet Init_\mathrm{A}(u') \wedge K(u', w'))\,. \tag{4.1}
$$

1. Relations between outputs and states ought to hold universally, and not just when H fails. Sharp retrenchment and output retrenchment address this, establishing in the consequent of the PO, in the former case $((H \vee D) \wedge V)$, and in the latter case $((H \wedge O) \vee D)$, where V, the nevertheless relation, or O the output relation, allow extra conjunctive properties to be expressed. See [5] and [7].

This is the same as (3.1). There is also the Op PO, which for a typical *Op* says

$$K(u, w) \wedge R_{Op}(i, k) \wedge stp_{Op_{\mathrm{F}}}(w, k, w', q) \Rightarrow$$
$$(\exists\, u', o \bullet stp_{Op_{\mathrm{A}}}(u, i, u', o) \wedge K(u', w') \wedge V_{Op}(o, q)) \,. \tag{4.2}$$

Note that V_{Op} enters the consequent of (4.2) conjunctively (which is why we call it a refinement), in contrast to the retrenchment case.

We return to our running example and introduce a simple refinement. We model the states of the concrete level ***Ref*** by injective sequences of NATs, therefore $\mathsf{W} = iseq(\mathsf{NAT})$. $\mathsf{I}_{Add_{\mathrm{A}}} = \mathsf{K}_{Add_{\mathrm{F}}}$, $\mathsf{O}_{Add_{\mathrm{A}}} = \mathsf{Q}_{Add_{\mathrm{F}}}$, $\mathsf{I}_{Rem_{\mathrm{A}}} = \mathsf{K}_{Rem_{\mathrm{F}}}$ and $\mathsf{O}_{Rem_{\mathrm{A}}} = \mathsf{Q}_{Rem_{\mathrm{F}}}$.

The retrieve relation is $K(u, w) = (u = rng(w))$ and thus associates each set with all its possible serialisations. Hence, for example, $u = \{1, 2, 3\}$ corresponds to $w = \langle 1, 2, 3\rangle$, or $w = \langle 2, 1, 3\rangle$, or to any of four other possibilities. The within and nevertheless relations are all the obvious identities. It remains to define the operations for ***Ref***. These exhibit the same behaviour as their ***Abs*** counterparts. Add_{F} is

$$w \text{ -}(k, Add_{\mathrm{F}})\text{-}\rightarrow w \;\hat{}\; \langle k \rangle, \quad k \notin ran(w)$$
$$w \text{ -}(k, Add_{\mathrm{F}})\text{-}\rightarrow w, \quad k \in ran(w), \tag{4.3}$$

and Rem_{F} is given by

$$\langle q \rangle \;\hat{}\; w \text{ -}(Rem_{\mathrm{F}}, q)\text{-}\rightarrow w \;. \tag{4.4}$$

With the self-evident initialisation, (4.2) holds for this arrangement.

5 The Reconciliation

In this section we take the retrenchment from ***Abs*** to ***Ret*** and the refinement from ***Abs*** to ***Ref***, and build a universal system ***Univ***, that both refines ***Ret*** and retrenches ***Ref***. See Fig. 1. Due to space limitations we present all material in this section without proof [2]. We begin by defining some equivalence relations. First let

$$HD(u, v) = H(u, v) \vee \bigvee_{Op} (\exists\, \underline{\underline{o}}, \underline{\underline{p}}, \underline{i}, \underline{j}, \underline{\underline{u}}, \underline{\underline{v}} \bullet D_{Op}(u, v, \underline{\underline{o}}, \underline{\underline{p}}; \underline{i}, \underline{j}, \underline{\underline{u}}, \underline{\underline{v}})) \tag{5.1}$$

$$QI_{Op}(i, j) = (\exists\, \underline{\underline{u}}, \underline{\underline{v}} \bullet Q_{Op}(i, j, \underline{\underline{u}}, \underline{\underline{v}})) \tag{5.2}$$

$$DO_{Op}(o, p) = (\exists\, \underline{\underline{u}}', \underline{\underline{v}}', \bullet D_{Op}(\underline{\underline{u}}', \underline{\underline{v}}', o, p; \underline{i}, \underline{j}, \underline{\underline{u}}, \underline{\underline{v}})) \tag{5.3}$$

Given these, we now introduce the following equivalence relations.

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

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

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

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

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

2. Proofs available online in `rec.ret.ref.1.prf.ps.gz` and `rec.ret.ref.1.prf.pdf` at `http://www.cs.man.ac.uk/~banach/some.pubs/`

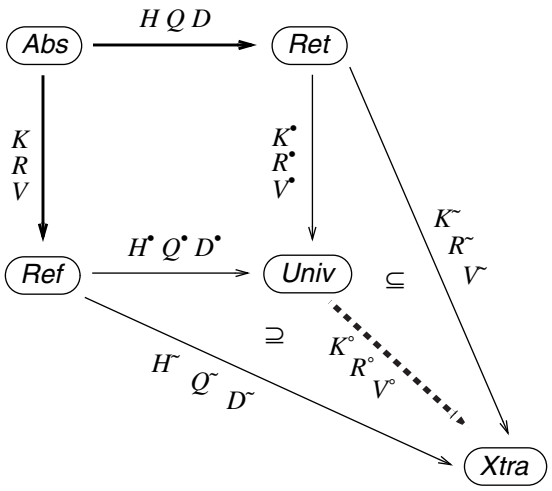

Fig. 1 All arrows labelled H, Q, D are retrenchments, all arrows labelled K, R, V are refinements.

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

The operation names set of *Univ* is $\mathsf{Ops}_{\mathrm{U}}$ with elements Op_{U}. The state space is T with elements t, inputs are $h \in \mathsf{H}$, outputs $s \in \mathsf{S}$. These are all constructed from the systems *Ret* and *Ref* as follows. Firstly $\mathsf{Ops}_{\mathrm{U}} = \mathsf{Ops}_{\mathrm{T}} = \mathsf{Ops}_{\mathrm{A}} \cup (\mathsf{Ops}_{\mathrm{U}} - \mathsf{Ops}_{\mathrm{A}})$. So each Op_{U} is either an Op_{A} or an $Op_{\mathrm{U}} \in (\mathsf{Ops}_{\mathrm{U}} - \mathsf{Ops}_{\mathrm{A}})$. Next $\mathsf{T} = \mathsf{V}/\sim_{\mathrm{V}} \times \mathsf{W}/\sim_{\mathrm{W}}$. For $Op_{\mathrm{U}} \in \mathsf{Ops}_{\mathrm{A}}$ the input and output spaces are $\mathsf{H}_{Op} = \mathsf{J}_{Op}/\sim_{\mathrm{J}Op} \times \mathsf{K}_{Op}/\sim_{\mathrm{K}Op}$ and $\mathsf{S}_{Op} = \mathsf{P}_{Op}/\sim_{\mathrm{P}Op} \times \mathsf{Q}_{Op}/\sim_{\mathrm{Q}Op}$, while for $Op_{\mathrm{U}} \notin \mathsf{Ops}_{\mathrm{A}}$, $\mathsf{H}_{Op} = \mathsf{J}_{Op}$ and $\mathsf{S}_{Op} = \mathsf{P}_{Op}$.

Now for some more definitions.

$$KH(\underline{w}, [v]) \;=\; (\forall\, u \bullet K(u, \underline{w}) \Rightarrow (\exists\, \underline{v} \bullet \underline{v} \in [v] \wedge H(u, \underline{v}))) \tag{5.10}$$

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

$$\begin{aligned} KD_{Op}(\underline{w}, [v]) \;=\; & (\forall\, u \bullet K(u, \underline{w}) \Rightarrow \\ & (\exists\, \underline{v} \bullet \underline{v} \in [v] \wedge (\exists\, \underline{\underline{o}}, \underline{\underline{p}}, \underline{\underline{i}}, \underline{\underline{j}}, \underline{\underline{u}}, \underline{\underline{v}} \bullet D_{Op}(u, \underline{v}, \underline{\underline{o}}, \underline{\underline{p}}; \underline{\underline{i}}, \underline{\underline{j}}, \underline{\underline{u}}, \underline{\underline{v}})))) \end{aligned} \tag{5.12}$$

$$\begin{aligned} & DK_{Op}([v], [w]) \;= \\ & \quad (\forall\, u, \underline{v} \bullet \underline{v} \in [v] \wedge (\exists\, \underline{\underline{o}}, \underline{\underline{p}}, \underline{\underline{i}}, \underline{\underline{j}}, \underline{\underline{u}}, \underline{\underline{v}} \bullet D_{Op}(u, \underline{v}, \underline{\underline{o}}, \underline{\underline{p}}; \underline{\underline{i}}, \underline{\underline{j}}, \underline{\underline{u}}, \underline{\underline{v}})) \Rightarrow \\ & \qquad (\exists\, \underline{w} \bullet \underline{w} \in [w] \wedge K(u, \underline{w}) \wedge KD_{Op}(\underline{w}, [v]))) \end{aligned} \tag{5.13}$$

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

$$QR_{Op}([j], [k]) =$$
$$(\forall\ i, u, \underline{j}, \underline{v}, v, w \bullet \underline{j} \in [j] \wedge H(u, \underline{v}) \wedge Q_{Op}(i, \underline{j}, u, \underline{v}) \wedge K^{\bullet}(\underline{v}, ([v], [w])) \Rightarrow$$
$$(\exists\ \underline{k}, \underline{w} \bullet \underline{k} \in [k] \wedge \underline{w} \in [w] \wedge R_{Op}(i, \underline{k}) \wedge K(u, \underline{w}) \wedge$$
$$RQ_{Op}(\underline{k}, \underline{w}, [j], [v])) \tag{5.15}$$

$$VD_{Op}(\underline{w}', \underline{q}, \underline{k}, \underline{w}, [v'], [p], [j], [v]) =$$
$$(\forall\ u', o, i, u \bullet K(u', \underline{w}') \wedge V_{Op}(o, \underline{q}) \wedge R_{Op}(i, \underline{k}) \wedge K(u, \underline{w}) \Rightarrow$$
$$(\exists\ \underline{v}', \underline{p}, \underline{j}, \underline{v} \bullet \underline{v}' \in [v'] \wedge \underline{p} \in [p] \wedge \underline{j} \in [j] \wedge \underline{v} \in [v] \wedge$$
$$H(u, \underline{v}) \wedge Q_{Op}(i, \underline{j}, u, \underline{v}) \wedge D_{Op}(u', \underline{v}', o, \underline{p}; i, \underline{j}, u, \underline{v}))) \tag{5.16}$$

$$DV_{Op}([p], [q]) =$$
$$(\forall\ u', o, i, u, \underline{v}', \underline{p}, \underline{j}, \underline{v}, v', w', j, k, v, w \bullet$$
$$\underline{p} \in [p] \wedge H(u, \underline{v}) \wedge Q_{Op}(i, \underline{j}, u, \underline{v}) \wedge D_{Op}(u', \underline{v}', o, \underline{p}; i, \underline{j}, u, \underline{v}) \wedge$$
$$K^{\bullet}(\underline{v}', ([v'], [w'])) \wedge R^{\bullet}{}_{Op}(\underline{j}, ([j], [k])) \wedge K^{\bullet}(\underline{v}, ([v], [w])) \Rightarrow$$
$$(\exists\ \underline{w}', \underline{q}, \underline{k}, \underline{w} \bullet \underline{w}' \in [w'] \wedge \underline{q} \in [q] \wedge \underline{k} \in [k] \wedge \underline{w} \in [w] \wedge$$
$$K(u', \underline{w}') \wedge V_{Op}(o, \underline{q}) \wedge R_{Op}(i, \underline{k}) \wedge K(u, \underline{w}) \wedge$$
$$VD_{Op}(\underline{w}', \underline{q}, \underline{k}, \underline{w}, [v'], [p], [j], [v]))) \tag{5.17}$$

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

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

$$H^{\bullet}(\underline{w}, ([v], [w])) \ = \ \underline{w} \in [w] \wedge (\exists\ u \bullet K(u, \underline{w})) \wedge KH(\underline{w}, [v]) \wedge$$
$$HK([v], [w]) \wedge \bigwedge_{Op} DK_{Op}([v], [w]) \tag{5.19}$$

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

$$Q^{\bullet}{}_{Op}(\underline{k}, ([j], [k]), \underline{w}, ([v], [w])) \ = \ \underline{k} \in [k] \wedge \underline{w} \in [w] \wedge (\exists\ i, u \bullet R_{Op}(i, \underline{k}) \wedge$$
$$K(u, \underline{w})) \wedge RQ_{Op}(\underline{k}, \underline{w}, [j], [v]) \wedge QR_{Op}([j], [k]) \tag{5.21}$$

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

$$D^{\bullet}{}_{Op}(\underline{w}', ([v'], [w']), \underline{q}, ([p], [q]); \underline{k}, ([j], [k]), \underline{w}, ([v], [w])) \ =$$
$$\underline{w}' \in [w'] \wedge \underline{q} \in [q] \wedge \underline{k} \in [k] \wedge \underline{w} \in [w] \wedge$$
$$(\exists\ u', o, i, u \bullet K(u', \underline{w}') \wedge V_{Op}(o, \underline{q}) \wedge R_{Op}(i, \underline{k}) \wedge K(u, \underline{w})) \wedge$$
$$VD_{Op}(\underline{w}', \underline{q}, \underline{k}, \underline{w}, [v'], [p], [j], [v]) \wedge DV_{Op}([p], [q]) \wedge$$
$$HK([v'], [w']) \wedge \bigwedge_{Op} DK_{Op}([v'], [w']) \tag{5.23}$$

In the above, *Op* ranges over $\mathsf{Ops}_{\mathrm{A}}$. For operations not in $\mathsf{Ops}_{\mathrm{A}}$, we define $R^{\bullet}{}_{Op}$ and $V^{\bullet}{}_{Op}$ as identities on inputs and outputs respectively.

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

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

Second $(H(u, \underline{v}) \wedge Q_{Op}(i, \underline{j}, u, \underline{v}));(K^\bullet(\underline{v}, ([v], [w])) \wedge R^\bullet{}_{Op}(\underline{j}, ([j], [k]))) = (K(u, \underline{w}) \wedge R_{Op}(i, \underline{k}));(H^\bullet(\underline{w}, ([v], [w])) \wedge Q^\bullet{}_{Op}(\underline{k}, ([j], [k]), \underline{w}, ([v], [w])))$, or more briefly,

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

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

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

Now we can give the transitions of *Univ*. The operation names set Ops_U decomposes as $\mathsf{Ops}_\mathrm{U} = \mathsf{Ops}_\mathrm{A} \cup (\mathsf{Ops}_\mathrm{U} - \mathsf{Ops}_\mathrm{A})$. Thus there are two possibilities. For $Op_\mathrm{U} \in (\mathsf{Ops}_\mathrm{U} - \mathsf{Ops}_\mathrm{A})$ we have a transition t -(h, Op_U, s)-› t' or, more explicitly,

$$([v], [w])\ \text{-}(\underline{j}, Op_\mathrm{U}, \underline{p})\text{-›}\ ([v'], [w']) \,, \tag{5.27}$$

iff $[v]$, $[w]$, $\underline{j}$, $\underline{p}$, $[v']$, $[w']$ satisfy

$$\begin{aligned}&\big(\ \forall\ \underline{v} \bullet \underline{v} \in [v] \Rightarrow \\ &\quad (\ \exists\ \underline{v}' \bullet stp_{Op_\mathrm{T}}(\underline{v}, \underline{j}, \underline{v}', \underline{p}) \wedge K^\bullet(\underline{v}', ([v'], [w']))\ )\ \big) \,.\end{aligned} \tag{5.28}$$

For $Op_\mathrm{A} \in \mathsf{Ops}_\mathrm{A}$, we have t -(h, Op_A, s)-› t' or

$$([v], [w])\ \text{-}(([j], [k]), Op_\mathrm{A}, ([p], [q]))\text{-›}\ ([v'], [w']) \tag{5.29}$$

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

$$\begin{aligned}&\big(\forall\ \underline{v}, \underline{j} \bullet \underline{v} \in [v] \wedge \underline{j} \in [j] \Rightarrow (\ \exists\ \underline{v}', \underline{p} \bullet stp_{Op_\mathrm{T}}(\underline{v}, \underline{j}, \underline{v}', \underline{p}) \wedge \\ &\qquad K^\bullet(\underline{v}', ([v'], [w'])) \wedge V^\bullet{}_{Op}(\underline{p}, ([p], [q]))\ )\big) \qquad \text{(a)} \\ &\wedge \\ &\big(\forall\ \underline{w}, \underline{k} \bullet H^\bullet(\underline{w}, ([v], [w])) \wedge Q^\bullet{}_{Op}(\underline{k}, ([j], [k]), \underline{w}, ([v], [w])) \Rightarrow \\ &\qquad (\ \exists\ \underline{w}', \underline{q} \bullet stp_{Op_\mathrm{F}}(\underline{w}, \underline{k}, \underline{w}', \underline{q}) \wedge (H^\bullet(\underline{w}', ([v'], [w'])) \vee \\ &\qquad\quad D^\bullet{}_{Op}(\underline{w}', ([v'], [w']), \underline{q}, ([p], [q]); \underline{k}, ([j], [k]), \underline{w}, ([v], [w])))\ )\big) \,. \qquad \text{(b)}\end{aligned} \tag{5.30}$$

The initialization predicate $Init_\mathrm{U}(t')$ sets t' to any $([v'], [w'])$ for which

$$\begin{aligned}&\big(\ \exists\ \underline{v}' \bullet Init_\mathrm{T}(\underline{v}') \wedge K^\bullet(\underline{v}', ([v'], [w']))\ \big)\ \wedge \\ &\big(\ \exists\ \underline{w}' \bullet Init_\mathrm{F}(\underline{w}') \wedge H^\bullet(\underline{w}', ([v'], [w']))\ \big)\end{aligned} \tag{5.31}$$

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

The components introduced above define an I/O-filtered refinement from *Ret* to *Univ* and a retrenchment from *Ref* to *Univ*, because they satisfy the appropriate POs, (5.32), (5.33) and (5.34), (5.35) respectively. Therefore the system *Univ* completes the square shown in Fig. 1.

$$Init_\mathrm{U}(t') \ \Rightarrow\ (\exists\ \underline{v}' \bullet Init_\mathrm{T}(\underline{v}') \wedge K^\bullet(\underline{v}', t')) \tag{5.32}$$

$$\begin{aligned}&K^\bullet(\underline{v}, t) \wedge R^\bullet{}_{Op}(\underline{j}, h) \wedge stp_{Op_\mathrm{U}}(t, h, t', s)\ \Rightarrow \\ &\quad (\exists\ \underline{v}', \underline{p} \bullet stp_{Op_\mathrm{T}}(\underline{v}, \underline{j}, \underline{v}', \underline{p}) \wedge K^\bullet(\underline{v}', t') \wedge V^\bullet{}_{Op}(\underline{p}, s))\end{aligned} \tag{5.33}$$

$$Init_{\mathrm{U}}(t') \;\Rightarrow\; (\exists\, \underline{w}' \bullet Init_{\mathrm{F}}(\underline{w}') \wedge H^{\bullet}(\underline{w}', t')) \tag{5.34}$$

$$\begin{aligned} &H^{\bullet}(\underline{w}, t) \wedge Q^{\bullet}{}_{Op}(\underline{k}, h, \underline{w}, t) \wedge stp_{Op_{\mathrm{U}}}(t, h, t', s) \;\Rightarrow \\ &\quad (\exists\, \underline{w}', \underline{q} \bullet stp_{Op_{\mathrm{F}}}(\underline{w}, \underline{k}, \underline{w}', \underline{q}) \wedge \\ &\quad (H^{\bullet}(\underline{w}', t') \vee D^{\bullet}{}_{Op}(\underline{w}', t', \underline{q}, s; \underline{k}, h, \underline{w}, t))) \end{aligned} \tag{5.35}$$

We also want the composition of the *Abs* to *Ret* retrenchment together with the *Ret* to *Univ* refinement on the one hand, and the *Abs* to *Ref* refinement together with the *Ref* to *Univ* retrenchment on the other, to be a retrenchment from *Abs* to *Univ*. This is in fact the case because for the composed retrenchment, POs (5.36) and (5.37) below hold, verifying that we have the required relationship. What is more, (5.24) to (5.26) show that both compositions give the *same* retrenchment, with retrieves, within, and concedes relations, G, P_{Op} and C_{Op} respectively.

$$Init_{\mathrm{U}}(t') \;\Rightarrow\; (\exists\, u' \bullet Init_{\mathrm{A}}(u') \wedge G(u', t')) \tag{5.36}$$

$$\begin{aligned} &G(u, t) \wedge P_{Op}(i, h, u, t) \wedge stp_{Op_{\mathrm{U}}}(t, h, t', s) \;\Rightarrow \\ &\quad (\exists\, u', o \bullet stp_{Op_{\mathrm{A}}}(u, i, u', o) \wedge (G(u', t') \vee C_{Op}(u', t', o, s; i, h, u, t))) \end{aligned} \tag{5.37}$$

Note also that *Univ* has the following properties.

$$K^{\bullet}(v', t') \wedge K^{\bullet}(\underline{v}', t') \;\Rightarrow\; v' \sim \underline{v}' \tag{U1}$$

$$(H^{\bullet}(w', t') \vee D^{\bullet}{}_{Op}(w', t', ...)) \wedge (H^{\bullet}(\underline{w}', t') \vee D^{\bullet}{}_{Op}(\underline{w}', t', ...)) \Rightarrow\; w' \sim \underline{w}' \tag{U2}$$

$$V^{\bullet}{}_{Op}(p, s) \wedge V^{\bullet}{}_{Op}(\underline{p}, s) \Rightarrow\; p \sim \underline{p} \tag{U3}$$

$$D^{\bullet}{}_{Op}(w', t', q, s; k, h, w, t) \wedge D^{\bullet}{}_{Op}(\underline{w}', t', \underline{q}, s; \underline{k}, h, \underline{w}, t) \;\Rightarrow\; q \sim \underline{q} \tag{U4}$$

$$K^{\bullet}(v', t') \wedge \underline{v}' \in [v'] \Rightarrow\; K^{\bullet}(\underline{v}', t') \tag{U5}$$

$$V^{\bullet}{}_{Op}(p, s) \wedge \underline{p} \in [p] \Rightarrow V^{\bullet}{}_{Op}(\underline{p}, s) \tag{U6}$$

$$K^{\bullet}(v', t') \;\Rightarrow\; (\, H^{\bullet}(w', t') \Leftrightarrow H^{\bullet}(w', ([v'], [w']))\,) \tag{U7}$$

$$\begin{aligned} &K^{\bullet}(v', t') \wedge V^{\bullet}{}_{Op}(p, s) \wedge R^{\bullet}{}_{Op}(j, h) \wedge K^{\bullet}(v, t) \Rightarrow \\ &\quad (\, H^{\bullet}(w, t) \wedge Q^{\bullet}{}_{Op}(k, h, w, t) \wedge D^{\bullet}{}_{Op}(w', t', q, s; k, h, w, t) \Leftrightarrow \\ &\qquad H^{\bullet}(w, ([v], [w])) \wedge Q^{\bullet}{}_{Op}(k, ([j], [k]), w, ([v], [w])) \wedge \\ &\qquad D^{\bullet}{}_{Op}(w', ([v'], [w']), q, ([p], [q]); k, ([j], [k]), w, ([v], [w]))\,) \end{aligned} \tag{U8}$$

$$K^{\bullet}(v', t') \;\Rightarrow\; (\exists\, w' \bullet K^{\bullet}(v', ([v'], [w']))) \tag{U9}$$

$$V^{\bullet}{}_{Op}(p, s^{\sim}) \;\Rightarrow\; (\exists\, q \bullet V^{\bullet}{}_{Op}(p, ([p], [q]))) \tag{U10}$$

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

Theorem 5.1 Let there be a retrenchment from *Abs* to *Ret*, and an I/O-filtered refinement from *Abs* to *Ref*, as shown in Fig. 1. Then the following hold.

(1) There is a universal system *Univ* such that there is a retrenchment from *Ref* to *Univ* and an I/O-filtered refinement from *Ret* to *Univ* whose compositions with the original refinement and retrenchment respectively are equal as retrenchments from *Abs* to *Univ*, and which satisfies (U1) to (U10) above.

(2) Whenever there is a system *Xtra* and a retrenchment from *Ref* to *Xtra* and an I/O-filtered refinement from *Ret* to *Xtra* whose compositions with the original refinement and retrenchment respectively are equal as retrenchments from *Abs* to

Xtra, and which satisfies (X1) to (X10) below, then there is an I/O-filtered refinement from *Univ* to *Xtra* such that $H^\bullet;K^\circ \Rightarrow H^\sim$, $(H^\bullet \wedge Q^\bullet);(K^\circ \wedge R^\circ) \Rightarrow (H^\sim \wedge Q^\sim)$, $(H^\bullet \wedge Q^\bullet \wedge D^\bullet);(K^{\circ\prime} \wedge V^\circ \wedge R^\circ \wedge K^\circ) \Rightarrow (H^\sim \wedge Q^\sim \wedge D^\sim)$, and such that $K^\bullet;K^\circ \Rightarrow K^\sim$, $R^\bullet;R^\circ \Rightarrow R^\sim$, $V^\bullet;V^\circ \Rightarrow V^\sim$.

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

We want *Univ* to be the most abstract possible completion of the square. Thus part (2) requires that for any other system *Xtra* which also completes the square (and belongs to the same class as *Univ* (see below)), there must be a I/O-filtered refinement from *Univ* to *Xtra*. It is this universal property which ensures *Univ* is the most abstract reconciliation possible. *Univ* is also the "best" solution, since it, and any other system *Univ** which can take its place, are mutually interrefinable, and thus equivalent in the context of this setting. This is part (3) of the theorem.

So suppose we have a system *Xtra* and an I/O-filtered refinement from *Ret* to *Xtra* with retrieve relation $K^\sim$, within relations $R^\sim{}_{Op}$, and nevertheless relations $V^\sim{}_{Op}$; and a retrenchment from *Ref* to *Xtra* with retrieve relation $H^\sim$, within relations $Q^\sim{}_{Op}$, and concedes relations $D^\sim{}_{Op}$. Let the state, input and output spaces of *Xtra* be given by $t^\sim \in \mathsf{T}^\sim$, $h^\sim \in \mathsf{H}^\sim{}_{Op}$, $s^\sim \in \mathsf{S}^\sim{}_{Op}$; and let the initialisation and step predicates of *Xtra* be $Init_{\mathrm{X}}$ and $stp_{Op_{\mathrm{X}}}$. Suppose also that (X1) to (X10) below hold.

$$K^\sim(v', t^{\sim\prime}) \wedge K^\sim(\underline{v}', t^{\sim\prime}) \;\Rightarrow\; v' \sim \underline{v}' \tag{X1}$$

$$(H^\sim(w', t^{\sim\prime}) \vee D^\sim{}_{Op}(w', t^{\sim\prime}, ...)) \wedge (H^\sim(\underline{w}', t^{\sim\prime}) \vee D^\sim{}_{Op}(\underline{w}', t^{\sim\prime}, ...)) \Rightarrow\; w' \sim \underline{w}' \tag{X2}$$

$$V^\sim{}_{Op}(p, s^\sim) \wedge V^\sim{}_{Op}(\underline{p}, s^\sim) \Rightarrow\; p \sim \underline{p} \tag{X3}$$

$$D^\sim{}_{Op}(w', t^{\sim\prime}, q, s^\sim; k, h^\sim, w, t^\sim) \wedge D^\sim{}_{Op}(\underline{w}', t^{\sim\prime}, \underline{q}, s^\sim; \underline{k}, h^\sim, \underline{w}, t^\sim) \Rightarrow\; q \sim \underline{q} \tag{X4}$$

$$K^\sim(v', t^{\sim\prime}) \wedge \underline{v}' \in [v'] \Rightarrow\; K^\sim(\underline{v}', t^{\sim\prime}) \tag{X5}$$

$$V^\sim{}_{Op}(p, s^\sim) \wedge \underline{p} \in [p] \Rightarrow V^\sim{}_{Op}(\underline{p}, s^\sim) \tag{X6}$$

$$K^\sim(v', t^{\sim\prime}) \;\Rightarrow\; (\, H^\sim(w', t^{\sim\prime}) \Leftrightarrow H^\bullet(w', ([v'], [w'])) \,) \tag{X7}$$

$$\begin{aligned} &K^\sim(v', t^{\sim\prime}) \wedge V^\sim{}_{Op}(p, s^\sim) \wedge R^\sim{}_{Op}(j, h^\sim) \wedge K^\sim(v, t^\sim) \Rightarrow \\ &\quad (\, H^\sim(w, t^\sim) \wedge Q^\sim{}_{Op}(k, h^\sim, w, t^\sim) \wedge D^\sim{}_{Op}(w', t^{\sim\prime}, q, s^\sim; k, h^\sim, w, t^\sim) \Leftrightarrow \\ &\qquad H^\bullet(w, ([v], [w])) \wedge Q^\bullet{}_{Op}(k, ([j], [k]), w, ([v], [w])) \wedge \\ &\qquad D^\bullet{}_{Op}(w', ([v'], [w']), q, ([p], [q]); k, ([j], [k]), w, ([v], [w])) \,) \end{aligned} \tag{X8}$$

$$K^\sim(v', t^{\sim\prime}) \;\Rightarrow\; (\exists\, w' \bullet K^\bullet(v', ([v'], [w']))) \tag{X9}$$

$$V^\sim{}_{Op}(p, s^\sim) \;\Rightarrow\; (\exists\, q \bullet V^\bullet{}_{Op}(p, ([p], [q]))) \tag{X10}$$

Notice properties (X1) to (X10) of *Xtra* correspond to (U1) to (U10) of *Univ*. Hence *Univ* and *Xtra* belong to the same class of systems that complete the square.

We now give the relations K°, $R^\circ{}_{Op}$ and $V^\circ{}_{Op}$. These satisfy the inclusions in part (2).

$$\begin{aligned} &K^\circ(t, t^\sim) \;=\; K^\circ(([v], [w]), t^\sim) \;= \\ &\quad (\, \forall\, \underline{v} \bullet \underline{v} \in [v] \Rightarrow K^\sim(\underline{v}, t^\sim) \,) \wedge (\, \forall\, \underline{w} \bullet H^\bullet(\underline{w}, ([v], [w])) \Rightarrow H^\sim(\underline{w}, t^\sim) \,) \,. \end{aligned} \tag{5.38}$$

For $Op \in \mathsf{Ops}_{\mathrm{A}}$ we have

$$\begin{aligned} &R^\circ{}_{Op}(h, h^\sim) \;=\; R^\circ{}_{Op}(([j], [k]), h^\sim) \;= \\ &\quad (\, \forall\, \underline{j} \bullet \underline{j} \in [j] \Rightarrow R^\sim{}_{Op}(\underline{j}, h^\sim) \,) \wedge \end{aligned}$$

$$(\forall \underline{k}, \underline{w}, t, t\tilde{} \bullet H^{\bullet}(\underline{w}, t) \wedge Q^{\bullet}{}_{Op}(\underline{k}, ([j], [k]), \underline{w}, t) \wedge K^{\circ}(t, t\tilde{}) \Rightarrow H\tilde{}(\underline{w}, t\tilde{}) \wedge Q\tilde{}{}_{Op}(\underline{k}, h\tilde{}, \underline{w}, t\tilde{})) , \tag{5.39}$$

$$\begin{aligned} &V^{\circ}{}_{Op}(s, s\tilde{}) = V^{\circ}{}_{Op}(([p], [q]), s\tilde{}) = \\ &\quad (\forall \underline{p} \bullet \underline{p} \in [p] \Rightarrow V\tilde{}{}_{Op}(\underline{p}, s\tilde{})) \wedge \\ &\quad (\forall \underline{w}', \underline{q}, \underline{k}, \underline{w}, t', t\tilde{}', h, h\tilde{}, t, t\tilde{} \bullet \\ &\qquad H^{\bullet}(\underline{w}, t) \wedge Q^{\bullet}{}_{Op}(\underline{k}, h, \underline{w}, t) \wedge D^{\bullet}{}_{Op}(\underline{w}', t', \underline{q}, ([p], [q]); \underline{k}, h, \underline{w}, t) \wedge \\ &\qquad K^{\circ}(t', t\tilde{}') \wedge R^{\circ}{}_{Op}(h, h\tilde{}) \wedge K^{\circ}(t, t\tilde{}) \Rightarrow \\ &\qquad\quad H\tilde{}(\underline{w}, t\tilde{}) \wedge Q\tilde{}{}_{Op}(\underline{k}, h\tilde{}, \underline{w}, t\tilde{}) \wedge D\tilde{}{}_{Op}(\underline{w}', t\tilde{}', \underline{q}, s\tilde{}; \underline{k}, h\tilde{}, \underline{w}, t\tilde{})) . \end{aligned} \tag{5.40}$$

For $Op \notin \mathsf{Ops}_{\mathrm{A}}$

$$R^{\circ}{}_{Op}(h, h\tilde{}) = (h = j \wedge R\tilde{}{}_{Op}(j, h\tilde{})) , \tag{5.41}$$

$$V^{\circ}{}_{Op}(s, s\tilde{}) = (s = p \wedge V\tilde{}{}_{Op}(p, s\tilde{})) . \tag{5.42}$$

Furthermore, K°, $R^{\circ}{}_{Op}$ and $V^{\circ}{}_{Op}$ are the component relations of the desired refinement from *Univ* to *Xtra* as they satisfy the necessary POs, (5.43) and (5.44) below.

$$Init_{\mathrm{X}}(t\tilde{}') \Rightarrow (\exists t' \bullet Init_{\mathrm{U}}(t') \wedge K^{\circ}(t', t\tilde{}')) . \tag{5.43}$$

$$\begin{aligned} &K^{\circ}(t, t\tilde{}) \wedge R^{\circ}{}_{Op}(h, h\tilde{}) \wedge stp_{Op_{\mathrm{X}}}(t\tilde{}, h\tilde{}, t\tilde{}', s\tilde{}) \Rightarrow \\ &\quad (\exists t', s \bullet stp_{Op_{\mathrm{U}}}(t, h, t', s) \wedge K^{\circ}(t', t\tilde{}') \wedge V^{\circ}{}_{Op}(s, s\tilde{})) \end{aligned} \tag{5.44}$$

This completes the details of the reconciliation.

6 A Closer Look at *Univ*

We return to our example and use it to explore the structure of *Univ*. The states t are pairs of equivalence classes $([v], [w])$ where $[v]$ contains sets of NATs and $[w]$ injective sequences of NATs. Thus $t = ([\{8, 111\}], [<7, 8961, 13>])$ is a valid state.

The transitions of *Univ* fall into two groups. To see this notice it is possible for $stp_{Op_{\mathrm{U}}}$ to be true with (5.30b) holding trivially. We refer to such transitions as junk. Let us concentrate on the other group, the non-junk transitions.

For Add_{U} an example of a non-boundary step is

$$([\{1, 2\}], [<1, 2>]) \text{ -}(([3], [3]), Add_{\mathrm{U}})\text{-}\rightarrow ([\{1, 2, 3\}], [<1, 2, 3>]) . \tag{6.1}$$

By (5.4) and (5.5), $[\{1, 2\}] = \{\{1, 2\}\}$, $[\{1, 2, 3\}] = \{\{1, 2, 3\}\}$, $[<1, 2>] = \{<1, 2>, <2, 1>\}$ and $[<1, 2, 3>]$ is all the permutations of $<1, 2, 3>$. So in *Univ* the equivalence classes are bunching together all *Ret* and *Ref* steps that correspond. Therefore, not surprisingly, $\{1, 2\}$ -$(3, Add_{\mathrm{T}})$-› $\{1, 2, 3\}$ matches both $<1, 2>$ -$(3, Add_{\mathrm{F}})$-› $<1, 2, 3>$ and $<2, 1>$ -$(3, Add_{\mathrm{F}})$-› $<2, 1, 3>$. Moving on, a typical boundary step is

$$([\{1..10\}], [<1..10>]) \text{ -}(([11], [11]), Add_{\mathrm{U}})\text{-}\rightarrow ([\{1..10\}], [<1..11>]) \tag{6.2}$$

where $[\{1..10\}] = \{\{1..10\}\}$, and $[<1..10>] = [<1..11>]$ and is the set of all serialisations of both $\{1..10\}$ and $\{1..11\}$. The latter follows from (5.5) because in addition to H, D_{Add} is now also involved, linking $u' = \{1..11\}$ and $v' = \{1..10\}$ in the retrench-

ment from *Abs* to *Ret*. Hence, {1..10} -(11, Add_{T})-› {1..10} matches <1..10> -(11, Add_{F})-› <1..11> and <10..1> -(11, Add_{F})-› <10..1, 11> among others. Also, observe that by (2.5) and (2.7), $stp_{Add_{\mathrm{T}}}$ is empty for steps with before-state of cardinality 11 or more. Hence, because (5.30a) is never trivially true, there can be no *Univ* steps for such situations. So what we actually have in *Univ* is a system with the limitations of *Ret* but with data at the level of *Ref*.

Let us return to the software engineering benefit of the reconciliation mentioned in the introduction and see how this works for (6.2). Let *Abs* be the original specification, *Ref* the software developed from it and *Ret* the changed specification. Let *New* be the altered software, which satisfies *Ret*. Given the interpretation we have for *Univ*, it is the step {1..10} -(11, Add_{T})-› {1..10} we need to rewrite in terms of the data of *Ref*. (6.2) associates [{1..10}] with [<1..11>]. Hence, since <1..10> and <10..1> (to name just two) in [<1..11>], identify with {1..10} of the Add_{T}-step, possible alternatives are <1..10> -(11, Add_{N})-› <1..10> and <10..1> -(11, Add_{N})-› <10..1> — just as we would expect.

To finish, we look at Rem_{U}. An example of a non-boundary step is

$$([\{1, 2, 3\}], [<1, 2, 3>]) \text{-}(Rem_{\mathrm{U}}, ([1], [1]))\text{-›} ([\{2, 3\}], [<2, 3>]) \ . \tag{6.3}$$

However, this time there are no non-junk steps for the boundary case: removal of an element when none are left. Although we have values that satisfy (5.30a), there are no values for which (5.30b) holds non-trivially. This is consistent, since *Abs* has no corresponding step in this case either, and so the boundary *Univ*-step we seek falls outside the scope of the retrenchment from *Abs* to *Univ*.

7 Conclusion

In the preceding sections we showed how to take a retrenchment and a refinement of the same system, and build a system which reconciles the two development steps. The construction we gave was universal within a class of similar reconciliations. Our main objective was to bring together retrenchments and refinements of as general a kind as possible, rather than look for a particularly elegant but restricted result.

The size of the example was dictated more by space considerations and a wish not to obscure the construction, than by its ability to convince the reader of the usefulness of retrenchment. Nevertheless the situation the example embodies typifies many of the cases in which retrenchment has its *raison d'etre*, namely where there are details of the real world application whose incorporation one wishes to postpone till a later stage of the development, but for which this would be precluded by the exigencies of the refinement POs. More persuasive problems in support of retrenchment can be found in [17], which looks at the specification of a program for dose calculation in radiotherapy, and in [18], which presents a control system case study.

The pushout-like construction of this paper is one key result in the full algebraic integration of retrenchment and refinement. Other results appear in the already cited works [1, 11], and yet others are work in progress. The complete suite will ensure retrenchment adds a truly fresh dimension to possible development routes, complementing refinement where it struggles to give a convincing account, rather than being a stand-alone technique that fragments development routes into incompatible paths. Ultimately the developer gains by having a richer integrated palette of tools with which to build specifications.

References

1. Banach R. Maximally Abstract Retrenchments. In *Proc. IEEE ICFEM-00*, 133-142, 2000.
2. 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`
3. Banach R., Poppleton M. Retrenchment and Punctured Simulation. In *Proc. IFM-99*, Taguchi, Galloway (eds.), Springer, 457-476, 1999.
4. Banach R., Poppleton M. Retrenchment, Refinement and Simulation. In *Proc. ZB-00*, Bowen, Dunne, Galloway, King (eds.), LNCS **1878**, 304-323, Springer (2000).
5. Banach R., Poppleton M. Sharp Retrenchment, Modulated Refinement and Simulation. *Form. Asp. Comp.* **11**, 498-540, 1999.
6. 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`.
7. 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`.
8. de Roever W.-P., Engelhardt K. *Data Refinement: Model-Oriented Proof Methods and their Comparison.* C.U.P., 1998.
9. Derrick J., Boiten E. *Refinement in Z and Object-Z: Foundations and Advanced Applications.* Springer (2001).
10. Hayes I. J., Sanders J. W. Specification by Interface Separation. *Form. Asp. Comp*. **7**, 430-439, 1995.
11. Jeske C., Banach R. Minimally and Maximally Abstract Retrenchments. In *Proc. IFM-02*, LNCS **2335**, 380-399, Springer (2002).
12. Liu S. Evolution: A More Practical Approach than Refinement for Software Development. In *Proc. ICECCS-97*, 142-151, IEEE, 1997.
13. 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.
14. Neilson D. S. *From Z to C: Illustration of a Rigorous Development Method.* PhD Thesis, Oxford University Programming Research Group, Monograph PRG-101, 1990.
15. 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.
16. Owe O. Partial Logics Reconsidered: A Conservative Approach. *Form. Asp. Comp.* **3**, 1-16, 1993.
17. 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.
18. 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.
19. Smith G. Stepwise Development from Ideal Specifications. In Edwards (ed.), *Aus. Comp. Sci. Conf.* **22**(1), 227-233. IEEE Computer Society, 2000.
20. 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.