Filtering Retrenchments into Refinements

Richard Banach
School of Computer Science
University of Manchester
Manchester, M13 9PL, U.K.
banach@cs.man.ac.uk

John Derrick
Computer Science Department
University of Sheffield
Sheffield, S1 4DP, U.K.
j.derrick@dcs.shef.ac.uk

Abstract

Retrenchment is a weakening of model based refinement that enables many development steps not expressible by refinement to be formally described nevertheless. The greater flexibility of retrenchment comes at the price of much feebler guarantees as compared with refinement, and so the interplay between retrenchment and refinement can hope to offer the best of both worlds. The paper explores the strategy of filtering the information in a retrenchment to yield a refinement under a suitable notion of observation. A general construction is given that enables a retrenchment, with its intrinsic notion of observability, to be filtered to produce a refinement with its intrinsic notion of observability. A simple running example illustrates the theory.

1. Introduction

Model based refinement has long been a highly respected method for developing executable code from (not necessarily executable) specifications [1, 12, 11]. The idea is that the system requirements are captured in as abstract a way as is possible, using mathematical modeling techniques that need not bear any close relation to the capabilities of any realistic computing device. Thence, using techniques proved reliable by the theory of refinement, the abstract model is transformed into code in a manner that can be depended on. This constitutes the traditional view of refinement as software development approach, and the mathematical integrity of the approach is unimpeachable. However it imposes a strict relationship between those models that can be related by refinement, and the strictness of this relationship can prove to be overly constraining in practical situations; these can often feature matters of detail in the models to be related by the prospective refinement that prevent an actual refinement from obtaining. This is a well known state of affairs, and is usually dealt with by ad hoc means that depend on the example at hand. (Typical reposts range from changing the abstract model to fit the exigencies of refinement, to making the 'mathematical' concrete model subtly inconsistent with the actual executable code, to simply neglecting certain low level details in all the mathematical development.)

To overcome the above difficulty, retrenchment was introduced as a formal technique less demanding than refinement, see eg. [5, 6, 15, 8, 4, 13, 7]. The intention was that if two models (abstract and concrete), embodied a useful step in the development of a system, but could not be related by a refinement for some technical reason, they might nevertheless be productively related by retrenchment.

In the light of the difficulties with refinement, retrenchment was deliberately designed as a very liberal notion in order to avoid simply encountering more subtle difficulties with the new technique further down the line. In fact it can be proved that subject to minimal constraints, any two systems can be related by a retrenchment of some sort [4]. This gives extreme descriptive flexibility to the technique, at the price of offering weak predictive power to a generic retrenchment. One way of improving the latter, less desirable, aspect, is to search for ways in which retrenchment and refinement can complement one another. There are in fact various ways of attempting to formulate such a complimentarity, some of which have already been discussed in the literature [2, 13, 14]. This paper is concerned with one hithero unexplored possibility.

We exploit the extreme descriptive flexibility of retrenchment to give a faithful description of a development step, but then stand back in order to stipulate a perspective from which the retrenchment ought to be viewed. The objective is to find a perspective from which the retrenchment relationship appears to be a refinement. In order that this programme can be carried through, some leeway may be required from the refinement side, which comes via the flexibility of defining what is regarded as observable within a given refinement.

To paraphrase this, one can view refinement as being concerned with consistent observations between abstract and concrete systems. To whatever extent constitutes an observation of the required system, the user ought not to be able to tell if it is the abstract model or concrete model doing the work. While this on the one hand leads to the strictness of the refinement relationships already pointed out, on

the other it allows some flexibility, as that which is regarded as observable is to a degree dependent on the context in which the required system is intended to be used. Taking the same attitude to retrenchments, suggests that a retrenchment too ought to be viewed from an appropriate perspective, one moreover, that if possible, lends itself to a reinterpretation of the observations as having been obtained via a refinement. So our desire in this paper is to see retrenchments emerging as consistent with refinements, thus helping to validate the sometimes cited slogan that retrenchment is 'like refinement except round the edges' [5, 8]. It will turn out that the restricted observations in question can be viewed as filters on the abstract and concrete systems, and in this paper we set up a framework in which the relevant filters are investigated, and the appropriate refinements are extracted.

In order that suitable relationships may be set up between them, retrenchment and refinement need to be placed on the same semantic footing. Retrenchment is usually investigated in a pure transition system framework (see cited references); refinement is usually investigated in some variant of total correctness, the contract semantics and behavioural semantics of [11] being typical. For simplicity, we explore our problem using a partial correctness formulation of refinement, which is semantically compatible with the usual transition setup for retrenchment.

The rest of the paper is structured as follows. Section 2 sets up our notation for system models. Section 3 covers refinement and Section 4 summarises retrenchment. Section 5 examines the retrenchment notions of fragments and multifragments, while Section 6 gives a technical outline of the goal of the filtering construction. Section 7 contains the main technical results of the paper, filtering an arbitrary retrenchment into a particular type of refinement. Section 8 gives an example; Section 9 concludes.

2. System Models

We will be dealing with several system models that participate in a development process to be described below. There will be an abstract system *Abs* and a concrete one *Conc* (that will be related by retrenchment), and a filtered abstract system *FAbs* and a filtered concrete one *FConc* (that will be related by refinement). We give our notational conventions for the *Abs* system in detail, the others being similar.

Underpinning all the semantics for all the system models we consider in this paper will be a transition system for the system model in question. For the *Abs* system, this has a set of operations $\mathsf{Ops}_{\mathrm{A}}$, typical element $Op_{\mathrm{A}} \in \mathsf{Ops}_{\mathrm{A}}$.[1] The transitions / steps of Op_{A} are written u -(i, Op_{A}, o)-› u' or $Op_{\mathrm{A}}(u, i, u', o)$. Here $u \in \mathsf{U}$ is the before-state, $i \in \mathsf{I}_{Op_{\mathrm{A}}}$ the input, $u' \in \mathsf{U}$ the after-state, and $o \in \mathsf{O}_{Op_{\mathrm{A}}}$ the output. Initialisation of *Abs* is given via the operation / predicate $Init_{\mathrm{A}}(u')$.[2]

In this paper we adopt an 'on the fly' treatment of I/O (cf. [17], Ch. 16). Inputs are consumed and outputs are produced as the operation takes place, rather than these being concealed in sequences in some special part of the state. This obviates the need for finalisations, and reduces the number of formal proof obligations to be considered later. Supplementing these system details will be notions of observation, discussed at the meta level, which delineate what aspects of the system's activities are actually taken note of by the surrounding environment.

The other systems have similar notations. For the concrete system *Conc*, we have operation names $\mathsf{Ops}_{\mathrm{C}}$, data spaces $\mathsf{V}, \mathsf{J}_{Op_{\mathrm{C}}}, \mathsf{P}_{Op_{\mathrm{C}}}$, initialisation $Init_{\mathrm{C}}(v')$, and transitions v -(j, Op_{C}, p)-› v'. For *FAbs* we have operation names $\mathsf{Ops}_{F\mathrm{A}}$, data spaces FU, $\mathsf{FI}_{Op_{F\mathrm{A}}}$, $\mathsf{FP}_{Op_{F\mathrm{A}}}$, initialisation via $Init_{F\mathrm{A}}(u'_F)$, and transitions u_F -$(i_F, Op_{F\mathrm{A}}, o_F)$-› u'_F. For *FConc* we have operation names $\mathsf{Ops}_{F\mathrm{C}}$, data spaces FV, $\mathsf{FJ}_{Op_{F\mathrm{C}}}$, $\mathsf{FP}_{Op_{F\mathrm{C}}}$, initialisation $Init_{F\mathrm{C}}(v'_F)$, and transitions v_F -$(j_F, Op_{F\mathrm{C}}, p_F)$-› v'_F.

3. Refinement

In this section we make precise the notion of refinement we need. In line with the previous section, we assume abstract and concrete systems *FAbs* and *FConc*, with a bijective correspondence between abstract operations in $\mathsf{Ops}_{F\mathrm{A}}$ and concrete operations in $\mathsf{Ops}_{F\mathrm{C}}$ indicated informally via the subscripts.[3] We assume the state spaces of *FAbs* and *FConc* are related by a retrieve relation $G_F(u_F, v_F)$ on the states, and we furthermore assume for each $Op \in \mathsf{Ops}_{F\mathrm{A}}$, an input relation $I_{F,Op}(i_F, j_F)$, and an output relation $O_{F,Op}(o_F, p_F)$. The relations $G_F(u_F, v_F)$, $I_{F,Op}(i_F, j_F)$, $O_{F,Op}(o_F, p_F)$, are read as relations from abstract to concrete in each case, and we will assume that $I_{F,Op}(i_F, j_F)$ and $O_{F,Op}(o_F, p_F)$ are total and onto.

For the transition semantics (aka partial correctness semantics) of this paper, we need that the above components satisfy the following proof obligations. Firstly there is the initialisation PO:

$$Init_{F\mathrm{C}}(v'_F) \Rightarrow (\exists\, u'_F \bullet Init_{F\mathrm{A}}(u'_F) \wedge G_F(u'_F, v'_F)) \quad (3.1)$$

and secondly there is the operation PO which for a typical Op reads:

$$\begin{aligned} G_F(u_F, v_F) \wedge I_{F,Op}(i_F, j_F) \wedge Op_{F\mathrm{C}}(v_F, j_F, v'_F, p_F) \;\Rightarrow \\ (\exists\, u'_F, o_F \bullet Op_{F\mathrm{A}}(u_F, i_F, u'_F, o_F) \wedge \\ G_F(u'_F, v'_F) \wedge O_{F,Op}(o_F, p_F)) \end{aligned} \quad (3.2)$$

These are equivalent to the relational inclusions (3.3)-(3.4),

1. In most works on retrenchment, Op denotes the *name* of an operation, which is mapped to its transition or step relation, which is written stp_{Op}. In most work on refinement, notation Op refers to the transition relation itself. To avoid confusion in citing familiar facts on refinement, we use the refinement convention in this paper. The lack of explicit names for operations will not cause us difficulties.

2. When we speak about initial states, $Init_{\mathrm{A}}(u')$ is the predicate that characterises them; when we speak about execution sequences commencing with initialisation, $Init_{\mathrm{A}}(u')$ is the (nondeterministic) assignment of the state to any value satisfying the predicate.

3. This confirms that the 'A', 'C', 'FA', 'FC', subscripts on operation names and other values are meta level tags. We will suppress them when appropriate and it does not cause confusion.

in which inputs and outputs are free and related by $I_{F,Op}$ and $O_{F,Op}$ respectively, and intermediate state values are (existentially) bound. In the notation of [11]:

$$Init_{FC} \subseteq Init_{FA} \,;\, G_F \tag{3.3}$$

$$(G_F \wedge I_{F,Op}) \,;\, Op_{FC} \subseteq Op_{FA} \,;\, (G_F \wedge O_{F,Op}) \tag{3.4}$$

The above constitute the forward simulation conditions for refinement, but as is well known, these are merely sufficient conditions for a more general notion in which arbitrary concrete programs can be simulated by corresponding abstract ones. A concrete complete program can be seen as a sequential composition of executed steps:

$$Init_{FC} \,;\, Op_{FC,0} \,;\, Op_{FC,1} \,;\, \ldots \,;\, Op_{FC,n} \tag{3.5}$$

where each $Op_{FC,k}$ stands for a tuple $v_{FC,k}$ -($j_{FC,k}$, $Op_{FC,k}$, $p_{FC,k}$)-› $v'_{FC,k}$ from the corresponding concrete transition relation Op_{FC}, and for each $0 \le k \le n-1$, $v'_{FC,k} = v_{FC,k+1}$, i.e. abutting states are equal. An abstract complete program conformal to (3.5) is given by:

$$Init_{FA} \,;\, Op_{FA,0} \,;\, Op_{FA,1} \,;\, \ldots \,;\, Op_{FA,n} \tag{3.6}$$

so that the same operations occur in the same order. The concrete system is a refinement of the abstract one iff for all concrete $Init_{FC} \,;\, Op_{FC,0} \,;\, Op_{FC,1} \,;\, \ldots \,;\, Op_{FC,n}$, a conformal abstract $Init_{FA} \,;\, Op_{FA,0} \,;\, Op_{FA,1} \,;\, \ldots \,;\, Op_{FA,n}$ can be found such that:

(i) the states resulting from the $Init_{FA}$ and $Init_{FC}$ operations are related by G_F
(ii) for each $0 \le k \le n$, the inputs of $Op_{FA,k}$ and $Op_{FC,k}$ are related via $I_{F,Op,k}$
(iii) for each $0 \le k \le n$, the outputs of $Op_{FA,k}$ and $Op_{FC,k}$ are related via $O_{F,Op,k}$
(iv) the after-states of $Op_{FA,n}$ and $Op_{FC,n}$ are related by G_F (3.7)

which is also written:

$$\begin{array}{l} Init_{FA} \,;\, Op_{FA,0} \,;\, Op_{FA,1} \,;\, \ldots \,;\, Op_{FA,n} \\ \qquad\qquad \sqsubseteq \\ Init_{FC} \,;\, Op_{FC,0} \,;\, Op_{FC,1} \,;\, \ldots \,;\, Op_{FC,n} \end{array} \tag{3.8}$$

The refinement criteria (3.7) define the notion of observability appropriate to the transition semantics of refinement. This turns out to be the main notion of observability for refinement used in this paper.

Amongst other things, the clauses in (3.7) confirm that $u'_{FC,0}$, $v'_{FC,0}$ … $u_{FC,n}$, $v_{FC,n}$, i.e. the intermediate states visited by the two programs, are not observed. They are existentially quantified bound values, the intermediate values in a sequential composition of relations. As noted above (3.1) and (3.2) are stronger than (3.8), since they also demand that these corresponding intermediate states are related by G_F, allowing (3.8) to be established by induction.

4. Retrenchment

Given the previously described context, a retrenchment from *Abs* to *Conc* is defined by three facts. Firstly, the previous bijective correspondence between Ops_A and Ops_C indicated via the subscripts, is loosened to an inclusion, i.e. there can be concrete operations that do not correspond to any abstract operation. We assume the state spaces of *Abs* and *Conc* are related by a retrieve relation $G(u, v)$, and we further assume for each $Op \in \mathsf{Ops}_\mathrm{A}$, a within relation $P_{Op}(i, j, u, v)$, an output relation $O_{Op}(o, p; u', v', i, j, u, v)$, and a concedes relation $C_{Op}(u', v', o, p; i, j, u, v)$. Note that the latter two relations feature predominantly after-values, but before-values can be mentioned too if this is desired (the two kinds of values are separated by a (purely cosmetic) semicolon). The relations are to be read from abstract to concrete as before, and this time, no assumptions are made about the totality etc. of any of $P_{Op}(i, j, u, v)$, $O_{Op}(o, p; u', v', i, j, u, v)$, $C_{Op}(u', v', o, p; i, j, u, v)$. This assembly of components is required to verify proof obligations as follows.

The initial states must satisfy:

$$Init_\mathrm{C}(v') \;\Rightarrow\; (\exists\, u' \bullet Init_\mathrm{A}(u') \wedge G(u', v')) \tag{4.1}$$

as in (3.1), and for every corresponding operation pair Op_A and Op_C, the abstract and concrete step relations must satisfy the operation PO:

$$\begin{array}{l} G(u, v) \wedge P_{Op}(i, j, u, v) \wedge Op_\mathrm{C}(v, j, v', p) \;\Rightarrow \\ \quad (\exists\, u', o \bullet Op_\mathrm{A}(u, i, u', o) \wedge \\ \qquad ((G(u', v') \wedge O_{Op}(o, p; u', v', i, j, u, v)) \vee \\ \qquad C_{Op}(u', v', o, p; i, j, u, v))) \end{array} \tag{4.2}$$

At this point a subtlety emerges. Because v, j occur free in both the antecedent and consequent of the PO, (4.2) is not equivalent to a straightforward inclusion of compositions of relations from (u, i) to (v', p), as was the case for (3.2) and (3.4). We will return to this point later. The preceding defines the usual transition semantics for retrenchment. Note that there was no mention of sequences of steps or of observability, which are discussed in the next section.

5. Fragments and Multifragments

The natural counterpart for retrenchment of the complete program for refinement is the multifragment. A fragment is similar to a program (in (3.5) or (3.6)) but without insisting on the initialisation at the beginning, and (crucially), without hiding the intermediate state values via existential quantification in a composition of relations. So a fragment is genuinely a sequence, and not just a relation that arises from a sequence of composable steps. A multifragment is a sequence of fragments.

To understand why retrenchment requires a different notion, observe that in refinement, in the simulation relationship:

$$\begin{array}{l} G_F(u_F, v_F) \wedge I_{F,Op}(i_F, j_F) \wedge Op_{FC}(v_F, j_F, v'_F, p_F) \wedge \\ Op_{FA}(u_F, i_F, u'_F, o_F) \wedge G_F(u'_F, v'_F) \wedge O_{F,Op}(o, q) \end{array} \tag{5.1}$$

the fact that the same relation, G, relates both the before-states and after-states leads to an induction that establishes (3.8).

Lemma 5.1 Suppose that u_F -(i_F, Op_{FA}, o_F)-› u'_F and v_F -(j_F, Op_{FC}, p_F)-› v'_F are in simulation, i.e. they satisfy (5.1). Then v_F -(j_F, Op_{FC}, p_F)-› v'_F refines u_F -(i_F, Op_{FA}, o_F)-›

u'_F, i.e. (3.2) is verified for v_F -(j_F, Op_{FC}, p_F)-› v'_F.

Proof. It is clear that (5.1) supplies the existential witnesses required by (3.2). ☺

In retrenchment the analogue of (5.1) is the simulation relationship:

$$\begin{array}{l} G(u, v) \wedge P_{Op}(i, j, u, v) \wedge Op_{\mathrm{C}}(v, j, v', p) \wedge \\ \quad Op_{\mathrm{A}}(u, i, u', o) \wedge \\ ((G(u', v') \wedge O_{Op}(o, p; u', v', i, j, u, v)) \vee \\ \quad C_{Op}(u', v', o, p; i, j, u, v)) \end{array} \tag{5.2}$$

which is written (u -(i, Op_{A}, o)-› u') Σ^1 (v -(j, Op_{C}, p)-› v'), and of which we say that u -(i, Op_{A}, o)-› u' and v -(j, Op_{C}, p)-› v' are in simulation. Now, because we admit C_{Op}, which need not guarantee $G \wedge P_{Op}$ for the next step, there is no induction a priori, and simulation can break down at any point, perhaps to restart later (or even immediately from an unrelated state); and thus simulability is in general an intermittent and unreliable phenomenon.

With a particular retrenchment from *Abs* to *Conc* in mind, Σ(*Abs*) is the set of *Abs* steps for which a *Conc* step exists such that (5.2) holds. Similarly Σ(*Conc*) is the set of *Conc* steps for which an *Abs* step exists such that (5.2) holds. Σ(*Abs*) and Σ(*Conc*) are the simulable steps of *Abs* and *Conc*.

Formally, a fragment is a sequence of execution steps and a multifragment is a sequence of fragments with all values free[4]. If there is more than one fragment in a multifragment, only the last one may be infinite; all its predecessors must be finite. If $\mathcal{S}$ is a multifragment of *Abs* and $\mathcal{T}$ is a multifragment of *Conc*, we write $\mathcal{S}\,\Sigma\,\mathcal{T}$ iff there is a non-empty ordered bijection[5] between (all) the simulable steps of $\mathcal{S}$ and (all) the simulable steps of $\mathcal{T}$, such that for each pair of steps in the bijection the abstract and concrete step are in simulation; and furthermore, that each fragment in $\mathcal{S}$ and in $\mathcal{T}$ contains at least one such simulable step.

Example 1 Fig. 1 gives some illustrations of how the $\mathcal{S}\,\Sigma\,\mathcal{T}$ relationship can appear globally. In each of (a)-(d), $\mathcal{S}$ is shown above $\mathcal{T}$, and the steps of $\mathcal{S}$ and $\mathcal{T}$ are shown as arrows. Consecutive arrows which abut belong to the same fragment of the multifagment. A space between consecutive arrows separates consecutive fragments of the multifagment. In each of (a)-(d) the shaded rectangles and rhombi represent occurrences of the one step simulability relation Σ^1 between the steps of $\mathcal{S}$ and $\mathcal{T}$. Arrows which are not part of a shaded rectangle / rhombus are assumed to be non-simulable. Note that all that is demanded is an ordered bijection between the simulable steps. It is **not** required for example that: (i) *all* steps of $\mathcal{S}$ or $\mathcal{T}$ are in simulation (eg. (a), (b), (d)); (ii) $\mathcal{S}$ or $\mathcal{T}$ (viewed as two directed graphs) are connected, even via the simulation (eg. (a)-(d)); (iii) the simulable steps in $\mathcal{S}$ or in $\mathcal{T}$ are consecutive (eg. (a), (b), (d)); (iv) $\mathcal{S}$ or $\mathcal{T}$ start in an initial state (or with an *Init* step).

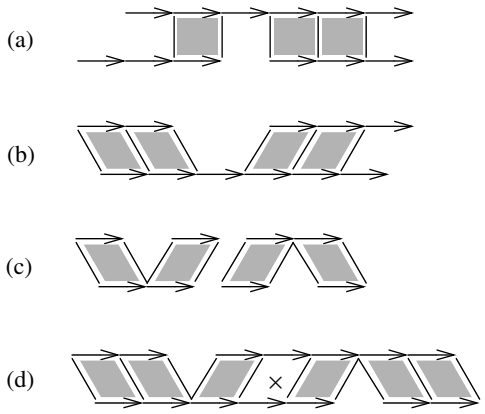

Fig. 1. Some examples of the $\mathcal{S}\,\Sigma\,\mathcal{T}$ relation.

The theory arising from simulability between multifragments is explored at length in [3]. There, system properties are defined as sets of multifragments which must be 'curt' and either 'Σ(*Abs*)-curt' or 'Σ(*Conc*)-curt' as appropriate. A multifragment is curt iff no two adjacent fragments of the multifragment can be concatenated to make a longer fragment (i.e. the last state of the former and the first state of the latter must differ). A multifragment of *Abs* is Σ(*Abs*)-curt iff the after-state of the last simulable step in any fragment of the multifragment, is different from the before-state of the first simulable step of the nearest following fragment that contains a simulable step. Σ(*Conc*)-curtness is defined similarly. The curtness conditions curtail excessive fragmentation within the multifragments considered, but are not of great significance for this paper, so their subsequent occurences may be overlooked on a first reading.

The $\mathcal{S}\,\Sigma\,\mathcal{T}$ relation captures the notion of observability appropriate to retrenchment. The fact that all the data occurring in $\mathcal{S}\,\Sigma\,\mathcal{T}$ is free, is in line with the idea that retrenchment, unlike refinement, must be used in a glass box manner. In the exploitation of retrenchment, with its toleration of the failure of simulability, it is vital that the participants are fully aware of the claims being made via the retrenchment POs, and just as importantly, what is not being claimed. In refinement, by contrast, it is at least permissible to delegate more responsibility to the guarantees that the refinement POs deliver, i.e. to take more of a black box approach.

Aside from the notion of simulation for retrenchment as described above, a more exacting variant will be useful to us in the constructions below. It is the strict simulation re-

4. All values are free as it is not a priori clear which steps may or may not be simulable in any given fragment. Moreover, the occurence both of outputs and inputs and also of state values in O_{Op} and C_{Op} (and P_{Op}), means that during a simulation of some steps by others, observing the outputs and inputs and not observing the intermediate state values, cannot be done without introducing extra, finegrained, quantification, which would need to be appropriately justified.

5. The ordering on the bijection must be a total ordering on the pairs of abstract and concrete steps whose projections on the abstract and concrete components are compatible with the respective orderings of steps in the abstract and concrete multifragments.

lationship:

$$G(u, v) \wedge P_{Op}(i, j, u, v) \wedge Op_{\mathrm{C}}(v, j, v', p) \wedge Op_{\mathrm{A}}(u, i, u', o) \wedge G(u', v') \wedge O_{Op}(o, p; u', v', i, j, u, v) \quad (5.3)$$

which is written $(u \text{ -}(i, Op_{\mathrm{A}}, o)\text{-›} u') \; \Sigma^{\mathrm{S1}} \; (v \text{ -}(j, Op_{\mathrm{C}}, p)\text{-›} v')$. This is much more like the simulation relationship of refinement, though not identical to it. It gives rise to a collection of notions as above, all written with an 'S' superscript thus: $\Sigma^{\mathrm{S}}(\mathit{Abs})$, $\Sigma^{\mathrm{S}}(\mathit{Conc})$, $\mathcal{S}\Sigma^{\mathrm{S}}\,\mathcal{T}$. Unlike (5.2), (5.3) *can* give rise to an induction that establishes a result like (3.8), which is the key to Section 7.

6. Filtering Retrenchments into Refinements

The principal objective of this paper is neatly represented in Fig. 2. On the left is a retrenchment from *Abs* to *Conc*, described by the relations G, P_{Op}, O_{Op}, C_{Op}. On the right is a refinement from *FAbs* to *FConc*, described by the relations G_F, $I_{F,Op}$, $O_{F,Op}$. Connecting the retrenchment to the refinement are two filtering functions; $AFil : \mathcal{CMF}_{\mathrm{A}} \to \mathcal{F}_{F\mathrm{A}}$ which maps $\mathcal{CMF}_{\mathrm{A}}$ (the curt and $\Sigma(\mathit{Abs})$-curt multifragments of *Abs*) to $\mathcal{F}_{F\mathrm{A}}$ (the fragments of *FAbs*), and $CFil : \mathcal{CMF}_{\mathrm{C}} \to \mathcal{F}_{F\mathrm{C}}$ which maps $\mathcal{CMF}_{\mathrm{C}}$ (the curt and $\Sigma(\mathit{Conc})$-curt multifragments of *Conc*) to $\mathcal{F}_{F\mathrm{C}}$ (the fragments of *FConc*). Some of the fragments of *FAbs* and *FConc* are extendable to complete programs, and the fact that intermediate states are free in (multi)fragments and bound in complete programs works in our favour. With this in place, any notion of observability which we now impose on the *FAbs* to *FConc* refinement, implicitly pulls back to a notion of observability for the *Abs* to *Conc* retrenchment.

Of course the diagram is intended to commute. Starting from a given retrenchment, we want to construct the refinement and filters to achieve this. Since the retrenchment defines just the left hand edge of the commuting square, it is to be expected that there will be many different ways of completing the square: some trivial, some non-trivial, some generic, some ad hoc, some applicable only to certain classes of retrenchments, etc.

One trivial solution would be to define both *FAbs* and *FConc* as systems with one state and no transitions, and with the obvious refinement between them, and to make *AFil* and *CFil* map all multifragments of *Abs* and *Conc* to the empty fragment; adopting a suitably trivial notion of observability for the refinement completes the picture. A marginally less trivial solution adds a single (skip) transition to *FAbs* and *FConc*, mapping all multifragments *Abs* and *Conc* to the one-step fragment. Obviously we want to do better than this. For example, a solution which insofar as possible preserves the extent to which the retrenchment is already a refinement in appropriate parts of its domain of definition, would be preferable. The remainder of this paper is concerned with describing one particular construction, but as mentioned above, it will be just one among many.

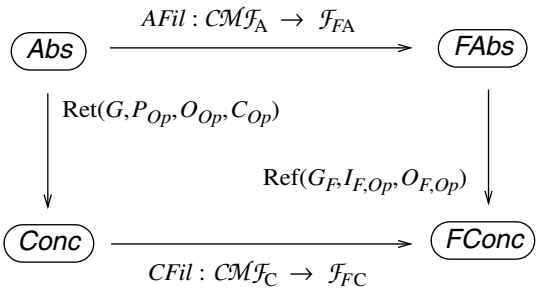

Fig. 2. Filtering refinements from retrenchments.

A desirable though not indispensable criterion for the filters *AFil* and *CFil* is that they should be homomorphisms of the multifragment structure, i.e. to respect concatenation of (multi)fragments to the extent permitted by the curt and $\Sigma(\ldots)$-curt restrictions. One way of ensuring this is to define them pointwise, i.e. to be (multi)fragment extensions of (perhaps partial) functions defined on individual steps. The no-transition solution above meets this criterion though the one-transition solution does not.

7. Retrenchment and Strictly Simulable Refinement

We take as given the retrenchment from *Abs* to *Conc*, and present a simple way of completing the construction of Fig. 2, based on the notion of strict simulation, (5.3).

The elements of the state spaces FU and FV of the systems *FAbs* and *FConc* are equivalence classes $[u] \in \mathsf{FU}$ and $[v] \in \mathsf{FV}$ of *Abs* and *Conc* states, given by the finest equivalence relations on U and V which satisfy the following collection of coupled properties:

u_0 -$(i_0, Op_{\mathrm{A},0}, o_1)$-› u_1 -$(i_1, Op_{\mathrm{A},1}, o_2)$-› u_2 -$(i_2, Op_{\mathrm{A},2}, o_3)$-› u_3 … … … u_{m-1} -$(i_{m-1}, Op_{\mathrm{A},m-1}, o_m)$-› u_m -$(i_m, Op_{\mathrm{A},m}, o_{m+1})$-› u_{m+1} is an *Abs* fragment such that u_0 -$(i_0, Op_{\mathrm{A},0}, o_1)$-› u_1 is strictly simulable and u_m -$(i_m, Op_{\mathrm{A},m}, o_{m+1})$-› u_{m+1} is strictly simulable, but all intervening steps (if any) are not strictly simulable
$\Rightarrow [u_1] = [u_m]$ (7.1)

u_{b} -$(i_{\mathrm{b}}, Op_{\mathrm{A,b}}, o_{\mathrm{b}})$-› u'_{b} is strictly simulable and there are u_{a}, v_{a}, v_{b} such that $[v_{\mathrm{a}}] = [v_{\mathrm{b}}]$ and $G(u_{\mathrm{a}}, v_{\mathrm{a}})$ and $G(u_{\mathrm{b}}, v_{\mathrm{b}})$ both hold
$\Rightarrow [u_{\mathrm{a}}] = [u_{\mathrm{b}}]$ (7.2)

v_0 -$(j_0, Op_{\mathrm{C},0}, p_1)$-› v_1 -$(j_1, Op_{\mathrm{C},1}, p_2)$-› v_2 -$(j_2, Op_{\mathrm{C},2}, p_3)$-› v_3 … … … v_{m-1} -$(j_{m-1}, Op_{\mathrm{C},m-1}, p_m)$-› v_m -$(j_m, Op_{\mathrm{C},m}, p_{m+1})$-› v_{m+1} is a *Conc* fragment such that v_0 -$(j_1, Op_{\mathrm{C},0}, p_1)$-› v_1 is strictly simulable and v_m -$(j_m, Op_{\mathrm{C},m}, p_{m+1})$-› v_{m+1} is strictly simulable, but all intervening steps (if any) are not strictly simulable
$\Rightarrow [v_1] = [v_m]$ (7.3)

v_{a} -$(j_{\mathrm{a}}, Op_{\mathrm{C,a}}, p_{\mathrm{a}})$-› v'_{a} and v_{b} -$(j_{\mathrm{b}}, Op_{\mathrm{C,b}}, p_{\mathrm{b}})$-› v'_{b} are strictly simulable and there are u_{a} and u_{b} such that $[u_{\mathrm{a}}] = [u_{\mathrm{b}}]$ and $G(u_{\mathrm{a}}, v'_{\mathrm{a}})$ and $G(u_{\mathrm{b}}, v_{\mathrm{b}})$ both hold
$\Rightarrow [v'_{\mathrm{a}}] = [v_{\mathrm{b}}]$ (7.4)

The provisions of (7.1)-(7.4) are illustrated in Fig. 3. Fig. 3.(a) shows (7.1) and (7.3), while Fig. 3.(b) shows (7.2) and

Fig. 3.(c) shows (7.4). The *ss* annotations on the transitions indicate strict simulability, and the heavy dashed arrows show the equivalences that are forced.

Given (7.1)-(7.4) we can define $G_F(u_F, v_F)$ as:

$$G_F(u_F, v_F) \;=\; (\exists\, u_a, v_b \bullet \\ u_a \in [u] = u_F \wedge v_b \in [v] = v_F \wedge G(u_a, v_b)) \qquad (7.5)$$

Let I^+_{Op} be the relation on $\mathsf{I}_{Op_A} \times \mathsf{J}_{Op_C}$ given by:

$$I^+_{Op}(i, j) \;=\; (\exists\, u, v \bullet P_{Op}(i, j, u, v)) \qquad (7.6)$$

Let O^+_{Op} be the relation on $\mathsf{O}_{Op_A} \times \mathsf{P}_{Op_C}$ given by:

$$O^+_{Op}(o, p) \;=\; (\exists\, u', v', i, j, u, v \bullet \\ O_{Op}(o, p; u', v', i, j, u, v)) \qquad (7.7)$$

and let $\mathsf{FO}_{Op_{FA}} = \mathrm{dom}(O^+_{Op})$ and $\mathsf{FP}_{Op_{FC}} = \mathrm{rng}(O^+_{Op})$; then we define $O_{F,Op}$ on $\mathsf{FO}_{Op_{FA}} \times \mathsf{FP}_{Op_{FC}}$ by:

$$O_{F,Op}(o_F, p_F) \;=\; (\exists\, o, p \bullet \\ o_F = o \wedge p_F = p \wedge O^+_{Op}(o, p)) \qquad (7.8)$$

The above are the ingredients of what will become a refinement between the *FAbs* and *FConc* systems whose transitions are given as follows:

u_F -(i_F, Op_{FA}, o_F)-› u'_F iff u -(i, Op_A, o)-› u'
and where:

$$u_F = [u],\ i_F = i \in \mathsf{FI}_{Op_{FA}},\ Op_{FA} = Op_A, \\ o_F = o \in \mathsf{FO}_{Op_{FA}},\ u'_F = [u'] \qquad (7.9)$$

v_F -(j_F, Op_{FC}, p_F)-› v'_F iff v -(j, Op_C, p)-› v'
is strictly simulable, and where:

$$v_F = [v],\ j_F = j \in \mathsf{FJ}_{Op_{FC}},\ Op_{FC} = Op_C, \\ p_F = p \in \mathsf{FP}_{Op_{FC}},\ v'_F = [v'] \qquad (7.10)$$

Finally $Init_{FA}([u']) \Leftrightarrow (\exists\, u \in [u'] \bullet Init_A(u))$, and $Init_{FC}([v']) \Leftrightarrow (\exists\, v \in [v'] \bullet Init_C(v))$.

Lemma 7.1 If the *Abs* and *Conc* steps u -(i, Op_A, o)-› u' and v -(j, Op_C, p)-› v' are in strict simulation, i.e. they satisfy (5.3), then there are associated *FAbs* and *FConc* steps u_F -(i_F, Op_{FA}, o_F)-› u'_F and v_F -(j_F, Op_{FC}, p_F)-› v'_F, given by (7.9)-(7.10), and they are in simulation, i.e. they satisfy (5.1).

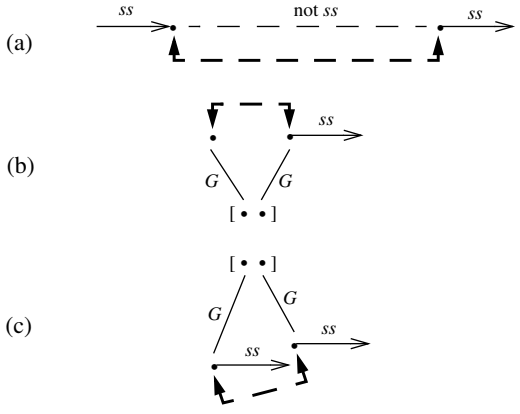

Fig. 3. Illustration of conditions (7.1)-(7.4).

Proof. It is sufficient to check that each of the criteria listed in (7.9)-(7.10) is valid, thus giving u_F -(i_F, Op_{FA}, o_F)-› u'_F and v_F -(j_F, Op_{FC}, p_F)-› v'_F, and that these *FAbs* and *FConc* transitions satisfy all the conjuncts of (5.1).

Regarding the former, for the abstract case, given u -(i, Op_A, o)-› u', it is clear that $u_F = [u]$, $Op_{FA} = Op_A$ and $u'_F = [u']$ are all trivial, while to check $i_F = i \in \mathsf{FI}_{Op_{FA}}$ it is enough to see that if u -(i, Op_A, o)-› u' and v -(j, Op_C, p)-› v' are in strict simulation, i.e. they satisfy (5.3), then $P_{Op}(i, j, u, v)$ is valid, and so $(\exists\, u, v \bullet P_{Op}(i, j, u, v))$ is valid, leading to $i \in \mathrm{dom}(I^+_{Op}) = \mathsf{FI}_{Op_{FA}}$; a similar derivation works for $o \in \mathrm{dom}(O^+_{Op}) = \mathsf{FO}_{Op_{FA}}$. Hence u_F -(i_F, Op_{FA}, o_F)-› u'_F is a transition of *FAbs*. We proceed similarly for the concrete v_F -(j_F, Op_{FC}, p_F)-› v'_F.

Regarding the latter, we note that all the component relations of the *FAbs* and *FConc* refinement are derived by existential quantification from the component relations of the *Abs* and *Conc* retrenchment ones. Thus, for the *FAbs* and *FConc* steps just established, and in the manner just noted, from $G(u, v)$ we derive $G_F(u_F, v_F)$, from $P_{Op}(i, j, u, v)$ we derive $I_{F,Op}(i_F, j_F)$, from $G(u', v')$ we derive $G_F(u'_F, v'_F)$, and from $O_{Op}(o, p; u', v', i, j, u, v)$ we derive $O_{F,Op}(o_F, p_F)$. This is enough for (5.1). ☺

Proposition 7.2 There is a refinement from *FAbs* to *FConc* given by (7.5), (7.7), (7.9). Moreover, for every complete *FConc* program, there is a complete *FAbs* program such that (3.8) holds.

Proof. First of all $I_{F,Op}$ and $O_{F,Op}$ are total and onto by construction of FI_{Op_A} and $\mathsf{FJ}_{Op_{FC}}$, and of FO_{Op_A} and $\mathsf{FP}_{Op_{FC}}$. It remains to consider the various POs.

For initialisation, if $Init_{FC}([v'])$ holds, there is a $v \in [v']$ such that $Init_C(v)$ holds. From the retrenchment initialisation PO, there is a u such that $Init_A(u)$ and $G(u, v)$ hold, so therefore we can derive $Init_{FA}([u'])$ and $G_F(u_F, v_F)$, where $u \in [u'] = u_F$.

For the operation PO, let $G_F(u_F, v_F) \wedge I_{F,Op}(i_F, j_F) \wedge Op_{FC}(v_F, j_F, v'_F, p_F)$ hold. From $Op_{FC}(v_F, j_F, v'_F, p_F)$ we infer a strictly simulable step v -(j, Op_C, p)-› v' with $v_F = [v], j_F = j$, etc. From strict simulability we deduce that there is some u -(i, Op_A, o)-› u' such that $(u$ -(i, Op_A, o)-› $u') \; \Sigma^{S1} \; (v$ -(j, Op_C, p)-› $v')$. By Lemma 7.1, the corresponding *FAbs* step u_F -(i_F, Op_{FA}, o_F)-› u'_F given by (7.9), is in simulation with v_F -(j_F, Op_{FC}, p_F)-› v'_F. Evidently u'_F and o_F provide the existential witness required for the refinement operation PO.

The claim about complete *FConc* and *FAbs* programs now follows by observing that the retrenchment initialisation PO is sufficient to prove the refinement initialisation PO, and then employing a simple induction. ☺

It remains to construct the filters $AFil : \mathcal{CM}_{\mathcal{F}_A} \to \mathcal{F}_{FA}$ and $CFil : \mathcal{CM}_{\mathcal{F}_C} \to \mathcal{F}_{FC}$ and to prove that Fig. 2 commutes.

Procedure 7.3 Let $\mathcal{S}$ be a mulifragment of *Abs*. We define $AFil(\mathcal{S})$ as the output of the following steps:

[1] Let $\mathcal{S}_1$ be the result of removing all not-strictly-simulable execution steps from $\mathcal{S}$ (keeping the remaining

ones in their original order). (N.B. This may increase the number of fragments in the multifragment by splitting some of the original fragments; other fragments may become empty.)

[2] Let $\mathcal{S}_2$ be the result of mapping all execution steps of $\mathcal{S}_1$ to the system *FAbs*. (Since only strictly simulable steps are left, all can be mapped, by Lemma 7.1.)

[3] Let $\mathcal{S}_3$ be the result of concatenating all consecutive concatenable fragments in $\mathcal{S}_2$, (so that $\mathcal{S}_3$ is curt).

[4] Let $\mathcal{S}_4$ be the first nonempty fragment of $\mathcal{S}_3$ if any, otherwise the empty one.

[5] Output $\mathcal{S}_4$.

The filter *CFil* on *Conc* mulifragments is defined by an identical procedure.

Example 7.4 Fig. 4 shows the effect of Procedure 7.3 on an example pair $\mathcal{S}$ and $\mathcal{T}$ which satisfy $\mathcal{S} \Sigma \mathcal{T}$. Fig. 4.(a) shows $\mathcal{S}_0$ and $\mathcal{T}_0$ with strictly simulable steps shown heavier, and with simulable but not strictly simulable steps, and nonsimulable steps shown light. The strict simulations are highlighted by shading. Fig. 4.(b) shows the result of erasing not-strictly-simulable steps from both $\mathcal{S}_0$ and $\mathcal{T}_0$. Fig. 4.(c) shows the result of concatenating the strictly simulable steps as far as possible (assuming in both abstract and concrete cases that the first four steps are concatenable).

The next proposition verifies that the concatenated strictly simulable steps always form up into equal length fragments in the two systems as in Fig. 4.(c).

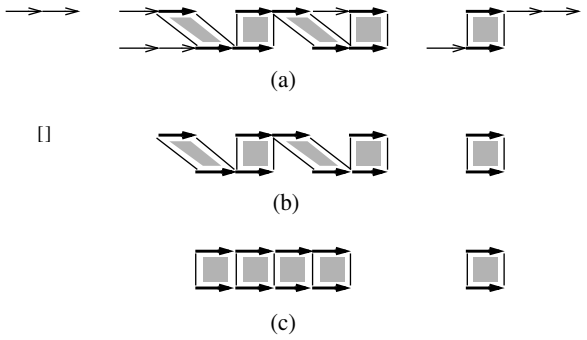

Fig. 4. The action of *AFil* and *CFil*.

Proposition 7.5 With the components defined, let $\mathcal{S}$ and $\mathcal{T}$ be abstract and concrete multifragments such that $\mathcal{S} \Sigma \mathcal{T}$ or $\mathcal{S} \Sigma^S \mathcal{T}$. Then Fig. 2 commutes in the sense that $CFil(\mathcal{T})$ is a refinement of $AFil(\mathcal{S})$. If $\mathcal{S} \Sigma^S \mathcal{T}$ then $CFil(\mathcal{T})$ and $AFil(\mathcal{S})$ are both non-empty. If $CFil(\mathcal{T})$ is extendable to a complete program $CFil(\mathcal{T})^{CP}$ by adding an initialisation, then $AFil(\mathcal{S})$ is similarly extendable to a complete program $AFil(\mathcal{S})^{CP}$, and $CFil(\mathcal{T})^{CP}$ is a refinement of $AFil(\mathcal{S})^{CP}$.

Proof. If $\mathcal{S} \Sigma^S \mathcal{T}$ then we have a non-empty bijection between the strictly simulable steps of $\mathcal{S}$ and $\mathcal{T}$, and if $\mathcal{S} \Sigma \mathcal{T}$ we have a bijection between the simulable steps of $\mathcal{S}$ and $\mathcal{T}$, which contains a (possibly empty) bijection between the strictly simulable steps of $\mathcal{S}$ and $\mathcal{T}$. When mapped to the systems *FAbs* and *FConc*, every pair of steps in strict simulation survives the filtering by Lemma 7.1 (hence verifying that $\mathcal{S} \Sigma^S \mathcal{T}$ implies $CFil(\mathcal{T})$ and $AFil(\mathcal{S})$ are both non-empty), and verifies the refinement simulation property (5.1), also by Lemma 7.1. It thus remains to check that the filters *AFil* and *CFil* agree on which pairs are to be retained in their entirety, and which are to be completely discarded; in particular that there are no dangling pairs, one element of which is discarded by *AFil* with its partner being retained by *CFil*, or vice versa.

For a contradiction, suppose that there was such a dangling pair. Let u_{m-1} -(i_{m-1}, $Op_{A,m-1}$, o_m)-› u_m and u_n -(i_n, $Op_{A,n}$, o_{n+1})-› u_{n+1} be abstract steps such that the former is the last strictly simulable abstract step retained by *AFil* and the latter is the first strictly simulable abstract step to be discarded by *AFil*. Then there can be no intervening strictly simulable abstract step between them. Let the corresponding concrete steps be v_{m-1} -(j_{m-1}, $Op_{C,m-1}$, p_m)-› v_m and v_n -(j_n, $Op_{C,n}$, p_{n+1})-› v_{n+1}, both retained by *CFil* by assumption; these can have no intervening strictly simulable step between them either. Now, because the two concrete steps are retained, and *CFil* discards all but the first fragment of the *FConc*-mapped multifragment, the *FConc*-mapped steps corresponding to v_{m-1} -(j_{m-1}, $Op_{C,m-1}$, p_m)-› v_m and v_n -(j_n, $Op_{C,n}$, p_{n+1})-› v_{n+1} must be concatenable, i.e. $v_{F,m} = [v_m] = [v_n] = v_{F,n}$. Therefore by (7.2), $u_{F,m} = [u_m] = [u_n] = u_{F,n}$, so the *FAbs*-mapped abstract steps are concatenable too, and so the second of them would not have been discarded, giving the contradiction. That $CFil(\mathcal{T})$ is a refinement of $AFil(\mathcal{S})$ now follows straightforwardly, as does the claim about extensions of $CFil(\mathcal{T})$ and $AFil(\mathcal{S})$ to $CFil(\mathcal{T})^{CP}$ and $AFil(\mathcal{S})^{CP}$. ☺

Given the straightforward way that the refinement has been extracted from the retrenchment, the appropriate notion of observability for the refinement is just the standard one, (3.7). Note furthermore that the above construction is in fact a homomorphism of the multifragment structure, in that it is defined pointwise via the strict simulability condition.

Given the observability for the refinement, (3.7), and the relevant one for the retrenchment, given by $\mathcal{S} \Sigma^S \mathcal{T}$ in this instance, how do we map the latter onto the former? The mechanism is a suitable existential quantification. Take $\mathcal{S}$. There must exist suitable (after- / before-) pairs of abstract states for successive strictly simulable abstract steps, equivalent in the relation generated by (7.1)-(7.4) and thus defining intermediate abstract states in *FAbs*, such that the following can be existentially quantified away:

- these intermediate states themselves,
- all non-strictly-simulable abstract steps,
- everything beyond the first fragment of concatenable *FAbs*-mapped steps,

leaving just the data that we naturally identify with the *FAbs* part of the refinement observation. Similarly for the

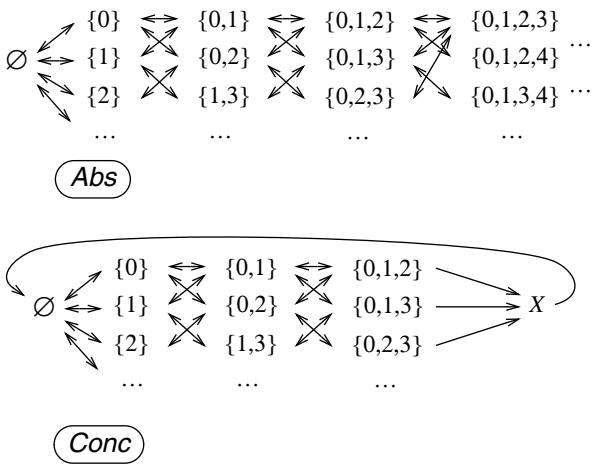

Fig. 5. Example *Abs* and *Conc* systems.

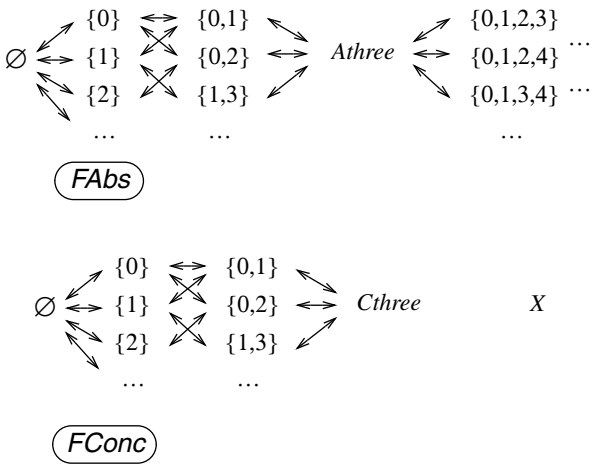

Fig. 6. Example *FAbs* and *FConc* systems.

concrete side. Clearly, expressing this in a properly formal manner would drag more or less the whole of the preceding material into the scope of the quantification, so we desist from elaborating the details.

8. Example

We now illustrate the construction with a very simple example. Consider the models *Abs* and *Conc* in Fig. 5, which we show directly as labelled transition systems. The states s of the abstract system *Abs* are subsets of NAT, initiallised to the empty set $\emptyset$. There are two operations, Put_{A} and Get_{A}. They can be defined by the Z schemas:

Put_{A}

$s : \mathcal{P}\,\mathsf{NAT}$
$n? : \mathsf{NAT}$

$n? \notin s$
$s' = s \cup \{n?\}$

Get_{A}

$s : \mathcal{P}\,\mathsf{NAT}$
$n! : \mathsf{NAT}$

$n! \in s$
$s' = s \setminus \{n!\}$

Fig. 5 shows this in essence. The rightward pointing part of each bidirectional arrow represents a Put_{A} going to a larger set; the leftward pointing part represents the corresponding Get_{A} going to a smaller set. In the model *Abs*, Put_{A} and Get_{A} are inverses of one another whenever their sequential composition is well defined.

The *Conc* model is similar but differs in detail. The states t are either subsets of NAT with cardinality at most 3, or a distinguished error state X. The initial state is the empty set as for *Abs*. There are three operations, Put_{C}, Get_{C} and $Reset_{\mathrm{C}}$, given by the following Z schemas:

Put_{C}

$t : \mathcal{P}\,\mathsf{NAT} \cup \{X\}$
$n? : \mathsf{NAT}$

$(t \neq X \land n? \notin t \land \# t < 3 \land t' = t \cup \{n?\}) \lor$
$(t \neq X \land n? \notin t \land \# t = 3 \land t' = X)$

Get_{C}

$t : \mathcal{P}\,\mathsf{NAT} \cup \{X\}$
$n! : \mathsf{NAT}$

$t \neq X \land n! : t$
$t' = t \setminus \{n!\}$

$Reset_{\mathrm{C}}$

$t : \mathcal{P}\,\mathsf{NAT} \cup \{X\}$

$t = X$
$t' = \emptyset$

Conc is clearly not a refinement of *Abs*. If it was, the initialisations at $\emptyset$ would have to be linked via the retrieve relation, by (3.1), and by following a suitable sequence of Put_{C}s, state X and cardinality 4 sets would also have to be related via the retrieve relation. Then the correctness condition for $Reset_{\mathrm{C}}$, i.e. equation (3.2), breaks down, regardless of which abstract operation we try to view $Reset_{\mathrm{C}}$ to be a refinement of.

Conc is, however, a retrenchment of *Abs*. With cardinality 3 sets retrieving to X this is trivially true, in that the

within, output and concedes relations for *Put* and *Get* can be given by the predicates true, true and false respectively, and we no longer need to relate $Reset_C$ to any abstract operation. The required POs are then easy to discharge.

The equivalence class construction can be seen as the least fixed point of the conditions (7.1)-(7.4). Since (7.1)-(7.4) are clearly monotonic, they can be interpreted as assignments that insist that certain proto-equivalence classes be merged when the one of the antecedents of (7.1)-(7.4) is true. The merging stops when there is no further change to the classes, and the least fixed point has been reached.

Viewed in this light, roughly speaking, (7.1) asks that two abstract states are made equivalent if one is reachable from the other via a non-simulable path between them, and they are in correspondence with concrete states. In our example, abstract state $s = \{0,1,2\}$ is non-trivially reachable from itself via a path consisting of bigger sets (eg. {0,1,2,3}); such a path is non-simulable. Furthermore {0,1,2} can also reach {4,5,6} via such a path. Therefore all abstract sets of cardinality 3 must be merged into the same class, called *Athree* in Fig. 6.

Moreover (7.4) forces a equivalence to be made between concrete states if they are both related to equivalent abstract states and there is a simulable transition from both of them. In our example this forces concrete sets of cardinality 3 to merge; the class is called *Cthree* in Fig. 6. Finally, (7.2) and (7.3) have no further effect.

The net result of the above is the transition systems *FAbs* and *FConc* illustrated in Fig. 6. In Fig. 6, for clarity, we have not shown the extra structure that turns an individual state from *Abs* or *Conc* into a singleton equivalence class state in *FAbs* and *FConc*, where this would be applicable. The transitions of *FAbs* and *FConc* are inherited in the expected way. Note that the transitions into and out of *X* have disappeared in *FConc*, as they are not strictly simulable. So *X* is unreachable in *FConc*. As we can see, the resulting diagrams represent the portions of the original behaviour which were in correspondence with one another. This was the desired effect of the filtering mechanism.

9. Conclusions

In this paper we have briefly sketched how retrenchments can be viewed as refinements. The approach to this has been to define a filtering mechanism, whereby two specifications which are related by retrenchment but not refinement are projected to specifications which are related by refinement. The projection forms equivalence classes of states based upon which portions of the specifications are in correspondence. The notion of correspondence is that provided by strictly simulable steps in the transition systems. Essentially the filtering mechanism tries to identify which portions of the two specifications have to be in correspondence with one another, and the resulting construction can then be related by inclusion of behaviour. We used a simple example to illustrate the process.

As indicated briefly in Section 6, a single retrenchment fixes only the left hand edge of Fig. 2. This leaves the other three sides of the square free to be constructed in many other ways. Let us mention some potential technical ingredients that could figure in such alternative constructions.

- The orientation of the horizontal arrows of Fig. 2 could be reversed. Thus instead of altering the retrenchment in order to reveal a refinement, as we did in this paper, one could look for the biggest refinement contained (in a suitable sense) inside the retrenchment. An interesting issue here is the extent to which the two approaches were dual (or perhaps adjoint) to one another.
- In connection with the preceding, one could explore more sensitive techniques than used in this paper, for ensuring refinability of individual steps extended to refinability of runs in their entirety.
- Instead of combining states into equivalence classes in order to mask inappropriate aspects of one or other state in the class, one could remove troublesome aspects of the systems involved, whether these were states or transitions, until a refinement was arrived at. Alternatively, one could consider adding sufficient new structure in a disciplined way so that a refinement emerged. Again there is an issue of duality or adjointness to explore.
- One could explore the possibility of broadening the notion of simulation employed in the basic construction. Instead of simply basing the construction on strict simulability, as above, the more general version could be used. This would entail paying attention to when primitive simulation elements (i.e. pairs of steps validating (5.2)) abutted, necessitating the use of the strict version, and when they didn't, opening the door to a modified version of the retrieve relation in the manufactured refinement.
- One could consider combinations of the above approaches, and the conditions needed in order that they may coexist in the same construction without conflicting with one another.

Possibilities such as the above are suggestive as regards the many possible ways that systems which are 'close', but not technically in a refinement, may be brought into a refinememt relationship — or to put it another way, how rigour might be brought to the informal slogan noted in the Introduction, that 'retrenchment is like refinement except round the edges'. The details of these will be explored in further papers.

Besides the preceding, a number of other authors have looked at issues related to those occurring here. In addition to retrenchment, work on approximations to refinement includes that on metric space approaches [9]. Work on dealing with extensions of behaviour includes alternative refinement orders as well as behavioural subtyping. For example, in a process algebraic context, the implementation relation EXT [10] has been defined to deal with situations whereby the behaviour of a specification has been extended (but is consistent on the unextended portion). This involves a quantification over the traces of the original specification in the consistency check (in fact, failures inclusion), allowing for additional traces to be present. Work on behavioural subtyping includes [16]. Here the concern is to define checks which allowed the behaviour to be extended with

new operations or actions, for example by allowing a buffer specification to be augmented with a delete operation. The concern is thus slightly different, in that our retrenchments can permit the concrete system to have behaviour which is an extension of abstract behaviour not only in new operations, but in the original operations as well. Our example bears this out.

References

[1] Back R., von Wright J. (1998); Refinement Calculus, A Systematic Introduction. Springer.

[2] Banach R. (2000); Maximally Abstract Retrenchments. *in*: Proc. IEEE ICFEM-00, 133-142.

[3] Banach R. (2003); Retrenchment and System Properties. `http://www.cs.man.ac.uk/~banach/some.pubs/Retrench.Props.ps.gz` *submitted*.

[4] Banach R., Jeske C. (2002); Output Retrenchments, Defaults, Stronger Compositions, Feature Engineering. `http://www.cs.man.ac.uk/~banach/some.pubs/Retrench.Def.Out.ps.gz` *submitted*.

[5] Banach R., Poppleton M. (1998); Retrenchment: An Engineering Variation on Refinement. *in*: Proc. B-98, Bert (ed.), LNCS **1393**, 129-147, Springer.

[6] Banach R., Poppleton M. (1999); Sharp Retrenchment, Modulated Refinement and Punctured Simulation. Form. Asp. Comp. **11**, 498-540.

[7] Banach R., Poppleton M. (2003); Retrenching Partial Requirements into System Definitions: A Simple Feature Interaction Case Study. Req. Eng. J. **8**, 266-288.

[8] Banach R., Poppleton M. (2005); Engineering and Theoretical Underpinnings of Retrenchment. `http://www.cs.man.ac.uk/~banach/some.pubs/Retrench.Underpin.ps.gz` *submitted*.

[9] Boiten E., Derrick J. (2005); Formal Program Development with Approximations. *in*: Proc. ZB-05, Treharne, King, Henson, Schneider (eds.), LNCS **3455**, 374-392, Springer.

[10] Brinksma E., Scollo G., Steenbergen C. (1987); LOTOS Specifications, their Implementations and their Tests. *in*: Proc. IFIP Protocol Specification, Testing and Verification (PSTV-6), Sarikaya, Bochmann (eds.), 349-360, Elsevier (North-Holland).

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

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

[13] Jeske C., Banach R. (2002); Minimally and Maximally Abstract Retrenchments. *in*: Proc. IFM-02, Butler, Petre, Sere (eds.), LNCS **2335**, 380-399, Springer.

[14] Jeske C. (2005); Algebraic Integration of Retrenchment and Refinement. PhD. Thesis, School of Computer Science, University of Manchester.

[15] Poppleton M., Banach R. (1999); Retrenchment: Extending the Reach of Refinement. *in*: Proc. IEEE ASE-99, 158-165.

[16] Wehrheim H. (2002); Checking Behavioural Subtypes via Refinement. *in*: Proc. FMOODS-02, Jacobs, Rensink (eds.), 79-93, Kluwer.

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