Fragmented Retrenchment, Concurrency and Fairness

R. Banach[a], M. Poppleton[a,b]

[a]Computer Science Dept., Manchester University, Manchester, M13 9PL, U.K.
[b]Faculty of Mathematics and Computing, Open University, Milton Keynes, MK7 6AL, U.K.
banach@cs.man.ac.uk , m.r.poppleton@open.ac.uk

Abstract

Retrenchment is presented in a simple relational framework as a more flexible development concept than refinement for capturing the early preformal stages of development, and briefly justified. Fragmented retrenchment permits the granularity of actions to decrease across a development step, many concrete steps retrenching a single abstract one. This generates the usual proliferation of interleavings of events at the concrete level. Event structures, particularly flow event structures, help to control these within the retrenchments of a single abstract step, while the concurrent reading of the fragmented retrenchment proof obligation permits acceptable interleavings of retrenchments of different steps. It is observed that retrenchment allows the convenient description of unfair behaviours when fairness is not guaranteed.

1. Introduction

Retrenchment [1, 2, 3, 4], is a technique that liberalises refinement for those cases where its proof obligations (POs) are too demanding to permit rigorous systems engineering. See the companion paper [5] for background and motivation. In this paper we examine some of the consequences for retrenchment, of allowing the granularity of individual actions to decrease, as often happens in refinements where the ultimate objective is a concurrent or distributed system; we call this kind of retrenchment fragmented retrenchment. We show that a fragmented retrenchment has a canonical factorisation into a (normal) retrenchment followed by a specific kind of refinement. Following the concurrency thread, we show how using an event based formalism can express the essence of fragmented retrenchment in a more compact manner. We also consider what happens when due to the smaller granularity of actions, constituent events from adjacent abstract actions interleave nontrivially, a possibility absent from normal retrenchment. Finally we examine fairness. The limit of a sequence of fair behaviours can be unfair. However retrenchment offers the prospect of capturing such limits in a way that appears natural with respect to higher levels of abstraction.

The rest of this paper is as follows. In Section 2 we introduce retrenchment, and a running example. In Section 3, we introduce fragmented retrenchment and its POs, and give the factorisation mentioned. In Section 4 we consider concurrency, the event based formulation of fragmented retrenchment, and interleaving. Section 5 considers fairness. Section 6 concludes.

2. Retrenchment

In this paper we work in a partial correctness framework. At the abstract level, we have a set of operations, Ops_A with typical element m_A, and our state space, input spaces, and output spaces, are U, $\mathsf{I}_{m_\mathrm{A}}$, $\mathsf{O}_{m_\mathrm{A}}$, respectively. Meta level values in U, $\mathsf{I}_{m_\mathrm{A}}$, $\mathsf{O}_{m_\mathrm{A}}$ will be written u, i, o respectively, with primes or subscripts or indeces to distinguish different values from the same space. (We will lighten the notation by not subscripting input and output values.) Transitions will be written as u -(i, m_A, o)-› u', where u and u' are the before- and after- states, i and o are the input and output, and m_A is the operation responsible for the transition. The totality of such transitions makes up the transition or step relation for m_A, $stp_{m_\mathrm{A}}(u, i, u', o)$. At the concrete level we have state, input and output spaces V, J, P, respectively, with values v, j, p, and similar conventions, except that we write operation name sets and operation names subscripted with C, eg. m_C. We assume each abstract operation m_A, has a corresponding concrete operation m_C, but there may also be other concrete operations, so that there is an injection from the set Ops_A to Ops_C, which associates m_A with m_C.

The relationship between abstract and concrete state spaces is given by the retrieve relation $G(u, v)$. The initialisation operation *Init* at abstract and concrete levels establishes G in corresponding after-states (as usual, the free variables are assumed implicitly universally quantified):

$$0 \longrightarrow (\varepsilon, Up_A, \varepsilon) \longrightarrow 3$$

$$0 \longrightarrow (\varepsilon, Up_C, Done) \longrightarrow 3$$
$$0 \longrightarrow (\varepsilon, Up_C, Error) \longrightarrow X$$

Fig. 1

$$Init_C(v') \Rightarrow (\exists\, u' \bullet Init_A(u') \wedge G(u', v')) \qquad (2.1)$$

Turning to the transition relation for operation m_A, beyond the retrieve relation G, we have the within relation $P_m(i, j, u, v)$, and concedes relation $C_m(u', v', o, p; i, j, u, v)$. The punctuation indicates that C_m is mainly concerned with after-values, but may refer to before-values too where necessary. These are combined into the retrenchment PO for steps which says that for each m_A:

$$\begin{aligned} &G(u, v) \wedge P_m(i, j, u, v) \wedge stp_{m_C}(v, j, v', p) \Rightarrow \\ &\quad (\exists\, u', o \bullet stp_{m_A}(u, i, u', o) \wedge \\ &\quad (G(u', v') \vee C_m(u', v', o, p; i, j, u, v))) \end{aligned} \qquad (2.2)$$

This PO affords considerable flexibility in relating different levels of abstraction, see [5] for a discussion. We consider a toy example; lack of space prevents consideration of something more realistic.

The abstract level is given by an initialisation operation $Init_A$, and one further operation Up_A. We have $\mathsf{U} = \{0, 3\}$, and $\mathsf{I}_{Up_A} = \varnothing = \mathsf{O}_{Up_A}$. $Init_A$ sets u to 0, and Up_A is given by 0 -(ε, Up_A, ε)-› 3, the only step in stp_{Up_A}. At the concrete level we have $Init_C$, and Up_C. The concrete state space is $\mathsf{V} = \{0, 3, X\}$, and $\mathsf{J}_{Up_C} = \varnothing$, $\mathsf{P}_{Up_C} = \{Done, Error\}$. $Init_C$ sets v to 0, and $stp_{Up_C} = \{0$ -(ε, Up_C, $Done$)-› 3, 0 -(ε, Up_C, $Error$)-› $X\}$, where ε is the empty input. The nontrivial steps are illustrated in Fig. 1.

The retrieve relation is given by the inclusion of U into V i.e. equality of abstract and concrete values, and the within relation for Up is $\mathsf{U} \times \mathsf{V}$ (i.e. we have a trivial within relation), where we also ignore the presence of the empty input spaces. There is some scope for choosing the concedes relation C_{Up}. The smallest possibility is:

$$C_1 = \{(u', v', p) \mid u' = 3 \wedge v' = X \wedge p = Error\}$$

while other possibilities include:

$$\begin{aligned} C_2 &= \{(u', v', p) \mid v' = X \wedge p = Error\} \\ C_3 &= \{(u', v', p) \mid (v' = X \wedge p = Error) \vee \\ &\qquad (v' = u' \wedge p = Done)\} \\ C_4 &= \{(u', v', p) \mid (p = Error \Rightarrow v' = X) \wedge \\ &\qquad (p = Done \Rightarrow v' = u')\} \end{aligned}$$

Note that $C_1 = C_2$ and $C_3 = C_4$ because of the smallness of the spaces involved. The different possibilities also illustrate some of what can be expressed using retrenchment in a more syntactically based framework. It is easy to check that the PO (2.2) holds for each of the C_i's.

3. Fragmented retrenchment and refinement

In (normal) retrenchment, one abstract step corresponds to one concrete step as we have seen. In fragmented retrenchment many steps at the concrete level correspond to a single abstract step; quite natural in many applications.

3.1. Fragmented retrenchment

With the usual notation, let $tt = [v$ -(j, $m_{C,0}$, p_1)-› v_1 -(j_1, $m_{C,1}$, p_2)-› v_2 …] be a finite or infinite sequence of concrete execution steps. We will write $ms_C = [m_{C,0}, m_{C,1}, \ldots]$ for the operation sequence of tt (i.e the sequence of operation names of tt). Then we further write v -(js, ms_C, ps)-› v' iff either:

(1) tt is finite, (v, js) are the initial state v and sequence of inputs $[j, j_1, \ldots, j_n]$ of tt, and (v', ps) are the last state and sequence of outputs $[p_1, p_2, \ldots, p_n]$ of tt, or:

(2) tt is infinite, (v, js) are the initial state v and length ω sequence of inputs of tt $[j, j_1, \ldots]$, (v', p) occur infinitely often in tt as the result state v' of a step with output p, and ps is the length ω+1 sequence $[p_1, p_2, \ldots, p]$ of outputs of tt with p appended.[1]

Let $\mathcal{T}$ be a set of (finite or infinite) sequences of concrete execution steps. Then we say that $stp_{\mathcal{T}}(v, js, v', ps)$ holds iff v -(js, ms_C, ps)-› v' for some tt in $\mathcal{T}$ with operation sequence ms_C. Thus the input and output spaces in which js and ps take values are:

$$\begin{aligned} \mathsf{JS} &= [\textstyle\bigcup_{m_C \in \mathsf{Ops}_C} \mathsf{J}_{m_C}]^{\leq \omega} \\ \mathsf{PS} &= [\textstyle\bigcup_{m_C \in \mathsf{Ops}_C} \mathsf{P}_{m_C}]^{\leq \omega+1} \end{aligned}$$

Fragmented retrenchment is stipulated by demanding firstly that both abstract and concrete levels have an $Init$ operation and that it obeys (2.1). Secondly, we demand that for each abstract operation m_A and corresponding concrete operation m_C, for each concrete step $t = v_*$ -(j_*, m_C, p_*)-› v_*', there is a set $\mathcal{T}_t$ of sequences of concrete steps, such that for each tt in $\mathcal{T}_t$, t occurs as some step in tt, and the following PO holds:

$$\begin{aligned} &G(u, v) \wedge P_m(i, js, u, v) \wedge stp_{\mathcal{T}_t}(v, js, v', ps) \Rightarrow \\ &\quad (\exists\, u', o \bullet stp_{m_A}(u, i, u', o) \wedge \\ &\quad (G(u', v') \vee C_m(u', v', o, ps; ms_C; i, js, u, v))) \end{aligned} \qquad (3.1)$$

where ms_C ranges over operation sequences that witness $stp_{\mathcal{T}_t}(v, js, v', ps)$. Thus whenever the retrieve and within relations hold and we can find a $stp_{\mathcal{T}_t}$ transition, there is an abstract step u -(i, m_A, o)-› u' such that for the after-states, either the retrieve relation is reestablished, or the concedes relation holds, taking into account the operation sequence and richer I/O of the concrete level. $\mathcal{T}_t$ is the set of amenable concrete sequences of t; and the union over concrete

1. We could easily admit (v', p) pairs that occur as limit points of infinite subsequences of tt too if the spaces of values carried a suitable topology, but we will ignore such richer possibilities in this paper.

steps t of the $\mathcal{T}_t$, is known as the set of amenable concrete sequences of the fragmented retrenchment.

We return to our small example to illustrate fragmented retrenchment. The abstract level will be as before. However this time the concrete level will feature a less powerful Up_C operation, capable only of incrementing by 2, so to remedy this there will also be an Inc_C operation, to increment by 1. Thus at the concrete level we will have $\mathsf{Ops}_C = \{Init_C, Up_C, Inc_C\}$ and $\mathsf{V} = \{0, 1, 2, 3, X\}$. The input and output spaces are given by $\mathsf{J}_{Up_C} = \emptyset = \mathsf{J}_{Inc_C}$ and $\mathsf{P}_{Up_C} = \{OK, Done, Error\} = \mathsf{P}_{Inc_C}$. $Init_C$ works as before, and the step relations for Up_C and Inc_C are:

$$stp_{Up_C} = \{0 \text{ -}(\varepsilon, Up_C, OK)\text{-}\rightarrow 2,\ 1 \text{ -}(\varepsilon, Up_C, Done)\text{-}\rightarrow 3,\ 2 \text{ -}(\varepsilon, Up_C, Error)\text{-}\rightarrow X\}$$
$$stp_{Inc_C} = \{0 \text{ -}(\varepsilon, Inc_C, OK)\text{-}\rightarrow 1,\ 1 \text{ -}(\varepsilon, Inc_C, OK)\text{-}\rightarrow 2,\ 2 \text{ -}(\varepsilon, Inc_C, Done)\text{-}\rightarrow 3\}$$

The within relation will be trivial as previously. For the concedes relation there are again many possibilities. Eg. we can take $C_i \times \mathsf{MS}$, where MS contains all relevant concrete operation sequences, and C_i is any of the C_i's of Section 2 (provided that in the definitions of the C_i's we replace occurrences of 'p' by '$\mathsf{last}(ps)$' to allow for the different output types here); any of these can be augmented by clauses that state that all non-last elements of ps are OK, or that are more specific about ms_C, etc.

There are several concrete execution sequences which may be related to the single abstract step 0 -(ε, Up_A, ε)-› 3. Fig. 2 illustrates. The abstract step itself is simulated by (i) and (ii), each consisting of an Inc_C and an Up_C. Also (iii) and (iv) lead to the error state X. To confirm that the POs are satisfied, we consider the elements of stp_{Up_C}, finding a suitable set $\mathcal{T}_t$ for each. Thus for 0 -(ε, Up_C, OK)-› 2, $\mathcal{T}_t$ is {(i), (iv)}. For 1 -(ε, Up_C, $Done$)-› 3, $\mathcal{T}_t$ is just {(ii)}, and for 2 -(ε, Up_C, $Error$)-› X, {(iii), (iv)} suffices. For all of these, it is easy to check that (3.1) holds for any of the C's described above. Of course, we also have a sequence of three concrete Inc_C steps that gets from 0 to 3, but since this does not contain an Up_C step, it falls outside the remit of the fragmented retrenchment. Discovering whether 'auxiliary operations' such as Inc_C are alone able to accomplish what would normally be done using the 'primary' operations inherited from the more abstract level, may be of vital interest, and open to investigation using model checking techniques [9, 10].

Now we note that the concrete level of Fig. 2 can be viewed as a refinement of the concrete level of Fig. 1, provided we adopt a suitable interpretation of 'refinement'. Thus for each concrete sequence in Fig. 2 there is a concrete step in Fig. 1 such that the states match appropriately, and the Fig. 2 outputs, filtered to discard OK, match the Fig. 1 outputs. In fact this is a wider phenomenon: a fragmented retrenchment can always be factored as a (normal) retrenchment followed by a suitable refinement. This result is a direct analogue in the fragmented domain of the main theorem of [5].

3.2. I/O-filtered fragmented refinement

An I/O-filtered fragmented refinement between an abstract and concrete system is given as follows. Let the abstract system be given by Ops_A, and the concrete system be given by Ops_C, with an injection from Ops_A to Ops_C as usual, and with a retrieve relation $G(u, v)$ between the abstract and concrete state spaces U and V. Let I_{m_A}, J_{m_C} and O_{m_A}, P_{m_C} be the abstract and concrete input and output spaces, and let two Ops_A-indexed families of total surjective relations be given, $\mathsf{Infilter}_m : \mathsf{I}_{m_A} \leftrightarrow \mathsf{JS}$ and $\mathsf{Outfilter}_m : \mathsf{O}_{m_A} \leftrightarrow \mathsf{PS}$, where JS and PS are the spaces constructed above from the J_{m_C} and P_{m_C}.

Suppose for each abstract operation m_A, for each concrete step $t = v_* \text{ -}(j_*, m_C, p_*)\text{-}\rightarrow v_*'$ of the corresponding concrete operation m_C, there is a nonempty set $\mathcal{T}_t$ of sequences of concrete steps, such that for each $tt = [v \text{ -}(j, m_{C,0}, p_1)\text{-}\rightarrow v_1 \text{ -}(j_1, m_{C,1}, p_2)\text{-}\rightarrow v_2 \ldots]$ in $\mathcal{T}_t$, t occurs as some step in tt, and we have:

$$G(u, v) \wedge \mathsf{Infilter}_m(i, js) \wedge stp_{T_t}(v, js, v', ps) \Rightarrow (\exists\, u', o \bullet stp_{m_A}(u, i, u', o) \wedge G(u', v') \wedge \mathsf{Outfilter}_m(o, ps)) \quad (3.2)$$

where js and ps are constructed from tt as previously. If in addition the initialisation PO (2.1) also holds, then we have an I/O-filtered fragmented refinement. We call $\mathcal{T}_t$ the set of amenable sequences of t. Furthermore the union of the $\mathcal{T}_t$ over concrete steps t, is the set of amenable sequences of the I/O-filtered fragmented refinement.

3.3. Universal factorisation

Now given our notation for abstract and concrete systems in a fragmented retrenchment, we define a canonical intermediate system as follows. Its set of operation names is Ops_U, in bijection with Ops_A, there being an m_U in Ops_U corresponding to each m_A in Ops_A. Its state space is the concrete state space V, and its input and output spaces (for every m_U) are the sets of sequences of concrete inputs and outputs constructed previously, i.e. JS and PS. The transitions of the intermediate system are given by letting stp_{m_U} be just the transitions $v \text{ -}(js, m_U, ps)\text{-}\rightarrow v'$ such that for some concrete step $t = v_* \text{ -}(j_*, m_C, p_*)\text{-}\rightarrow v_*'$, there is a step $v \text{ -}(js, ms_C, ps)\text{-}\rightarrow v'$ in stp_{T_t} coming from an amenable sequence tt containing t in $\mathcal{T}_t$ with operation sequence ms_C, in the fragmented retrenchment.

Theorem 3.1 Let there be a fragmented retrenchment with retrieve relation $G(u, v)$, within relations $P_m(i, js, u, v)$, and concedes relations $C_m(u', v', o, ps; ms_C; i, js, u, v)$, between an abstract system and a concrete system with concrete operations Ops_C and abstract operations Ops_A. Let Ops_U be

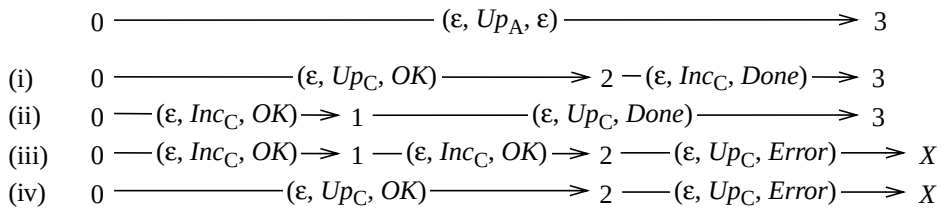

Fig. 2

the intermediate system constructed above. Then there is a (normal) retrenchment between the abstract and intermediate systems, and an I/O-filtered fragmented refinement between the intermediate system and the amenable sequences of the concrete system.

Proof. Between the abstract and intermediate systems, we define the same retrieve relation $G(u, v)$ as before, the same within relations $P_m(i, js, u, v)$, and concedes relations $(\exists\, ms_C \bullet C_m(u', v', o, ps; ms_C; i, js, u, v))$, interpreting js, v, ps as values in intermediate spaces. It is immediate that the initialisation PO (2.1) is satisfied. That the operation PO (2.2) holds, follows readily from the validity of the operation PO (3.1) for the fragmented retrenchment.

Turning to the I/O-filtered fragmented refinement between intermediate and concrete systems, we let the retrieve relation be the identity on V, and we take for all m_U in Ops_U, $\mathsf{Infilter}_m$ to be the identity on JS, and $\mathsf{Outfilter}_m$ to be the identity on PS. It is clear that the initialisation PO (2.1) holds, and that the operation PO (3.2) holds too for the amenable sequences of the concrete system. ☺

Note that our example above was not precisely an instance of this, since the inputs and outputs for the intermediate system (i.e. the concrete system of Fig. 1) were not as prescribed in the theorem. However the intermediate system of Theorem 3.1 has a useful initial property whereby if there is an extra system that factors a fragmented retrenchment into a normal retrenchment followed by an I/O-filtered fragmented refinement, the latter with filters In_m' and Out_m' say, then there is an I/O-filtered fragmented refinement from the intermediate to the extra system, which factors the extra I/O-filtered fragmented refinement through the canonical one, and it has unique filters ϕ_I and ϕ_O which factor In_m' and Out_m' through the canonical $\mathsf{Infilter}_m$ and $\mathsf{Outfilter}_m$, i.e.:

$$\mathsf{In}_m' = \mathsf{Infilter}_m \circ \phi_I \ , \ \mathsf{Out}_m' = \mathsf{Outfilter}_m \circ \phi_O$$

This is because the canonical filters are identities. In our running example, it is easy to check that the concrete system of Fig. 1 provides a factorisation of the fragmented retrenchment of Fig. 2 with the expected retrieves relation, trivial input filter In_m', and with output filter:

$$\mathsf{Out}_m'(o, ps) \ = \ (o = \mathsf{last}(ps))$$

3.4. Loose fragmented retrenchment

Up to this point, we have presented fragmented retrenchment by saying that each concrete step t determines a set T_t, which further determines a set of abstract m_A steps that validate the PO (3.1) for members of T_t. Let

$$\mathsf{T}_{m_A} \ = \ \cup\{\mathsf{T}_t \mid t \text{ is a step of } m_C\}$$

Fragmented retrenchment thus determines a relation from T_{m_A} to abstract m_A steps given by demanding that (3.1) holds. We define a *loose* fragmented retrenchment by giving directly a relation between a set of concrete sequences T_{m_A} given a priori, and abstract m_A steps, such that (3.1) holds with 'stp_{T_t}' replaced by '$stp_{\mathsf{T}_{m_A}}$'. The loose form is principally suited to situations where there is no naturally arising injection from Ops_A to Ops_C that one can exploit (though (2.1) must clearly also still hold). In the remainder of the paper, we use the loose form of the theory in Section 5, and in Section 4 we could use either variant of the theory, as we only need T_{m_A} sets, regardless of the means by which they were derived.

4. Concurrency

While the strategy of the last section is fine at a semantic level, the burden of explicitly recording all the amenable sequences in a mechanised syntactically based approach can quickly become overwhelming for sizeable examples. There is often a proliferation of concrete sequences when the granularity of atomic actions decreases, due to inconsequential interleavings of independent actions. The literature contains much work on this problem; see eg. papers in [11]. In this section we take an event based approach for a more efficient encoding of the amenable sequences.

4.1. Flow event structures

Events are defined as occurrences of actions, actions being certain fragments of the syntactic description of the system, assumed executed atomically. The events of a system can be organised into event structures to represent possible system behaviours in a wide variety of ways, as [12] shows. For us, the main property of any event structure formalism, is the ability to represent an arbitrary collection of event sequences; in fact provided we are willing to accept a certain amount of 'event duplication', almost any of the proposed

formalisms would do. We will focus on flow event structures [13], as they can translate fairly directly into reasonable syntactic representations.

A flow event structure (FES) consists of a set of events E, an irreflexive flow relation $\prec$ on E where $e_1 \prec e_2$ says that e_1 is a possible immediate cause of e_2, and a symmetric conflict relation # on E where $e_1 \# e_2$ says that neither may happen once the other has happened.

Let $\leq_X$ be the transitive closure of $\prec$ restricted to $X \subseteq E$. A configuration of the FES is a subset X of E such that:

(1) $\leq_X$ is acyclic,
(2) X contains no #-related pairs,
(3) for all $e \in X$, $\{e' \mid e' \in X, e' \leq_X e\}$ is finite,
(4) for all $e \in X$, if $e' \prec e$, then either $e' \in X$, or there is an $e'' \in X$ such that $e'' \prec e$ and $e'' \# e'$.

A configuration X moves to another configuration X' iff there is an element $e' \notin X$ such that $X' = X \cup \{e'\}$. Sequences of moves starting from the empty configuration are the way that FESs encode sequences of events. Let $\mathcal{T}$ be a set of sequences. We can encode $\mathcal{T}$ using an FES as follows. Let $E = \{(e, i, tt) \mid tt \in \mathcal{T}, i \in \mathsf{dom}(tt), tt(i) = e\}$. Let $(e_1, i_1, tt_1) \prec (e_2, i_2, tt_2)$ iff $tt_1 = tt_2$ and $i_1{+}1 = i_2$. Let $(e_1, i_1, tt_1) \# (e_2, i_2, tt_2)$ iff $tt_1 \neq tt_2$. This shows that an arbitrary $\mathcal{T}$ can be encoded by an FES but the construction is very inefficient. Normally we would exploit the flexibility of FESs to gain efficiency. We will see an example shortly.

Returning to fragmented retrenchment, our theory contains just enough syntax to allow actions to be defined. For us a (concrete) action will be a triple (j, m_C, p) with m_C a concrete operation name and j and p an input and output for m_C. Let $tt = [v \text{ -}(j, m_{C,0}, p_1)\text{-›} v_1 \text{ -}(j_1, m_{C,1}, p_2)\text{-›} v_2 \ldots]$ be a concrete execution sequence. Then $\mathsf{ev}(tt)$, the sequence of events of tt, is $[(j, m_{C,0}, p_1), (j_1, m_{C,1}, p_2) \ldots]$. Given a fragmented retrenchment, let:

$$\mathcal{E}_{m_C} = \{\mathsf{ev}(tt) \mid tt \in \mathcal{T}_{m_C}\}$$

Let $(E, \prec, \#)$ be a FES that encodes $\mathcal{E}_{m_C}$. Then we can equally well express the fragmented retrenchment by saying that the PO (3.1) holds in the obvious way for every maximal sequence of moves through $(E, \prec, \#)$. Returning to our running example, Fig. 3 shows the relevant FES. The solid directed arrows represent the elements of the flow relation and the bidirected broken arrows represent the conflict relation. It is not hard to see that the maximal sequences of moves through Fig. 3 correspond exactly to the sequences of Fig. 2. Note the additional tagging on the two (ε, Inc_C, OK) actions to make them into distinct events.

When actions are deterministic, as in our example, it can often be more economical to regard outputs of an event as derivable from the operation name, input and current state, the latter being derivable from the previous history of the execution. In such a case one can suppress output data from event definitions, leading to more compact event

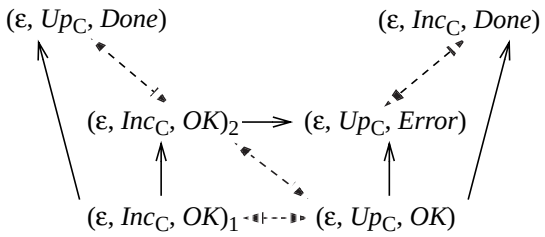

Fig. 3

structures. Fig. 4a shows the FES for our example when this is done (we suppress inputs too). Fig. 4b shows that further compaction that can occur if we allow our event structures to feature an *asymmetric* conflict relation, ⦊ instead of #, so that if a ⦊ b, then b cannot occur after a (⦊ is represented by a directed broken arrow in Fig. 4b). (In the formal definition of configurations for asymmetric FESs, (2) is replaced by 'no ⦊-cycles' and in (4) 'e'' # e'' is replaced by 'e'' ⦊ e''.) Note that the efficiency of the representations in Fig. 4 relies on permitting the same event to produce different outputs depending on preceding history. (For asymmetric conflict see eg. [14] which does not use a flow relation, or [12] which considers 'possible' events.)

4.2. Concurrent readings

Now we discuss a complication of fragmented retrenchment not possessed by the unfragmented theory. Consider the cartesian product of our running example with itself. The abstract state space is $\{0, 3\} \times \{0, 3\} = \mathsf{U}^X \times \mathsf{U}^Y$, with non-initial operations $Up^X{}_A$ and $Up^Y{}_A$, the former acting on the first component of the state and the latter on the second. The concrete system is likewise two copies of the former concrete system, with state space $\{0, 1, 2, 3, X\} \times \{0, 1, 2, 3, X\} = \mathsf{V}^X \times \mathsf{V}^Y$, and X- and Y- labelled operations. The retrieve relation is again inclusion. As the fragmented retrenchment between abstract and concrete systems, we take X- and Y- labelled versions of the retrenchment presented above, i.e. the previous relations extended by the universal relation on irrelevant coordinates. Thus, $Up^X{}_A$ has within relation $P_{Up^X} = P_{Up}(i^X, j^X, u^X, v^X) \times (\mathsf{I}^Y \times \mathsf{J}^Y \times \mathsf{U}^Y \times \mathsf{V}^Y)$ (still trivial overall of course), and concedes relation C_{Up^X} manufactured similarly from any candidate C_{Up}. Symmetrically for $Up^Y{}_A$. Suitable $\mathcal{T}_t$ sets are more of a problem however. Consider the following.

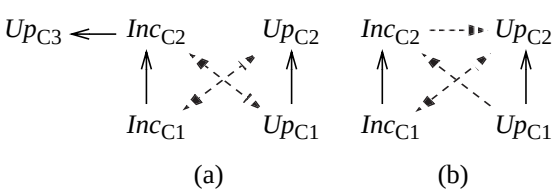

Fig. 4

The concrete system possesses the execution sequence tt_c = [(0, 0) -(ε, Up^X_C, *OK*)-› (2, 0) -(ε, Inc^X_C, *Done*)-› (3, 0) -(ε, Inc^Y_C, *OK*)-› (3, 1) -(ε, Inc^Y_C, *OK*)-› (3, 2) -(ε, Up^Y_C, *Error*)-› (3, *X*)]. This fits well with the retrenchment: the first two steps retrench an X abstract step, and the remainder retrench a Y abstract step. However the system also has the perfectly reasonable execution sequence in which the first two Inc^X_C and Inc^Y_C steps of the former are interchanged, namely tt_{nc} = [(0, 0) -(ε, Up^X_C, *OK*)-› (2, 0) -(ε, Inc^Y_C, *OK*)-› (2, 1) -(ε, Inc^X_C, *Done*)-› (3, 1) -(ε, Inc^Y_C, *OK*)-› (3, 2) -(ε, Up^Y_C, *Error*)-› (3, *X*)]. This is harder to reconcile with the retrenchment, as the first three steps, ostensibly retrenching an X abstract step, lead to a state (3, 1), outside the retrieve relation, and the last four steps, ostensibly retrenching a Y abstract step, start at the state (2, 0), also outside the retrieve relation. This being so, the retrenchment POs fail for these subsequences.

The cause of the problem, the interleaving, is evident. And when genuine concurrency gives the users of a system less control over the actual ordering of operation occurrences, one cannot guarantee that only sequences like tt_c will occur. Fragmented retrenchment must confront this.

We are going to introduce concurrent readings of the basic PO, which focus in a safe manner on just those parts of the state and other spaces that are vital for a particular step and its retrenchment, ignoring the rest. For this we need some terminology, starting with some remarks on syntax.

We assume a flat syntactic name space; no clashes between any identifiers. For any syntactic variable $\mathbf{x}$ (state or I/O), let X be its space of values. We call the pair $(\mathbf{x}, \mathsf{X})$ a basic component. We assume that our system is ultimately defined by syntactic means, so that the basic components lead to a fixed cartesian decomposition of all spaces of interest, eg. for our abstract state space where the state variables[2] are say $\mathbf{u}_1, \mathbf{u}_2 \ldots \mathbf{u}_k$ and which thus lead to basic components $(\mathbf{u}_1, \mathsf{U}_1)$, $(\mathbf{u}_2, \mathsf{U}_2)$ … $(\mathbf{u}_k, \mathsf{U}_k)$, we will have:

$$\mathsf{U} = \mathsf{U}_1 \times \mathsf{U}_2 \times \ldots \times \mathsf{U}_k$$

Usually an operation *m* affects some variables and not the remainder. Say *m* affects $\mathbf{i}_1 \ldots$, $\mathbf{u}_1 \ldots \mathbf{u}_j$, $\mathbf{o}_1 \ldots$, i.e. it affects *all* the inputs for *m*, *some* of the state and *all* the outputs; and it doesn't affect the rest of the state $\mathbf{u}_{j+1} \ldots \mathbf{u}_k$. Then the stp_m relation will be of the form:

$$stp_m = stp^\bullet_m \times stp^\circ_m \tag{4.1}$$

where $stp^\bullet_m$ will normally be nontrivial, and expresses the real work done by the step (including I/O), and stp°_m is an identity relation on the values of remaining state variables $\mathsf{U}_{j+1} \ldots \mathsf{U}_k$, (the identity expresses the interleaving perspective of our semantics). Thus $stp^\bullet_m$ will be nontrivial for (values in) the basic components $\mathsf{bas}(stp^\bullet_m) = \{(\mathbf{i}_1, \mathsf{I}_1), \ldots (\mathbf{u}_1, \mathsf{U}_1) \ldots (\mathbf{u}_j, \mathsf{U}_j), (\mathbf{o}_1, \mathsf{O}_1) \ldots\}$, and stp°_m will be an identity on (values in) the basic components $\mathsf{bas}(stp^\circ_m) = \{(\mathbf{u}_{j+1}, \mathsf{U}_{j+1}) \ldots (\mathbf{u}_k, \mathsf{U}_k)\}$. As we will not need $\mathsf{bas}(stp^\circ_m)$ below, we will abbreviate $\mathsf{bas}(stp^\bullet_m)$ to $\mathsf{bas}(m)$.

More generally, for any relation $R = R^\bullet \times R^\circ$ of the theory, (with R° being an identity or universal for 'unused' variables as appropriate), because *R* is ultimately described using the same syntactic elements, there will be similar collections, $\mathsf{bas}(R^\bullet)$ and $\mathsf{bas}(R^\circ)$, of basic components arising from the spaces of values of these elements. We will abbreviate $\mathsf{bas}(R^\bullet)$ to $\mathsf{bas}(R)$ as before.

Given such an *R* (eg. in particular, relations *G*, P_m, C_m of our theory), there will typically also be a finest cartesian decomposition of $R^\bullet$ (possibly trivial, but in any case reflecting the natural structure of the system), such that $R^\bullet = R_1 \times R_2 \times \ldots \times R_n$. We will assume that distinct R_i's are built from distinct collections of syntactic variables, so that for each R_i, there will be a distinct smallest collection of basic components, written as $\mathsf{bas}(R_i)$, arising from the spaces of values of the syntactic variables used to define R_i; so that R_i is a relation on $\mathsf{bas}(R_i)$.

When we have several relations R_a, R_b … R_l, then we define $\mathsf{bas}(R_a, R_b \ldots R_l)$ by:

$$\mathsf{bas}(R_a, R_b \ldots R_l) = \mathsf{bas}(R_a) \cup \mathsf{bas}(R_b) \cup \ldots \cup \mathsf{bas}(R_l) \tag{4.2}$$

In particular, this defines $\mathsf{bas}(m_a, m_b \ldots m_l)$ when we have several operations m_a, m_b … m_l.

Given relations *S* and *R*, with *R* decomposed as a cartesian product $R = R_1 \times R_2 \times \ldots \times R_n$, we define $R_{\mathsf{bas}(S)}$ by:

$$R_{\mathsf{bas}(S)} = R_{i_1} \times \ldots \times R_{i_q} \tag{4.3}$$

where $R_{i_1} \times \ldots \times R_{i_q}$ is the smallest cartesian factor of *R* with respect to the given decomposition such that $\mathsf{bas}(S) \subseteq \mathsf{bas}(R_{i_1} \ldots R_{i_q})$. Similarly when we have several relations $S_a, S_b \ldots S_l$, we let $R_{\mathsf{bas}(S_a, S_b \ldots S_l)}$ be the smallest cartesian factor $R_{i_1} \times \ldots \times R_{i_r}$ of *R* with respect to the given decomposition such that for all *j*, $\mathsf{bas}(S_j) \subseteq \mathsf{bas}(R_{i_1} \ldots R_{i_r})$.

In our product example, the decompositions are into X- and Y- labelled components; thus $stp_{Up^X_A} = stp^\bullet_{Up^X_A} \times stp^\circ_{Up^X_A}$. Here $stp^\bullet_{Up^X_A}(u, i, u', o) = stp_{Up_A}(\pi^X(u), i, \pi^X(u'), o)$ where π^X is the canonical projection of $u = (u^X, u^Y)$ onto its first component, and $stp^\circ_{Up^X_A} = \mathsf{Id}(\mathsf{U}^Y)$. Likewise for the retrieve relation $G^{XY}(u, v)$ we have the decomposition $G^{XY}(u, v) = \mathsf{Incl}(\mathsf{U}^X, \mathsf{V}^X) \times \mathsf{Incl}(\mathsf{U}^Y, \mathsf{V}^Y)$. As a consequence we have $G^{XY}{}_{\mathsf{bas}(Up^X_A)} = \mathsf{Incl}(\mathsf{U}^X, \mathsf{V}^X)$, a tidy state of affairs. (Note that there are no 'unused' variables in G^{XY}.) The *P* and *C* relations decompose equally conveniently into X- and Y- labelled components, as discussed above, and the remainder of the example is similar.

Now given a fragmented retrenchment, let $\mathsf{bas}(T_{m_A})$ be $\mathsf{bas}(m_A) \cup \mathsf{bas}(m_{C,a}, m_{C,b} \ldots m_{C,l})$ where $\{m_{C,a}, m_{C,b} \ldots m_{C,l}\}$ are the names of all the concrete operations that occur as elements of sequences in T_{m_A}. We say that the retrench-

2. We assume the same variables are used for before- and after- states.

ment is normal iff for all m_A:

$$\mathsf{bas}(T_{m_A}) = \mathsf{bas}(G_{\mathsf{bas}(T_{m_A})}, P_{m\ \mathsf{bas}(T_{m_A})}, C_{m\ \mathsf{bas}(T_{m_A})}) \qquad (4.4)$$

This says that the spaces of values (i.e. types) of all the operations that occur in T_{m_A} are just those in the relations that define the fragmented retrenchment for m_A and m_C. Observe that the prefix [$Up^X{}_C$, $Inc^Y{}_C$, $Inc^X{}_C$] and the suffix [$Inc^Y{}_C$, $Inc^X{}_C$, $Inc^Y{}_C$, $Up^Y{}_C$] of tt_{nc} above, do not satisfy this criterion, as the former mentions a Y-labelled component in the fragmented retrenchment of an X-labelled operation, and the latter mentions an X-labelled component in the fragmented retrenchment of a Y-labelled operation. Thus these (or indeed any strictly larger) fragments of tt_{nc} cannot be elements of $T_{Up^X{}_A}$ or $T_{Up^Y{}_A}$ respectively in a normal fragmented retrenchment. All this being aside from the fact that the operation PO fails for them too.

The preceding terminology gives us the means to speak about appropriate cartesian factors of the spaces of values of interest. To exploit it we need to dissect execution sequences of operation occurrences in a corresponding manner, in order to unpick the effects of interleaving.

Given a sequence *ee*, a window of *ee* is an unbroken sequence of consecutive elements of *ee*, indexed by a subrange of the indeces of *ee* between a lower and an upper limit $lw \le up$ inclusive, (we admit the cases of *up* a limit of the indeces of *ee*, and also of empty windows). For example, [5, 6, 7, 8, 9] is a window of the natural numbers.

A subsequence of a sequence *ee* is (as usual) an increasing sequence of elements of *ee* with possible breaks. For example, the even numbers are a subsequence of the natural numbers. Obviously 'every window is a subsequence, but not vice versa'. If *ff* is a subsequence or window of a sequence or window *ee*, then *ee* – *ff* is the obvious subsequence of *ee* containing all the elements of *ee* not in *ff*.

Now let *ee* be a concrete execution sequence of a normal fragmented retrenchment. Let *ss* be a window of *ee*; let m_A be an abstract operation and *t* be a step of m_C; let *tt* be an element of T_t. We define ss_{m_A} and tt_{m_A}, the $\mathsf{bas}(T_{m_A})$-projections of *ss* and *tt*, as the sequences of projections of the (*stp* relation) elements of *ss* and *tt* onto the basic components $\mathsf{bas}(T_{m_A})$. These projections are well defined because of our previous assumptions. Suppose there is an injective map λ from the indeces of tt_{m_A} into those of ss_{m_A} which defines a subsequence, and such that for all the indeces *i* of tt_{m_A}: $tt_{m_A}(i)$ and $ss_{m_A}(\lambda(i))$ have the same operation name, input value, and output value, these being the *i*'th elements of ms_C, *js*, and *ps*, respectively.

For each basic component χ in $\mathsf{bas}(T_{m_A})$, we define the window of χ in tt_{m_A}, $\mathsf{win}(\chi, tt_{m_A})$, to be the smallest window in tt_{m_A} such that $\chi \notin \mathsf{bas}(m_{\tilde{t}})$ for all steps $\tilde{t}$ in $(tt_{m_A} - \mathsf{win}(\chi, tt_{m_A}))$, where $m_{\tilde{t}}$ is the operation name of $\tilde{t}$; or empty if there is no step $\tilde{t}$ of tt_{m_A} with χ in $\mathsf{bas}(m_{\tilde{t}})$. We define $\mathsf{win}(\chi, ss_{m_A}, tt_{m_A})$, the window of χ in ss_{m_A} with respect to $\mathsf{win}(\chi, tt_{m_A})$, to be the smallest window of ss_{m_A} which includes $\lambda(\mathsf{win}(\chi, tt_{m_A}))$, where λ applied to a window yields the subsequence of elements at the indeces in its range. Window *ss* is said to be acceptable iff for every χ in $\mathsf{bas}(T_{m_A})$, if a step $\tilde{s}$ of ss_{m_A}, with operation name $m_{\tilde{s}}$, is in $(\mathsf{win}(\chi, ss_{m_A}, tt_{m_A}) - \lambda(\mathsf{win}(\chi, tt_{m_A})))$, then we have $\chi \notin \mathsf{bas}(m_{\tilde{s}})$. So far we have said that the types of any non-tt_{m_A} operations interleaved into the λ image of tt_{m_A}, are disjoint from those of the retrenchment of m_A. Refer to Fig. 5.

For each individual concrete state variable $\mathbf{v}_\alpha$ with basic component $\chi_\alpha = (\mathbf{v}_\alpha, V_\alpha) \in \mathsf{bas}(T_{m_A})$, let v_α be the before-value of $\mathbf{v}_\alpha$ in the first step of $\mathsf{win}(\chi_\alpha, ss_{m_A}, tt_{m_A})$, and let v'_α be the after-value of v_α in the last step of $\mathsf{win}(\chi_\alpha, ss_{m_A}, tt_{m_A})$, provided $\mathsf{win}(\chi_\alpha, ss_{m_A}, tt_{m_A})$ is nonempty. Let $\hat{v}$ be a tuple of the v_α, and $\hat{v}'$ a corresponding tuple of the v'_α. Let $\check{v}, \check{v}'$ be tuples of values for the concrete state variables $\mathbf{v}_\alpha$ with $\mathsf{win}(\chi_\alpha, ss_{m_A}, tt_{m_A})$ empty. So $(\hat{v}, \check{v})$ and $(\hat{v}', \check{v}')$ are the $\mathsf{bas}(T_{m_A})$ concrete before- and after- states.

We say that the window *ss* of *ee* contains a concurrent reading (via *tt* and λ) of a step of the normal fragmented retrenchment iff it is acceptable and the following holds:

$$\begin{aligned}
&(\forall\ \check{v}, \check{v}' \bullet G_{\mathsf{bas}(T_{m_A})}(u, (\hat{v}, \check{v})) \wedge \\
&\quad P_{m\ \mathsf{bas}(T_{m_A})}(i, js, u, (\hat{v}, \check{v})) \wedge \\
&\quad stp_{T_t\ \mathsf{bas}(T_{m_A})}((\hat{v}, \check{v}), js, (\hat{v}', \check{v}'), ps)) \Rightarrow \\
&\qquad (\exists\ u', o \bullet \forall\ \check{v}, \check{v}' \bullet stp_{m_A\ \mathsf{bas}(T_{m_A})}(u, i, u', o) \wedge \\
&\qquad\quad (G_{\mathsf{bas}(T_{m_A})}(u', (\hat{v}', \check{v}')) \vee \\
&\qquad\qquad C_{m\ \mathsf{bas}(T_{m_A})}(u', (\hat{v}', \check{v}'), o, ps; ms_C; \\
&\qquad\qquad\qquad i, js, u, (\hat{v}, \check{v}))))
\end{aligned} \qquad (4.5)$$

Note that the above interprets the previously given PO (3.1) by gathering the relevant before- and after- values asynchronously into the tuples $\hat{v}$ and $\hat{v}'$, rather than needing to have them all fixed at a specific point in time. Operations that affect values $\check{v}$ of concrete variables that are not needed by *tt* but are mentioned in *G*, *P*, *C*, may interleave *ss* provided that the PO is insensitive to such values and any changes they might undergo. Most importantly the failure of the retrieve relation to hold at crucial points is overcome. Evidently, the concept of concurrent reading brings many

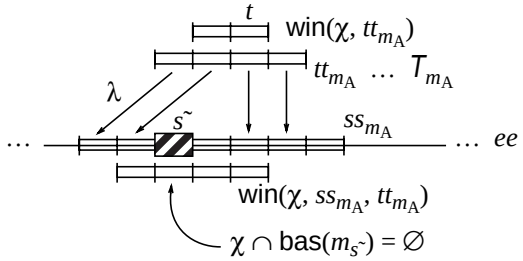

Fig. 5

more execution sequences into the fold encompassed by a fragmented retrenchment than is possible using the mechanisms of Section 3 alone, without suffering combinatorial explosion. This gives rise to notions of simulation via concurrent readings, whereby the simulable subsequences of a concrete execution are matched up with abstract steps via concurrent readings of the relevant operation POs. As the details are somewhat technical, we will explore them elsewhere.

For an example, the prefix $[Up^{X}_{C}, Inc^{Y}_{C}, Inc^{X}_{C}]$ and the $[Inc^{Y}_{C}, Inc^{X}_{C}, Inc^{Y}_{C}, Up^{Y}_{C}]$ suffix of tt_{nc} mentioned above, contain obvious concurrent readings of the normal fragmented retrenchment relating to X- and Y- labelled versions of sequences (i) and (iii) of Fig. 2 respectively.

5. Fairness and Retrenchment

It is well known that fair behaviours can be topologically problematic in the sense that a limit of fair behaviours may be unfair: notions of fairness typically arise as non-closed sets in naturally prevailing topologies for the system [15], and this can be awkward when continuity plays an important role in the system's semantic description. One can overcome this to some extent by working harder at the topology [16]. We show that loose fragmented retrenchments can provide a natural vehicle for describing what typically happens in such situations. The reason for this is that firstly, given a collection of fair and unfair behaviours of some system, since fair behaviours are 'good' and unfair behaviours are 'bad', there will always be some 'good' property that one can abstract away, thus spawning the upper level of abstraction seen in a retrenchment. Secondly, since the 'bad' behaviours exist at the lower level, the concedes relation is ideally suited to describing them.

Consider the following example. Suppose we modify the concrete system of Section 3 by removing the error state *X*, and altering the step 2 -(ε, Up_C, *Error*)-› *X* to 2 -(ε, Up_C, *Error*)-› 0. Now the set of execution sequences containing Up_C is described by the regular-like expression:

$$((Inc_C;Inc_C + Up_C);Up_C)^{\leq\omega} ; (Up_C;Inc_C + Inc_C;Up_C)$$

We can represent the situation using an iterative variation of the flow event structures we have been using, where by an iterative flow event structure we mean a flow event structure with return pseudoevent ↵. (The configurations of an iterative FES are sets of proper events with each event indexed by an iteration tag, initialised at 0, and whenever ↵ is encountered in generating the next configuration, no change in the configuration occurs but the current iteration tag is incremented, and subsequent events are generated by executing the iterative FES from the beginning.)

Fig. 6 is an iterative FES that corresponds to the behaviours just described in an obvious way. The infinite sequences of our example fail to reach a state retrieve-related to the abstract state 3. However states 0, 1, 2 are visited infinitely often, which we can conveniently express in a simple concedes relation:

$$C(u', v') = \{(u', v') \mid u' = 3 \wedge v' \in \{0, 1, 2\}\}$$

Now the fragmented retrenchment operation PO expresses exactly that either a concrete execution sequence conforms to the retrieve relation (and must therefore be finite), or it conforms to the concedes relation (and is necessarily infinite). The former case applies when the execution is fair; the latter only applies in the unfair case.

Now consider the general case. Suppose a system (which will become the concrete system of a loose fragmented retrenchment) possesses a set of fair behaviours F and a set of unfair ones U, with $F \cap U = \varnothing$. We assume the system is reasonable in the sense that each prefix *ee* of each unfair bahaviour *uu* in U has a fair extension *ff* in F. We also assume that there is some predicate Φ, in the concrete variables permitted to occur in concedes relations, that separates F and U, i.e. members of F verify Φ and members of U falsify Φ (were it not for the restriction on allowed variables, in general not severe, '$tt \in F$' itself would do for Φ). We use these components to build a canonical loose fragmented retrenchment.

For each fair concrete execution sequence in F with operation sequence ms_C, let $\underline{ms}_C$ be a fresh operation name. Ops_A will consist of all the $\underline{ms}_C$'s thus constructed. The abstract state space U will just be a synonym for the concrete state space V. Likewise I is a synonym for J and O is a synonym for P; so that IS and JS, and OS and PS are synonyms. The abstract steps are u -(i, $\underline{ms}_C$, o)-› u', where for some fair concrete execution sequence $tt = [v$ -(j, $m_{C,0}$, p_1)-› v_1 -(j_1, $m_{C,1}$, p_2)-› v_2 …] and with standard notation, u is a synonym for v, i is a synonym for js, u' is a synonym for v', o is a synonym for ps, and $\underline{ms}_C$ is the abstract name built from ms_C.

The loose fragmented retrenchment is founded on the retrieve relation $G(u, v)$, a synonym for the identity between U and V, the within relation P, a synonym for the universal relation $\mathsf{U} \times \mathsf{V} \times \mathsf{I} \times \mathsf{JS}$, and the concedes relation C given by:

$$(\neg\Phi \wedge (\exists\, n \bullet i{\uparrow}n = js{\uparrow}n \wedge o{\uparrow}n = ps{\uparrow}n))$$

The first term of C expresses the unfairness of the executions in U, while the second term expresses the fair prefix

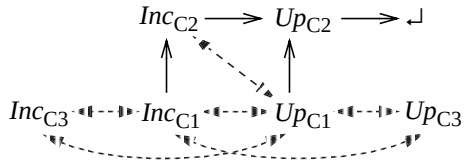

Fig. 6

property. It is clear that the PO (3.1) holds in the loose sense, with concrete executions $F \cup U$ related to abstract steps made from F. A strong reason for not attempting to construct a fragmented retrenchment in the normal sense, is that one common fairness property Φ, says that 'operation m_C occurs in the execution, given that it is sufficiently enabled during the execution'. In this scenario an abstract step with operation name named after m_C makes sense for a *tt* validating such a Φ. However the naturally related unfair concrete sequences, are characterised precisely by the absence of m_C from ms_C, making the choice of an operation name in ms_C to relate to m_C in the fragmented retrenchment problematic. Finally we note that our canonical loose fragmented retrenchment possesses initial properties similar to those in Theorem 3.1, which we do not elaborate for lack of space.

6. Conclusions

Accepting that refinement is not always ideal the higher you move up the modelling/specification hierarchy, a position discussed at greater length in [1] and [5], retrenchment was introduced to enable useful high level precursor models to be incorporated into a formal development process. In its original form, retrenchment preserved the granularity of actions, which is the most appropriate for introducing complexity into individual operations. In this paper, we have explored the consequences of relaxing this restriction, giving fragmented retrenchment. To keep the connection with the original notion as strong as possible, we formulated fragmented retrenchment in terms of a relation between single abstract steps and sequences of concrete steps. This led to a canonical factorisation of a fragmented retrenchment into a normal retrenchment followed by an I/O-filtered fragmented refinement. Although this means that strictly speaking, fragmented retrenchment can be done away with, doing so may not be natural from an engineering standpoint, and retrenchment is nothing if not intended for the alleviation of 'engineering unnaturalness'.

The sequence-oriented definition of fragmented retrenchment leads immediately to familiar combinatorial explosion problems arising through interleaving: firstly *within* the fragmented retrenchment of a single operation, and secondly *between* fragmented retrenchments of distinct operations. For the former, we saw that partial order methods could help, and we made use of flow event structures in particular. For the latter, we introduced concurrent readings of the operation PO, which overcame the point that interleaving can cause the retrieve relation to fail at the two ends of a retrenching sequence of concrete steps. This preserved a firm connection with the original mechanisable PO, and we were able to retain the simplicity and economy of the original description, since in a concurrent reading, a single sequence of the retrenchment can represent a potentially unbounded number of its interleavings with other activities.

Finally we considered fairness, and showed that the concedes relation of fragmented retrenchment conveniently describes the unfair executions, when a system can exhibit both fair and unfair ones. For this we used the loose form of fragmented retrenchment, which is particularly useful when we do not have one specific concrete step in a retrenching sequence *tt* to naturally identify with the abstract operation.

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*: Tech. Rep. 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.), 457-476, Springer, 1999.
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. Banach R. Maximally Abstract Retrenchments. *this volume*
6. Clarke E. M., Burch J. R., Dill D. L., McMillan K. L., Hwang J. Symbolic Model Checking: 10^{20} States and Beyond. Inf. and Comp. **98**, 142-170, 1992.
7. McMillan K. L. Symbolic Model Checking. Kluwer, 1993.
8. Proc. CAV-90, Clarke. Kurshan (eds.), Springer, 1990, LNCS **531**.
9. Pinna G. M., Poigne A. On the nature of Events: Another Perspective in Concurrency. Theor. Comp. Sci., **138**, 425-454, 1995.
10. Boudol G. Flow Event Structures and Flow Nets. *in*: Semantics and Systems of Concurrent Processes, Proc. LITP-90, Guessarian (ed.), Springer, 1990, 62-95, LNCS **469**.
11. Baldan P., Corradini A., Montanari U. An Event Structure Semantics for P/T Contextual Nets: Asymmetric Event Structures. *in*: Proc. FOSSACS-98, Nivat (ed.), Springer, 1998, 63-80, LNCS **1378**.
12. Kwiatkowska M. Z. On Topological Charecterization of Behavioural Properties. *in*: Topology and Category Theory in Computer Science. Reed, Roscoe, Wachter (eds.), 153-177, O.U.P., 1991.
13. Priese L. Approaching Fair Computations by Ultra Metrics. *in*: Semantics and Systems of Concurrent Processes, Proc. LITP-90, Guessarian (ed.), Springer, 1990, 420-433, LNCS **469**.