Maximally Abstract Retrenchments

R. Banach

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

Abstract

The more obvious and well known drawbacks of using refinement as the sole means of progressing from an abstract model to a concrete implementation are reviewed. Retrenchment is presented in a simple partial correctness framework as a more flexible development concept for formally capturing the early and otherwise preformal stages of development, and briefly justified.

Given a retrenchment from an abstract to a concrete model, the problem of finding a model at the level of abstraction of the abstract model, but refinable to the concrete one, is examined. A construction is given that solves the problem in a universal manner, there being a canonical factorisation of the original retrenchment, into a retrenchment to the universal system followed by an I/O-filtered refinement. The universality amounts to the observation that the retrenchment component of any similar factorisation, factors uniquely through the universal model. The construction's claim to be at the right level of abstraction is supported by an idempotence property. The consequences of including termination criteria in the formal models is briefly explored.

1. Introduction

Retrenchment was introduced in [1], and subsequently developed in [2, 3, 4], as a means of overcoming the relatively widely perceived (if the truth be admitted) drawbacks of using refinement alone as the means of going from an abstract model of a system to a concrete one, in a completely formal manner. Refinement, particularly in formulations that emphasise total correctness, imposes stringent constraints on the relationship that can hold between the related models, and this can restrict the applicability of the technique quite severely. This phenomenon is most keenly felt in applications 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 situations, it is commonly found that all except the last few steps of the development process have to be performed informally — because the proof obligations 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, such as mechanical checkability, for the majority of the development effort.

By refinement, we understand its usual forward simulation incarnation; see [5] for a comprehensive review of refinement from both forward and backward simulation perspectives, and in both syntactic and semantic frameworks. Unlike refinement, a retrenchment step from a more abstract to a more concrete level of abstraction admits strengthening of the precondition and weakening of the postcondition, and the mingling of state and I/O information between the levels of abstraction in question. This is accomplished by having two extra predicates per retrenched operation, the WITHIN and CONCEDES clauses. (That is what they were called in the B-Method [6, 7, 8], within which the original work on retrenchment was done.) The former expresses the precondition strengthening, and the latter expresses the postcondition weakening. In particular, non-refinement-like behaviour can be accomodated within the framework via the weakened postcondition. This in turn permits inconvenient low level detail of the true system from interfering with an idealised model at a high level of abstraction, leading to hopefully cleaner, more comprehensible, earlier formalisable development routes.

Retrenchment can thus be seen as serving two related purposes. The first is to aid in a formally controllable manner, the postponement of the consideration of levels of detail during the construction of the actual system definition *because it is more perspicuous to do the development that way* (and *that way* would be impossible using refinement alone, even if some other, less transparent, way would yield to pure refinement). The second is to accomplish the same

objective *because accepted engineering practice demands consideration of models not relatable by refinement* (and thus only a fraction of some accepted development route could ever be formally expressed). In the first area, retrenchment gives developers *scope for building specifications in a more understandable manner* (though a formal route would be available anyway); in the second retrenchment *makes formality usable in situations where it hitherto has not been so*.

In both senses, retrenchment postpones the introduction of details that for various reasons one does not wish to consider immediately. This postponement normally goes hand in hand with data concretisation — for instance the limitations of a discrete approximation to the solution of a problem expressed in continuous mathematics do not normally become of interest till the representation of the data for the problem is itself sufficiently concrete and discrete. Nevertheless the question can be put, as to what is the counterpart at the original level of abstraction, of the concrete problem description arrived at through one or more retrenchment steps.

It is with this issue that this paper is concerned. We give a universal construction that can be seen as porting the additional structure introduced at the lower level of a retrenchment, to the higher level.

What do we gain by doing this? Three things. Firstly, we give an affirmative answer to the question of whether, given a retrenchment, there is a model at the abstract level which could have been *refined* to the given concrete level; in fact there is more than one way of answering this question (see the Conclusions). Secondly, we integrate retrenchment and refinement, an important consideration for practitioners, for whom the flexibility of choosing between retrenchment and refinement development steps no longer risks fracturing the development process into incompatible and irreconcilable paths. Thirdly, integrating all the additional detail introduced during retrenchment steps into the abstract level of one big refinement, yields — at least in principle — a technique for validating that all the requirements of the system have been adequately taken into account.

We say 'at least in principle' because we rather doubt whether practitioners would actually feel much inclined to do requirements validation in this way. If retrenchment has been used in a natural manner to introduce detail and complexity into a development at appropriate points, it is unlikely to be more appealing to validate the requirements at issue by translating them to a less appropriate level of abstraction. It is far more likely that a validation process would be content to confront the collection of requirements against the actual places in the development where they have been addressed, relying on the validation of the specific retrenchment proof obligations to ensure consistency.

Having said the above, the author is in no doubt that the process of validating requirements via the route opened up by the results of this paper would be a useful thing to do, if for no other reason than that it is *always useful* to force oneself to view a problem from a different perspective — and integrating the issues addressed during the various retrenchment steps of a development has exactly this character. Still, it is not the purpose of this paper to debate the merits or otherwise of this *primarily methodological* aspect of requirements validation; we are more concerned with setting out the technical foundations on which such an approach would rest. We truncate the discussion of it here.

In outline the rest of this paper is as follows. In Section 2 we review some of the reasons why refinement alone is too stringent a technique to capture all the development steps we might consider to be desirable. Section 3 introduces retrenchment, in a simple partial correctness form that makes for a transparent theory. Section 4 presents I/O-filtered refinements, the kind of refinements needed later. Section 5 develops maximally abstract retrenchments, discussing their universality properties. Section 6 discusses junk-freedom and idempotence, while Section 7 treats an example. Section 8 briefly considers guaranteed termination, as needed for general and total correctness. Section 9 concludes.

Notation. In the sequel we will view systems 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

Almost all applications of refinement are of the forward simulation incarnation of the concept. For this reason we will concentrate on forward simulation in this paper.

To introduce a running example, let us examine a system whose state u is a set of NATs, and which has an operation $AddEl(n)$ to add an element n to the set. Its description (in a suitable syntactic framework, which will be of no further concern to us) constitutes a simple specification, our abstract model. At a more concrete level, we model sets of NATs by injective sequences of NATs, so $u = \{1, 2, 3\}$ would correspond to $w = [1, 2, 3]$, or $w = [2, 1, 3]$, or to any of four other possibilities. There is an obvious retrieve relation relating sets and all possible serialisations of their elements. For pragmatic reasons, the length of any representative sequence can be no greater than 10, while no such restriction applies to the cardinality of the set. Thus not all sets have concrete representations.

The $AddEl(n)$ operation at the concrete level must test the length of the sequence and for the prior presence of n. If n is new and the length is 10, the modelling of set union, for example by appending the extra element to the end of

the sequence, is forbidden. It would break the bound on the length of w. Whatever the concrete $AddEl(n)$ operation does in this situation, we claim it cannot be a refinement of the abstract $AddEl(n)$. For to be a refinement, in the given situation, it would have to satisfy:

$$G(u, w) \wedge stp_{AddEl_C}(w, i, w') \Rightarrow (\exists\, u' \bullet stp_{AddEl_A}(u, i, u') \wedge G(u', w')) \quad (2.1)$$

where G is the retrieve relation, $stp_{AddEl_C}(w, i, w')$ is the transition or step relation of the concrete $AddEl$ (similarly for the abstract one), and primes indicate the after-state.

If the concrete operation did nothing, i.e. did a $skip$, (2.1) would fail as $skip$ is not an option at the abstract level. In the same way, if the concrete operation output an error message, (2.1) would fail as a change in signature is not allowed in conventional refinement. These two possibilities exhaust the relatively sensible options for the concrete operation. Even if the concrete operation did something else within its constraints, it could not tie up with the abstract operation, which would produce a set of cardinality 11.

The above shows one very simple scenario in which refinement fails to adequately describe a desirable development step. Many situations involving a finite computable subdomain of a mathematically ideal and infinite one, follow the same pattern, and a number of techniques have been described in the literature, designed to address the same issue. For example, there is Neilson's thesis [9], which presents the concept of acceptably inadequate refinements. These tackle the above problem by observing that the infinite ideal domains usually arise as well behaved limits of the finite ones, and thus refinement in the idealised case can be understood as the limit of a finite version. The failure of refinement observed above becomes merely the manifestation of an incompletely taken limit. Another avenue of attack by Owe is to be found in [10, 11] where he proposes a logical approach, based on a careful analysis of the effects of ill-definedness on a programming logic.

Another issue hinted at above is the desirability of changing I/O signatures during the course of a development step. 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 in refinement. In the literature the question has been pondered by a number of authors amongst which [12, 13, 14, 15].

Retrenchment, introduced below, brings all of these aspects together into a single generic and flexible development step, quantified by a pair of predicates per operation. Interestingly enough, a similar goal has been tackled in the work of Liu on evolution [16]. This is a technique whereby the issues discussed above falling outside of conventional refinement, are incorporated by demanding that the pre- and post- conditions of the abstract specification in a development step be semantically equivalent to subformulae of the pre- and post- conditions at the more concrete level. The point here is that any predicate Q can be viewed as a subformula of a formula equivalent to any other predicate P, since Q is a subformula of $((P \wedge Q) \vee (P \wedge \neg Q))$ which is equivalent to P. This fact tends to rob the evolution notion of semantic bite, and without further ado, relates any two models (with corresponding operations). Of course the same can be said to apply for retrenchment if one makes the within and concedes relations strong enough; but in retrenchment, because these relations form part of the definition of the retrenchment step, in making these clauses strong, one is being explicit about just how extreme the relationship between the models is. The 'is a subformula of something equivalent to' property lacks this element of discrimination.

3. Retrenchment

For the majority of this paper we adhere to a partial correctness framework for system description. A good comparison between the partial and total correctness frameworks, and also a critical evaluation of many approaches to refinement, can be found in [5]. In our framework, a system will be given by a state space, a set of operation names each with its own I/O signature, and a transition relation for which each step is labelled by one of the operation names. To discuss retrenchment we need two systems, an abstract one Abs and a concrete one $Conc$.

At the abstract level, the set of operation names is Ops_A, with typical element Op_A. The state space is U, having typical element u. For an $Op_A \in \mathsf{Ops}_A$, the input and output spaces will be I_{Op_A} and O_{Op_A} with typical elements i, o respectively (the anticipated subscripts on i and o to indicate which Op_A they belong to will be suppressed, without causing confusion). Primes, indeces and other decorations will be used to distinguish different elements of the same space. A typical system transition will be written using the notation u -(i, Op_A, o)-› u', where u and u' are the before- and after- states, i and o are the input and output values, and Op_A is the name of the operation responsible for the transition. The set of all such transitions makes up the transition or step relation for the operation Op_A, $stp_{Op_A}(u, i, u', o)$, which we will always assume non-empty.

At the concrete level we have a similar setup. The operation names are $Op_C \in \mathsf{Ops}_C$. States are $w \in \mathsf{W}$, inputs $k \in \mathsf{K}$, outputs $q \in \mathsf{Q}$. The transitions at the concrete level are written w -(k, Op_C, q)-› w', and are members of the step relation $stp_{Op_C}(w, k, w', q)$. In retrenchment it is assumed that there is a distinct Op_C corresponding to each distinct Op_A, but not necessarily vice versa, so the concrete level may contain additional operations. We will assume this correspondence is an inclusion $\mathsf{Ops}_A \subseteq \mathsf{Ops}_C$. Thus we have the decomposition $\mathsf{Ops}_C = \mathsf{Ops}_A \cup (\mathsf{Ops}_C - \mathsf{Ops}_A)$.

The relationship between abstract and concrete state spaces is given by the retrieve relation $G(u, w)$. We assume that there are initialisation operations $Init_{\mathrm{A}}$ and $Init_{\mathrm{C}}$ at abstract and concrete levels that establish G in corresponding after-states thus:

$$Init_{\mathrm{C}}(w') \Rightarrow (\exists\, u' \bullet Init_{\mathrm{A}}(u') \wedge G(u', w')) \qquad (3.1)$$

Speaking conventionally, refinement now sets itself the goal of preserving the retrieve relation G through executions at the abstract and concrete levels, and different assumptions about how this might be accomplished lead to differing properties of the process; these then become the proof obligations (POs) of the particular approach, which when discharged for some specific application, prove that the application scenario is indeed a refinement. See once more [5] for a survey.

Retrenchment differs from refinement in that G alone is not enough to fix the relationship between the two levels. We also have for all operation names in $\mathsf{Ops}_{\mathrm{A}}$, a within relation $P_{Op}(i, k, u, w)$ and a concedes relation $C_{Op}(u', w', o, q; i, k, u, w)$, where, exploiting the inclusion of $\mathsf{Ops}_{\mathrm{A}} \subseteq \mathsf{Ops}_{\mathrm{C}}$, the A/C subscripts on Op have been suppressed because these relations are relevant to both the abstract and concrete levels. The punctuation in C_{Op} is intended to highlight that it is mainly concerned with after-values, but may refer to the before-values too if required. These relations are combined into the retrenchment operation proof obligation for steps of the $\mathsf{Ops}_{\mathrm{A}}$ operations which says that for each such Op:

$$\begin{aligned} &G(u, w) \wedge P_{Op}(i, k, u, w) \wedge stp_{Op_{\mathrm{C}}}(w, k, w', q) \Rightarrow \\ &\quad (\exists\, u', o \bullet stp_{Op_{\mathrm{A}}}(u, i, u', o) \wedge \\ &\quad (G(u', w') \vee C_{Op}(u', w', o, q; i, k, u, w))) \end{aligned} \qquad (3.2)$$

Compared to the conventional forward simulation refinement PO, the import of this is the following. We assert the consequent of the implication, but only provided both G and P hold. This enables us to restrict via the within relation P, 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 G, or failing that, satisfies the concedes relation C. Again the additional flexibility allowed by C 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] (also before- entities if appropriate).

The structure of the PO (3.2) also yields two predicates, the abstract and concrete core predicates of the operation Op, $cor_{Op_{\mathrm{A}}}(u, i)$ and $cor_{Op_{\mathrm{C}}}(w, k)$, identifying those parts of the abstract and concrete state and input spaces for Op, for which the PO (3.2) is non-trivial:

$$cor_{Op_{\mathrm{A}}}(u, i) = (\exists\, w, k \bullet G(u, w) \wedge P_{Op}(i, k, u, w)) \qquad (3.3)$$
$$cor_{Op_{\mathrm{C}}}(w, k) = (\exists\, u, i \bullet G(u, w) \wedge P_{Op}(i, k, u, w)) \qquad (3.4)$$

Clearly what happens in either model outside of the cores has no bearing on the retrenchment and can thus be arbitrary. A set of transitions of a system forming part of a retrenchment is said to be abstract (resp. concrete) core bounded iff all the (before-state, input) pairs of the set satisfy the abstract (resp. concrete) core predicates. These concepts will be needed in the universality results below; they will be used analogously to totality assumptions.

We now reconsider our running example as a retrenchment. Predictably $\mathsf{Ops}_{\mathrm{A}} = \mathsf{Ops}_{\mathrm{C}} = \{Init, AddEl\}$, $\mathsf{U} = \mathsf{P}(\mathsf{NAT})$, $\mathsf{W} = \mathsf{iseq}(\mathsf{NAT})$, $\mathsf{I}_{AddEl} = \mathsf{K}_{AddEl} = \mathsf{NAT}$, $\mathsf{O}_{AddEl} = \emptyset$, and Q_{AddEl} would depend on how we defined the concrete $AddEl$ operation. In the simple case in which $AddEl_{\mathrm{C}}$ is *skip* for exceptional cases, $\mathsf{Q}_{AddEl} = \emptyset$, otherwise it contains an overflow message; we concentrate on the former.

To describe the *skip* case fully, we need to give the within and concedes relations for $AddEl(n)$. Some examples follow, the first three in order of increasing strength:

$$\begin{aligned} P_1 &= \mathit{true}\ ,\ C_1 = (n \notin u \Rightarrow w' = w) \\ P_2 &= \mathit{true}\ ,\ C_2 = (n \notin u \wedge w' = w) \\ P_3 &= \mathit{true}\ ,\ C_3 = (|u| = 10 \wedge n \notin u \wedge w' = w) \\ P_4 &= (|u| \leq 10)\ ,\ C_4 = (n \notin u \wedge w' = w) \end{aligned}$$

There are several similar possibilities. Alternatively:

$$P_5 = (|u| < 10 \vee (|u| = 10 \wedge n \in u))\ ,\ C_5 = \mathit{false}$$

Note that option 5 describes fewer corresponding pairs of steps than the others, since the cases when $|u| = 10 \wedge n \notin u$ are not in the scope of the relationship between the two models. In fact the triviality of the concedes relation here means that option 5 is describing a particular kind of refinement, and shows that if one restricts one's focus suitably, one can often find a refinement lurking inside a retrenchment. This point of view is explored also in [3, 4].

The unfortunate consequence of this is of course that as one seeks to avoid mentioning the details that accumulate as one descends through the development process, the relationship between the top and bottom levels of abstraction as

1. Relations between outputs and states ought to hold universally, and not just when G fails. The truth is thus stronger than (3.2). Sharp retrenchment addresses this, establishing in the consequent of the proof obligation $((G \vee C) \wedge V)$ where V, the nevertheless relation, allows extra conjunctive properties to be expressed. See [4]. We will ignore the nevertheless relation in this paper, aside from its use in I/O-filtered refinements, in which concedes relations in their turn do not appear.

expressed by such a refinement is increasingly noncommital; only a small portion of the two models is spoken about.

4. I/O-Filtered Refinements

In this section we make precise the particular notion of refinement we need so that our subsequent results go through unproblematically. The conventional notion of forward simulation in which I/O signatures remain unchanged is not quite enough, as I/O signatures can be changed during retrenchment and this effect leaks through into the universal result we seek in this paper.

For economy's sake, we reuse the notation set up already. Thus we assume abstract and concrete systems *Abs* and *Conc*, with $\mathsf{Ops}_\mathrm{A} = \mathsf{Ops}_\mathrm{C}$, and with the state spaces related by a retrieve relation $G(u, w)$ as before. For an I/O-filtered refinement, we furthermore have for each $Op \in \mathsf{Ops}_\mathrm{A}$, a within relation $R_{Op}(i, k)$, and a nevertheless relation $V_{Op}(o, q)$. All three relations are constrained to be total on their first components and onto on their second components; i.e. they are relations S satisfying firstly $\forall a \exists b \bullet S(a, b)$ and secondly $\forall b \exists a \bullet S(a, b)$. They are relations from abstract to concrete states, from abstract to concrete inputs, and from abstract to concrete outputs respectively. Unlike the retrenchment case, the within and nevertheless relations are not permitted to involve the states, thus separating concerns. This assembly of components is required to verify the following proof obligations.

Firstly there is the initialisation PO. This is the same as (3.1) so we do not restate it here. Secondly there is the operation PO which for a typical Op reads:

$$\begin{aligned} &G(u, w) \wedge R_{Op}(i, k) \wedge stp_{Op_\mathrm{C}}(w, k, w', q) \Rightarrow \\ &\quad (\exists\, u', o \bullet stp_{Op_\mathrm{A}}(u, i, u', o) \wedge \\ &\quad (G(u', w') \wedge V_{Op}(o, q))) \end{aligned} \tag{4.1}$$

From this we can derive the core predicates for I/O-filtered refinements:

$$cor_{Op_\mathrm{A}}(u, i) = \mathit{true} \tag{4.2}$$

$$cor_{Op_\mathrm{C}}(w, k) = \mathit{true} \tag{4.3}$$

Note also that V_{Op} enters the consequent of (4.1) conjunctively, in contrast to the retrenchment case.

5. Maximally Abstract Retrenchments

In this section we take the retrenchment from *Abs* to *Conc* in Section 3, and manufacture a third, universal system *Univ*, enjoying the claimed universal properties. The operation names set of *Univ* is Ops_U with elements Op_U. The state space is V with elements v, inputs are $j \in \mathsf{J}$, outputs $p \in \mathsf{P}$. These are all constructed from the *Abs* and *Conc* systems as follows.

Firstly $\mathsf{Ops}_\mathrm{U} = \mathsf{Ops}_\mathrm{C}$. Each Op_U is in fact one of the Op_Cs. The spaces are $\mathsf{V} = \mathsf{U} \times \mathsf{W}$, $\mathsf{J} = \mathsf{I} \times \mathsf{K}$ and $\mathsf{P} = \mathsf{O} \times \mathsf{Q}$. To give the transitions of *Univ* we firstly observe that Ops_U decomposes as $\mathsf{Ops}_\mathrm{U} = \mathsf{Ops}_\mathrm{A} \cup (\mathsf{Ops}_\mathrm{U} - \mathsf{Ops}_\mathrm{A})$.

For an operation $Op_\mathrm{U} \in (\mathsf{Ops}_\mathrm{U} - \mathsf{Ops}_\mathrm{A})$, we have a transition $v \text{-}(j, Op_\mathrm{U}, p)\text{-}\!\rightarrow v'$ or more explicitly:

$$(u, w) \text{-}((i, k), Op_\mathrm{U}, (o, q))\text{-}\!\rightarrow (u', w') \tag{5.1}$$

for arbitrary u, i, o, u', w, k, q, w'; so that the non-Op_A transitions of *Univ* form a universal relation.

For an $Op_\mathrm{A} \in \mathsf{Ops}_\mathrm{A}$, we have $v \text{-}(j, Op_\mathrm{A}, p)\text{-}\!\rightarrow v'$ or:

$$(u, w) \text{-}((i, k), Op_\mathrm{A}, (o, q))\text{-}\!\rightarrow (u', w') \tag{5.2}$$

iff u, i, o, u', w, k, q, w' satisfy:

$$\begin{aligned} &G(u, w) \wedge P_{Op}(i, k, u, w) \Rightarrow \\ &\quad stp_{Op_\mathrm{A}}(u, i, u', o) \wedge \\ &\quad (G(u', w') \vee C_{Op}(u', w', o, q; i, k, u, w)) \end{aligned} \tag{5.3}$$

Note that the implicational form admits many 'junk' transitions, namely $\overline{(G \wedge P)} \times (\mathsf{U} \times \mathsf{W} \times \mathsf{O} \times \mathsf{Q})$. We will return to this later.

Finally, $Init_\mathrm{U}(v')$ in *Univ* assigns v' to any value (u', w') such that $Init_\mathrm{A}(u') \wedge G(u', w')$ holds.

We now define a retrenchment from *Abs* to *Univ* and an I/O-filtered refinement from *Univ* to *Conc*, by giving the data for these constructions and showing that the appropriate POs are satisfied. We further show that the composition of this new retrenchment and I/O-filtered refinement yields the original retrenchment.

The data for the retrenchment consists of the retrieve and within relations (which yield the core predicates) and concedes relation. The retrieve relation is:

$$H(u, v) = (v = (u, w) \wedge G(u, w)) \tag{5.4}$$

For an operation Op the within relation is:

$$\begin{aligned} &Q_{Op}(i, j, u, v) = \\ &\quad (j = (i, k) \wedge v = (u, w) \wedge P_{Op}(i, k, u, w)) \end{aligned} \tag{5.5}$$

A little calculation gives the abstract core:

$$\begin{aligned} &cor_{Op_\mathrm{A}}(u, i) = (\exists\, v, j \bullet H(u, v) \wedge Q_{Op}(i, j, u, v)) = \\ &\quad (\exists\, w, k \bullet G(u, w) \wedge P_{Op}(i, k, u, w)) \end{aligned} \tag{5.6}$$

which is identical to (3.3), while the concrete core is:

$$cor_{Op_\mathrm{U}}(v, j) = (\exists\, u, i \bullet H(u, v) \wedge Q_{Op}(i, j, u, v)) \tag{5.7}$$

The concedes relation is:

$$\begin{aligned} &D_{Op}(u', v', o, p; i, j, u, v) = \\ &\quad (v' = (u', w') \wedge p = (o, q) \wedge j = (i, k) \wedge v = (u, w) \wedge \\ &\quad C_{Op}(u', w', o, q; i, k, u, w)) \end{aligned} \tag{5.8}$$

To prove that we have a retrenchment we must show the appropriate POs are valid.

The initialisation PO is $Init_\mathrm{U}(v') \Rightarrow (\exists\, u' \bullet Init_\mathrm{A}(u') \wedge H(u', v'))$. Now $Init_\mathrm{U}$ assigns v' to any value (u', w') such that $Init_\mathrm{A}(u')$, and $G(u', w')$ hold. Noting that $H(u', v')$ iff $G(u', w')$ and $v' = (u', w')$, we see that for any initial v' there is indeed a u' as required.

For the operation PO we have to show that:

$$H(u, v) \wedge Q_{Op}(i, j, u, v) \wedge stp_{Op_U}(v, j, v', p) \Rightarrow (\exists\, u', o \bullet stp_{Op_A}(u, i, u', o) \wedge (H(u', v') \vee D_{Op}(u', v', o, p; i, k, u, v)))$$

We assume the antecedents. Now $H(u, v) \wedge Q_{Op}(i, j, u, v)$ implies that $v = (u, w)$ and $j = (i, k)$, and that moreover $G(u, w) \wedge P_{Op}(i, k, u, w)$ holds. Next, knowing $stp_{Op_U}(v, j, v', p)$, by (5.3), from $G(u, w) \wedge P_{Op}(i, k, u, w)$ we can deduce that $stp_{Op_A}(u, i, u', o) \wedge (G(u', w') \vee C_{Op}(u', w', o, q; i, k, u, w))$ is true. Therefore there are u', o such that $stp_{Op_A}(u, i, u', o)$ holds, namely the ones just mentioned, and for these u' and o, having $G(u', w') \vee C_{Op}(u', w', o, q; i, k, u, w)$, we easily derive $H(u', v') \vee D_{Op}(u', v', o, p; i, k, u, v)$. This yields the consequent of the retrenchment PO.

Turning to the I/O-filtered refinement, its data consists of the retrieve and within relations, and the nevertheless relation. The retrieve relation is:

$$K(v, w) \;=\; (v = (u, w)) \tag{5.9}$$

For an operation Op the within relation is:

$$R_{Op}(j, k) \;=\; (j = (i, k)) \tag{5.10}$$

and we note that the abstract and concrete cores will be trivial. The nevertheless relation is:

$$V_{Op}(p, q) \;=\; (p = (o, q)) \tag{5.11}$$

Noting that these three relations are projection functions onto the second component, we immediately conclude that they are total and surjective relations.

The POs for the I/O-filtered refinement are dealt with as follows. This time the initialisation PO demands that we have $Init_C(w') \Rightarrow \exists\, v' \bullet Init_U(v') \wedge K(v', w')$. Now the initialisation PO of the original retrenchment (3.1) ensures that for an initial w' there is an initial u' such that $G(u', w')$ holds. Combining this u' with w' gives a $v' = (u', w')$ such that $Init_U(v') \wedge K(v', w')$.

To establish the operation PO we need to show that:

$$K(v, w) \wedge R_{Op}(j, k) \wedge stp_{Op_C}(w, k, w', q) \Rightarrow (\exists\, v', p \bullet stp_{Op_U}(v, j, v', p) \wedge (K(v', w') \wedge V_{Op}(p, q)))$$

For operations $Op_C \in (\mathsf{Ops}_U - \mathsf{Ops}_A)$, assuming the antecedents, choosing any v', p such that $K(v', w') \wedge V_{Op}(p, q)$ holds will satisfy the predicate, since the surjectivity of K and V_{Op} makes this possible, and $stp_{Op_U}(v, j, v', p)$ is universal for such operations.

For operations $Op_C \in \mathsf{Ops}_A$, assume the antecedents. Now $K(v, w) \wedge R_{Op}(j, k)$ implies that $v = (u, w)$ and $j = (i, k)$ for some u and i; and we know $stp_{Op_C}(w, k, w', q)$ holds. Then either $G(u, w) \wedge P_{Op}(i, k, u, w)$ holds or not. If it does, then we have all the antecedents of the original retrenchment PO (3.2), which asserts that values u', o exist such that $stp_{Op_A}(u, i, u', o) \wedge (G(u', w') \vee C_{Op}(u', w', o, q; i, k, u, w))$ holds. With such values (5.3) is satisfied, allowing us to assert that there are v', p such that $stp_{Op_U}(v, j, v', p)$ holds, and such that $K(v', w') \wedge V_{Op}(p, q)$ holds too (this being easy to check), which together yields the consequent for this case.

Otherwise, $G(u, w) \wedge P_{Op}(i, k, u, w)$ does not hold and we are in the 'junk' case of (5.3). This is trivial, so we need only choose v', p such that $K(v', w') \wedge V_{Op}(p, q)$ holds. This is always possible since K and V_{Op} are surjective. These v', p automatically satisfy $stp_{Op_U}(v, j, v', p)$ so we are done.

It remains to define the composition of the retrenchment and I/O-filtered refinement just constructed. This requires care since not only are retrenchments and refinements different concepts, but also the collections of variables of the intermediate system occurring in abutting relations are not the same. We define the composition to be a retrenchment for which the component relations are given as follows.

The retrieve relation is the (usual) composition of the component retrieve relations:

$$(H;K)(u, w) \;=\; (\exists\, v \bullet H(u, v) \wedge K(v, w)) \tag{5.12}$$

The within relation is the composition of the component within and retrieve relations in the following sense:

$$(Q_{Op};(R_{Op} \wedge K))(i, k, u, w) \;=\; (\exists\, v, j \bullet Q_{Op}(i, j, u, v) \wedge R_{Op}(j, k) \wedge K(v, w)) \tag{5.13}$$

The concedes relation is a combination of the component concedes, retrieve, nevertheless, and within relations in the following manner:

$$(D_{Op};(K' \wedge V_{Op} \wedge R_{Op} \wedge K))(u', w', o, q; i, k, u, w) \;= (\exists\, v, j, v', p \bullet D_{Op}(u', v', o, p; i, j, u, v) \wedge K(v', w') \wedge V_{Op}(p, q) \wedge R_{Op}(j, k) \wedge K(v, w)) \tag{5.14}$$

For the retrenchment and refinement constructed above it is relatively evident that this notion of composition recovers the relations of Section 3.

Theorem 5.1 Let there be a retrenchment as above from the *Abs* to *Conc*. Then (see Fig. 1):

(1) There is a universal system *Univ* such that: there is a retrenchment from *Abs* to *Univ* and an I/O-filtered refinement from *Univ* to *Conc* whose composition is the given retrenchment.

(2) Whenever there is a system *Xtra* and a retrenchment from *Abs* to *Xtra* and an I/O-filtered refinement from *Xtra* to *Conc* whose composition is the given retrenchment, then there is an I/O-filtered refinement from *Univ* to the concrete core bound transitions of *Xtra*; such that $H^{\sim} \Rightarrow H;K^{\circ}$, $Q^{\sim} \Rightarrow Q;R^{\circ} \wedge K^{\circ}$, $D^{\sim} \Rightarrow D;K^{\circ\prime} \wedge V^{\circ} \wedge R^{\circ} \wedge K^{\circ}$, and such that $K \Rightarrow K^{\circ};K^{\sim}$, $R \Rightarrow R^{\circ};R^{\sim}$, $V \Rightarrow V^{\circ};V^{\sim}$.

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

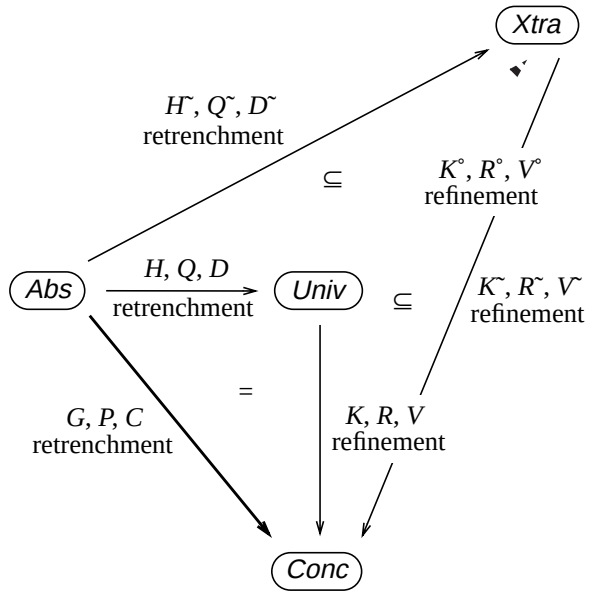

Fig. 1

Proof. We have proved (1) already, defining what we needed along the way. Now assume a retrenchment from *Abs* to *Xtra* given by retrieve relation $H^{\breve{}}$, within relation $Q^{\breve{}}$, and concedes relation $D^{\breve{}}$; and an I/O-filtered refinement from *Xtra* to *Conc* given by retrieve relation $K^{\breve{}}$, within relation $R^{\breve{}}$, and nevertheless relation $V^{\breve{}}$. Let the state, input and output spaces of *Xtra* be given by $v^{\breve{}} \in \mathsf{V}^{\breve{}}$, $j^{\breve{}} \in \mathsf{J}^{\breve{}}$, $p^{\breve{}} \in \mathsf{P}^{\breve{}}$. Let $Init_{\mathrm{X}}$ and $stp_{Op_{\mathrm{X}}}$ be the initialisation and step predicates for *Xtra*.

We define relations K°, $R^{\circ}{}_{Op}$, $V^{\circ}{}_{Op}$, indicate that they satisfy the inclusions stated, and prove that they are the retrieve, within and nevertheless relations of an I/O-filtered refinement from *Univ* to the concrete core bound *Xtra* transitions. Thus:

$$K^{\circ}(v, v^{\breve{}}) \;=\; (\exists\, w \bullet K(v, w) \wedge K^{\breve{}}(v^{\breve{}}, w)) \qquad (5.15)$$
$$R^{\circ}{}_{Op}(j, j^{\breve{}}) \;=\; (\exists\, k \bullet R_{Op}(j, k) \wedge R^{\breve{}}{}_{Op}(j^{\breve{}}, k)) \qquad (5.16)$$
$$V^{\circ}{}_{Op}(p, p^{\breve{}}) \;=\; (\exists\, q \bullet V_{Op}(p, q) \wedge V^{\breve{}}{}_{Op}(p^{\breve{}}, q)) \qquad (5.17)$$

Firstly K° is total and surjective since K is a total surjective function and $K^{\breve{}}$ is total and surjective. Likewise for $R^{\circ}{}_{Op}$ and $V^{\circ}{}_{Op}$. Proving that $H^{\breve{}} \Rightarrow H;K^{\circ}$, $Q^{\breve{}}{}_{Op} \Rightarrow Q_{Op};R^{\circ}{}_{Op}$, $D^{\breve{}}{}_{Op} \Rightarrow D_{Op};V^{\circ}{}_{Op}$, and that $K \Rightarrow K^{\circ};K^{\breve{}}$, $R_{Op} \Rightarrow R^{\circ}{}_{Op};R^{\breve{}}{}_{Op}$, $V_{Op} \Rightarrow V^{\circ}{}_{Op};V^{\breve{}}{}_{Op}$ is now an easy exercise and is left to the reader.

Next, we examine the initialisation PO. We need to show that $Init_{\mathrm{X}}(v^{\breve{}\prime}) \Rightarrow \exists\, v' \bullet Init_{\mathrm{U}}(v') \wedge K^{\circ}(v', v^{\breve{}\prime})$, so let us choose an initial $v^{\breve{}\prime}$. We know for this $v^{\breve{}\prime}$ that $Init_{\mathrm{X}}(v^{\breve{}\prime}) \Rightarrow (\exists\, u' \bullet Init_{\mathrm{A}}(u') \wedge H^{\breve{}}(u', v^{\breve{}\prime}))$ from the *Abs* to *Xtra* retrenchment, so pick a suitable u'. Let w' be such that $G(u', w')$ holds; such a w' must exist because $K^{\breve{}}$ is total and $G = H^{\breve{}};K^{\breve{}}$. Then $v' = (u', w')$ is easily shown to be a suitable v'.

Now we establish the operation PO, namely that for the concrete core bound steps of *Xtra* we have:

$$K^{\circ}(v, v^{\breve{}}) \wedge R^{\circ}{}_{Op}(j, j^{\breve{}}) \wedge stp_{Op_{\mathrm{X}}}(v^{\breve{}}, j^{\breve{}}, v^{\breve{}\prime}, p^{\breve{}}) \;\Rightarrow$$
$$\quad (\exists\, v', p \bullet stp_{Op_{\mathrm{U}}}(v, j, v', p) \wedge$$
$$\quad (K^{\circ}(v', v^{\breve{}\prime}) \wedge V^{\circ}{}_{Op}(p, p^{\breve{}})))$$

For an operation $Op_{\mathrm{X}} \in (\mathsf{Ops}_{\mathrm{X}} - \mathsf{Ops}_{\mathrm{A}})$, assuming the antecedents, then choosing any v', p such that $K^{\circ}(v', v^{\breve{}\prime}) \wedge V^{\circ}{}_{Op}(p, p^{\breve{}})$ will do, since the surjectivity of K° and $V^{\circ}{}_{Op}$ makes this possible, and $stp_{Op_{\mathrm{U}}}(v, j, v', p)$ is universal for such operations.

For operations $Op_{\mathrm{X}} \in \mathsf{Ops}_{\mathrm{A}}$, let us assume the antecedents. If $v = (u, w)$ and $j = (i, k)$ are such that $G(u, w) \wedge P_{Op}(i, k, u, w)$ does not hold, then the preceding argument works, since $stp_{Op_{\mathrm{U}}}(v, j, v', p)$ is still universal.

Finally suppose $Op_{\mathrm{X}} \in \mathsf{Ops}_{\mathrm{A}}$ and that $v = (u, w)$ and $j = (i, k)$ are such that $G(u, w) \wedge P_{Op}(i, k, u, w)$ does hold. We assume that $v^{\breve{}}$ -$(j^{\breve{}}, Op_{\mathrm{X}}, p^{\breve{}})$-› $v^{\breve{}\prime}$ is a concrete core bound step, since otherwise there is nothing to prove. Therefore $H^{\breve{}}(u, v^{\breve{}}) \wedge Q^{\breve{}}{}_{Op}(i, j^{\breve{}}, u, v^{\breve{}})$ holds. Since we now have the antecedents of the *Abs* to *Xtra* retrenchment, we infer that there is a step u -(i, Op_{X}, o)-› u' such that $H^{\breve{}}(u', v^{\breve{}\prime}) \vee D^{\breve{}}{}_{Op}(u', v^{\breve{}\prime}, o, p^{\breve{}}; i, j^{\breve{}}, u, v^{\breve{}})$ holds. Now $K^{\breve{}}$ and $V^{\breve{}}{}_{Op}$ are total, and $G = H^{\breve{}};K^{\breve{}}$ and $C_{Op} = D^{\breve{}}{}_{Op};K^{\breve{}\prime} \wedge V^{\breve{}}{}_{Op} \wedge R^{\breve{}}{}_{Op} \wedge K^{\breve{}}$, by our assumptions. Therefore we can deduce $G(u', w') \vee C_{Op}(u', w', o, q; i, k, u, w)$ for suitable w' and q. We have now assembled enough pieces to use (5.3) to conclude that $stp_{Op_{\mathrm{U}}}(v, j, v', p)$ holds, where $v' = (u', w')$ and $p = (o, q)$. Noting that K and V_{Op} are projections allows us to quickly deduce $K^{\circ}(v', v^{\breve{}\prime}) \wedge V^{\circ}{}_{Op}(p, p^{\breve{}})$, completing this case and part (2) of the theorem.

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 the concrete core bound steps of *Univ** and an I/O-filtered refinement from *Univ** to the concrete core bound steps of *Univ*. We are done. ☺

Note that the mutual interrefinability in Theorem 5.1.(3) is not isomorphism in the conventional set theoretical sense. To illustrate, the reader might have noticed that of the 'junk' transitions afforded by (5.3), we only ever used ones satisfying $(G \vee C)$. Therefore an alternative definition of $stp_{Op_{\mathrm{U}}}$ as $(G \vee C) \wedge ((G \wedge P) \Rightarrow stp_{Op_{\mathrm{A}}})$ would have sufficed above, and this and (5.3) are certainly not equivalent in the sense of conventional set theoretical isomorphism. In fact the mutual interrefinability we showed yields a much looser notion of equivalence of systems.

6. Idemopotence and Junk-Freedom

We claimed that the system *Univ* represented a system at the level of abstraction of *Abs* that captured the constraints demanded by *Conc*, and supported this statement by showing any other retrenchment of *Abs* that refined to *Conc* was refinable through *Univ*. But *Univ* looks pretty complicated. Can we be sure its level of abstraction is high enough? One test of this is to apply the construction of *Univ* to the *Abs* to *Univ* retrenchment itself, to see if we get something substantially different. The steps of a system *UUniv* constructed in pursuit of such an objective must satisfy the analogue of (5.3):

$$H(u, v) \wedge Q_{Op}(i, j, u, v) \Rightarrow \\ stp_{Op_A}(u, i, u', o) \wedge \\ (H(u', v') \vee D_{Op}(u', v', o, p; i, j, u, v)) \qquad (7.1)$$

But substituting for H, Q_{Op}, D_{Op} in the familiar manner leads right back to (5.3), thus the set of non-junk transitions of (7.1) is isomorphic to those of (5.3), since they are of the form $(u, (u, w))$ -$((i, (i, k)), Op_A, (o, (o, q)))$-› $(u', (u', w'))$ and having two copies of the same *Abs* transition in a *UUniv* transition is isomorphic to having one copy in the corresponding *Univ* transition. This supports our claim to have indeed found the appropriate level of abstraction.

The junk transitions permitted by (7.1) are not however isomorphic in the same way. Different *Abs* (u, i, u', o) quadruples (they needn't be *Abs* transitions of course) may reside in the inner and outer positions of a *UUniv* junk transition. Iterating the construction further multiplies the number of different *Abs* quadruples that could reside in a U^n*niv* junk transition. Nevertheless, those U^n*niv* junk transitions, related via the appropriate retrieve, within and nevertheless relations to *Univ* junk transitions, *do* contain just one *Univ* junk transition, as the iteration of (5.9)-(5.11) reveals. More generally the junk transitions of U^n*niv* are embedded in those of U^{n+m}*niv*. These phenomena tend to suggest that junk transitions might somehow be marginal to the core business of Theorem 5.1. Could we possibly do without them? The answer is a qualified yes.

We note that the junk transitions came in useful in providing refinement targets for *Conc* transitions for which v was not related to w via $G \wedge P$. To do without junk transitions we must therefore exclude such v, w pairs from $K \wedge R$. This requires abandoning I/O-filtered refinements and moving to the richer notion of modulated refinements, in which the within (resp. nevertheless) relation can involve the before-states (resp. after-states), see [3, 4]. Focusing thus on just the well behaved v, w pairs, and adding sufficient extra hypotheses to ensure that the resulting relations plug together suitably (since the within and nevertheless relations will no longer automatically be total and surjective), a junk-free version of Theorem 5.1 can be carried through, the steps of its universal system being given by replacing $\Rightarrow$ in (5.3) with $\wedge$. The version we gave is slightly simpler, and also made the non-*Abs* operations refinable in a natural manner, a property we consider worthwhile.

7. An Example

Let us return to our running example and see what the universal construction means in this setting. The states of the *Univ* system are pairs (u, w) such that u is a set of NATs and w is a sequence of NATs, eg. $(\{1, 2\}, [2, 5, 3])$. Those states which serve as before-states of non-junk steps are such that w is a serialisation of u, for example $(\{1, 3, 4\}, [4, 1, 3])$. The non-junk steps of *Univ* in nonboundary cases are eg.:

$$(\{3, 4\}, [3, 4])\ \text{-}((1, 1), AddEl)\text{-›}\ (\{1, 3, 4\}, [4, 1, 3])$$

Note that (as illustrated) there is no necessity for the concrete component of this step to agree with any specific $AddEl_C$ step, given that we have suggested that $AddEl_C$ should act only by appending to the end of the sequence.

The boundary steps, in the case that $AddEl_C$ is a *skip*, look for example like:

$$(\{1\ldots10\}, [1\ldots10])\ \text{-}((11, 11), AddEl)\text{-›}\ (\{1\ldots11\}, [1\ldots10])$$

and we see clearly why the general product structure is needed for the universal state space; evidently $[1\ldots10]$ can never be a serialisation of $\{1\ldots11\}$.

Focusing now on the nonboundary case for simplicity, it is worth observing that regarding the abstract level information, all steps $(u, \pi(u))$ -$((i, i), AddEl)$-› $(u', \pi'(u'))$ are equivalent, where $u' = u \cup \{i\}$, and π, π' are arbitrary permutations of the standard orderings of u, u'. What are we to deduce from this?

Note that the notion of universality expressed in Theorem 5.1 is only up to interrefinability by total surjective relations. Ordinary set theoretic isomorphisms are total surjective relations, and noting that all the states that we are implicitly suggesting are equivalent are set theoretically isomorphic, suggests that there will be a reformulation of the universal construction which deals with *sets* of concrete states rather than individual ones as we used. This turns out to be true, and is an example of Theorem 5.1.(3) in action. The two forms of universal construction are equivalent up to interrefinability by total surjective relations. The given presentation is somewhat simpler, and the general situation will be explored more fully elsewhere.

Returning to our example, and taking boundary cases into account, the non-junk part of the universal system is equivalent to a modification of the abstract level alone, restricting the abstract transitions to ones satisfying $|u'| \leq 10$. This is of course obvious, and represents an ideal state of affairs, i.e. a situation in which all the extra detail that the original retrenchment introduced at the concrete level, may be absorbed into a modified model at the abstract level, as

discussed in the Introduction. Not all situations are necessarily ideal however, and the fact that in an arbitrary retrenchment, the within and concedes relations can feature arbitrary interdependencies between the levels, means that such a clean lifting of detail to the abstract level will not always be available without further restrictions.

8. General and Total Correctness

Many refinement techniques take account of the guaranteed termination properties of operations, leading to general and total correctness formalisms. In this section we assume that each *Abs* operation Op_A possesses a predicate $trm_{Op_A}(u, i)$ whose truth indicates the before-states and inputs where the outcome of the operation is guaranteed to be successfully terminating. Similarly for the other systems in our discourse. The termination PO arising from the original retrenchment is:

$$G(u, w) \wedge P_{Op}(i, k, u, w) \wedge trm_{Op_C}(w, k) \Rightarrow trm_{Op_A}(u, i) \quad (8.1)$$

Those for the factorising retrenchment and refinement are:

$$H(u, v) \wedge Q_{Op}(i, j, u, v) \wedge trm_{Op_U}(v, j) \Rightarrow trm_{Op_A}(u, i) \quad (8.2)$$

and

$$K(v, w) \wedge R_{Op}(j, k) \wedge trm_{Op_U}(v, j) \Rightarrow trm_{Op_C}(w, k) \quad (8.3)$$

Note that both (8.2) and (8.3) have $trm_{Op_U}(v, j)$ in the antecedents, highlighting the different directions of the *trm* dependencies in retrenchments and refinements. Given the relationships already established, it is not hard to see that the general solution of (8.2) and (8.3) for $trm_{Op_U}(v, j)$ is:

$$trm_{Op_U}(v, j) \Rightarrow (\forall\, w, k \bullet K(v, w) \wedge R_{Op}(j, k) \Rightarrow trm_{Op_C}(w, k)) \quad (8.4)$$

(From (8.4), (8.3) follows readily, while (8.2) requires noting also that K and R are total.) In particular, the solution is not unique, allowing an arbitrarily small $trm_{Op_U}(v, j)$. Consider now the analogous conditions for *Xtra*, whose *trm* must satisfy:

$$H\tilde{}(u, v\tilde{}) \wedge Q\tilde{}_{Op}(i, j\tilde{}, u, v\tilde{}) \wedge trm_{Op_X}(v\tilde{}, j\tilde{}) \Rightarrow trm_{Op_A}(u, i) \quad (8.5)$$

and

$$K\tilde{}(v\tilde{}, w) \wedge R\tilde{}_{Op}(j\tilde{}, k) \wedge trm_{Op_X}(v\tilde{}, j\tilde{}) \Rightarrow trm_{Op_C}(w, k) \quad (8.6)$$

This allows $trm_{Op_X}(v\tilde{}, j\tilde{})$ to be arbitrarily small too as, for the same reasons, it needs only to satisfy:

$$trm_{Op_X}(v\tilde{}, j\tilde{}) \Rightarrow (\forall\, w, k \bullet K\tilde{}(v\tilde{}, w) \wedge R\tilde{}_{Op}(j\tilde{}, k) \Rightarrow trm_{Op_C}(w, k)) \quad (8.7)$$

Now in a total correctness framework, one combines the implications arising from termination and simulation of steps into a single property, by conjoining all the antecedents and conjoining all the consequents, making a single implication. Letting *trm* be arbitrarily small thus weakens the impact of the joint simulation relation discussed above, in the limit making it completely trivial. In this case it is natural to focus on those factorising systems *Xtra* for which *trm* is the biggest possible; for these the outer implication in (8.7) becomes an equality and the refinement from *Xtra* to *Conc* becomes *trm*-preserving on trm_{Op_C}. This restriction allows us to make the outer implication of (8.4) an equality too. We find:

Proposition 8.1 In a total correctness framework, let the factorising systems *Xtra* of Fig. 1 all be *trm*-preserving on trm_{Op_C}. Then *Univ* may be endowed with the *trm* predicate:

$$trm_{Op_U}(v, j) = (\forall\, w, k \bullet K(v, w) \wedge R_{Op}(j, k) \Rightarrow trm_{Op_C}(w, k))$$

Proof. For the required *Univ* to *Xtra* refinement to work, we need to confirm additionally that with our assumptions, $K^{\circ}(v, v\tilde{}) \wedge R^{\circ}_{Op}(j, j\tilde{}) \wedge trm_{Op_U}(v, j) \Rightarrow trm_{Op_X}(v\tilde{}, j\tilde{})$.

To do this, we assume the antecedents of this implication, and call these the outer hypotheses. The consequent, $trm_{Op_X}(v\tilde{}, j\tilde{})$, is itself an implication (i.e. the consequent of (8.7)); we call its antecedents the inner hypotheses and assume these also.

To exploit the inner hypotheses, assume $v\tilde{}$, $j\tilde{}$ are such that $K\tilde{}(v\tilde{}, w) \wedge R\tilde{}_{Op}(j\tilde{}, k)$ holds for some w, k. Next, we exploit the outer hypotheses by assuming v, j are such that $K^{\circ}(v, v\tilde{}) \wedge R^{\circ}_{Op}(j, j\tilde{})$ holds, witnessed by the aforementioned w, k — this is always possible by (5.12) and (5.13) since K and R are surjective, and generates as a side effect $K(v, w) \wedge R_{Op}(j, k)$. Now we can use $trm_{Op_U}(v, j)$ from the outer hypotheses, itself a quantified implication, which enables us to infer $trm_{Op_C}(w, k)$. This in turn is the consequent under the quantification of $trm_{Op_X}(v\tilde{}, j\tilde{})$. We can now use the deduction principle, and then quantify over w, k to get our goal, $trm_{Op_X}(v\tilde{}, j\tilde{})$ itself. We are done. ☺

In a general correctness setting, there is a much looser coupling between the simulation properties and *trm* properties. In fact it is unhelpful to assume any interdependence at all. Accordingly we require a solution to (8.2) and (8.3) which is as general as possible in terms of the systems *Xtra* that we admit. Since (the *trm* properties of) these will now be unrestricted, the only solution for trm_{Op_U} will be the empty one. Now there is nothing to prove to deduce that the *Univ* to *Xtra* refinement works in the following.

Proposition 8.2 In a general correctness framework, *Univ* may be endowed with the *trm* predicate:

$$trm_{Op_U}(v, j) = \mathit{false}$$

Lack of space prevents us developing these full correctness formalisations more fully, but we have shown the essential ingredients out of which they are built. Furthermore Proposition 8.1 and Proposition 8.2 will apply equally well in the context of the junk-free formulations of step simulation indicated in Section 7.

9. Conclusions

In the preceding sections we gave a universal construction for decomposing a retrenchment development step into a canonical retrenchment followed by a refinement. We showed that this decomposition enjoyed a universal factorisation property, that its construction was idempotent (up to isomorphism of non-junk), and we indicated that there was a slightly more complicated junk-free version of the result along the same lines. We also briefly examined how guaranteed termination predicates fared in this setup, and indicated that the minimal and maximal solutions to the termination predicate for the universal system were respectively more appropriate for general and total correctness; these issues will be explored in greater detail elsewhere.

The intention was always for the universal system to reflect at the abstract level, the detail introduced at the concrete level. In the ideal situation this is achieved by building a more complex model within the abstract state and I/O spaces, and the example we discussed showed this, up to interrefinability. However, the fact that the within and concedes relations may contain nontrivial dependencies between abstract and concrete spaces, means that there will be cases where one cannot restrict purely to the abstract spaces without quantifying over the concrete spaces in some way. That is why the general construction had to involve both. Some of the issues in this area regarding quantification are discussed in [4], which deals with a different way of relating retrenchment and refinement, by looking for a refinement inside a retrenchment.

The construction we presented can be bootstrapped, to lift a retrenchment all the way to the top of a development path, as follows. The given construction yields a retrenchment from Abs to $Univ$ that we may describe as 'horizontal'. If Abs is itself the result of an earlier refinement step, then the composition of that refinement and the horizontal retrenchment yields another 'diagonal' retrenchment as in Fig. 1. The universal construction may then be reapplied to lift the retrenchment to a higher horizontal level, and so on. The end result may, if desired, be used for requirements validation, as suggested in the Introduction.

It is important to explore the interplay between retrenchment and refinement, in order to show that they are not techniques that are somehow in tension or in competition with each other, but rather ones that naturally complement one another. The result presented in this paper is merely one avenue that needed to be explored. Other results regarding retrenchments and refinements that abut each other in various ways will be the subject of further publications. And the aim of all this activity is to alleviate 'engineering unnaturalness'; to supply the developer with a rich suite of specification development techniques, not constrained by the tyranny of pure refinement.

References

1. Banach R., Poppleton M. Retrenchment: An Engineering Variation on Refinement. *in*: Proc. B-98, Bert (ed.), Springer, 1998, 129-147, LNCS **1393**. *See also*: UMCS Technical Report UMCS-99-3-2, `http://www.cs.man.ac.uk/cstechrep`
2. Banach R., Poppleton M. Retrenchment and Punctured Simulation. *in*: Proc. IFM-99, Taguchi, Galloway (eds.), Springer, 1999, 457-476.
3. Banach R., Poppleton M. Retrenchment, Refinement and Simulation. *in*: Proc. ZB-00, Bowen, Dunne, Galloway, King (eds.), Springer, 2000, 304-323, LNCS **1878**.
4. Banach R., Poppleton M. Sharp Retrenchment, Modulated Refinement and Simulation. Form. Asp. Comp. **11**, 498-540, 1999.
5. de Roever W.-P., Engelhardt K. Data Refinement: Model-Oriented Proof Methods and their Comparison. C.U.P., 1998.
6. Abrial J. R. The B-Book. C.U.P., 1996.
7. Wordsworth J. B. Software Engineering with B. Addison-Wesley, 1996.
8. Lano K., Haughton H. Specification in B: An Introduction Using the B-Toolkit. Imperial College Press, 1996.
9. Neilson D. S. From Z to C: Illustration of a Rigorous Development Method. PhD. Thesis, Oxford University Computing Laboratory Programming Research Group, Technical Monograph PRG-101, 1990.
10. 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. ISBN 82-90230-88-5, 1985.
11. Owe O. Partial Logics Reconsidered: A Conservative Approach. Form. Asp. Comp. **3**, 1-16, 1993.
12. Hayes I. J., Sanders J. W. Specification by Interface Separation. Form. Asp. Comp. **7**, 430-439, 1995.
13. Boiten E., Derrick J. IO-Refinement in Z. *in*: Proc. Third BCS-FACS Northern Formal Methods Workshop. Ilkley, U.K., 1998.
14. 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.
15. 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.
16. Liu S. Evolution: A More Practical Approach than Refinement for Software Development. *in*: Proc. ICECCS-97, 142-151, IEEE, 1997.