A Multiply Fibred Automaton Semantics for IWIM

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

F. Arbab
Software Engineering Dept., CWI, Kruislaan 413, 1098 SJ Amsterdam, Netherlands
farhad@cwi.nl

G. A. Papadopoulos
Computer Science Dept., University of Cyprus, 75 Kallipoleos St., Nicosia, Cyprus
george@cs.ucy.ac.cy

J. R. W. Glauert
School of Information Systems, University of East Anglia, Norwich, NR4 7TJ, U.K.
J.Glauert@sys.uea.ac.uk

Abstract. The drawbacks of programming coordination activities directly within the applications software that needs them are briefly reviewed. Coordination programming helps to separate concerns, making complex coordination protocols into standalone entities; permitting separate development, verification, maintenance, and reuse. The IWIM coordination model is described, and a formal automata theoretic version of the model is developed, capturing the essentials of the framework in a fibration based approach. Specifically, families of worker automata have their communication governed by a state of a manager automaton, whose transitions correspond to reconfigurations. To capture the generality of processes in IWIM systems, the construction is generalised so that process automata can display both manager and worker traits. IWIM systems possess a large number of algebraic properties, a range of which are described. The relationship with other formalisations of the IWIM conception of the coordination principle is explored.
Keywords. Coordination, IWIM, Automata, Fibration.

1 Introduction

The massively parallel systems that can be built today require programming models that explicitly deal with the concurrency of cooperation among large numbers of entities in a single application. Today's concurrent applications typically use ad hoc templates to coordinate the cooperation of their components, and this is symptomatic of a lack of proper coordination frameworks for describing complex cooperation protocols in terms of simple primitives and structuring constructs.

In most real applications, there is no paradigm in which we can systematically talk about cooperation of active entities, and in which we can compose cooperation scenarios such as client-server, workers pool, etc., out of a set of more basic concepts. Consequently, applications programmers must deal directly with the lower-level

communication primitives that instantiate the cooperation model of a concurrent application. These primitives are generally scattered throughout the source code, interspersed with non-communication application code, and the cooperation model never manifests itself in a tangible form. Thus it is not an identifiable piece of source code that can be designed, developed, debugged, maintained, and reused, in isolation from the rest of the application. This inability to deal with the cooperation model of a concurrent application explicitly, contributes to the difficulty of developing working concurrent applications containing large numbers of actively cooperating entities.

Despite the fact that the implementation of complex protocols is often the most difficult part of a development, the end result is typically so nebulous that it cannot be recognized as a commodity in its own right. This makes maintenance and modification of the cooperation protocols much more difficult than necessary, and their reuse next to impossible.

The two most popular models of communication within highly concurrent applications are shared memory and message passing. In the shared memory model, interprocess synchronisation primitives play the dominant role, with interprocess communication subordinate, whereas in the message passing model, interprocess communication is dominant, and synchronisation subordinate. The latter makes the message passing model somewhat more flexible than the shared memory model and, therefore, it is the dominant model used in concurrent applications. However, both paradigms are too low-level to serve as a proper foundation for systematic construction of cooperation protocols as explicit, tangible pieces of software.

Such observations have led in recent years to an upsurge in activity in so-called coordination frameworks and languages. An early survey is [Malone and Crowston (1994)] which characterisies coordination as an emerging discipline. Various approaches with roots in eg. the actor model [Agha (1986)], or in logic programming [Shapiro (1989)], were instrumental in establishing coordination as an independent discipline. See [Ciancarini and Hankin (1996), Garlan and Le Metayer (1997), Papadopoulos and Arbab (1998), Ciancarini and Wolf (1999), Porto and Roman (2000), Omicini (2002)] for representative contemporary work. A number of higher level perspectives have emerged. Among these are the tuple based approaches such as Linda [Gelernter (1985), Carriero and Gelernter (1989)], and by contrast, the connection control based approaches amongst which we find the IWIM model. It is with this model that this paper is concerned.

The rest of this paper contains the following. In Section 2 we survey the IWIM model informally. With this motivation covered, in Section 3 we develop a theoretical automaton-based model for IWIM, which we call the IWIM systems model. This is developed gradually, as it is a fairly complicated construction, aiming to reflect the essentials of IWIM in a credible manner. The underlying idea is that families of worker automata perform their tasks under the supervision of a manager automaton. Change of state of the manager corresponds to reconfiguration, whereupon a different family of worker automata shoulders the burden. This basic idea is elaborated to enable arbitrarily complex hierarchies to be modelled. Although our model is reasonably in-

volved, it falls short of capturing everything about IWIM or any specific implementation of the IWIM idea, such as is to be found in the formal specification of the MANIFOLD language [Arbab et al. (1993), Bonsangue et al. (2000)]. In particular we abstract away from the ability of workers to continue with internal actions on their own, which in the full IWIM model they can do irrespective of the attentions of any manager. Our main purpose could be seen as being to explore the viability of fibration based ideas in the arena of reconfiguration problems.

In Section 4 we describe some algebraic properties of our IWIM systems. These are based primarily on the categorical ideas of pullbacks and pushouts, suitably interpreted in the present context. A number of variations on these ideas are possible, and we consider a number of them. The completeness of the algebraic constructions offered turns out to be a relatively straightforward issue and also receives some attention. In Section 5 we discuss how the instantaneous reconfiguration aspect of our IWIM systems can be generalised to model the asynchronous event based reconfigurations characteristic of real IWIM frameworks. In Section 6 we show how the model of Arbab, de Boer and Bonsangue [Arbab et al. (2000a)], a theoretical model featuring aspects of reconfiguration, can be captured within IWIM systems; and in Section 7 we show how the model of Katis, Sabadini and Walters [Katis et al. (2000)], a significantly different theoretical account, can also be captured within IWIM systems. Section 8 concludes.

2 The IWIM Model

In this section we review the generic coordination framework known as the Ideal Worker Ideal Manager (IWIM) model [Arbab (1995), Arbab (1996), Arbab et al. (1998)]. The basic concepts in the IWIM model are processes, events, ports, and channels. A process is a black box with well defined ports of connection through which it exchanges units of information with the other processes in its environment. A port is a named opening in the bounding walls of a process through which units of information are exchanged using standard I/O primitives such as read and write; we assume that each port is used for the exchange of information in only one direction: either into the process (input port) or out of the process (output port).

The interconnections between the ports of processes are made through channels. A channel connects a port of a producer process to a port of a consumer process. Independent of the channels, there is an event mechanism for information exchange in IWIM. Events are broadcast by their sources into their environment, yielding event occurrences. In principle, any process in an environment can pick up a broadcast event occurrence. In practice, usually only a few processes pick up occurrences of each event, because only they are tuned in to the relevant sources.

The IWIM model supports anonymous communication: in general, a process does not, and need not, know the identity of the processes with which it exchanges information. This concept reduces the dependence of a process on its environment and makes processes more reusable; it also makes the protocols governing such communication more reusable.

A process in IWIM can be regarded as a worker process or a manager (or coordinator) process. The responsibility of a worker process is to perform a task. A worker process is not responsible for the communication that is necessary for it to obtain the proper input it requires to perform its task, nor is it responsible for the communication that is necessary to deliver the results it produces to their proper recipients. In general, no process in IWIM is responsible for its own communication with other processes. It is always the responsibility of a manager process to arrange for and to coordinate the necessary communications among a set of worker processes.

There is always a bottom layer of worker processes, called atomic workers, in an application. In the IWIM model, an application is built as a (dynamic) hierarchy of worker and manager processes on top of this layer. Aside from the atomic workers, the categorization of a process as a worker or a manager process is subjective: a manager process *man* that coordinates the communication among a number of worker processes, may itself be considered as a worker process by another manager process responsible for coordinating the communication of *man* with other processes.

In IWIM, a channel is a communication link that carries a sequence of bits, grouped into units. A channel represents a reliable, directed, and perhaps buffered, flow of information in time. Here, reliable means that the bits placed into a channel are guaranteed to flow through without loss, error, or duplication, and with their order preserved; and directed means that there are always two identifiable ends in a channel: a source and a sink. Once a channel is established between a producer process and a consumer process, it operates autonomously and transfers the units from its source to its sink.

If we make no assumptions about the internal operation of the producer and the consumer of a channel *c*, we must consider the possibility that *c* may contain some pending units. The pending units of a channel *c* are the units that have already been delivered to *c* by its producer, but not yet delivered by *c* to its consumer. The possibility of the existence of pending units in a channel gives it an identity of its own, independent of its producer and consumer. It makes it meaningful for a channel to remain connected at one of its ends, after it is disconnected from the other. The full details of the IWIM model codify a number of variations on this theme, but for our purposes, a channel will stay alive as long as one end or another is connected to a process.

Worker processes have two means of communication: via ports, and via events. The communication primitives that allow a process to exchange data through its ports are conventional read and write primitives. A process can attempt to read data from one of its input ports. It hangs if no data is presently available through that port, and continues once data is made available. Similarly, a process can attempt to write data to one of its output ports. It hangs if the port is presently not connected to any channel, and continues once a channel connection is made to accept the data.

A process *proc* can also broadcast an event *e* to all other processes in its environment by raising that event. The identity of the event *e* together with the identity of the process *proc* comprise the event occurrence. A process can also pick up event occurrences broadcast by other processes and react to them. Certain events are guaranteed to

be broadcast in special circumstances; for example, termination of a process instance always raises a special event to indicate its death. Our formal model in the rest of the paper will be quite limited in that we only model reconfiguration events. Even then, for simplicity, the modelling will be synchronous, a defect we address later.

A manager process can create new instances of processes (including itself) and broadcast and react to event occurrences. It can also create and destroy channel connections between various ports of the process instances it knows, including its own. Creation of new process instances, as well as installation and dismantling of communication channels are done dynamically. Specifically, these actions may be prompted by event occurrences it detects. Each manager process typically controls the communications among a dynamic family of process instances in a data-flow like network. The processes themselves are generally unaware of their patterns of communication, which may change in time, according to the decisions of a coordinator process.

In our formal model, again for reasons of simplicity, we eschew the full generality of these concepts. Our process networks will turn out to be statically defined, though the execution trajectory through this stucture will be dynamically determined. As such they may be viewed as the static unwinding of an implicit but more succinct syntactic specification of dynamic behaviour, and the unwinding enables us to restrict discussion to the semantic level alone, a welcome simplification.

3 IWIM Automata

In this section, we distil the essentials of the ideas just described, to create the model which will serve as the basis for the semantics of IWIM in the rest of the paper. We build the model up in two steps. The first is based on a fibration-inspired strategy, to reflect the way that IWIM events tear down and rebuild interconnections between families of processes. Accordingly, elementary IWIM automata will have in the base a manager automaton, describing how the manager part of an elementary IWIM system moves, and above each state of the manager automaton, there will be a collection of worker automata, connected together according to the prescription contained in the manager state. The various worker collections are then integrated into a single elementary IWIM system using an 'above' relation describing how workers relate to states of the manager, a construction inspired in essence by the Grothendieck construction. As a result of this, each configuration of the overall automaton can be projected down onto the relevant state of the manager in the manner of a fibration.

The capacity of IWIM systems to reconfigure themselves via events that provoke managers into reconfiguration activities, is here modelled by mappings of certain worker moves (that represent the raising of the event) to manager moves (that represent the reception and processing of the event, resulting in reconfiguration). Unlike genuine IWIM systems, this is a synchronous activity in our model, but we will show in Section 5 that the asynchronous aspects can be recaptured within our framework.

Fig. 1 illustrates in pictures what we have just described in words for elementary IWIM automata. It shows a collection of worker automata $\{A, B, C, D, E, S\}$ sitting

above a manager *Man*, forming an elementary IWIM system. The states of *Man* i.e. {*l*, *m*, *n*} each map to communication networks consisting of directed graphs of ports and channels. The ports of these networks correspond bijectively to input and output ports in the workers, who are ignorant of whence come their input messages and where their output messages are destined. Input ports are shown solid, while output ports are hollow. Furthermore these bijections in large part mimic the substructuring of individual ports in IWIM into their private and public parts. Also following these bijections up to the workers reveals which workers are above which management states. Note that worker *B* is above more than one management state. This means that when *Man* makes a transition from *l* to *m*, *B* is unaffected and continues to work as before. Attached to each channel is a queue of messages illustrated for just one channel for *l* in the figure. Some of the channels can be external, such as the external

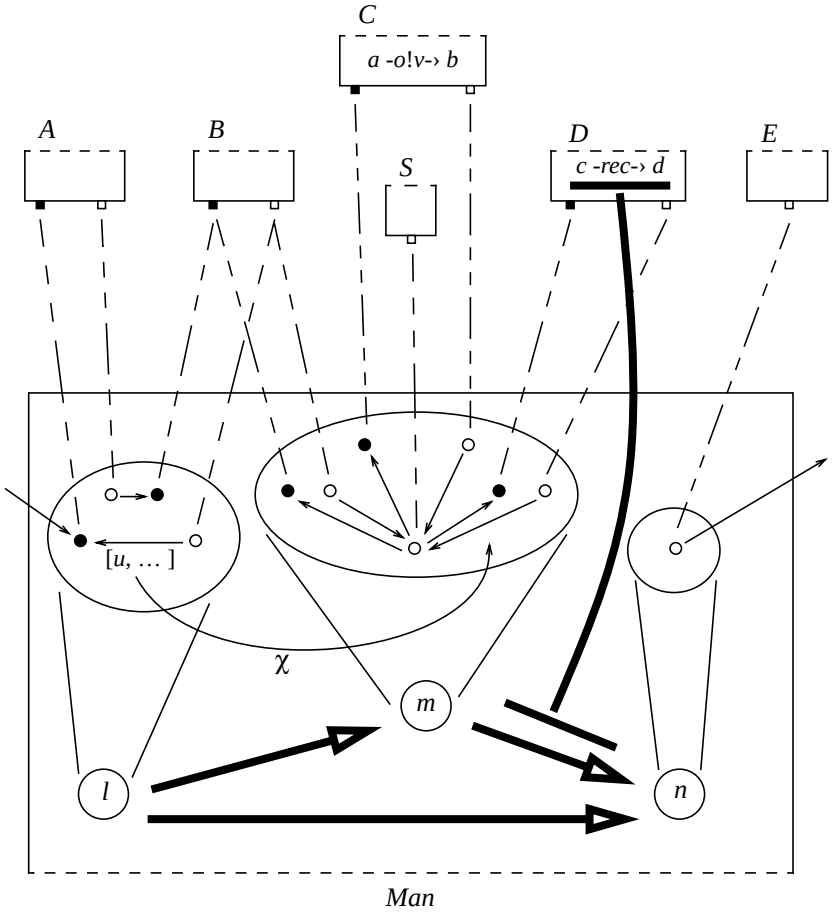

Fig. 1

input channel for state *l*, and the external output channel for *n*; these allow connection to and exchange of information with the outside world. Note however that external input can only take place when *l* is the current management state, and external output can only take place when *n* is the current management state. The management transitions must specify what happens to the message queues. These are mapped by additional data illustrated by χ in the figure and merged into the destination queues.

Worker *C* shows a typical worker output transition; there are similar worker input transitions. The port of worker *S* shows that ports are really quite general purpose concepts in IWIM, able to accomodate several incoming and outgoing channels. Worker *S* itself can be seen as providing a serialisation service for *B*, *C*, *D*. Worker *D* shows a reconfiguration event transition. The thick line from the transition to the manager illustrates that the atomic transition label *rec* is mapped to the manager transition from *m* to *n*. In this manner the workers can provoke reconfigurations implemented by the manager.

In the second step of the two step strategy for building our IWIM system model, the elementary IWIM system construction just described is generalised to take account of the more flexible nature of real IWIM systems. Now, processes may manifest both manager and worker roles, worker processes may enjoy the attentions of more than one manager, and manager processes may enjoy the benefits of more than one worker. To cope with this, we define IWIM worker-manager automata as asynchronous products of individual worker and manager automata. Also the relation connecting workers and managers becomes global. In this manner we get unrestricted IWIM systems. The previously mentioned properties continue to hold. In particular, configurations of an unrestricted IWIM system can be projected down onto configurations of their mangers.

Let us illustrate all this in another Figure. Fig 2 shows four worker-manager automata, *W, X, Y, Z*. These are drawn as rectangles with the dashed horizontal line representing the division between the worker and manager facets, the manager facet being uppermost. The worker structure is suppressed in all cases, and the fact that the manager parts of *X* and *Y* are empty is intended to indicate that these automata are atomic workers, with trivial manager facets. The arrows emanating from manager states point to the worker facets under their control. Fig 2 illustrates that (almost) completely general management relationships are permitted between worker-manager automata. In fact the only restriction is that an automaton's manager facet cannot manage it's own worker facet. Of course in realistic settings, the kind of contorted and cyclic dependencies occurring in Fig. 2 do not really arise. Far more plausible are regularly structured hierarchies with atomic workers in the bottommost layer.

3.1 Elementary IWIM Systems

Definition 3.1 An IWIM manager automaton is a triple (M, m_{I}, R), where M is a set of management states, $m_{\mathrm{I}} \in M$ is an initial state, and R is a set of reconfiguration transitions. These components are further stuctured as follows. Each management state m is itself the name of a pair (P_m, C_m), where P_m is a set of port names, and C_m is a

set of channel names. There are two partial functions s_m, $t_m : C_m \to P_m$ that send channels to source and target port names where they are defined. They satisfy $\mathrm{dom}(s_m) \cup \mathrm{dom}(t_m) = C_m$, i.e. each channel is connected to at least one port — channels not in $\mathrm{dom}(s_m)$ are called external input channels, and channels not in $\mathrm{dom}(t_m)$ are called external output channels; channels in both $\mathrm{dom}(s_m)$ and $\mathrm{dom}(t_m)$ are called internal channels. In a reconfiguration transition, written m -r-› n, the r is shorthand for a partial injection on the channel names $\chi_{m,n} : C_m \to C_n$. Also for each management state m, we have an identity transition m -id_m-› m in which the $\chi_{m,m}$ partial injection is a total identity.

The above definition characterises states of the manager automaton as connection networks in which the ports do not have a unique orientation (as input or output ports). Different states m, n may refer to the same connection network. Reconfigurations identify some channels of the source state with some channels of the target.

Definition 3.2 An IWIM worker automaton is a triple (I, O, A), where I is a set of input ports, disjoint from O a set of output ports; and $A = (St, Init, Tr)$ is an automaton with states St, of which $Init \in St$ is an initial state, and $Tr \subseteq St \times Act \times St$ is a transition relation, where Act is a set of actions of the form $in?v$ or $out!v$ or rec. In the first two kinds of action, $in \in I$, $out \in O$, and we assume that there is a global alphabet of val-

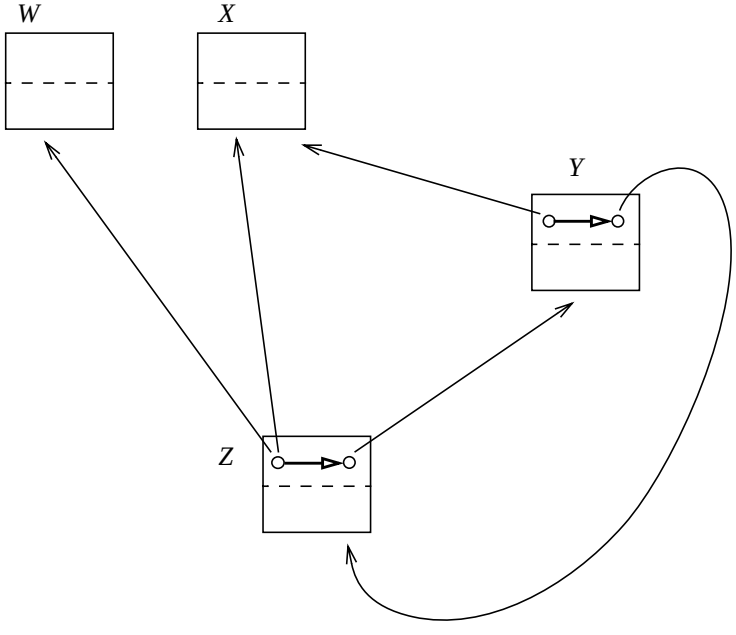

Fig. 2

ues *Val* containing *v*. In the last kind, *rec* is just a name (intended to be the name of a reconfiguration transition as in Definition 3.1). Where convenient below, we will write transitions using the notation a *-in*?v-› b or a *-out*!v-› b or a *-rec-›* b. We define $Tr_{\mathrm{I}} = \{a \text{ -in?}v\text{-›} b \in Tr\}$, $Tr_{\mathrm{O}} = \{a \text{ -out!}v\text{-›} b \in Tr\}$, $Tr_{\mathrm{R}} = \{a \text{ -rec-›} b \in Tr\}$, so that $Tr = Tr_{\mathrm{I}} \cup Tr_{\mathrm{O}} \cup Tr_{\mathrm{R}}$, the union being evidently disjoint. Additionally we define $Rec = \{rec \mid a \text{ -rec-›} b \in Tr\}$ the alphabet of reconfiguration events of the worker.

So far, workers are automata of a fairly standard kind. Now we show how workers and managers are glued together.

Definition 3.3 An elementary IWIM system (*Man*, *Wor*) consists of an IWIM manager automaton *Man*, an elementary workforce *Wor*, and ancillary data to be described below. *Wor* is a set of worker names together with a map *wor*, which yields for each worker $w \in Wor$, an IWIM worker automaton $wor(w)$. Furthermore we have:

(1) There is a relation ^ between *Wor* and the management states of *Man*. We write $w \hat{} m$ to say that a worker w is *above* a management state m if the pair is in the relation.

(2) If a worker w is above a management state m, then there is a map $r_{w\hat{}m}$ from the *rec* actions of $wor(w)$, into reconfiguration transitions m *-r-›* n of *Man*.

(3) For each management state $m \in Man$, there is a total bijection $\lambda_m : P_m \to IO_m$ where IO_m is the disjoint union of all of the input and output ports of all workers above m; i.e. $IO_m = \biguplus_{k\hat{}m}\{i \mid i \in I_{wor(k)}\} \uplus \biguplus_{k\hat{}m}\{o \mid o \in O_{wor(k)}\}$.

(4) Associated to each channel $c \in C_m$ (where m is a management state), there is a queue of messages which we write $c{:}[u_0, u_1, \ldots]$. Each u_i is in *Val*. The front of this queue is u_0.

A configuration of an elementary IWIM system (*Man*, *Wor*) consists of:

(1) a state m of *Man*;

(2) a set $ests = \{a_k \mid a_k \in St_{wor(k)}, k \in Wor\}$ of states a_k one for each worker k;

(3) a set $qs = \{c{:}q_c \mid c{:}q_c = c{:}[u_0, u_1, \ldots], c \in C_n, n \in M\}$ of queues of messages $c{:}[u_0, u_1, \ldots]$ one for each channel of each management state.

Note that in the above, *ests* may equivalently be viewed as the range of a function which maps each worker to one of its states, so that a_k is formally an ordered pair. Since we are overwhelmingly concerned with the states and how they change, we will not use the more cumbersome functional apparatus. Similar remarks apply to *qs* though here some of the indexing information is routinely suppressed.

A configuration of an elementary IWIM system (*Man*, *Wor*) is initial iff: m is initial, the a_k are also all initial, and the queues associated with all channels are empty.

A transition of an elementary IWIM system (*Man*, *Wor*) in state (m, *ests*, *qs*) is one of the following six kinds:

(ENVI) The environment adds a value to the input end of a queue whose source end is not attached to any port (an external input channel's queue).

$$\frac{c \notin \mathrm{dom}(s_m)\ ,\quad c \in \mathrm{dom}(t_m)\ ,\quad qs_{\mathrm{rest}} = qs - \{c{:}[\ \ldots\ , u_n]\}}{\begin{array}{l} m \longrightarrow m\ , \\ ests \longrightarrow ests\ , \\ qs \longrightarrow qs_{\mathrm{rest}} \cup \{c{:}[\ \ldots\ , u_n\ , u]\} \end{array}}$$

(ENVO) The environment removes a value from the output end of a queue whose target end is not attached to any port (an external output channel's queue).

$$\frac{c \notin \mathrm{dom}(t_m)\ ,\quad c \in \mathrm{dom}(s_m)\ ,\quad qs_{\mathrm{rest}} = qs - \{c{:}[u, u_1, \ldots\ ]\}}{\begin{array}{l} m \longrightarrow m\ , \\ ests \longrightarrow ests\ , \\ qs \longrightarrow qs_{\mathrm{rest}} \cup \{c{:}[u_1, \ldots\ ]\} \end{array}}$$

(IN) A worker automaton performs an input on one of its input ports, removing the front element from an input queue attached to the port, of which there must be at least one.

$$\frac{k\text{^}m\ ,\ a_k \in ests\ ,\ a_k\ \text{-}i?u\text{-}\!\rightarrow b_k\ ,\quad \lambda_m(p) = i \in I_{wor(k)}\ ,\ t_m(c) = p\ ,\quad ests_{\mathrm{rest}} = ests - \{a_k\}\ ,\quad qs_{\mathrm{rest}} = qs - \{c{:}[u, u_1, \ldots\ ]\}}{\begin{array}{l} m \longrightarrow m\ , \\ ests \longrightarrow ests_{\mathrm{rest}} \cup \{b_k\}\ , \\ qs \longrightarrow qs_{\mathrm{rest}} \cup \{c{:}[u_1, \ldots\ ]\} \end{array}}$$

(OUT) A worker automaton performs an output on one of its output ports, adding a value to the end of any output queue attached to the port, of which there must be at least one.

$$\frac{k\text{^}m\ ,\ a_k \in ests\ ,\ a_k\ \text{-}o!u\text{-}\!\rightarrow b_k\ ,\quad \lambda_m(p) = o \in O_{wor(k)}\ ,\quad \varnothing \neq Out = \{d \mid s_m(d) = p\}\ ,\quad ests_{\mathrm{rest}} = ests - \{a_k\}\ ,\quad qs_{\mathrm{rest}} = qs - \{d{:}[\ \ldots\ , u_{d,n_d}] \mid d \in Out\}}{\begin{array}{l} m \longrightarrow m\ , \\ ests \longrightarrow ests_{\mathrm{rest}} \cup \{b_k\}\ , \\ qs \longrightarrow qs_{\mathrm{rest}} \cup \{d{:}[\ \ldots\ , u_{d,n_d}, u] \mid d \in Out\} \end{array}}$$

(FOR) A port performs a forwarding action, removing the front element from an input queue attached to the port and inserting (a copy of) it to all output queues attached to the port, of which there must be at least one.

$t_m(c) = p$,
$\emptyset \neq Out = \{d \mid s_m(d) = p\}$,
$qs_{\text{rest}} = qs - (\{c{:}[u, u_1, \ldots]\} \cup \{d{:}[\ldots, u_{d,n_d}] \mid d \in Out\})$

$m \longrightarrow m$,
$ests \longrightarrow ests$,
$qs \longrightarrow qs_{\text{rest}} \cup \{c{:}[u_1, \ldots]\} \cup \{d{:}[\ldots, u_{d,n_d}, u] \mid d \in Out\}$

NB. The above notation is intended to include the case that $c \in Out$, whereupon the front message of c's queue is moved to its tail.

(REC) A worker automaton k_r performs a *rec* action a_{k_r} *-rec-›* b_{k_r}, provoking a reconfiguration m *-r-›* n of the elementary IWIM system, given by the function $r_{k_r \wedge m}$. The manager automaton makes a transition to the new state. Worker automaton k_r completes its transition. Worker automata other than k_r who are above both the old and new manager state remain as before. Worker automata above the old but not the new manager state go into suspension. Worker automata not above the old but above the new manager state are awakened. The queues of channels above the old manager state which are reassigned via the channel reconfiguration data are moved according to that data, being merged with the existing queues at target channels and leaving the queues at originating channels empty. The queues at other channels remain as before.

$k_r \wedge m$, $a_{k_r} \in ests$, a_{k_r} *-rec-›* b_{k_r} ,
$r_{k_r \wedge m}(rec) = m$ *-r-›* $n = \chi_{m,n} : C_m \to C_n$,
$ests_{\text{rest}} = ests - \{a_{k_r}\}$,
$qs_{\text{del}} = \{c{:}q_c \mid c \in C_m, c \in \text{dom}(\chi_{m,n})\} \cup \{d{:}q_d \mid d \in C_n, d \in \text{rng}(\chi_{m,n})\}$,
$qs_{\text{rest}} = qs - qs_{\text{del}}$,
$qs_{\text{dom}} = \{c{:}[] \mid c \in C_m, c \in \text{dom}(\chi_{m,n})\}$,
$qs_{\text{merge}} = \{d{:}q_{cd} \mid c{:}q_c, c \in C_m, c \in \text{dom}(\chi_{m,n}),$
$d{:}q_d, \chi_{m,n}(c) = d \in C_n, d \in \text{rng}(\chi_{m,n}),$
$q_{cd} \in \text{merge}(q_c, q_d)\}$

$m \longrightarrow n$,
$ests \longrightarrow ests_{\text{rest}} \cup \{b_{k_r}\}$,
$qs \longrightarrow qs_{\text{rest}} \cup qs_{\text{dom}} \cup qs_{\text{merge}}$

This transition system has some features that deserve comment. Note firstly that input/output and forwarding activities are completely decoupled. For this reason it makes little sense for the manager to connect up a port to use simultaneously as a broadcasting device, and as an input device to the relevant worker, since the input messages and forwarded messages are necessarily disjoint. Thus since even forwarding ports have to belong to some worker, it is best to invent special purpose dummy workers just for the purpose.

A second issue concerns the creation and destruction of processes. IWIM is entirely virtuous regarding matters of life and death: there is no murder, only suicide. The most that managers can accomplish is anasthesia. When a reconfiguration transition

takes a worker out of the current configuration because that worker is not above the new current management state, the worker sleeps, because being above the current management state is a hypothesis of all six transition types. When the current management state once more becomes one which the worker is above, it wakes and is able to participate in worker transitions again. It is the worker's own responsibility to enter a state out of which no transitions emerge if it wishes to die.

Thirdly there arises the issue of queue management during reconfiguration transitions. We have elected to merge assigned queues with existing ones (for given source and target ports) as representing an abstraction of the potential presence of several independent queues from the source to the target. The latter would require a more complex notion of reconfiguration transition than we wish to get embroiled in.

Let *EConfs*(*Man*, *Wor*) be the set of all configurations of (*Man*, *Wor*). Equipping it with the transitions just described makes it into a transition system. We regard this transition system as unlabelled, it being the case that the kind of step involved is always deducible from the pair of configurations in question.

A run of (*Man*, *Wor*) is, in the normal manner, a sequence of contiguous transitions of *EConfs*(*Man*, *Wor*), starting with an initial configuration:

$$(m, ests, qs) \longrightarrow (m', ests', qs') \longrightarrow (m'', ests'', qs'') \longrightarrow \ldots$$

Let *Mngr*(*Man*, *Wor*) be the set of manager states of configurations in *EConfs*(*Man*, *Wor*). These are given by a function $e\pi_{man}$ where $e\pi_{man}(m, ests, qs) = m$. The set *Mngr*(*Man*, *Wor*) can be equipped with transitions derived from the (REC) transitions of *EConfs*(*Man*, *Wor*). Thus to the transition $(m, ests, qs) \longrightarrow (m', ests', qs')$ corresponds the *Mngr*(*Man*, *Wor*) transition $e\pi_{man}(m, ests, qs) \longrightarrow e\pi_{man}(m', ests', qs')$, i.e. $m \longrightarrow m'$, (we regard these transition as unlabelled too). We also add an identity transition $m \longrightarrow m$ to each manager state in *Mngr*(*Man*, *Wor*).

Now although a particular worker may be above several manager states, making problematic the definition of a projection from the static structure of the elementary IWIM system to its manager, the same is not true of the set of configurations of the elementary IWIM system and its transition system, *EConfs*(*Man*, *Wor*), as it relates to the set of manager states. In *EConfs*(*Man*, *Wor*), some specific manager state always indexes any worker state that forms part of a configuration, and so we obtain the following result.

Proposition 3.4 Let (*Man*, *Wor*) be an elementary IWIM system. Let *EConfs*(*Man*, *Wor*) be the associated transition system and *Mngr*(*Man*, *Wor*) be the corresponding set of manager transitions. Then there is a projection:

$$\Pi_e : EConfs(Man, Wor) \to Mngr(Man, Wor)$$

which maps states by:

$$(m, ests, qs) \mapsto m = e\pi_{man}(m, ests, qs)$$

and which maps (REC) transitions by:

$$(m, ests, qs) \longrightarrow (m', ests', qs')$$
$$\mapsto$$
$$m \longrightarrow m' = e\pi_{man}(m, ests, qs) \longrightarrow e\pi_{man}(m', ests', qs')$$

and which maps (ENVI), (ENVO), (IN), (OUT), transitions to identity transitions:

$$(m, ests, qs) \longrightarrow (m, ests', qs')$$
$$\mapsto$$
$$m \longrightarrow m$$

Proof. Obvious. ☺

3.2 Unrestricted IWIM Systems

The previous section captures the essence of the process by which an individual manager automaton manages a group of worker automata. However the IWIM model does not restrict worker management to a single layer. Managers may themselves be workers managed by others, in time honoured hierarchical fashion. We model this here by allowing managers to themselves acquire a worker facet. The result is effectively a product of the two preceding constructions.

Definition 3.5 An IWIM worker-manager automaton is the asynchronous product of an IWIM worker automaton (I, O, A) as in Definition 3.2, and an IWIM manager automaton (M, m_I, R) as in Definition 3.1. That is to say, an IWIM worker-manager automaton is of the form $(I, O, A) \otimes (M, m_I, R)$, where (I, O, A) is called the worker facet and (M, m_I, R) is called the manger facet. The set of states of the worker-manager automaton is $St \times M$, with initial state $(Init, m_I)$, and there are two kinds of transitions: worker transitions such as (a, m) -w-› (b, m) where a -w-› b is a transition of (I, O, A) (and the manager facet remains unchanged), and manager transitions such as (a, m) -r-› (a, n) where m -r-› n is a transition of (M, m_I, R) (and the worker facet remains unchanged).

The following is evident.

Proposition 3.6 An IWIM worker-manager automaton for which the worker facet is a single (initial) state IWIM worker automaton with empty transition relation is strongly bisimilar to an IWIM manager automaton. Also an IWIM worker-manager automaton for which the manager facet is a single (initial) state IWIM manager automaton whose port and channel sets are empty, and with transition relation consisting of just the obligatory (in this case empty) identity function, is strongly bisimilar to an IWIM worker automaton.

In view of this, we can refer to IWIM worker-manager automata with trivial worker facets as pure mangers, and to IWIM worker-manager automata with trivial manager facets as pure workers.

Now that individual automata are capable of both worker and manager behaviour, we can define an unrestricted IWIM system as a community of automata where the manager facets of individual automata manage their individual workforces drawn from the same community, and the worker facets of individual automata each do their jobs

coordinated by one or more manager facets, since we place no restriction on the number of bosses any poor labourer might have. In keeping with the best industrial practice, no worker is ever his own manager (no selfdetermination — no one sets their own salary, nor signs off their own expense claims). Since the moves of the whole system are the moves of the individual elements, we need no additional restrictions beyond the no selfdetermination rule and the restrictions that apply to elementary IWIM systems, to have consistency.

Definition 3.7 An unrestricted IWIM system *WM* is a set of IWIM worker-manager automaton names called *WM*, a subset $Initial_{WM} \subseteq WM$, together with ancillary data described below. There are three maps: *worman*, *wor*, *man*, where for each $wm \in WM$, *worman*(*wm*) is an IWIM worker-manager automaton, *wor*(*wm*) is its worker facet, and *man*(*wm*) is its manager facet. We write m_{wm} to say that state *m* is a state of a facet of automaton *wm*, the facet intended being clear from the context; formally m_{wm} is an ordered pair, just as before. The states of a worker-manager automaton *wm* are thus written (a_{wm}, m_{wm}), where *a* is the state of the worker facet and *m* is the state of the manager facet.

Moreover, other aspects of the notation for elementary IWIM systems acquire additional subscripting to indicate what part of the unrestricted IWIM system they refer to. Thus we have $P_{m_{wm}}$ for the set of port names of state *m* of the manager facet *man*(*wm*) of *wm*; likewise $C_{m_{wm}}$ is the corresponding set of channel names.

There is a binary above relation $\wedge$ where $wm' \wedge m_{wm}$ means that the worker facet *wor*(*wm*′) of automaton *wm*′ is above state *m* of the nontrivial manger facet *man*(*wm*) of automaton *wm*. The no selfdetermination rule implies that whenever $wm' \wedge m_{wm}$, then $wm' \neq wm$. The workforce $\{wm_1, \ldots, wm_n\}$ of automata whose worker facets are above states of the manager facet of *wm* is refered to as an elementary IWIM subsystem of *WM*, and is an elementary IWIM system in the sense of Definition 3.3 when we disregard the manger facets of the workers and the worker facet of the manager. Thus $IO_{m_{wm}}$ is the set of input and output ports of the workforce above m_{wm}. Specifically for an elementary IWIM subsystem:

(1) The above relation is inherited from the global one, and we will assume henceforth that no automaton is above the unique state of a trivial manager.

(2) There is a map $r_{wm' \wedge m_{wm}}$ of the *rec* transitions of worker facets into reconfiguration transitions of the corresponding nontrivial manager facet.

(3) The total bijection property of manager ports to workforce input/output ports holds via a map $\lambda_{m_{wm}} : P_{m_{wm}} \to IO_{m_{wm}}$.

(Note that the no selfdetermination rule is consistent with the asynchronous product structure of the transitions for worker-manager automata. Otherwise some $r_{wm \wedge m_{wm}}$ could force moves of *wm* that were worker and manager moves simultaneously.)

Let *WM* be an unrestricted IWIM system. Then we define $WM^{\#} = \{wm \in WM \mid wm$ has a nontrivial manager facet$\}$.

A configuration (sts, qs) of an unrestricted IWIM system consists of:

(1) a set $sts = \{(a_{wm}, m_{wm}) \mid wm \in WM\}$ of states (a_{wm}, m_{wm}) one for each automaton in WM;

(2) a set $qs = \{c{:}q_c \mid c \in C_{m_{wm}}, \exists\ a \bullet (a_{wm}, m_{wm}) \in sts\}$ of queues of messages $c{:}[u_0, u_1, \ldots]$ one for each channel $c \in C_{m_{wm}}$ of each management state m_{wm} of each nontrivial manager facet $man(wm)$.

As before, these configuration components are really the ranges of suitable functions.

A configuration (sts, qs) of an unrestricted IWIM system WM is initial iff: all states in sts are initial in both facets, and all channel queues in qs are empty.

Let (sts, qs) be a configuration of an unrestricted IWIM system WM. Then we can define the manager part of (sts, qs) to be $\pi_{\text{man}}(sts) = \{m_{wm} \mid \exists\ a_{wm} \bullet (a_{wm}, m_{wm}) \in sts, wm \in WM^{\#}\}$.

A transition of an unrestricted IWIM system WM in configuration (sts, qs) is one of six kinds, patterned after elementary IWIM system transitions:

(ENVI) The environment adds a value to the end of an external input queue.

$$\frac{\begin{array}{l} c \notin \bigcup\{\text{dom}(s_{m'_{wm'}}) \mid m'_{wm'} \in \pi_{\text{man}}(sts)\}\ , \\ c \in \text{dom}(t_{m_{wm}})\ ,\ m_{wm} \in \pi_{\text{man}}(sts)\ , \\ qs_{\text{rest}} = qs - \{c{:}[\ \ldots\ , u_n]\} \end{array}}{\begin{array}{l} sts \longrightarrow sts\ , \\ qs \longrightarrow qs_{\text{rest}} \cup \{c{:}[\ \ldots\ , u_n, u]\} \end{array}}$$

(ENVO) The environment removes a value from the end of an external output queue.

$$\frac{\begin{array}{l} c \notin \bigcup\{\text{dom}(t_{m'_{wm'}}) \mid m'_{wm'} \in \pi_{\text{man}}(sts)\}\ , \\ c \in \text{dom}(s_{m_{wm}})\ ,\ m_{wm} \in \pi_{\text{man}}(sts)\ , \\ qs_{\text{rest}} = qs - \{c{:}[u, u_1, \ldots]\} \end{array}}{\begin{array}{l} sts \longrightarrow sts\ , \\ qs \longrightarrow qs_{\text{rest}} \cup \{c{:}[u_1, \ldots]\} \end{array}}$$

(IN) A worker facet of an automaton performs an input on one of its input ports, of which there must be at least one.

$$\frac{\begin{array}{l} k{\wedge}m_{wm}\ ,\ m_{wm} \in \pi_{\text{man}}(sts)\ , \\ (a_k, n_k) \in sts\ ,\ (a_k, n_k)\ \text{-}i?u\text{-}\!\rightarrow (b_k, n_k)\ , \\ \lambda_{m_{wm}}(p) = i \in I_{wor(k)}\ ,\ t_{m_{wm}}(c) = p\ , \\ sts_{\text{rest}} = sts - \{(a_k, n_k)\}\ , \\ qs_{\text{rest}} = qs - \{c{:}[u, u_1, \ldots]\} \end{array}}{\begin{array}{l} sts \longrightarrow sts_{\text{rest}} \cup \{(b_k, n_k)\}\ , \\ qs \longrightarrow qs_{\text{rest}} \cup \{c{:}[u_1, \ldots]\} \end{array}}$$

(OUT) A worker facet of an automaton performs an output on one of its output ports, of which there must be at least one.

$(a_k, n_k) \in sts$, (a_k, n_k) -o!u-› (b_k, n_k) ,
$\emptyset \neq Out = \{d \mid \exists\ m_{wm} \in \pi_{\text{man}}(sts), p \bullet k^\wedge m_{wm},$
$\lambda_{m_{wm}}(p) = o \in O_{wor(k)}, s_{m_{wm}}(d) = p\}$,
$sts_{\text{rest}} = sts - \{(a_k, n_k)\}$,
$qs_{\text{rest}} = qs - \{d{:}[\ \ldots\ , u_{d,n_d}] \mid d \in Out\}$

$sts \longrightarrow sts_{\text{rest}} \cup \{(b_k, n_k)\}$,
$qs \longrightarrow qs_{\text{rest}} \cup \{d{:}[\ \ldots\ , u_{d,n_d}, u] \mid d \in Out\}$

(FOR) A port performs a forwarding action.

$k^\wedge m'_{wm'}$, $m'_{wm'} \in \pi_{\text{man}}(sts)$, $t_{m'_{wm'}}(c) = p$,
$\emptyset \neq Out = \{d \mid \exists\ m_{wm} \in \pi_{\text{man}}(sts), p \bullet k^\wedge m_{wm},$
$\lambda_{m_{wm}}(p) = o \in O_{wor(k)}, s_{m_{wm}}(d) = p\}$,
$qs_{\text{rest}} = qs - (\{c{:}[u, u_1, \ldots\ ]\} \cup \{d{:}[\ \ldots\ , u_{d,n_d}] \mid d \in Out\})$

$sts \longrightarrow sts$,
$qs \longrightarrow qs_{\text{rest}} \cup \{c{:}[u_1, \ldots\ ]\} \cup \{d{:}[\ \ldots\ , u_{d,n_d}, u] \mid d \in Out\}$

NB. The above notation is intended to include the case that $c \in Out$, whereupon the front message of c's queue is moved to its tail.

(REC) The worker facet of automaton k_r performs a *rec* action a_{k_r} -*rec*-› b_{k_r}, moving to state b_{k_r}, and provoking reconfigurations of all the elementary IWIM subsystems managed by manager facets above a current state of which k_r sits. All these manager facets move to their respective new management states. The queues of the channels managed by these manager facets are mapped via the channel reconfiguration data for their particular manager facet.

$\emptyset \neq Rm_{\text{man}} = \{m_{wm} \mid m_{wm} \in \pi_{\text{man}}(sts) \bullet k_r^\wedge m_{wm}\}$,
$(a_{k_r}, m_{k_r}) \in sts$, (a_{k_r}, m_{k_r}) -*rec*-› (b_{k_r}, m_{k_r}) ,
$Rn_{\text{man}} = \{n_{wm} \mid m_{wm} \in \pi_{\text{man}}(sts) \bullet k_r^\wedge m_{wm},$
$r_{k_r^\wedge m_{wm}}(rec) = m_{wm}$ -*r*-› $n_{wm} = \chi_{m_{wm},n_{wm}} : C_{m_{wm}} \to C_{n_{wm}}\}$,
$sts_{\text{rest}} = sts - (\{(a_{k_r}, m_{k_r})\} \cup$
$\{(a_{wm}, m_{wm}) \mid (a_{wm}, m_{wm}) \in sts, m_{wm} \in Rm_{\text{man}}\})$,
$sts_{\text{post}} = \{(b_{k_r}, m_{k_r})\} \cup \{(a_{wm}, n_{wm}) \mid (a_{wm}, m_{wm}) \in sts,$
$m_{wm} \in Rm_{\text{man}}, n_{wm} \in Rn_{\text{man}}\}$,
$qs_{\text{del}} = \{c{:}q_c \mid c \in C_{m_{wm}}, c \in \text{dom}(\chi_{m_{wm},n_{wm}}), m_{wm} \in Rm_{\text{man}}\} \cup$
$\{d{:}q_d \mid d \in C_{m_{wm}}, d \in \text{rng}(\chi_{m_{wm},n_{wm}}), m_{wm} \in Rm_{\text{man}}, n_{wm} \in Rn_{\text{man}}\}$,
$qs_{\text{rest}} = qs - qs_{\text{del}}$,
$qs_{\text{dom}} = \{c{:}[] \mid c \in C_{m_{wm}}, c \in \text{dom}(\chi_{m_{wm},n_{wm}}), m_{wm} \in Rm_{\text{man}}\}$,
$qs_{\text{merge}} = \{d{:}q_{cd} \mid c{:}q_c, c \in C_{m_{wm}}, c \in \text{dom}(\chi_{m_{wm},n_{wm}}),$
$d{:}q_d, \chi_{m_{wm},n_{wm}}(c) = d \in C_{m_{wm}}, d \in \text{rng}(\chi_{m_{wm},n_{wm}}),$
$m_{wm} \in Rm_{\text{man}}, n_{wm} \in Rn_{\text{man}},$
$q_{cd} \in \text{merge}(q_c, q_d)\}$

$sts \longrightarrow sts_{\text{rest}} \cup sts_{\text{post}}$,
$qs \longrightarrow qs_{\text{rest}} \cup qs_{\text{dom}} \cup qs_{\text{merge}}$

The remarks made following the elementary IWIM subsystems transition system description apply with equal or greater force here. Thus all transitions have hypotheses that ensure that any active worker is being actively managed by being above at least one current mangement state. Also there is no murder, only anasthesia and suicide. Moreover, reconfiguration events simultaneously affect all mangers who might be managing a particular worker facet. The structure of the model ensures that they can all do this without adversely interfering with each other.

Let *Confs*(*WM*) be the set of all configurations of *WM*. Equipping it with the transitions just described makes it into a transition system.

A run of *WM* is a sequence of contiguous transitions of *Confs*(*WM*) starting with an initial configuration:

$$(sts, qs) \longrightarrow (sts', qs') \longrightarrow (sts'', qs'') \longrightarrow \ldots$$

Let (*sts*, *qs*) be a configuration of *WM*. Let *Mngrs*(*WM*) be the set of manager parts of configurations in *Confs*(*WM*). It can be equipped with transitions derived from those of *Confs*(*WM*). Thus whenever (*sts*, *qs*) $\longrightarrow$ (*sts*′, *qs*′) is a (REC) transition of *Confs*(*WM*), there is a *Mngrs*(*WM*) transition $\pi_{\text{man}}(sts) \longrightarrow \pi_{\text{man}}(sts')$. We also add an identity transition $\pi_{\text{man}}(sts) \longrightarrow \pi_{\text{man}}(sts)$ to each manager part in *Mngrs*(*WM*). As previously, all of these transitions are unlabelled.

It will now not be surprising that despite the greater complexity we have here, the projection that we had in Section 3.1 can be recovered.

Proposition 3.8 Let *WM* be an unrestricted IWIM system. Let *Confs*(*WM*) be the associated transition system, and *Mngrs*(*WM*) be the associated manager parts transition system. Then there is a projection:

$$\Pi : Confs(WM) \to Mngrs(WM)$$

which maps states by:

$$(sts, qs) \mapsto \pi_{\text{man}}(sts)$$

and which maps (REC) transitions by:

$$\begin{array}{l} (sts, qs) \longrightarrow (sts', qs') \\ \qquad \mapsto \\ \pi_{\text{man}}(sts) \longrightarrow \pi_{\text{man}}(sts') \end{array}$$

and which maps (ENVI), (ENVO), (IN), (OUT), transitions to identity transitions:

$$\begin{array}{l} (sts, qs) \longrightarrow (sts', qs') \\ \qquad \mapsto \\ \pi_{\text{man}}(sts) \longrightarrow \pi_{\text{man}}(sts') = \pi_{\text{man}}(sts) \end{array}$$

Proof. Obvious. ☺

In the remainder of the paper we will be concerned only with unrestricted IWIM systems, and will henceforth just refer to them as IWIM systems.

4 Algebraic Properties of IWIM Systems

The relatively clean structure of IWIM systems gives rise to a number of algebraic properties. In this section we describe a selection of these from among the large number of possibilities. We start by defining suitable notions of homomorphism for worker, manager, and worker-manager automata, and move on to pullback and pushout contructions using them; these being things focused on automata themselves. We repeat the exercise for weakened notions of homomorphism, for reasons that become clear when we subsequently consider contructions focused on IWIM systems. Finally, we consider completeness.

4.1 Contructions Centred on Automata

We start with the most obvious constructions.

Definition 4.1 (Worker Homomorphisms) Let $wor_1 = (I_1, O_1, A_1 = (St_1, Init_1, Tr_1))$ and $wor_2 = (I_2, O_2, A_2 = (St_2, Init_2, Tr_2))$ be worker automata. A worker homomorphism $f_\mathsf{w} : wor_1 \to wor_2$ is given by the functions: $f_\mathsf{w} : St_1 \to St_2$ (overloading the name f_w), and $\varphi : I_1 \to I_2$, $\kappa : O_1 \to O_2$, where φ and κ are bijections, $f_\mathsf{w}(Init_1) = Init_2$, and whenever there is a transition of the form a -in?v-› b or a -out!v-› b or a -rec-› b in Tr_1, then we have a transition $f_\mathsf{w}(a)$ -$\varphi(in)$?v-› $f_\mathsf{w}(b)$ or $f_\mathsf{w}(a)$ -$\kappa(out)$!v-› $f_\mathsf{w}(b)$ or $f_\mathsf{w}(a)$ -rec-› $f_\mathsf{w}(b)$ respectively in Tr_2. The worker homomorphism $f_\mathsf{w} : wor_1 \to wor_2$ is said to be injective, surjective, bijective etc., iff the set function $f_\mathsf{w} : St_1 \to St_2$ has (any of) these properties. Below we will normally save on notation by assuming that the bijections φ and κ are strict identities.

Definition 4.2 (Manager Homomorphisms) Let $man_1 = (M_1, m_{\mathrm{I},1}, R_1)$ and $man_2 = (M_2, m_{\mathrm{I},2}, R_2)$ be manager automata. A manager homomorphism $f_\mathsf{m} : man_1 \to man_2$ is given by the functions: $f_\mathsf{m} : M_1 \to M_2$ (overloading this time f_m), and the set of functions $\{f_{\mathsf{m},mP}, f_{\mathsf{m},mC} \mid m \in M_1\}$, such that $f_\mathsf{m}(m_{\mathrm{I},1}) = m_{\mathrm{I},2}$, and all the following hold:

- whenever $f_m(m_1) = m_2$ then if m_1 maps to (P_{m_1}, C_{m_1}) and m_2 maps to (P_{m_2}, C_{m_2}) then $f_{\mathsf{m},m_1P} : P_{m_1} \to P_{m_2}$ is a bijection, which further restricts to bijections between $\mathrm{rng}(s_{m_1})$ and $\mathrm{rng}(s_{m_2})$, $\mathrm{rng}(t_{m_1})$ and $\mathrm{rng}(t_{m_2})$; and $f_{\mathsf{m},m_1C} : C_{m_1} \to C_{m_2}$ is a bijection, which further restricts to bijections between $\mathrm{dom}(s_{m_1})$ and $\mathrm{dom}(s_{m_2})$, $\mathrm{dom}(t_{m_1})$ and $\mathrm{dom}(t_{m_2})$; and that:

  $$f_{\mathsf{m},m_1P} \circ s_{m_1} = s_{m_2} \circ f_{\mathsf{m},m_1C} \quad \text{and} \quad f_{\mathsf{m},m_1P} \circ t_{m_1} = t_{m_2} \circ f_{\mathsf{m},m_1C}$$

- whenever m_1 -r-› n_1 is a transition of R_1 given by $\chi_{m_1,n_1} : C_{m_1} \to C_{n_1}$, then we have a transition $f_\mathsf{m}(m_1) = m_2$ -r-› $n_2 = f_\mathsf{m}(n_1)$ of R_2 given by $\chi_{m_2,n_2} : C_{m_2} \to C_{n_2}$, such that f_{m,m_1C} restricts to a bijection between $\mathrm{dom}(\chi_{m_1,n_1})$ and $\mathrm{dom}(\chi_{m_2,n_2})$, and f_{m,n_1C} restricts to a bijection between $\mathrm{rng}(\chi_{m_1,n_1})$ and $\mathrm{rng}(\chi_{m_2,n_2})$; and that:

  $$f_{\mathsf{m},n_1C} \circ \chi_{m_1,n_1} = \chi_{m_2,n_2} \circ f_{\mathsf{m},m_1C}$$

The manager homomorphism $f_\mathsf{m} : man_1 \to man_2$ is said to be injective, surjective, bijective etc., iff the set function $f_\mathsf{m} : M_1 \to M_2$ has these properties. Below we will

normally save on notation by assuming that the family of bijections $\{f_{\mathsf{m},mP}, f_{\mathsf{m},mC} \mid m \in M_1\}$ actually consists of strict identities.

Definition 4.3 (Worker-Manager Homomorphisms) Let $wm_1 = (I_1, O_1, A_1) \otimes (M_1, m_{\mathrm{I},1}, R_1)$ and $wm_2 = (I_2, O_2, A_2) \otimes (M_2, m_{\mathrm{I},2}, R_2)$ be worker-manager automata. A worker-manager homomorphism $(f_{\mathsf{w}}, f_{\mathsf{m}}) : wm_1 \to wm_2$ consists of a worker homomorphism f_{w} acting on the worker facets, and a manager homomorphism f_{m} acting on the manager facets. Also the worker-manager homomorphism $(f_{\mathsf{w}}, f_{\mathsf{m}}) : wm_1 \to wm_2$ is said to be injective, surjective, bijective etc., iff the component worker and manager homomorphisms both are.

Definition 4.4 (Worker Pullbacks) Let $wor_1 = (I, O, A_1 = (St_1, Init_1, Tr_1))$, $wor_2 = (I, O, A_2 = (St_2, Init_2, Tr_2))$, and $wor^\bullet = (I, O, A^\bullet = (St^\bullet, Init^\bullet, Tr^\bullet))$ be worker automata. Let $f_{\mathsf{w},1}{}^\bullet : wor_1 \to wor^\bullet$ and $f_{\mathsf{w},2}{}^\bullet : wor_2 \to wor^\bullet$ be two worker homomorphisms. We define the worker automaton $wor = (I, O, A = (St, Init, Tr))$, the worker pullback of wor_1 and wor_2 with respect to $f_{\mathsf{w},1}{}^\bullet$ and $f_{\mathsf{w},2}{}^\bullet$, as follows.

$$St = f_{\mathsf{w},1}{}^{\bullet -1}(St^{\bullet\cap}) \times f_{\mathsf{w},2}{}^{\bullet -1}(St^{\bullet\cap}) \quad \text{where} \quad St^{\bullet\cap} = f_{\mathsf{w},1}{}^\bullet(St_1) \cap f_{\mathsf{w},2}{}^\bullet(St_2)$$

$$Init = (Init_1, Init_2)$$

$$\begin{aligned} Tr = \; & \{(a_1, a_2) \text{-}in?v\text{-}\!\rightarrow (b_1, b_2) \mid (a_1, a_2), (b_1, b_2) \in St, \\ & \qquad a_1 \text{-}in?v\text{-}\!\rightarrow b_1 \in Tr_{1,\mathrm{I}}, a_2 \text{-}in?v\text{-}\!\rightarrow b_2 \in Tr_{2,\mathrm{I}}\} \cup \\ & \{(a_1, a_2) \text{-}out!v\text{-}\!\rightarrow (b_1, b_2) \mid (a_1, a_2), (b_1, b_2) \in St, \\ & \qquad a_1 \text{-}out!v\text{-}\!\rightarrow b_1 \in Tr_{1,\mathrm{O}}, a_2 \text{-}out!v\text{-}\!\rightarrow b_2 \in Tr_{2,\mathrm{O}}\} \cup \\ & \{(a_1, a_2) \text{-}rec\text{-}\!\rightarrow (b_1, b_2) \mid (a_1, a_2), (b_1, b_2) \in St, \\ & \qquad a_1 \text{-}rec\text{-}\!\rightarrow b_1 \in Tr_{1,\mathrm{R}}, a_2 \text{-}rec\text{-}\!\rightarrow b_2 \in Tr_{2,\mathrm{R}}\} \end{aligned}$$

Evidently the above is consistent, and there are projections $f_{\mathsf{w},1} : wor \to wor_1$ and $f_{\mathsf{w},2} : wor \to wor_2$ that respectively delete the wor_2 aspects and wor_1 aspects from wor in the expected way.

Definition 4.5 (Manager Pullbacks) Let $man_1 = (M_1, m_{\mathrm{I},1}, R_1)$, $man_2 = (M_2, m_{\mathrm{I},2}, R_2)$, and $man^\bullet = (M^\bullet, m_{\mathrm{I}}{}^\bullet, R^\bullet)$ be manager automata. Let $f_{\mathsf{m},1}{}^\bullet : man_1 \to man^\bullet$ and $f_{\mathsf{m},2}{}^\bullet : man_2 \to man^\bullet$ be two manager homomorphisms. We define the manager automaton $man = (M, m_{\mathrm{I}}, R)$, the manager pullback of man_1 and man_2 with respect to $f_{\mathsf{m},1}{}^\bullet$ and $f_{\mathsf{m},2}{}^\bullet$, as follows.

$$M = f_{\mathsf{m},1}{}^{\bullet -1}(M^{\bullet\cap}) \times f_{\mathsf{m},2}{}^{\bullet -1}(M^{\bullet\cap}) \quad \text{where} \quad M^{\bullet\cap} = f_{\mathsf{m},1}{}^\bullet(M_1) \cap f_{\mathsf{m},2}{}^\bullet(M_2)$$

$$m_{\mathrm{I}} = (m_{\mathrm{I},1}, m_{\mathrm{I},2})$$

$$\begin{aligned} (m_1, m_2) \in M \Rightarrow \; & (m_1, m_2) \text{ maps to } (P_m, C_m) \text{ in } man \text{ iff} \\ & (\, m_1 \text{ maps to } (P_m, C_m) \text{ in } man_1 \text{ and} \\ & \; m_2 \text{ maps to } (P_m, C_m) \text{ in } man_2 \text{ and} \\ & \; f_{\mathsf{m},1}{}^\bullet(m_1) = f_{\mathsf{m},2}{}^\bullet(m_2) \text{ maps to } (P_m, C_m) \text{ in } man^\bullet \,) \end{aligned}$$

$$\begin{aligned} R = \; & \{(m_1, m_2) \text{-}r\text{-}\!\rightarrow (n_1, n_2) \mid (m_1, m_2), (n_1, n_2) \in M, \\ & \qquad m_1 \text{-}r\text{-}\!\rightarrow n_1 \in R_1, m_2 \text{-}r\text{-}\!\rightarrow n_2 \in R_2\} \end{aligned}$$

$$(m_1, m_2) \text{-}r\text{-}\!\rightarrow (n_1, n_2) \in R \Rightarrow \chi_{(m_1, m_2),(n_1, n_2)} = \chi_{m_1,n_1} = \chi_{m_2,n_2}$$

Note that this generates identity reconfigurations on (m_1, m_2) as identities on C_m in the appropriate way. Also the above is consistent, our notational saving coming into its own in the mapping of states of *man* to port-channel networks and their reconfigurations. There are also projections $f_{\mathsf{m},1} : man \to man_1$ and $f_{\mathsf{m},2} : man \to man_2$ that respectively delete the man_2 aspects and man_1 aspects from *man* in the expected way.

Definition 4.6 (Worker-Manager Pullbacks) Let $wm_1 = (I, O, A_1) \otimes (M_1, m_{\mathrm{I},1}, R_1)$, $wm_2 = (I, O, A_2) \otimes (M_2, m_{\mathrm{I},2}, R_2)$, and $wm^\bullet = (I, O, A^\bullet) \otimes (M^\bullet, m_{\mathrm{I},}{}^\bullet, R^\bullet)$ be worker-manager automata. Let $(f_{\mathsf{w},1}{}^\bullet, f_{\mathsf{m},1}{}^\bullet) : wm_1 \to wm^\bullet$ and $(f_{\mathsf{w},2}{}^\bullet, f_{\mathsf{m},2}{}^\bullet) : wm_2 \to wm^\bullet$ be two worker-manager homomorphisms. Then we define the worker-manager automaton $wm = (I, O, A) \otimes (M, m_{\mathrm{I}}, R)$, the worker-manager pullback of wm_1 and wm_2 with respect to $(f_{\mathsf{w},1}{}^\bullet, f_{\mathsf{m},1}{}^\bullet)$ and $(f_{\mathsf{w},2}{}^\bullet, f_{\mathsf{m},2}{}^\bullet)$, as the asynchronous product of the worker pullback of $f_{\mathsf{w},1}{}^\bullet$ and $f_{\mathsf{w},2}{}^\bullet$ acting on the worker facets, and the manager pullback of $f_{\mathsf{m},1}{}^\bullet$ and $f_{\mathsf{m},2}{}^\bullet$ acting on the manager facets, in the natural manner. Inevitably we have projections $f_{\mathsf{wm},1} : wm \to wm_1$ and $f_{\mathsf{wm},2} : wm \to wm_2$ that act in the expected way.

We move now to the pushout constructions. In order to avoid cumbersome technical details, we assume that henceforth all the unions we mention are disjoint, so it is clear for each element of such a union, which component it arises from. As is usual in algebraic discussions, we can always arrange for unions to be disjoint by choosing appropriate (set theoretically) isomorphic variants of the structures we consider.

Definition 4.7 (Worker Pushouts) Let $wor_1 = (I, O, A_1 = (St_1, Init_1, Tr_1))$, $wor_2 = (I, O, A_2 = (St_2, Init_2, Tr_2))$, and $wor^\bullet = (I, O, A^\bullet = (St^\bullet, Init^\bullet, Tr^\bullet))$ be disjoint worker automata. Let $f_{\mathsf{w},1}{}^\bullet : wor^\bullet \to wor_1$ and $f_{\mathsf{w},2}{}^\bullet : wor^\bullet \to wor_2$ be two worker homomorphisms. We define the worker automaton $wor = (I, O, A = (St, Init, Tr))$, the worker pushout of wor_1 and wor_2 with respect to $f_{\mathsf{w},1}{}^\bullet$ and $f_{\mathsf{w},2}{}^\bullet$, as follows.

$$St = St_1 \cup St_2 \,/\sim_{\mathsf{w}} \quad \text{where } \sim_{\mathsf{w}} \text{ is the finest equivalence relation generated by the propositions } a_1 = f_{\mathsf{w},1}{}^\bullet(a^\bullet) \wedge f_{\mathsf{w},2}{}^\bullet(a^\bullet) = a_2 \Rightarrow a_1 \sim_{\mathsf{w}} a_2 \text{ and we write } [a]_{\mathsf{w}} \text{ for the equivalence class containing } a$$

$$Init = [Init_1]_{\mathsf{w}} = [Init_2]_{\mathsf{w}}$$

$$\begin{aligned} Tr = \; & \{[a]_{\mathsf{w}} \text{ -}in?v\text{-}\rangle\ [b]_{\mathsf{w}} \mid [a]_{\mathsf{w}}, [b]_{\mathsf{w}} \in St,\ a \text{ -}in?v\text{-}\rangle\ b \in Tr_{1,\mathrm{I}} \cup Tr_{2,\mathrm{I}}\} \cup \\ & \{[a]_{\mathsf{w}} \text{ -}out!v\text{-}\rangle\ [b]_{\mathsf{w}} \mid [a]_{\mathsf{w}}, [b]_{\mathsf{w}} \in St,\ a \text{ -}out!v\text{-}\rangle\ b \in Tr_{1,\mathrm{O}} \cup Tr_{2,\mathrm{O}}\} \cup \\ & \{[a]_{\mathsf{w}} \text{ -}rec\text{-}\rangle\ [b]_{\mathsf{w}} \mid [a]_{\mathsf{w}}, [b]_{\mathsf{w}} \in St,\ a \text{ -}rec\text{-}\rangle\ b \in Tr_{1,\mathrm{R}} \cup Tr_{2,\mathrm{R}}\} \end{aligned}$$

Evidently the above is consistent, and there are homomorphisms $f_{\mathsf{w},1} : wor_1 \to wor$ and $f_{\mathsf{w},2} : wor_2 \to wor$ that identify wor_1 aspects and wor_2 aspects inside *wor* in the expected way.

Definition 4.8 (Manager Pushouts) Let $man_1 = (M_1, m_{\mathrm{I},1}, R_1)$, $man_2 = (M_2, m_{\mathrm{I},2}, R_2)$, and $man^\bullet = (M^\bullet, m_{\mathrm{I},}{}^\bullet, R^\bullet)$ be manager automata. Let $f_{\mathsf{m},1}{}^\bullet : man^\bullet \to man_1$ and $f_{\mathsf{m},2}{}^\bullet : man^\bullet \to man_2$ be two manager homomorphisms. To save on notation we will assume that the bijections $\{f_{\mathsf{m},1}{}^\bullet{}_{,mP}, f_{\mathsf{m},1}{}^\bullet{}_{,mC} \mid m \in M_1\}$ and $\{f_{\mathsf{m},2}{}^\bullet{}_{,mP}, f_{\mathsf{m},2}{}^\bullet{}_{,mC} \mid m \in M_2\}$ are strict identities as previously. We define the manager automaton $man = (M, m_{\mathrm{I}}, R)$, the manager pushout of man_1 and man_2 with respect to $f_{\mathsf{m},1}{}^\bullet$ and $f_{\mathsf{m},2}{}^\bullet$, as follows.

$M = M_1 \cup M_2 / \sim_{\mathsf{m}}$ where $\sim_{\mathsf{m}}$ is the finest equivalence relation generated by the propositions $m_1 = f_{\mathsf{m},1}{}^{\bullet}(m^{\bullet}) \wedge f_{\mathsf{m},2}{}^{\bullet}(m^{\bullet}) = m_2 \Rightarrow m_1 \sim_{\mathsf{m}} m_2$ and we write $[m]_{\mathsf{m}}$ for the equivalence class containing m

$m_{\mathrm{I}} = [m_{\mathrm{I},1}]_{\mathsf{m}} = [m_{\mathrm{I},2}]_{\mathsf{m}}$

$[m]_{\mathsf{m}} \in M \Rightarrow [m]_{\mathsf{m}}$ maps to (P_m, C_m) in man iff
(m maps to (P_m, C_m) in man_1 or
m maps to (P_m, C_m) in man_2 (or both))

$R = \{[m]_{\mathsf{m}} \text{ -}r\text{-}\rangle [n]_{\mathsf{m}} \mid [m]_{\mathsf{m}}, [n]_{\mathsf{m}} \in M, m \text{ -}r\text{-}\rangle n \in R_1 \cup R_2\}$

$[m]_{\mathsf{m}} \text{ -}r\text{-}\rangle [n]_{\mathsf{m}} \in R \Rightarrow \chi_{[m]_{\mathsf{m}},[n]_{\mathsf{m}}} =$
$\begin{cases} \chi_{m_1,n_1} & \text{if } m_1 \in [m]_{\mathsf{m}}, n_1 \in [n]_{\mathsf{m}}, m_1 \text{ -}r\text{-}\rangle n_1 \in R_1 \text{ or} \\ \chi_{m_2,n_2} & \text{if } m_2 \in [m]_{\mathsf{m}}, n_2 \in [n]_{\mathsf{m}}, m_2 \text{ -}r\text{-}\rangle n_2 \in R_2 \\ & \text{(or both)} \end{cases}$

Note that this also generates identity reconfigurations on $[m]_{\mathsf{m}}$ as identities on C_m in the appropriate way. Evidently the above is consistent, and there are homomorphisms $f_{\mathsf{m},1} : man_1 \to man$ and $f_{\mathsf{m},2} : man_2 \to man$ that identify man_1 aspects and man_2 aspects inside man in as expected.

Definition 4.9 (Worker-Manager Pushouts) Let $wm_1 = (I, O, A_1) \otimes (M_1, m_{\mathrm{I},1}, R_1)$, $wm_2 = (I, O, A_2) \otimes (M_2, m_{\mathrm{I},2}, R_2)$, and $wm^{\bullet} = (I, O, A^{\bullet}) \otimes (M^{\bullet}, m_{\mathrm{I}}{}^{\bullet}, R^{\bullet})$ be worker-manager automata. Let $(f_{\mathsf{w},1}{}^{\bullet}, f_{\mathsf{m},1}{}^{\bullet}) : wm^{\bullet} \to wm_1$ and $(f_{\mathsf{w},2}{}^{\bullet}, f_{\mathsf{m},2}{}^{\bullet}) : wm^{\bullet} \to wm_2$ be two worker-manager homomorphisms. Then we define the worker-manager automaton $wm = (I, O, A) \otimes (M, m_{\mathrm{I}}, R)$, the worker-manager pushout of wm_1 and wm_2 with respect to $(f_{\mathsf{w},1}{}^{\bullet}, f_{\mathsf{m},1}{}^{\bullet})$ and $(f_{\mathsf{w},2}{}^{\bullet}, f_{\mathsf{m},2}{}^{\bullet})$, as the asynchronous product of the worker pushout of $f_{\mathsf{w},1}{}^{\bullet}$ and $f_{\mathsf{w},2}{}^{\bullet}$ acting on the worker facets, and the manager pushout of $f_{\mathsf{m},1}{}^{\bullet}$ and $f_{\mathsf{m},2}{}^{\bullet}$ acting on the manager facets, in the natural manner. Inevitably we have homomorphisms $f_{\mathsf{wm},1} : wm_1 \to wm$ and $f_{\mathsf{wm},2} : wm_2 \to wm$ that act in the expected way.

As far as they go, the above constructions work well. There's a snag however when we come to try to utilise them within the context of an IWIM system. There, the fact that homomorphisms identify the manager interconnection structures 'on the nose' conflicts in pullback/pushout situations with the properties demaded of the $\lambda_{m_{wm}}$ and $r_{wm' \wedge m_{wm}}$ functions of the IWIM system. We will see this in detail below. We consequently introduce alternative constructions that work better in this regard, based on the idea of asynchronous products that we have seen already.

Definition 4.10 (Asynchronous Worker Homomorphisms) Let $wor_1 = (I, O, A_1 = (St_1, Init_1, Tr_1))$ and $wor_2 = (I, O, A_2 = (St_2, Init_2, Tr_2))$ be worker automata. An asynchronous worker homomorphism $f_{\mathsf{aw}} : wor_1 \to wor_2$ exists iff there is a function $f_{\mathsf{aw}} : St_1 \to St_2$ such that whenever there is a transition (of any kind) from a to b in Tr_1, then there is a transition from $f_{\mathsf{aw}}(a)$ to $f_{\mathsf{aw}}(b)$ (and not necessarily of the same kind) in Tr_2. The asynchronous worker homomorphism is said to be injective, surjective, bijective etc., iff the set function $f_{\mathsf{aw}} : St_1 \to St_2$ is. Note that we have adopted immediately a strict identity perspective on the input and output channels, optimising away the bijections that would otherwise be needed.

Definition 4.11 (Asynchronous Manager Homomorphisms) Let $man_1 = (M_1, m_{\mathrm{I},1}, R_1)$ and $man_2 = (M_2, m_{\mathrm{I},2}, R_2)$ be manager automata. An asynchronous manager homomorphism $f_{\mathsf{am}} : man_1 \to man_2$ exists iff there is a function $f_{\mathsf{am}} : M_1 \to M_2$ such that whenever there is a transition from m to n in R_1, then there is a transition from $f_{\mathsf{am}}(m)$ to $f_{\mathsf{aw}}(n)$ in R_2. The asynchronous manager homomorphism is said to be injective, surjective, bijective etc., iff the set function $f_{\mathsf{am}} : M_1 \to M_2$ is.

Definition 4.12 (Asynchronous Worker-Manager Homomorphisms) Let $wm_1 = (I, O, A_1) \otimes (M_1, m_{\mathrm{I},1}, R_1)$ and $wm_2 = (I, O, A_2) \otimes (M_2, m_{\mathrm{I},2}, R_2)$ be worker-manager automata. An asynchronous worker-manager homomorphism $(f_{\mathsf{aw}}, f_{\mathsf{am}}) : wm_1 \to wm_2$ consists of an asynchronous worker homomorphism f_{aw} acting on the worker facets, and an asynchronous manager homomorphism f_{am} acting on the manager facets. The asynchronous worker-manager homomorphism $(f_{\mathsf{aw}}, f_{\mathsf{am}}) : wm_1 \to wm_2$ is said to be injective, surjective, bijective etc., iff the component worker and manager homomorphisms both are.

Definition 4.13 ((Left and Right) Asynchronous Worker Pullbacks) Let $wor_1 = (I, O, A_1 = (St_1, Init_1, Tr_1))$, $wor_2 = (I, O, A_2 = (St_2, Init_2, Tr_2))$, and $wor^\bullet = (I, O, A^\bullet = (St^\bullet, Init^\bullet, Tr^\bullet))$ be worker automata. Let $f_{\mathsf{aw},1}{}^\bullet : wor_1 \to wor^\bullet$ and $f_{\mathsf{aw},2}{}^\bullet : wor_2 \to wor^\bullet$ be two asynchronous worker homomorphisms. We define three kinds of worker automata all denoted $wor = (I, O, A = (St, Init, Tr))$, namely the left, right, and arbitrary (i.e. with chosen initial state) asynchronous worker pullbacks of wor_1 and wor_2 with respect to $f_{\mathsf{aw},1}{}^\bullet$ and $f_{\mathsf{aw},2}{}^\bullet$, as follows. (Here as below, we economise on notation by using *wor* for all three types of automata, the left and right versions being of by far the most interest and thus highlighted in the definition's name; the context or other supplementary remarks, will clarify which is intended in each individual case).

$$St = f_{\mathsf{aw},1}{}^{\bullet -1}(St^{\bullet\cap}) \times f_{\mathsf{aw},2}{}^{\bullet -1}(St^{\bullet\cap}) \quad \text{where} \quad St^{\bullet\cap} = f_{\mathsf{aw},1}{}^\bullet(St_1) \cap f_{\mathsf{aw},2}{}^\bullet(St_2)$$

$$Init \quad \begin{cases} = (Init_1, a_2) \in St & \text{for a left asynchronous pullback} \\ = (a_1, Init_2) \in St & \text{for a right asynchronous pullback} \\ \in St & \text{for an arbitrary asynchronous pullback} \end{cases}$$

$$\begin{aligned} Tr = \; & \{(a_1, a_2) \text{ -}in?v\text{-}\!\rightarrow (b_1, a_2) \mid (a_1, a_2), (b_1, a_2) \in St, a_1 \text{ -}in?v\text{-}\!\rightarrow b_1 \in Tr_{1,\mathrm{I}}\} \cup \\ & \{(a_1, a_2) \text{ -}in?v\text{-}\!\rightarrow (a_1, b_2) \mid (a_1, a_2), (a_1, b_2) \in St, a_2 \text{ -}in?v\text{-}\!\rightarrow b_2 \in Tr_{2,\mathrm{I}}\} \cup \\ & \{(a_1, a_2) \text{ -}out!v\text{-}\!\rightarrow (b_1, a_2) \mid (a_1, a_2), (b_1, a_2) \in St, a_1 \text{ -}out!v\text{-}\!\rightarrow b_1 \in Tr_{1,\mathrm{O}}\} \cup \\ & \{(a_1, a_2) \text{ -}out!v\text{-}\!\rightarrow (a_1, b_2) \mid (a_1, a_2), (a_1, b_2) \in St, a_2 \text{ -}out!v\text{-}\!\rightarrow b_2 \in Tr_{2,\mathrm{O}}\} \cup \\ & \{(a_1, a_2) \text{ -}rec\text{-}\!\rightarrow (b_1, b_2) \mid (a_1, a_2), (b_1, b_2) \in St, \\ & \qquad a_1 \text{ -}rec\text{-}\!\rightarrow b_1 \in Tr_{1,\mathrm{R}}, a_2 \text{ -}rec\text{-}\!\rightarrow b_2 \in Tr_{2,\mathrm{R}}\} \end{aligned}$$

Note that the choice of initial state is not canonically determined because of the relatively undemanding notion of homomorphism that we are using. Even the left and right asynchronous pullbacks are not themselves unique without further conditions; eg. the choice of the initial state for the left asynchronous pullback is not unique unless $f_{\mathsf{aw},2}{}^{\bullet -1}(f_{\mathsf{aw},1}{}^\bullet(Init_1))$ is a singleton. Analogous considerations apply for the right asynchronous pullback. Note furthermore that while input and output transitions are inherited individually from Tr_1 and Tr_2, *rec* transitions are only inherited if they match up in both Tr_1 and Tr_2. This is for later convenience.

The asynchronous pullback worker automata *wor* possess partial asynchronous worker projection homomorphisms $\pi_{aw,1} : wor \to wor_1$ and $\pi_{aw,2} : wor \to wor_2$, and given in the case of $\pi_{aw,1}$ by:

$$\pi_{aw,1}((a_1, a_2)) = a_1$$

$$\pi_{aw,1}((a_1, a_2) \text{-}in?v\text{-}\rangle (b_1, a_2)) = a_1 \text{-}in?v\text{-}\rangle b_1 \quad \text{where } a_1 \text{-}in?v\text{-}\rangle b_1 \in Tr_{1,I}$$
$$\pi_{aw,1}((a_1, a_2) \text{-}out!v\text{-}\rangle (b_1, a_2)) = a_1 \text{-}out!v\text{-}\rangle b_1 \quad \text{where } a_1 \text{-}out!v\text{-}\rangle b_1 \in Tr_{1,O}$$
$$\pi_{aw,1}((a_1, a_2) \text{-}rec\text{-}\rangle (b_1, b_2)) = a_1 \text{-}rec\text{-}\rangle b_1 \quad \text{where } a_1 \text{-}rec\text{-}\rangle b_1 \in Tr_{1,R}$$

with the Tr_2 based input and output transitions of *wor* being outside the domain of $\pi_{aw,1}$. The definition of $\pi_{aw,2}$ is symmetric.

The partial projections $\pi_{aw,1}$ and $\pi_{aw,2}$, though partial on the static description of *wor*, extend to total projections, $\pi_{aw,1}^*$ and $\pi_{aw,2}^*$, from runs of *wor* to runs of wor_1 and wor_2 and given for $\pi_{aw,1}^*$ by:

$$\pi_{aw,1}^*([tran_0, tran_1, tran_2, \ldots]) = \begin{cases} \pi_{aw,1}(tran_0) :: \pi_{aw,1}^*([tran_1, tran_2, \ldots]) & \text{if } tran_0 \in \mathrm{dom}(\pi_{aw,1}) \\ \pi_{aw,1}^*([tran_1, tran_2, \ldots]) & \text{otherwise} \end{cases}$$

where the $tran_i$ are the individual transitions of the run. Symmetrically for $\pi_{aw,2}^*$.

There is of course the special case of this construction where $wor^\bullet$ is a one-state automaton with a self-loop, the result being called an **asynchronous worker product automaton**. This has a distinguished initial state, namely $(Init_1, Init_2)$.

Definition 4.14 ((Left and Right) Asynchronous Manager Pullbacks) Let $man_1 = (M_1, m_{I,1}, R_1)$, $man_2 = (M_2, m_{I,2}, R_2)$, and $man^\bullet = (M^\bullet, m_{I,}{}^\bullet, R^\bullet)$ be two disjoint manager automata. Let $f_{am,1}{}^\bullet : man_1 \to man^\bullet$ and $f_{am,2}{}^\bullet : man_2 \to man^\bullet$ be two asynchronous manager homomorphisms. We define the manager automata $man = (M, m_I, R)$, the left, right, and arbitrary (i.e. with chosen initial state) asynchronous manager pullbacks of man_1 and man_2 with respect to $f_{am,1}{}^\bullet$ and $f_{am,2}{}^\bullet$, as follows.

$$M = f_{am,1}{}^{\bullet -1}(M^{\bullet\cap}) \times f_{am,2}{}^{\bullet -1}(M^{\bullet\cap}) \quad \text{where } M^{\bullet\cap} = f_{am,1}{}^\bullet(M_1) \cap f_{am,2}{}^\bullet(M_2)$$

$$m_I \quad \begin{cases} = (m_{I,1}, m_2) \in M & \text{for a left asynchronous pullback} \\ = (m_1, m_{I,2}) \in M & \text{for a right asynchronous pullback} \\ \in M & \text{for an arbitrary asynchronous pullback} \end{cases}$$

$$(m_1, m_2) \in M \Rightarrow (m_1, m_2) \text{ maps in } man \text{ to } (P_{(m_1, m_2)}, C_{(m_1, m_2)}) = (P_{m,1} \uplus P_{m,2}, C_{m,1} \uplus C_{m,2}) \text{ iff } (m_1 \text{ maps to } (P_{m,1}, C_{m,1}) \text{ in } man_1 \text{ and } m_2 \text{ maps to } (P_{m,2}, C_{m,2}) \text{ in } man_2)$$

$$R = \{(m_1, m_2) \text{-}r\text{-}\rangle (n_1, m_2) \mid (m_1, m_2), (n_1, m_2) \in M, m_1 \text{-}r\text{-}\rangle n_1 \in R_1\} \cup \{(m_1, m_2) \text{-}r\text{-}\rangle (m_1, n_2) \mid (m_1, m_2), (m_1, n_2) \in M, m_2 \text{-}r\text{-}\rangle n_2 \in R_2\}$$

$$(m_1, m_2) \text{-}r\text{-}\rangle (n_1, m_2) \in R \Rightarrow \chi_{(m_1, m_2),(n_1, m_2)} = \chi_{m_1,n_1} \uplus \mathrm{id}_{C_{m_2}} \text{ if } m_1 \text{-}r\text{-}\rangle n_1 \in R_1$$
$$(m_1, m_2) \text{-}r\text{-}\rangle (m_1, n_2) \in R \Rightarrow \chi_{(m_1, m_2),(m_1, n_2)} = \mathrm{id}_{C_{m_1}} \uplus \chi_{m_1,n_1} \text{ if } m_2 \text{-}r\text{-}\rangle n_2 \in R_2$$

Note that we need disjoint unions in the definitions of $P_{(m_1, m_2)}$ and $C_{(m_1, m_2)}$ as otherwise there is a risk that the source or target function of some $c \in C_{(m_1, m_2)}$ might be ambiguous. Note also that the identities come out correctly without extra work. The same considerations as for workers also pertain to the initial states here; thus the initial state for the left asynchronous pullback is not unique unless ${f_{\mathsf{am},2}}^{\bullet -1}({f_{\mathsf{am},1}}^{\bullet}(m_{\mathrm{I},1}))$ is a singleton, etc.

Because all manager automaton states are stipulated to have at least an identity self-transition, there are total asynchronous worker projection homomorphisms $\pi_{\mathsf{am},1} : man \to man_1$ and $\pi_{\mathsf{am},2} : man \to man_2$, given for $\pi_{\mathsf{am},1}$ by:

$$\pi_{\mathsf{am},1}((m_1, m_2)) = m_1$$

$$\pi_{\mathsf{am},1}((m_1, m_2) \text{-}r\text{-}\rangle (n_1, m_2)) = m_1 \text{-}r\text{-}\rangle n_1 \quad \text{where } m_1 \text{-}r\text{-}\rangle n_1 \in R_1$$
$$\pi_{\mathsf{am},1}((m_1, m_2) \text{-}r\text{-}\rangle (m_1, n_2)) = m_1 \text{-id}_{m_1}\text{-}\rangle m_1 \quad \text{where } m_2 \text{-}r\text{-}\rangle n_2 \in R_2$$

(and symmetrically for $\pi_{\mathsf{am},2}$). It now goes without saying that $\pi_{\mathsf{am},1}$ and $\pi_{\mathsf{am},2}$ extend to runs in the predicted manner.

Equally obvious is the degenerate case of a one-state $man^{\bullet}$, giving rise to the **asynchronous manager product automaton** with distinguished initial state $(m_{\mathrm{I},1}, m_{\mathrm{I},2})$.

Definition 4.15 ((Left and Right) Asynchronous Worker-Manager Pullbacks) Let $wm_1 = (I, O, A_1) \otimes (M_1, m_{\mathrm{I},1}, R_1)$, $wm_2 = (I, O, A_2) \otimes (M_2, m_{\mathrm{I},2}, R_2)$, and $wm^{\bullet} = (I, O, A^{\bullet}) \otimes (M^{\bullet}, m_{\mathrm{I}}{}^{\bullet}, R^{\bullet})$ be worker-manager automata. Let $({f_{\mathsf{aw},1}}^{\bullet}, {f_{\mathsf{am},1}}^{\bullet}) : wm_1 \to wm^{\bullet}$ and $({f_{\mathsf{aw},2}}^{\bullet}, {f_{\mathsf{am},2}}^{\bullet}) : wm_2 \to wm^{\bullet}$ be two asynchronous worker-manager homomorphisms. Then we define the worker-manager automata $wm = (I, O, A) \otimes (M, m_{\mathrm{I}}, R)$, the left, right, and arbitrary (i.e. with chosen initial state) asynchronous worker-manager pullbacks of wm_1 and wm_2 with respect to $({f_{\mathsf{aw},1}}^{\bullet}, {f_{\mathsf{am},1}}^{\bullet})$ and $({f_{\mathsf{aw},2}}^{\bullet}, {f_{\mathsf{am},2}}^{\bullet})$, as the asynchronous products of: the (left, right, arbitrary) asynchronous worker pullbacks of the worker facets with respect to ${f_{\mathsf{aw},1}}^{\bullet}$ and ${f_{\mathsf{aw},2}}^{\bullet}$, and the (left, right, arbitrary) asynchronous manager pullbacks of the manager facets with respect to ${f_{\mathsf{am},1}}^{\bullet}$ and ${f_{\mathsf{am},2}}^{\bullet}$, in the natural manner.

The initial state and projection properties of asynchronous worker-manager pullback automata are inherited naturally from those of their constituents. Thus in the case of the latter, there are partial asynchronous worker-manager projection homomorphisms $(\pi_{\mathsf{aw},1}, \pi_{\mathsf{am},1}) : wm \to wm_1$ and $(\pi_{\mathsf{aw},2}, \pi_{\mathsf{am},2}) : wm \to wm_2$, such that for $(\pi_{\mathsf{aw},1}, \pi_{\mathsf{am},1})$, all transitions except worker transitions of the form $((a_1, a_2), (m_1, m_2))$ -*act*-$\rangle$ $((a_1, b_2), (m_1, m_2))$, where *act* is a non-*rec* action of Tr_2, are in $\mathrm{dom}((\pi_{\mathsf{am},1}, \pi_{\mathsf{am},1}))$, and symmetrically for $(\pi_{\mathsf{aw},2}, \pi_{\mathsf{am},2})$.

Obviously we also have in the expected way the degenerate case of a one-state $wm^{\bullet}$, giving rise to the **asynchronous worker-manager product automaton** with distinguished initial state $((Init_1, Init_2), (m_{\mathrm{I},1}, m_{\mathrm{I},2}))$.

Definition 4.16 ((Left and Right) Asynchronous Worker Pushouts) Let $wor_1 = (I, O, A_1 = (St_1, Init_1, Tr_1))$, $wor_2 = (I, O, A_2 = (St_2, Init_2, Tr_2))$, and $wor^{\bullet} = (I, O, A^{\bullet} = (St^{\bullet}, Init^{\bullet}, Tr^{\bullet}))$ be disjoint worker automata. Let ${f_{\mathsf{aw},1}}^{\bullet} : wor^{\bullet} \to wor_1$ and ${f_{\mathsf{aw},2}}^{\bullet} : wor^{\bullet} \to wor_2$ be two asynchronous worker homomorphisms. We define the worker

automata $wor = (I, O, A = (St, Init, Tr))$, the left, right, and arbitrary (i.e. with chosen initial state) asynchronous worker pushouts of wor_1 and wor_2 with respect to $f_{aw,1}{}^\bullet$ and $f_{aw,2}{}^\bullet$, as follows.

$St = St_1 \cup St_2 \,/\sim_{aw}$ where $\sim_{aw}$ is the finest equivalence relation generated by the propositions $a_1 = f_{aw,1}{}^\bullet(a^\bullet) \wedge f_{aw,2}{}^\bullet(a^\bullet) = a_2 \Rightarrow a_1 \sim_{aw} a_2$ and we write $[a]_{aw}$ for the equivalence class containing a

$$Init \begin{cases} = [Init_1]_{aw} & \text{for the left asynchronous pushout} \\ = [Init_2]_{aw} & \text{for the right asynchronous pushout} \\ \in St & \text{for an arbitrary asynchronous pushout} \end{cases}$$

$$\begin{aligned} Tr = {} & \{[a]_{aw} \text{-}in?v\text{-}\rangle\ [b]_{aw} \mid [a]_{aw}, [b]_{aw} \in St,\ a \text{-}in?v\text{-}\rangle\ b \in Tr_{1,I} \cup Tr_{2,I}\} \cup \\ & \{[a]_{aw} \text{-}out!v\text{-}\rangle\ [b]_{aw} \mid [a]_{aw}, [b]_{aw} \in St,\ a \text{-}out!v\text{-}\rangle\ b \in Tr_{1,O} \cup Tr_{2,O}\} \cup \\ & \{[a]_{aw} \text{-}rec\text{-}\rangle\ [b]_{aw} \mid [a]_{aw}, [b]_{aw} \in St,\ a \text{-}rec\text{-}\rangle\ b \in Tr_{1,R} \cup Tr_{2,R}\} \end{aligned}$$

Note that this time we have exactly two canonical choices for intial state, namely $[Init_1]_{aw}$ and $[Init_2]_{aw}$. The 'arbitrary' possibility is retained for completeness' sake.

Evidently there are (total) asynchronous worker homomorphisms $f_{aw,1} : wor_1 \to wor$ and $f_{aw,2} : wor_2 \to wor$ that identify wor_1 aspects and wor_2 aspects inside wor in the expected way. These also have extensions $f_{aw,1}{}^*$ and $f_{aw,2}{}^*$ to runs.

Just as for pullbacks we have degenerate cases. When $wor^\bullet$ is the empty worker automaton, and $f_{aw,1}{}^\bullet$ and $f_{aw,2}{}^\bullet$ are empty maps, we get the **left, right** and **arbitrary asynchronous sum worker automata**. Note though, that despite the fact that they constitute a very natural limiting case, asynchronous sum automata are not terribly useful in themselves. Since the state space is the disjoint union of the two components, whichever component contains the nominated initial state will contain all of the subsequent dynamics of the sum, and the other component becomes a useless bystander as its states are not accessible from the first component without some element of pushout-like gluing.

Definition 4.17 ((Left and Right) Asynchronous Manager Pushouts) Let $man_1 = (M_1, m_{I,1}, R_1)$, $man_2 = (M_2, m_{I,2}, R_2)$, and $man^\bullet = (M^\bullet, m_{I,}{}^\bullet, R^\bullet)$ be disjoint manager automata. Let $f_{am,1}{}^\bullet : man^\bullet \to man_1$ and $f_{am,2}{}^\bullet : man^\bullet \to man_2$ be two asynchronous manager homomorphisms. We define the manager automata $man = (M, m_I, R)$, the left, right, and arbitrary (i.e. with chosen initial state) asynchronous manager pushouts of man_1 and man_2 with respect to $f_{am,1}{}^\bullet$ and $f_{am,2}{}^\bullet$, as follows.

$M = M_1 \cup M_2 \,/\sim_{am}$ where $\sim_{am}$ is the finest equivalence relation generated by the propositions $m_1 = f_{am,1}{}^\bullet(m^\bullet) \wedge f_{aw,2}{}^\bullet(m^\bullet) = m_2 \Rightarrow m_1 \sim_{am} m_2$ and we write $[m]_{am}$ for the equivalence class containing m

$$m_I \begin{cases} = [m_{I,1}]_{am} & \text{for the left asynchronous pushout} \\ = [m_{I,2}]_{am} & \text{for the right asynchronous pushout} \\ \in M & \text{for an arbitrary asynchronous pushout} \end{cases}$$

$$[m]_{am} \in M \Rightarrow [m]_{am} \text{ maps to } (P_{[m]_{am}}, C_{[m]_{am}}) = (\biguplus\{P_m \mid m \in [m]_{am}\},\ \biguplus\{C_m \mid m \in [m]_{am}\})$$

$R = \{[m]_{\mathsf{am}} \text{-}r\text{-}\rangle [n]_{\mathsf{am}} \mid [m]_{\mathsf{am}}, [n]_{\mathsf{am}} \in M, m \text{-}r\text{-}\rangle n \in R_1 \cup R_2\}$

$[m]_{\mathsf{am}} \text{-}r\text{-}\rangle [n]_{\mathsf{am}} \in R \Rightarrow$

$$\chi_{[m]_{\mathsf{am}},[n]_{\mathsf{am}}} = \begin{cases} \chi_{m_1,n_1} & \text{if } m_1 \in [m]_{\mathsf{am}}, n_1 \in [n]_{\mathsf{am}}, m_1 \text{-}r\text{-}\rangle n_1 \in R_1 \text{ or} \\ \chi_{m_2,n_2} & \text{if } m_2 \in [m]_{\mathsf{am}}, n_2 \in [n]_{\mathsf{am}}, m_2 \text{-}r\text{-}\rangle n_2 \in R_2 \end{cases}$$

$[m]_{\mathsf{am}} \text{-id-}\rangle [m]_{\mathsf{am}} \in R \Rightarrow$

$$\chi_{[m]_{am},[n]_{am}} = \mathrm{id}_{[m]_{am}} : C_{[m]_{am}} \to C_{[m]_{am}}$$

We need disjoint unions in the definitions of $P_{[m]_{\mathsf{am}}}$ and $C_{[m]_{\mathsf{am}}}$ exactly as before. Note also the reconfiguration transitions $\chi_{[m]_{\mathsf{am}},[n]_{\mathsf{am}}}$ of the pushouts are just the reconfiguration transitions of the components, seen as partial injections on $C_{[m]_{\mathsf{am}}}$. However in this instance, unlike for the preceding manager constructions, we must add explicit identities on the states, as they do not arise naturally otherwise.

As in the previous case, we have exactly two canonical choices of initial state. Also there are total asynchronous manager homomorphisms $f_{\mathsf{am},1} : man_1 \to man$ and $f_{\mathsf{am},2} : man_2 \to man$ that identify man_1 aspects and man_2 aspects inside man as expected, and which also have extensions $f_{\mathsf{am},1}{}^*$ and $f_{\mathsf{am},2}{}^*$ to runs.

Definition 4.18 ((Left and Right) Asynchronous Worker-Manager Pushouts) Let $wm_1 = (I, O, A_1) \otimes (M_1, m_{\mathrm{I},1}, R_1)$, $wm_2 = (I, O, A_2) \otimes (M_2, m_{\mathrm{I},2}, R_2)$, and $wm^\bullet = (I, O, A^\bullet) \otimes (M^\bullet, m_{\mathrm{I}}{}^\bullet, R^\bullet)$ be worker-manager automata. Let $(f_{\mathsf{aw},1}{}^\bullet, f_{\mathsf{am},1}{}^\bullet) : wm^\bullet \to wm_1$ and $(f_{\mathsf{aw},2}{}^\bullet, f_{\mathsf{am},2}{}^\bullet) : wm^\bullet \to wm_2$ be two asynchronous worker-manager homomorphisms. Then we define the worker-manager automaton $wm = (I, O, A) \otimes (M, m_{\mathrm{I}}, R)$, the left, right, and arbitrary (i.e. with chosen initial state) asynchronous worker-manager pushouts of wm_1 and wm_2 with respect to $(f_{\mathsf{aw},1}{}^\bullet, f_{\mathsf{am},1}{}^\bullet)$ and $(f_{\mathsf{aw},2}{}^\bullet, f_{\mathsf{am},2}{}^\bullet)$, as the asynchronous products of: the (left, right, arbitrary) asynchronous worker pushouts of the worker facets with respect to $f_{\mathsf{aw},1}{}^\bullet$ and $f_{\mathsf{aw},2}{}^\bullet$, and the (left, right, arbitrary) asynchronous worker pushouts of the manager facets with respect to $f_{\mathsf{am},1}{}^\bullet$ and $f_{\mathsf{am},2}{}^\bullet$, in the natural manner. Inevitably we have asynchronous homomorphisms $(f_{\mathsf{aw},1}, f_{\mathsf{am},1}) : wm_1 \to wm$ and $(f_{\mathsf{aw},2}, f_{\mathsf{am},2}) : wm_2 \to wm$ that act in the expected way.

One natural application for an asynchronous pushout, is that of imitating sequential composition of automata. If one identifies a suitable 'final' state of automaton A with the initial state of automaton B, and forms the left asynchronous pushout, nominating the initial state of A as the initial state of the pushout, then the pushout automaton admits a run that reaches the final state of A to continue on into B. However this idea is not completely robust. If the initial state of automaton B has in-transitions and the final state of automaton A has out-transitions, the run may eventually return to the initial state of B and continue back into A once more. A more bulletproof way of modelling sequential composition will be discussed below.

We now give constructions that we call condensations. They can be seen as special cases of the asynchronous pushout constructions.

Definition 4.19 (Worker State Condensation) Let $wor = (I, O, A = (St, Init, Tr))$ be a worker automaton, and let θ_{w} be an equivalence relation on St. We define the condensed worker automaton $wor/\theta_{\mathsf{w}} = (I, O, A/\theta_{\mathsf{w}} = (St/\theta_{\mathsf{w}}, [Init]_{\theta_{\mathsf{w}}}, Tr/\theta_{\mathsf{w}}))$, where

$[Init]_{\theta_w}$ is the equivalence class of *Init* under θ_w, and Tr/θ_w is given by a -*act*-› $b \in Tr$ iff $[a]_{\theta_w}$ -*act*-› $[b]_{\theta_w} \in Tr/\theta_w$.

Thus the state condensation simply goups states together and the transitions are mapped to transitions from the source equivalence class to the target one. It is not hard to see this as (isomorphic to) a special case of the asynchronous worker pushout of two copies of *wor*, wor_1 and wor_2, with respect to a $wor^\bullet$ and asynchronous homomorphisms $f_{aw,1}{}^\bullet : wor^\bullet \to wor_1$ and $f_{aw,2}{}^\bullet : wor^\bullet \to wor_2$, whose structure we sketch next (though the direct construction is easier to comprehend).

The states $St^\bullet$ of $wor^\bullet$ are pairs (a_1, a_2) such that $a_1\ \theta_w\ a_2$. The maps $f_{aw,1}{}^\bullet$ and $f_{aw,2}{}^\bullet$ are the left and right projections on these pairs. The initial state is $(Init_1, Init_2)$. Transitions are inherited componentwise in the usual way.

Definition 4.20 (Manager State Condensation) Let $man = (M, m_I, R)$ be a manager automaton, and let θ_m be an equivalence relation on M. We define the condensed manager automaton $man/\theta_m = (M/\theta_m, [m_I]_{\theta_m}, R/\theta_m)$, where $[m_I]_{\theta_m}$ is the equivalence class of m_I under θ_m, and R/θ_m is given by m -*r*-› $n \in R$ iff $[m]_{\theta_m}$ -*r*-› $[n]_{\theta_m} \in R/\theta_m$. Above each $[m]_{\theta_m}$ in M/θ_m we have the port-channel network:

$$(P_{[m]_{\theta_m}}, C_{[m]_{\theta_m}}) = (\biguplus\{P_m \mid m \in [m]_{\theta_m}\},\ \biguplus\{C_m \mid m \in [m]_{\theta_m}\})$$

where we insist that the union operations are disjoint as previously. Furthermore each transition $[m]_{\theta_m}$ -*r*-› $[n]_{\theta_m}$ of R/θ_m, corresponds to the reconfiguration partial injection:

$$\chi_{[m]_{\theta_m},[n]_{\theta_m}} = \chi_{m,n} \quad \text{if } m \text{ -}r\text{-› } n \in R$$

As above, there is no difficulty in interpreting this as isomorphic to an asynchronous manager pushout construction, and in harmony with that observation, we note that we must explicitly add identity reconfiguration transitions in the form:

$$[m]_{\theta_m} \text{ -id-› } [m]_{\theta_m} = \mathrm{id}_{[m]_{\theta_m}} : C_{[m]_{\theta_m}} \to C_{[m]_{\theta_m}}$$

to make it into a well defined manager.

For determinism reflecting relations θ_m, i.e. ones such that:

$$m, m' \in [m]_{\theta_m},\ n, n' \in [n]_{\theta_m},\ m \text{ -}r\text{-› } n,\ m' \text{ -}r'\text{-› } n' \in R$$
$$\Rightarrow$$
$$m = m',\ n = n',\ r = r' \ \text{ or } \ m \neq m',\ n \neq n',\ r \neq r'$$

there is an alternative construction of some interest, which however is not isomorphic to a special case of asynchronous manager pushout.

Definition 4.21 (Determinism Reflecting Manager State Condensation) Let $man = (M, m_I, R)$ be a manager automaton, and let θ_m be a determinism reflecting equivalence relation on M. We define the determinism reflecting condensed manager automaton $man/_D\theta_m = (M/_D\theta_m, [m_I]_{\theta_m}, R/_D\theta_m)$, in which $M/_D\theta_m = M/\theta_m$, $[m_I]_{\theta_m}$ is the equivalence class of m_I under θ_m, and $R/_D\theta_m$ is given by the equivalence m -*r*-› $n \in R$ iff $[m]_{\theta_m}$ -**R**-› $[n]_{\theta_m} \in R/_D\theta_m$, where $\mathbf{R} = \bigcup\{r \mid m \text{ -}r\text{-› } n \in R,\ m \in [m]_{\theta_m},\ n \in [n]_{\theta_m}\}$, i.e. we accumulate all reconfiguration transitions between states in $[m]_{\theta_m}$ and $[n]_{\theta_m}$ to

build a transition of $R/_{D}\theta_{\mathsf{m}}$. Above each $[m]_{\theta_{\mathsf{m}}}$ in $M/_{D}\theta_{\mathsf{m}}$ we have the port-channel network:

$$(P_{[m]_{\theta_{\mathsf{m}}}}, C_{[m]_{\theta_{\mathsf{m}}}}) = (\biguplus\{P_m \mid m \in [m]_{\theta_{\mathsf{m}}}\},\ \biguplus\{C_m \mid m \in [m]_{\theta_{\mathsf{m}}}\})$$

where we need the union operations to be disjoint as always. Furthermore each transition $[m]_{\theta_{\mathsf{m}}}$ -**R**-› $[n]_{\theta_{\mathsf{m}}}$ of $R/_{D}\theta_{\mathsf{m}}$, corresponds to the reconfiguration partial injection:

$$\chi_{[m]_{\theta_{\mathsf{m}}},[n]_{\theta_{\mathsf{m}}}} = \bigcup\{\chi_{m,n} \mid m \in [m]_{\theta_{\mathsf{m}}},\ n \in [n]_{\theta_{\mathsf{m}}}\}$$

which are well defined by the determinism reflecting property. This time, the required identities come for free, as is easy enough to see.

The alert reader may be wondering why not, instead of insisting on the determinism reflecting property, to define a transition $[m]_{\theta_{\mathsf{m}}}$ -**R**-› $[n]_{\theta_{\mathsf{m}}}$ we did not simply consider a collection of individual transitions m -r-› $n \in R$ that made the union definition of $\chi_{[m]_{\theta_{\mathsf{m}}},[n]_{\theta_{\mathsf{m}}}}$ unproblematic. For given $[m]_{\theta_{\mathsf{m}}}$ and $[n]_{\theta_{\mathsf{m}}}$, one could have taken the set of these possibilities as the family of transitions from $[m]_{\theta_{\mathsf{m}}}$ to $[n]_{\theta_{\mathsf{m}}}$. The answer to this will come below.

Definition 4.22 (Worker-Manager State Condensation) Let $wm = wor \otimes man = (I, O, A = (St, Init, Tr)) \otimes (M, m_{\mathrm{I}}, R)$ be a worker-manager automaton, and let θ_{w} and θ_{m} be equivalence relations on St and M respectively. Then we define the condensed worker-manager automaton $wm/(\theta_{\mathsf{w}},\theta_{\mathsf{m}}) = wor/\theta_{\mathsf{w}} \otimes man/\theta_{\mathsf{m}}$ as the asynchronous product of the condensed worker automaton wor/θ_{w} and the condensed manager automaton man/θ_{m}.

Definition 4.23 (Determinism Reflecting Worker-Manager State Condensation) Let $wm = wor \otimes man = (I, O, A = (St, Init, Tr)) \otimes (M, m_{\mathrm{I}}, R)$ be a worker-manager automaton, and let θ_{w} and θ_{m} be equivalence relations on St and M respectively with θ_{m} determinism reflecting. Then we define the determinism reflecting condensed worker-manager automaton $wm/_{D}(\theta_{\mathsf{w}},\theta_{\mathsf{m}}) = wor/\theta_{\mathsf{w}} \otimes man/_{D}\theta_{\mathsf{m}}$ as the asynchronous product of the condensed worker automaton wor/θ_{w} and the determinism reflecting condensed manager automaton $man/_{D}\theta_{\mathsf{m}}$.

The preceding completes the description of our automata-centred notions. We note that these featured at times disjoint unions and at other times normal ones, and we consider here the significance of the two different kinds. While mathematically there is no special significance one way or the other, the two types of union having slightly different theoretical properties, the difference becomes more acute if we suppose that we are dealing with mathematical models of actual computing systems. In the real world distinct systems have a tendency to retain their distinct identities unless one takes active steps to obscure them. This makes disjoint union the more natural notion. However one can understand conventional union as arising from a disjoint union via the identification, under a partial equivalence relation, of distinct copies of 'the same thing'. This is just a pushout in Set. In the real world one would have to construct some aparatus in order to implement the identification, but in general this is feasible. It is on this reading of conventional union (i.e. the tacit assumption of the

existence of the requisite partial equivalence relation), that the rest of this paper should be understood.

4.2 Contructions Centred on Systems

The next definition enables two IWIM systems to be brought together into one, and to work alongside one another.

Definition 4.24 (Asynchronous Product of Systems) Let WM_1, WM_2 be disjoint IWIM systems. We define the asynchronous product IWIM system $WM_1 \otimes WM_2$ as follows. Its set of automaton names is $WM_1 \cup WM_2$. Similarly, all the other components are given by (disjoint) unions. Thus $worman_\otimes = worman_1 \cup worman_2$; $\wedge_\otimes = \wedge_1 \cup \wedge_2$; $r_\otimes = r_1 \cup r_2$; $\lambda_\otimes = \lambda_1 \cup \lambda_2$. A configuration of $WM_1 \otimes WM_2$ is of the form $(sts_1 \cup sts_2, qs_1 \cup qs_2)$, which, because of the disjointness of WM_1 and WM_2, can be decomposed into a configuration (sts_1, qs_1) of WM_1 and a configuration (sts_2, qs_2) of WM_2. Among these configurations, the initial configurations are those configurations $(sts_{I,1} \cup sts_{I,2}, qs_{I,1} \cup qs_{I,2})$ of $WM_1 \otimes WM_2$ built out of initial configurations $(sts_{I,1}, qs_{I,1})$ of WM_1 and $(sts_{I,2}, qs_{I,2})$ of WM_2. Finally, the dynamics of $WM_1 \otimes WM_2$ is easily given by the following rules:

$$\frac{(sts_1, qs_1) \longrightarrow (sts_1', qs_1') \ ; \ (sts_2, qs_2) \text{ a config of } WM_2}{(sts_1 \cup sts_2, qs_1 \cup qs_2) \longrightarrow (sts_1' \cup sts_2, qs_1' \cup qs_2)}$$

and

$$\frac{(sts_1, qs_1) \text{ a config of } WM_1 \ ; \ (sts_2, qs_2) \longrightarrow (sts_2', qs_2')}{(sts_1 \cup sts_2, qs_1 \cup qs_2) \longrightarrow (sts_1 \cup sts_2', qs_1 \cup qs_2')}$$

We see that the transitions of the asynchronous product are the individual transitions of the component systems interpreted in the context of the product system. The two components thus evolve independently of one another. This property leads to a total surjective relation between pairs of runs of WM_1 and WM_2, and runs of $WM_1 \otimes WM_2$ given by arbitrarily interleaving the steps of the run of WM_1 and the run of WM_2. The states of the two runs are just combined in union in the obvious way. Thus if we have for WM_1: $(sts_1, qs_1) \longrightarrow (sts_1', qs_1') \longrightarrow (sts_1'', qs_1'') \longrightarrow \ldots$, and for WM_2 we have: $(sts_2, qs_2) \longrightarrow (sts_2', qs_2') \longrightarrow (sts_2'', qs_2'') \longrightarrow \ldots$, then one possible interleaving yields for $WM_1 \otimes WM_2$: $(sts_1 \cup sts_2, qs_1 \cup qs_2) \longrightarrow (sts_1' \cup sts_2, qs_1' \cup qs_2) \longrightarrow (sts_1' \cup sts_2', qs_1' \cup qs_2') \longrightarrow (sts_1' \cup sts_2'', qs_1' \cup qs_2'') \longrightarrow \ldots$. One consequence of this structure is that the converse relation, from runs of $WM_1 \otimes WM_2$ to pairs of runs of WM_1 and WM_2, is a pair of projections, given by simply striking out all WM_2 steps and portions of state/queue sets to get the WM_1 run, and striking out all WM_1 steps and portions of state/queue sets to get the WM_2 run.

Corresponding to the product notion we have a sum notion. This is less pointless than the corresponding notion for automata for reasons indicated below.

Definition 4.25 (Left and Right Asynchronous Sum of Systems) Let WM_1, WM_2 be disjoint IWIM systems. We define the the left and right asynchronous sum IWIM systems $WM_1 \leftthreetimes WM_2$ and $WM_1 \rightthreetimes WM_2$ respectively, exactly as we do asynchronous

products, *except* for the initial configurations. Instead, an initial configuration of $WM_1 \langle\oplus WM_2$ is of the form $(sts_{\mathrm{I},1} \cup \varnothing_2, qs_{\mathrm{I},1} \cup \varnothing_2)$ with $(sts_{\mathrm{I},1}, qs_{\mathrm{I},1})$ initial in WM_1, while an initial configuration of $WM_1 \oplus\rangle WM_2$ is of the form $(\varnothing_1 \cup sts_{\mathrm{I},2}, \varnothing_1 \cup qs_{\mathrm{I},2})$ with $(sts_{\mathrm{I},2}, qs_{\mathrm{I},2})$ initial in WM_2.

Given the decoupled way that the dynamics of the two components in $WM_1 \langle\oplus WM_2$ and $WM_1 \oplus\rangle WM_2$ (and in $WM_1 \otimes WM_2$ also) evolve, it is clear that in $WM_1 \langle\oplus WM_2$ the WM_2 component is inactive, since the WM_2 component of an initial configurations of $WM_1 \oplus\rangle WM_2$ is $\varnothing$ and consequently remains $\varnothing$ throughout any run. In $WM_1 \oplus\rangle WM_2$ the roles of WM_1 and WM_2 are reversed, and it is WM_1 that is useless. What makes the definitions of $WM_1 \langle\oplus WM_2$ and $WM_1 \oplus\rangle WM_2$ (and to an extent $WM_1 \otimes WM_2$ also) not purposeless, is the fact that by using constructions from the preceding subsection on the automata in the asynchronous sum or product, the inactive part may be nontrivially coupled to the active one. For this to work in a well defined way we need to check appropriate conditions for each of the constructions. The rest of this section states, in the form of a series of propositions, sufficient conditions under which application of these various constructions keeps an IWIM system well defined. As one might imagine when working with sufficient conditions, these are not unique, and we restrict ourselves to relatively straightforward ones not requiring fixed point constructions, in keeping with the rest of the paper.

First we need some notation. Let R be a relation from A to B, i.e. $R \subseteq A \times B$, and $D \subseteq A$, $E \subseteq B$. Then we define:

$$D \lhd R = R - D \times B$$
$$R \rhd E = R - A \times E$$

Proposition 4.26 (Worker-Manager Pullbacks in Systems) Let WM be an IWIM system, and let $wm_1 = (I, O, A_1) \otimes (M_1, m_{\mathrm{I},1}, R_1)$ and $wm_2 = (I, O, A_2) \otimes (M_2, m_{\mathrm{I},2}, R_2)$ be worker-manager automata of WM. Let $wm^\bullet$ be another worker-manager automaton, and $(f_{\mathrm{w},1}{}^\bullet, f_{\mathrm{m},1}{}^\bullet) : wm_1 \to wm^\bullet$, $(f_{\mathrm{w},2}{}^\bullet, f_{\mathrm{m},2}{}^\bullet) : wm_2 \to wm^\bullet$ be two worker-manager homomorphisms. Let $wm = (I, O, A) \otimes (M, m_{\mathrm{I}}, R)$ be the worker-manager pullback of wm_1 and wm_2 with respect to $(f_{\mathrm{w},1}{}^\bullet, f_{\mathrm{m},1}{}^\bullet)$ and $(f_{\mathrm{w},2}{}^\bullet, f_{\mathrm{m},2}{}^\bullet)$ with attendant projections $f_{\mathrm{wm},1} : wm \to wm_1$ and $f_{\mathrm{wm},2} : wm \to wm_2$. Suppose the following hold:

(1) For $i \neq j \in \{1, 2\}$, $wm_i{}^\wedge m'_{wm'} \Rightarrow \neg\, wm_j{}^\wedge m'_{wm'}$.

(2) For $i \neq j \in \{1, 2\}$, $\neg\, wm_j{}^\wedge m_{wm_i}$ for any $m \in M_i$.

(3) For $i \in \{1, 2\}$, $wm'{}^\wedge m_{wm_i} \Rightarrow f_{\mathrm{m},i}{}^\bullet(m) \in M^{\bullet\cap}$.

(4) For $(m_1, m_2) \in M$, $wm'{}^\wedge m_{1,wm_1} \Leftrightarrow wm'{}^\wedge m_{2,wm_2}$, and
$f_{\mathrm{m},1}{}^\bullet \circ r_{wm'{}^\wedge m_{1,wm_1}} = f_{\mathrm{m},2}{}^\bullet \circ r_{wm'{}^\wedge m_{2,wm_2}}$

Then $WM^* = (WM - \{wm_1, wm_2\}) \cup \{wm\}$ with ancillary data given by:

$$\begin{aligned} \wedge^* = {} & (\{wm_1, wm_2\} \lhd \wedge \rhd (M_{1,wm_1} \cup M_{2,wm_2})) \cup \\ & \{wm^\wedge{}^* m'_{wm'} \mid wm_i{}^\wedge m'_{wm'}, i \in \{1, 2\}\} \cup \\ & \{wm'{}^\wedge{}^*(m_1, m_2)_{wm} \mid (m_1, m_2) \in M, wm'{}^\wedge m_{1wm_1}, wm'{}^\wedge m_{2wm_2}\} \end{aligned}$$

$$\begin{aligned}
\lambda^* \;=\; & ((M_{1,wm_1} \cup M_{2,wm_2}) \lhd \lambda \rhd (IO_{wm_1} = IO_{wm_2})) \cup \\
& \{\lambda^*_{m'_{wm'}}(p) = io \in IO_{wm} \mid \lambda_{m'_{wm'}}(p) = io \in IO_{wm_i},\ i \in \{1, 2\}\} \cup \\
& \{\lambda^*_{(m_1,\, m_2)_{wm}}(p) = io \in IO_{wm'} \mid (m_1, m_2) \in M,\ p \in P_{m_{1wm_1}} = P_{m_{2wm_2}},\\
& \qquad \lambda_{m_{1wm_1}}(p) = \lambda_{m_{2wm_2}}(p) = io \in IO_{wm'}\}
\end{aligned}$$

$$\begin{aligned}
r^* \;=\; & ((Rec_1 \cup Rec_2) \lhd r \rhd (R_{1,wm_1} \cup R_{2,wm_2})) \cup \\
& \{r^*_{wm\hat{}*m'_{wm'}}(rec) = m'_{wm'} \text{ -}r\text{-}\!\rightarrow n'_{wm'} \mid \\
& \qquad r_{wm_i\hat{}m'_{wm'}}(rec) = m'_{wm'} \text{ -}r\text{-}\!\rightarrow n'_{wm'},\ i \in \{1, 2\}\} \cup \\
& \{r^*_{wm'\hat{}*(m_1,\, m_2)_{wm}}(rec) = (m_1, m_2)_{wm} \text{ -}r\text{-}\!\rightarrow (n_1, n_2)_{wm} \mid \\
& \qquad r_{wm'\hat{}m_{1wm_1}}(rec) = m_{1,wm_1} \text{ -}r\text{-}\!\rightarrow n_{1,wm_1},\\
& \qquad r_{wm'\hat{}m_{2wm_2}}(rec) = m_{2,wm_2} \text{ -}r\text{-}\!\rightarrow n_{2,wm_2},\\
& \qquad f_{\mathrm{m},1}{}^{\bullet}(m_{1,wm_1} \text{ -}r\text{-}\!\rightarrow n_{1,wm_1}) = f_{\mathrm{m},2}{}^{\bullet}(m_{2,wm_2} \text{ -}r\text{-}\!\rightarrow n_{2,wm_2})\}
\end{aligned}$$

$$\begin{aligned}
Initial_{WM^*} \;=\; & (sts^*, qs^*) \\
& \text{where} \\
& \qquad sts^* = \begin{cases} (sts - INIS) \cup \{(Init_{wm}, m_{\mathrm{I},wm})\} & \text{if } sts \cap INIS \neq \emptyset \\ sts & \text{otherwise} \end{cases} \\
& \qquad qs^* = \begin{cases} (qs - INIQ) \cup \{d{:}[\,] \mid d \in C_{m_{\mathrm{I},wm}}\} & \text{if } sts \cap INIS \neq \emptyset \\ qs & \text{otherwise} \end{cases} \\
& \text{and where} \quad INIS = \{(Init_{wm_1}, m_{\mathrm{I},wm_1}), (Init_{wm_2}, m_{\mathrm{I},wm_2})\}, \\
& \qquad\qquad\quad INIQ = \{d{:}[\,] \mid d \in C_{m_{\mathrm{I},1wm_1}} \cup C_{m_{\mathrm{I},2wm_2}}\}, \\
& \qquad\qquad\quad Initial_{WM} = (sts, qs)
\end{aligned}$$

is a well defined IWIM system.

Proof. It is sufficient to check four things. First, that $\hat{}*$ is well defined. For this we observe that replacing $wm_i\hat{}m'_{wm'}$ with $wm\hat{}*m'_{wm'}$ is well defined since (2) guarantees that wm' can never be wm_j. Likewise, replacing $wm'\hat{}m_{1wm_1}$ and $wm'\hat{}m_{2wm_2}$ by $wm'\hat{}*(m_1, m_2)_{wm}$ for pairs (m_1, m_2) is well defined since (2) guarantees that wm' can never be wm_i or wm_j, (3) guarantees that any m_{1wm_1} or m_{2wm_2} below wm' gets paired in the construction of the manager pullback, and (4) guarantees that wm' is above one of m_{1wm_1} or m_{2wm_2} iff it is above the other.

Second, that λ^* is a bijection. For this we see that replacing $\lambda_{m'_{wm'}}(p) = io \in IO_{wm_i}$ by the corresponding $io \in IO_{wm}$ is well defined since (2) guarantees that wm' can never be wm_i or wm_j, (1) guarantees that at most one of them is above wm', and the pullback construction guarantees that $IO_{wm_i} = IO_{wm}$. Likewise, mapping $\lambda^*_{(m_1,\, m_2)_{wm}}(p)$ to $io \in IO_{wm'}$ whenever both $\lambda_{m_{1wm_1}}(p)$ and $\lambda_{m_{2wm_2}}(p)$ map to it is sound since (2) guarantees that wm' can never be wm_1 or wm_2, (3) guarantees that any m_{1wm_1} or m_{2wm_2} below wm' gets paired, (4) guarantees that wm' is above one of m_{1wm_1} or m_{2wm_2} iff it is above the other, and the pullback construction guarantees that m_{1wm_1} and m_{2wm_2} (and hence $(m_1, m_2)_{wm}$) have the same port channel network. (N.B. In the definition of λ^* we used the notation $\ldots \lambda \rhd (IO_{wm_1} = IO_{wm_2})) \cup \ldots$ with the obvious interpretation, for emphasis. Similarly below.)

Third, r^* is a function. On the one hand, any *rec* transition of $wor(wm)$ comes from *rec* transitions in wm_1 and wm_2, exactly one of which will have an $r_{wm_i\hat{}m'_{wm'}}$ image by (1); so mapping the $wor(wm)$ transition in the same way under $r^*_{wm\hat{}*m'_{wm'}}$ is well

defined. On the other hand, by (3) any *rec* transition of a wm' above any m_{wm_i}, is above an m that forms a pair (m_1, m_2) in the pullback. By (4) the $f_{\mathsf{m},1}{}^{\bullet} \circ r_{wm'^{\wedge}m_{1,wm_1}}$ and $f_{\mathsf{m},2}{}^{\bullet} \circ r_{wm'^{\wedge}m_{2,wm_2}}$ images of this *rec* transition will coincide in $wm^{\bullet}$; therefore we get a unique $(m_1, m_2)_{wm}$ -*r*-› $(n_1, n_2)_{wm}$ reconfiguration transition in *wm* to which to map the *rec* transition in wm'.

Finally, if the initial state of either of wm_1, wm_2 is in the *sts* component of $Initial_{WM}$, then the wm_i in question must either have a nontrivial manager facet, or be above some $m_{\mathrm{I},wm'}$ with $wm' \in WM^{\#}$, by the conditions for initial configurations. In such a case the *sts** component of $Initial_{WM^*}$ must contain the $(Init_{wm}, m_{\mathrm{I},wm})$ state of *wm* to satisfy the same conditions; otherwise not. For the initial queues, we merely replace any queues belonging to wm_1, wm_2 with ones for *wm* as required. ☺

Proposition 4.27 (Worker-Manager Pushouts in Systems) Let *WM* be an IWIM system, and let $wm_1 = (I, O, A_1) \otimes (M_1, m_{\mathrm{I},1}, R_1)$ and $wm_2 = (I, O, A_2) \otimes (M_2, m_{\mathrm{I},2}, R_2)$ be worker-manager automata of *WM*. Let $wm^{\bullet}$ be another worker-manager automaton, and $(f_{\mathsf{w},1}{}^{\bullet}, f_{\mathsf{m},1}{}^{\bullet}) : wm^{\bullet} \to wm_1$, $(f_{\mathsf{w},2}{}^{\bullet}, f_{\mathsf{m},2}{}^{\bullet}) : wm^{\bullet} \to wm_2$ be two worker-manager homomorphisms. Let $wm = (I, O, A) \otimes (M, m_{\mathrm{I}}, R)$ be the worker-manager pushout of wm_1 and wm_2 with respect to $(f_{\mathsf{w},1}{}^{\bullet}, f_{\mathsf{m},1}{}^{\bullet})$ and $(f_{\mathsf{w},2}{}^{\bullet}, f_{\mathsf{m},2}{}^{\bullet})$ with attendant homomorphisms $f_{\mathsf{wm},1} : wm_1 \to wm$ and $f_{\mathsf{wm},2} : wm_2 \to wm$. Suppose the following hold:

(1) For $i \neq j \in \{1, 2\}$, $wm_i{}^{\wedge}m'_{wm'} \Rightarrow \neg\, wm_j{}^{\wedge}m'_{wm'}$.

(2) For $i \neq j \in \{1, 2\}$, $\neg\, wm_j{}^{\wedge}m_{wm_i}$ for any $m \in M_i$.

(3) For $i, j \in \{1, 2\}$, $m_1, m_2 \in [m]_{\mathsf{m}} \in M$, $wm'{}^{\wedge}m_{1,wm_i} \Leftrightarrow wm'{}^{\wedge}m_{2,wm_j}$ and $f_{\mathsf{m},i} \circ r_{wm'^{\wedge}m_{1,wm_i}} = f_{\mathsf{m},j} \circ r_{wm'^{\wedge}m_{2,wm_j}}$

(4) $Rec_1 = Rec_2$.

Then $WM^* = (WM - \{wm_1, wm_2\}) \cup \{wm\}$ with ancillary data given by:

$$
\begin{aligned}
{}^{\wedge}* \;=\; & (\{wm_1, wm_2\} \;\lhd\, {}^{\wedge} \,\rhd\; (M_{1,wm_1} \cup M_{2,wm_2})) \cup \\
& \{wm^{\wedge *} m'_{wm'} \mid wm_i{}^{\wedge}m'_{wm'},\ i \in \{1, 2\}\} \cup \\
& \{wm'{}^{\wedge *}[m]_{\mathsf{m},wm} \mid [m]_{\mathsf{m}} \in M,\ (wm'{}^{\wedge}m_{wm_1} \text{ or } wm'{}^{\wedge}m_{wm_2})\}
\end{aligned}
$$

$$
\begin{aligned}
\lambda^* \;=\; & ((M_{1,wm_1} \cup M_{2,wm_2}) \;\lhd\, \lambda \,\rhd\; (IO_{wm_1} = IO_{wm_2})) \cup \\
& \{\lambda^*_{m'_{wm'}}(p) = io \in IO_{wm} \mid \lambda_{m'_{wm'}}(p) = io \in IO_{wm_i},\ i \in \{1, 2\}\} \cup \\
& \{\lambda^*_{[m]_{\mathsf{m},wm}}(p) = io \in IO_{wm'} \mid [m]_{\mathsf{m}} \in M,\ p \in P_{m_{wm_i}}, \\
& \qquad\qquad \lambda_{m_{wm_i}}(p) = io \in IO_{wm'},\ i \in \{1, 2\}\}
\end{aligned}
$$

$$
\begin{aligned}
r^* \;=\; & ((Rec_1 \cup Rec_2) \;\lhd\, r \,\rhd\; (R_{1,wm_1} \cup R_{2,wm_2})) \cup \\
& \{r^*_{wm^{\wedge *} m'_{wm'}}(rec) = m'_{wm'} \text{ -}r\text{-›} n'_{wm'} \mid \\
& \qquad r_{wm_i{}^{\wedge}m'_{wm'}}(rec) = m'_{wm'} \text{ -}r\text{-›} n'_{wm'},\ i \in \{1, 2\}\} \cup \\
& \{r^*_{wm'{}^{\wedge *}[m]_{\mathsf{m},wm}}(rec) = [m]_{\mathsf{m},wm} \text{ -}r\text{-›} [n]_{\mathsf{m},wm} \mid \\
& \qquad r_{wm'{}^{\wedge}m_{wm_i}}(rec) = m_{wm_i} \text{ -}r\text{-›} n_{wm_i},\ i \in \{1, 2\}\}
\end{aligned}
$$

$$
\begin{aligned}
Initial_{WM^*} \;=\; & (sts^*, qs^*) \\
& \text{where} \\
& \qquad sts^* = \begin{cases} (sts - INIS) \cup \{(Init_{wm}, m_{\mathrm{I},wm})\} & \text{if } sts \cap INIS \neq \varnothing \\ sts & \text{otherwise} \end{cases}
\end{aligned}
$$

$$qs^* = \begin{cases} (qs - INIQ) \cup \{d{:}[\,] \mid d \in C_{m_{\mathrm{I},wm}}\} & \text{if } sts \cap INIS \neq \emptyset \\ qs & \text{otherwise} \end{cases}$$

and where $INIS = \{(Init_{wm_1}, m_{\mathrm{I},wm_1}), (Init_{wm_2}, m_{\mathrm{I},wm_2})\}$,
$INIQ = \{d{:}[\,] \mid d \in C_{m_{\mathrm{I},1wm_1}} \cup C_{m_{\mathrm{I},2wm_2}}\}$,
$Initial_{WM} = (sts, qs)$

is a well defined IWIM system.

Proof. There are four things to establish. First, that $\wedge^*$ is well defined. Neither of wm_1 or wm_2 is above the other by (2). Therefore it is sufficient to replace $wm_i \wedge m'_{wm'}$ with $wm \wedge^* m'_{wm'}$. Likewise replacing $wm' \wedge m_{1wm_1}$ or $wm' \wedge m_{2wm_2}$ by $wm' \wedge^* [m]_{\mathsf{m},wm}$ is well defined since (2) ensures that wm' can never be wm_1 or wm_2, and (3) guarantees that whenever wm' is above some m_{1wm_1} or m_{2wm_2} contributing to $[m]_{\mathsf{m}}$, then it is above all such $m \in [m]_{\mathsf{m}}$.

Second, that λ^* is a bijection. We see that replacing $\lambda_{m'_{wm'}}(p) = io \in IO_{wm_i}$ by the corresponding $io \in IO_{wm}$ is well defined since (2) guarantees that wm' can never be wm_i or wm_j, (1) guarantees that at most one of them is above wm', and the pushout construction guarantees that $IO_{wm_i} = IO_{wm}$. Likewise, mapping $\lambda^*_{[m]_{\mathsf{m},wm}}(p)$ to $io \in IO_{wm'}$ whenever $\lambda_{m_{wm_i}}(p)$ maps to it for some $m_{wm_i} \in [m]_{\mathsf{m},wm}$ is sound, since (2) guarantees that wm' can never be wm_1 or wm_2, (3) guarantees that wm' is above all $m_{wm_i} \in [m]_{\mathsf{m},wm}$ or none of them, and the pushout construction guarantees that all of them (and hence $[m]_{\mathsf{m},wm}$) have the same port channel network.

Third, r^* is a function. On the one hand, any *rec* transition of $wor(wm)$ comes from a *rec* transition in either wm_1 or wm_2 (or both), and for exactly one of these will an $r_{wm_i \wedge m'_{wm'}}$ be defined by (1). By (4), the sets of *rec* events of $wor(wm_1)$ and $wor(wm_2)$ are equal, so that a *rec* event of $wor(wm)$ will be in the domain of either $r_{wm_i \wedge m'_{wm'}}$, making the definition of $r^*_{wm \wedge^* m'_{wm'}}(rec)$ unambiguous. On the other hand, any *rec* transition of a wm' above any m_{wm_i}, either ends up above a singleton $[m]_{\mathsf{m}}$ in wm, in which case the replacement of $r_{wm' \wedge m_{wm_i}}(rec)$ by $r^*_{wm' \wedge^* [m]_{\mathsf{m},wm}}(rec)$ is immediately unambiguous, or not. If not, we know by (3) that all the $r_{wm' \wedge m_{wm_i}}(rec)$ map via $f_{\mathsf{m},i}$ to the same wm reconfiguration transition $[m]_{\mathsf{m},wm}$ -r-› $[n]_{\mathsf{m},wm}$, making the replacement unambiguous also. Finally, for the initial configurations, the argument is as in the previous proposition. ☺

It is clear that in the preceding constructions some fairly demanding condition have to hold. For greater flexibility with pullbacks and pushouts, we now consider their asynchronous analogues.

Proposition 4.28 ((Left and Right) Asynchronous Worker-Manager Pullbacks in Systems) Let WM be an IWIM system, and let $wm_1 = (I, O, A_1) \otimes (M_1, m_{\mathrm{I},1}, R_1)$ and $wm_2 = (I, O, A_2) \otimes (M_2, m_{\mathrm{I},2}, R_2)$ be worker-manager automata of WM. Let $wm^\bullet$ be another worker-manager automaton, and $(f_{\mathsf{aw},1}{}^\bullet, f_{\mathsf{am},1}{}^\bullet) : wm_1 \to wm^\bullet$, $(f_{\mathsf{aw},2}{}^\bullet, f_{\mathsf{am},2}{}^\bullet) : wm_2 \to wm^\bullet$ be two asynchronous worker-manager homomorphisms. Let $wm = (I, O, A) \otimes (M, m_{\mathrm{I}}, R)$ be the left or right asynchronous worker-manager pullback of wm_1 and wm_2 with respect to $(f_{\mathsf{aw},1}{}^\bullet, f_{\mathsf{am},1}{}^\bullet)$ and $(f_{\mathsf{aw},2}{}^\bullet, f_{\mathsf{am},2}{}^\bullet)$ with attendant projections $f_{\mathsf{awm},1} : wm \to wm_1$ and $f_{\mathsf{awm},2} : wm \to wm_2$. Suppose the following hold:

(1) For $i \neq j \in \{1, 2\}$, $wm_i{}^\wedge m'_{wm'} \Rightarrow \neg\, wm_j{}^\wedge m'_{wm'}$.

(2) For $i \neq j \in \{1, 2\}$, $\neg\, wm_j{}^\wedge m_{wm_i}$ for any $m \in M_i$.

(3) For $i \in \{1, 2\}$, $wm'{}^\wedge m_{wm_i} \Rightarrow f_{\mathsf{am},i}{}^\bullet(m) \in M^{\bullet\frown}$.

(4) For $i \neq j \in \{1, 2\}$, $wm'{}^\wedge m_{i,wm_i} \Rightarrow \neg\, wm'{}^\wedge m_{j,wm_j}$.

Then $WM^* = (WM - \{wm_1, wm_2\}) \cup \{wm\}$ with ancillary data given by:

$$
\begin{aligned}
{}^\wedge{}^* \;=\; & (\{wm_1, wm_2\} \lhd {}^\wedge \rhd (M_{1,wm_1} \cup M_{2,wm_2})) \cup \\
& \{wm{}^\wedge{}^* m'_{wm'} \mid wm_i{}^\wedge m'_{wm'},\ i \in \{1, 2\}\} \cup \\
& \{wm'{}^\wedge{}^*(m_1, m_2)_{wm} \mid (m_1, m_2) \in M,\ (wm'{}^\wedge m_{1wm_1} \text{ or } wm'{}^\wedge m_{2wm_2})\}
\end{aligned}
$$

$$
\begin{aligned}
\lambda^* \;=\; & ((M_{1,wm_1} \cup M_{2,wm_2}) \lhd \lambda \rhd (IO_{wm_1} = IO_{wm_2})) \cup \\
& \{\lambda^*_{m'_{wm'}}(p) = io \in IO_{wm} \mid \lambda_{m'_{wm'}}(p) = io \in IO_{wm_i},\ i \in \{1, 2\}\} \cup \\
& \{\lambda^*_{(m_1, m_2)_{wm}}(p) = io \in IO_{wm'} \mid (m_1, m_2) \in M,\ p \in P_{m_{1wm_1}} \uplus P_{m_{2wm_2}}, \\
& \quad (wm'{}^\wedge m_{1wm_1},\ \lambda_{m_{1wm_1}}(p) = io \in IO_{wm'} \text{ or} \\
& \quad\ wm'{}^\wedge m_{2wm_2},\ \lambda_{m_{2wm_2}}(p) = io \in IO_{wm'})\}
\end{aligned}
$$

$$
\begin{aligned}
r^* \;=\; & ((Rec_1 \cup Rec_2) \lhd r \rhd (R_{1,wm_1} \cup R_{2,wm_2})) \cup \\
& \{r^*_{wm{}^\wedge{}^* m'_{wm'}}(rec) = m'_{wm'} \text{-}r\text{-}\rangle\, n'_{wm'} \mid \\
& \quad r_{wm_i{}^\wedge m'_{wm'}}(rec) = m'_{wm'} \text{-}r\text{-}\rangle\, n'_{wm'},\ i \in \{1, 2\}\} \cup \\
& \{r^*_{wm'{}^\wedge{}^*(m_1, m_2)_{wm}}(rec) = (m_1, m_2)_{wm} \text{-}r\text{-}\rangle\, (n_1, m_2)_{wm} \mid \\
& \quad r_{wm'{}^\wedge m_{1wm_1}}(rec) = m_{1,wm_1} \text{-}r\text{-}\rangle\, n_{1,wm_1}\} \cup \\
& \{r^*_{wm'{}^\wedge{}^*(m_1, m_2)_{wm}}(rec) = (m_1, m_2)_{wm} \text{-}r\text{-}\rangle\, (m_1, n_2)_{wm} \mid \\
& \quad r_{wm'{}^\wedge m_{2wm_2}}(rec) = m_{2,wm_2} \text{-}r\text{-}\rangle\, n_{2,wm_2}\}
\end{aligned}
$$

$$
\begin{aligned}
Initial_{WM^*} \;=\; & (sts^*, qs^*) \\
& \text{where} \\
& \quad sts^* = \begin{cases} (sts - INIS) \cup \{(Init_{wm}, m_{\mathrm{I},wm})\} & \text{if } sts \cap INIS \neq \varnothing \\ sts & \text{otherwise} \end{cases} \\
& \quad qs^* \;=\; \begin{cases} (qs - INIQ) \cup \{d{:}[\,] \mid d \in C_{m_{\mathrm{I},wm}}\} & \text{if } sts \cap INIS \neq \varnothing \\ qs & \text{otherwise} \end{cases} \\
& \text{and where} \quad INIS = \{(Init_{wm_1}, m_{\mathrm{I},wm_1}), (Init_{wm_2}, m_{\mathrm{I},wm_2})\}, \\
& \qquad\qquad\ INIQ = \{d{:}[\,] \mid d \in C_{m_{\mathrm{I},1wm_1}} \cup C_{m_{\mathrm{I},2wm_2}}\}, \\
& \qquad\qquad\ Initial_{WM} = (sts, qs)
\end{aligned}
$$

is a well defined IWIM system.

Proof. As usual there are four things to establish. First, that ${}^\wedge{}^*$ is well defined. Neither of wm_1 or wm_2 is above the other by (2). Therefore it is sufficient to replace $wm_i{}^\wedge m'_{wm'}$ with $wm{}^\wedge{}^* m'_{wm'}$. Likewise, replacing $wm'{}^\wedge m_{1wm_1}$ by $wm'{}^\wedge{}^*(m_1, m_2)_{wm}$ for all m_2 such that (m_1, m_2) is a state of $man(wm)$ is well defined since (3) guarantees that any m_{1wm_1} or m_{2wm_2} below wm' gets paired in the construction of the manager pullback.

Second, that λ^* is a bijection. Given a management state $(m_1, m_2)_{wm}$ of wm, then with the definition of ${}^\wedge{}^*$, $\lambda^*_{(m_1, m_2)_{wm}}$ becomes the disjoint union of $\lambda_{m_{1wm_1}}$ and

$\lambda_{m_{2wm_2}}$. This succeeds since (3) guarantees that any management states m_{1wm_1} or m_{2wm_2} with nonempty communication network get paired, and (4) ensures that the families of workers above any m_{1wm_1} and m_{2wm_2} are disjoint, so that the disjoint union of the bijections is a bijection. Also by (1) at most one of wm_1, wm_2 is above any $m'_{wm'}$, so that replacing any $io \in IO_{wm_i}$ in the range of $\lambda_{m'_{wm'}}(p)$ by the corresponding $io \in IO_{wm}$ generates no problems.

Third, r^* is a function. By (1) again, for at most one $i \in \{1, 2\}$ does $r_{wm_i \wedge m'_{wm'}}$ exist. Thus defining the $r^*_{wm \wedge * m'_{wm'}}$ image of a *rec* event accordingly is sound. Also, given some $(m_1, m_2)_{wm}$, replacing the $r_{wm' \wedge m_{1wm_1}}$ or $r_{wm' \wedge m_{2wm_2}}$ image of *rec* by the reconfiguration transition $(m_1, m_2)_{wm}$ -*rec*-› $(n_1, m_2)_{wm}$ or $(m_1, m_2)_{wm}$ -*rec*-› $(m_1, n_2)_{wm}$ respectively, is well defined because (4) ensures that exactly one of these cases exists (thus making $r^*_{wm' \wedge *(m_1, m_2)_{wm}}$ single valued). Finally for the initial configurations, the argument is as in previous cases. ☺

Proposition 4.29 ((Left and Right) Asynchronous Worker-Manager Pushouts in Systems) Let *WM* be an IWIM system, and let $wm_1 = (I, O, A_1) \otimes (M_1, m_{I,1}, R_1)$ and $wm_2 = (I, O, A_2) \otimes (M_2, m_{I,2}, R_2)$ be worker-manager automata of *WM*. Let $wm^\bullet$ be another worker-manager automaton, and $(f_{aw,1}{}^\bullet, f_{am,1}{}^\bullet) : wm^\bullet \to wm_1$, $(f_{aw,2}{}^\bullet, f_{am,2}{}^\bullet) : wm^\bullet \to wm_2$ be two asynchronous worker-manager homomorphisms. Let $wm = (I, O, A) \otimes (M, m_I, R)$ be the left or right asynchronous worker-manager pushout of wm_1 and wm_2 with respect to $(f_{aw,1}{}^\bullet, f_{am,1}{}^\bullet)$ and $(f_{aw,2}{}^\bullet, f_{am,2}{}^\bullet)$ with attendant homomorphisms $f_{awm,1} : wm \to wm_1$ and $f_{awm,2} : wm \to wm_2$. Suppose the following hold:

(1) For $i \neq j \in \{1, 2\}$, $wm_i \wedge m'_{wm'} \Rightarrow \neg\, wm_j \wedge m'_{wm'}$.

(2) For $i \neq j \in \{1, 2\}$, $\neg\, wm_j \wedge m_{wm_i}$ for any $m \in M_i$.

(3) $|\{m \mid m \in [m]_{am} \in M, (wm' \wedge m_{wm_1} \text{ or } wm' \wedge m_{wm_2})\}| \leq 1$.

Then $WM^* = (WM - \{wm_1, wm_2\}) \cup \{wm\}$ with ancillary data given by:

$$\begin{aligned}
\wedge^* \;=\; & (\{wm_1, wm_2\} \;\lhd\; \wedge \;\rhd\; (M_{1,wm_1} \cup M_{2,wm_2})) \cup \\
& \{wm \wedge^* m'_{wm'} \mid wm_i \wedge m'_{wm'},\ i \in \{1, 2\}\} \cup \\
& \{wm' \wedge^* [m]_{am} \mid [m]_{am} \in M,\ (wm' \wedge m_{wm_1} \text{ or } wm' \wedge m_{wm_2})\}
\end{aligned}$$

$$\begin{aligned}
\lambda^* \;=\; & ((M_{1,wm_1} \cup M_{2,wm_2}) \;\lhd\; \lambda \;\rhd\; (IO_{wm_1} = IO_{wm_2})) \cup \\
& \{\lambda^*_{m'_{wm'}}(p) = io \in IO_{wm} \mid \lambda_{m'_{wm'}}(p) = io \in IO_{wm_i},\ i \in \{1, 2\}\} \cup \\
& \{\lambda^*_{[m]_{am,wm}}(p) = io \in IO_{wm'} \mid [m]_{am} \in M,\ \lambda_{m_{wm_i}}(p) = io \in IO_{wm'}, \\
& \qquad\qquad p \in \ \biguplus \{P_{m_{wm_i}} \mid m \in [m]_{am},\ i \in \{1, 2\}\}\}
\end{aligned}$$

$$\begin{aligned}
r^* \;=\; & ((Rec_1 \cup Rec_2) \;\lhd\; r \;\rhd\; (R_{1,wm_1} \cup R_{2,wm_2})) \cup \\
& \{r^*_{wm \wedge^* m'_{wm'}}(rec) = m'_{wm'} \text{-}r\text{-›} n'_{wm'} \mid \\
& \qquad r_{wm_i \wedge m'_{wm'}}(rec) = m'_{wm'} \text{-}r\text{-›} n'_{wm'},\ i \in \{1, 2\}\} \cup \\
& \{r^*_{wm' \wedge^* [m]_{am,wm}}(rec) = [m]_{am,wm} \text{-}r\text{-›} [n]_{am,wm} \mid \\
& \qquad r_{wm' \wedge m_{wm_i}}(rec) = m_{wm_i} \text{-}r\text{-›} n_{wm_i},\ i \in \{1, 2\}\}
\end{aligned}$$

$$
\begin{aligned}
Initial_{WM^*} \;=\; & (sts^*, qs^*) \\
& \text{where} \\
& \quad sts^* = \begin{cases} (sts - INIS) \cup \{(Init_{wm}, m_{\mathrm{I},wm})\} & \text{if } sts \cap INIS \neq \varnothing \\ sts & \text{otherwise} \end{cases} \\
& \quad qs^* \;= \begin{cases} (qs - INIQ) \cup \{d{:}[\,] \mid d \in C_{m_{\mathrm{I},wm}}\} & \text{if } sts \cap INIS \neq \varnothing \\ qs & \text{otherwise} \end{cases} \\
& \text{and where} \quad INIS = \{(Init_{wm_1}, m_{\mathrm{I},wm_1}), (Init_{wm_2}, m_{\mathrm{I},wm_2})\}, \\
& \qquad\qquad\;\; INIQ = \{d{:}[\,] \mid d \in C_{m_{\mathrm{I},1wm_1}} \cup C_{m_{\mathrm{I},2wm_2}}\}, \\
& \qquad\qquad\;\; Initial_{WM} = (sts, qs)
\end{aligned}
$$

is a well defined IWIM system.

Proof. As usual there are four things to establish. First, that $\wedge^*$ is well defined. Neither of wm_1 or wm_2 is above the other by (2). So it is sufficient to replace $wm_i \wedge m'_{wm'}$ with $wm \wedge^* m'_{wm'}$; and to replace $wm' \wedge m_{1wm_1}$ or $wm' \wedge m_{2wm_2}$ with $wm' \wedge^* [m]_{\mathsf{am},wm}$ for the $[m]_{\mathsf{am}}$ that contains m_1 or m_2.

Second, that λ^* is a bijection. We replace the individual bijections $\lambda_{m_{1wm_1}}$ and $\lambda_{m_{2wm_2}}$ by aggregates of them, $\lambda^*_{[m]_{wm}}$, a process which leaves none out because every state in $M_1 \cup M_2$ enters some equivalence class or other in M, and causes no overlap of aggregated codomains by (3), preserving bijectiveness. Also by (1) at most one of wm_1, wm_2 is above any $m'_{wm'}$, so that replacing any $io \in IO_{wm_i}$ in the range of $\lambda_{m'_{wm'}}(p)$ by the corresponding $io \in IO_{wm}$ generates no problems either.

Third, r^* is a function. By (1) again, for at most one $i \in \{1, 2\}$ does $r_{wm_i \wedge m'_{wm'}}$ exist. Thus defining the $r^*_{wm \wedge^* m'_{wm'}}$ image of a *rec* event in agreement with that case is sound. Equally, substituting the $r_{wm' \wedge m_{1wm_1}}$ or $r_{wm' \wedge m_{2wm_2}}$ image of some *rec* event by the transition $[m]_{\mathsf{am},wm}$ -*r*-› $[n]_{\mathsf{am},wm}$, where the latter comes from m_{wm_i} -*r*-› n_{wm_i} via $r_{wm' \wedge m_{wm_i}}$, is uniquely defined, because for any wm' there is only one m_{wm_i} for which $r_{wm' \wedge m_{wm_i}}$ exists by (3). Finally, for the initial configurations, the argument is as in the previous cases. ☺

The preceding results illustrate that various pullback and pushout constructions acting on automata can be placed in the context of systems to give well defined algebraic operations on systems. However what has been described does not exhaust the possibilities. One could always imagine different ways of plumbing up the $\wedge^*$, λ^*, and r^* data, especially if other useful properties obtained in the system.

On a different tack, one could consider a hybrid notion of homomorphism for workers, which while insisting that input, output, and *rec* transitions mapped to input, output, and *rec* transitions respectively, did not insist that the data for these corresponded exactly. This would yield the opportunity of using the pair of values involved, to label a transition of the worker facet of a yet other notion of pullback or pushout.

More intriguingly, since the worker and manager facets of a worker-manager automaton are as independent as they are here, one could consider hybrid constructions on automata featuring say an asynchronous pushout on the manager facets and a (not asynchronous) pullback on the worker facets. Given the variety of component constructions that we have hinted at above, a large number of potential system level con-

structions can be contemplated this manner, and we leave their further investigation to the enthusiastic reader.

We turn now to the remaining automaton level constructions and examine their system level consequences.

Proposition 4.30 (Worker-Manager State Condensation in Systems) Let WM be an IWIM system and let $wm = wor \otimes man = (I, O, A = (St, Init, Tr)) \otimes (M, m_I, R)$ be a worker-manager automaton of WM. Let θ_w and θ_m be equivalence relations on St and M respectively and let $wm/(\theta_w,\theta_m) = wor/\theta_w \otimes man/\theta_m$ be the corresponding condensed worker-manager automaton. Suppose the following holds:

(1) $|\{m \mid m \in [m]_{\theta_m}, [m]_{\theta_m} \in M/\theta_m, (wm' \wedge m_{wm})\}| \leq 1$.

Then $WM^* = (WM - \{wm\}) \cup \{wm/(\theta_w,\theta_m)\}$ with ancillary data given by:

$$\wedge^* = (\{wm\} \lhd \wedge \rhd M) \cup \{wm/(\theta_w,\theta_m)\wedge^* m'_{wm'} \mid wm \wedge m'_{wm'}\} \cup \{wm' \wedge^* [m]_{\theta_m} \mid [m]_{\theta_m} \in M/\theta_m,\ wm' \wedge m_{wm}\}$$

$$\lambda^* = (M \lhd \lambda \rhd IO_{wm}) \cup \{\lambda^*_{m'_{wm'}}(p) = io \in IO_{wm/(\theta_w,\theta_m)} \mid \lambda_{m'_{wm'}}(p) = io \in IO_{wm}\} \cup \{\lambda^*_{[m]_{\theta_m}}(p) = io \in IO_{wm'} \mid [m]_{\theta_m} \in M/\theta_m,\ \lambda_{m_{wm}}(p) = io \in IO_{wm'},\ p \in \biguplus\{P_{m_{wm}} \mid m \in [m]_{\theta_m}\}\}$$

$$r^* = (Rec \lhd r \rhd R) \cup \{r^*_{wm/(\theta_w,\theta_m)\wedge^* m'_{wm'}}(rec) = m'_{wm'} \text{ -}r\text{-}\rightarrow n'_{wm'} \mid r_{wm \wedge m'_{wm'}}(rec) = m'_{wm'} \text{ -}r\text{-}\rightarrow n'_{wm'}\} \cup \{r^*_{wm' \wedge^* [m]_{\theta_m, wm/(\theta_w,\theta_m)}}(rec) = [m]_{\theta_m, wm/(\theta_w,\theta_m)} \text{ -}r\text{-}\rightarrow [n]_{\theta_m, wm/(\theta_w,\theta_m)} \mid r_{wm' \wedge m_{wm}}(rec) = m_{wm} \text{ -}r\text{-}\rightarrow n_{wm}\}$$

$$Initial_{WM^*} = (sts^*, qs^*)$$

where

$$sts^* = \begin{cases} (sts - INIS) \cup \{(Init_{wm/(\theta_w,\theta_m)}, m_{I,wm/(\theta_w,\theta_m)})\} \\ \quad \text{if } sts \cap INIS \neq \emptyset \\ sts \text{ otherwise} \end{cases}$$

$$qs^* = \begin{cases} (qs - INIQ) \cup \{d{:}[\,] \mid d \in C_{m_{I,wm/(\theta_w,\theta_m)}}\} \\ \quad \text{if } sts \cap INIS \neq \emptyset \\ qs \text{ otherwise} \end{cases}$$

and where $INIS = \{(Init_{wm}, m_{I,wm})\}$, $INIQ = \{d{:}[\,] \mid d \in C_{m_{I,wm}}\}$, $Initial_{WM} = (sts, qs)$

is a well defined IWIM system.

Proof. Mostly this is a simple adaptation of Proposition 4.29 so we will be brief. The definition of $\wedge^*$ is unproblematic. For λ^*, (1) assures bijectiveness of the $\lambda^*_{[m]_{\theta_m}}(p)$ terms, while the $\lambda^*_{m'_{wm'}}(p)$ terms are bijective since wm and $wm/(\theta_w,\theta_m)$ have the same input and output channel sets. Also it is easy to prove r^* is a function. For the initial configurations, we replace wm components by $wm/(\theta_w,\theta_m)$ components if required. ☺

Proposition 4.31 (Determinism Reflecting Worker-Manager State Condensation in Systems) Let WM be an IWIM system and let $wm = wor \otimes man = (I, O, A = (St, Init, Tr)) \otimes (M, m_I, R)$ be a worker-manager automaton of WM. Let θ_w and θ_m be equivalence relations on St and M respectively and suppose that θ_m is determinism reflecting. Let $wm/_D(\theta_w,\theta_m) = wor/\theta_w \otimes man/_D\theta_m$ be the corresponding determinism reflecting condensed worker-manager automaton. Suppose the following holds:

(1) $|\{m \mid m \in [m]_{\theta_m}, [m]_{\theta_m} \in M/_D\theta_m, (wm' \wedge m_{wm})\}| \leq 1$.

Then $WM^* = (WM - \{wm\}) \cup \{wm/_D(\theta_w,\theta_m)\}$ with ancillary data given by:

$$\begin{aligned}
\wedge^* \;=\; & (\{wm\} \lhd \wedge \rhd M) IO_{wm} \{wm/_D(\theta_w,\theta_m)^{\wedge *} m'_{wm'} \mid wm \wedge m'_{wm'}\} \cup \\
& \{wm' \wedge^* [m]_{\theta_m} \mid [m]_{\theta_m} \in M/_D\theta_m,\ wm' \wedge m_{wm}\}
\end{aligned}$$

$$\begin{aligned}
\lambda^* \;=\; & (M \lhd \lambda \wedge \rhd IO_{wm}) \cup \\
& \{\lambda^*_{m'_{wm'}}(p) = io \in IO_{wm/_D(\theta_w,\theta_m)} \mid \lambda_{m'_{wm'}}(p) = io \in IO_{wm}\} \cup \\
& \{\lambda^*_{[m]_{\theta_m}}(p) = io \in IO_{wm'} \mid [m]_{\theta_m} \in M/_D\theta_m,\ \lambda_{m_{wm}}(p) = io \in IO_{wm'}, \\
& \qquad p \in \textstyle\bigcup \{P_{m_{wm}} \mid m \in [m]_{\theta_m}\}\}
\end{aligned}$$

$$\begin{aligned}
r^* \;=\; & (Rec \lhd r \rhd R) \cup \\
& \{r^*_{wm/_D(\theta_w,\theta_m)^{\wedge *} m'_{wm'}}(rec) = m'_{wm'} \text{-}r\text{-}\rangle\ n'_{wm'} \mid \\
& \quad r_{wm \wedge m'_{wm'}}(rec) = m'_{wm'} \text{-}r\text{-}\rangle\ n'_{wm'}\} \cup \\
& \{r^*_{wm' \wedge^* [m]_{\theta_m, wm/_D(\theta_w,\theta_m)}}(rec) = [m]_{\theta_m, wm/_D(\theta_w,\theta_m)} \text{-}\mathbf{R}\text{-}\rangle\ [n]_{\theta_m, wm/_D(\theta_w,\theta_m)} \mid \\
& \quad r_{wm' \wedge m_{wm}}(rec) = m_{wm} \text{-}r\text{-}\rangle\ n_{wm}, \\
& \quad r \in \mathbf{R} = \{r \mid m \text{-}r\text{-}\rangle\ n \in R,\ m \in [m]_{\theta_m}, [m]_{\theta_m} \in M/_D\theta_m, \\
& \qquad\qquad n \in [n]_{\theta_m}, [n]_{\theta_m} \in M/_D\theta_m\}\}
\end{aligned}$$

$$\begin{aligned}
Initial_{WM^*} \;=\; & (sts^*, qs^*) \\
& \text{where} \\
& \quad sts^* = \begin{cases} (sts - INIS) \cup \{(Init_{wm/_D(\theta_w,\theta_m)}, m_{I,wm/_D(\theta_w,\theta_m)})\} \\ \qquad \text{if } sts \cap INIS \neq \varnothing \\ sts \text{ otherwise} \end{cases} \\
& \quad qs^* = \begin{cases} (qs - INIQ) \cup \{d{:}[\,] \mid d \in C_{m_{I,wm/_D(\theta_w,\theta_m)}}\} \\ \qquad \text{if } sts \cap INIS \neq \varnothing \\ qs \text{ otherwise} \end{cases} \\
& \text{and where } INIS = \{(Init_{wm}, m_{I,wm})\}, \\
& \qquad\quad INIQ = \{d{:}[\,] \mid d \in C_{m_{I,wm}}\}, \\
& \qquad\quad Initial_{WM} = (sts, qs)
\end{aligned}$$

is a well defined IWIM system.

Proof. This is almost identical to Proposition 4.30 and is omitted. ☺

Incidentally, Proposition 4.31 solves the riddle posed after Definition 4.21, i.e. why not define a transition $[m]_{\theta_m}$ -**R**-› $[n]_{\theta_m}$ as any set of transitions m -r-› $n \in R$ that makes the union definitions of $\rho_{[m]_{\theta_m},[n]_{\theta_m}}$ and $\chi_{[m]_{\theta_m},[n]_{\theta_m}}$ sound. The answer is that without a canonical choice for the transition $[m]_{\theta_m}$ -**R**-› $[n]_{\theta_m}$, there is no canonical way to make $r^*_{wm' \wedge [m]_{\theta_m}, wm/_D(\theta_w,\theta_m)}$ into a function.

We end this subsection with three almost trivial but useful constructions. The first merely glues the free end of an external output to the free end of an external input, to make a new internal channel. The second, removes a tuple from the partial injection on channels in a reconfiguration transition; and the third augments the domain and range of the partial injection on channels in a reconfiguration transition with a fresh tuple; enabling the benefits of the first construction to be felt after a reconfiguration.

Proposition 4.32 (External Channel Piping in Systems) Let WM be an IWIM system and let $wm = wor \otimes man = (I, O, A = (St, Init, Tr)) \otimes (M, m_I, R)$ be a worker-manager automaton of WM. Let $m \in M$ be a manager state of man, which maps to $(P_{m_{wm}}, C_{m_{wm}})$. Suppose $\{c_{ei}, c_{eo}\} \subseteq C_{m_{wm}}$ that c_{ei} is an external input channel, and that c_{eo} is an external input channel. Let c_{io} be fresh. Then $WM^* = (WM - \{wm\}) \cup \{wm^*\}$, given below, is a well defined IWIM system.

$$
\begin{aligned}
wm^* \;=\; & wor \otimes man^* \text{ where} \\
& man^* = (M, m_I, R^*) \text{ and } m \mapsto (P_{m_{wm}}, C_{m_{wm}}{}^*) \\
& \text{where } C_{m_{wm}}{}^* = (C_{m_{wm}} - \{c_{ei}, c_{eo}\}) \cup \{c_{io}\} \text{ and} \\
& \qquad s_{m_{wm^*}} = (\{c_{eo}\} \lhd s_{m_{wm}}) \cup \{c_{io} \mapsto s_{m_{wm}}(c_{eo})\} \\
& \qquad t_{m_{wm^*}} = (\{c_{ei}\} \lhd s_{m_{wm}}) \cup \{c_{io} \mapsto t_{m_{wm}}(c_{ei})\}
\end{aligned}
$$

Proposition 4.33 (Restricted Reconfiguration in Systems) Let WM be an IWIM system and let $wm = wor \otimes man = (I, O, A = (St, Init, Tr)) \otimes (M, m_I, R)$ be a worker-manager automaton of WM. Let m_{wm} -r-› $n_{wm} \in R$ be a reconfiguration transition of wm with $r = \chi_{m_{wm},n_{wm}} : C_{m_{wm}} \to C_{n_{wm}}$. Suppose further that $\chi_{m_{wm},n_{wm}}(c_{m_{wm}}) = c_{n_{wm}}$. Then $WM^* = (WM - \{wm\}) \cup \{wm^*\}$, given below, is a well defined IWIM system.

$$
\begin{aligned}
wm^* \;=\; & wor \otimes man^* \text{ where} \\
& man^* = (M, m_I, R^*) \text{ and} \\
& R^* = (R - \{m_{wm} \text{-}r\text{-›} n_{wm} = \chi_{m_{wm},n_{wm}} : C_{m_{wm}} \to C_{n_{wm}}\}) \cup \\
& \qquad \{m_{wm} \text{-}r^*\text{-›} n_{wm} = \chi^*{}_{m_{wm},n_{wm}} = \\
& \qquad\; \chi_{m_{wm},n_{wm}} - \{c_{m_{wm}} \mapsto c_{n_{wm}}\} : C_{m_{wm}} \to C_{n_{wm}}\}
\end{aligned}
$$

Note that the restricting operation can be applied unconditionally (assuming there is a tuple to remove in the first place). Even with an empty resulting $\chi_{m_{wm},n_{wm}}$ there is still a transition r^* to act as target for any needed $r_{wm \hat{} m'_{wm'}}$ function.

Proposition 4.34 (Extended Reconfiguration in Systems) Let WM be an IWIM system and let $wm = wor \otimes man = (I, O, A = (St, Init, Tr)) \otimes (M, m_I, R)$ be a worker-manager automaton of WM. Let m_{wm} -r-› $n_{wm} \in R$ be a reconfiguration transition of wm with $r = \chi_{m_{wm},n_{wm}} : C_{m_{wm}} \to C_{n_{wm}}$. Suppose further that $c_{m_{wm}} \in C_{m_{wm}}$ although $c_{m_{wm}} \notin \mathrm{dom}(\chi_{m_{wm},n_{wm}})$, and $c_{n_{wm}} \in C_{n_{wm}}$ although $c_{n_{wm}} \notin \mathrm{ran}(\chi_{m_{wm},n_{wm}})$. Then WM^* $= (WM - \{wm\}) \cup \{wm^*\}$, given below, is a well defined IWIM system.

$$
\begin{aligned}
wm^* \;=\; & wor \otimes man^* \text{ where} \\
& man^* = (M, m_I, R^*) \text{ and} \\
& R^* = (R - \{m_{wm} \text{-}r\text{-›} n_{wm} = \chi_{m_{wm},n_{wm}} : C_{m_{wm}} \to C_{n_{wm}}\}) \cup \\
& \qquad \{m_{wm} \text{-}r^*\text{-›} n_{wm} == \chi^*{}_{m_{wm},n_{wm}} = \\
& \qquad\; \chi_{m_{wm},n_{wm}} \cup \{c_{m_{wm}} \mapsto c_{n_{wm}}\} : C_{m_{wm}} \to C_{n_{wm}}\}\}
\end{aligned}
$$

4.3 Completeness

In this subsection we consider a question converse to those dealt with hitherto, i.e. to what extent can an arbitrary IWIM system be assembled from more primitive components using the operations already described. Now intuitively, an arbitrary worker-manager automaton $wm = wor \otimes man$ can be seen (up to isomorphism) as an asynchronous pushout of *p-wor* and *p-man*, where *p-wor* is a pure worker containing *wm*'s worker facet, and *p-man* is a pure manager containing *wm*'s manager facet. This thought allows us to pull apart an arbitrary entanglement of worker-manager automata into what are effectively disjoint elementary IWIM subsystems. These in turn can be built up out of smaller primitives, and this provides the basis of our completeness result.

To cope with the requirement that an asynchronous worker pushout only works when the I and O channel sets are exactly the same, we define an (I, O)-pure manager to be a worker-manager automaton in which the worker facet is a one state automaton without transitions, but equipped nevertheless with input and output channel sets I and O.

Proposition 4.35 (Worker-Manager Pull-Apart in Systems) Let WM be an IWIM system and let $wm = wor \otimes man = (I, O, A = (St, Init, Tr)) \otimes (M, m_I, R)$ be a worker-manager automaton of WM. Let *p-wor* be a pure worker with worker facet *wor*, and *p-man* be an (I, O)-pure manager with manager facet *man*.

Then $WM^* = (WM - \{wm\}) \cup \{p\text{-}wor, p\text{-}man\}$ with ancillary data given by:

$$\begin{aligned}
\wedge^* \;=\; & (\{wm\} \;\triangleleft\; \wedge \;\triangleright\; M) \cup \{p\text{-}wor \wedge^* m'_{wm'} \mid wm \wedge m'_{wm'}\} \cup \\
& \{wm' \wedge^* m_{p\text{-}man} \mid m \in M,\ wm' \wedge m_{wm}\}
\end{aligned}$$

$$\begin{aligned}
\lambda^* \;=\; & (M \;\triangleleft\; \lambda \;\triangleright\; IO) \cup \\
& \{\lambda^*_{m'_{wm'}}(p) = io \in IO_{p\text{-}wor} \mid \lambda_{m'_{wm'}}(p) = io \in IO_{wm}\} \cup \\
& \{\lambda^*_{m_{p\text{-}man}}(p) = io \in IO_{wm'} \mid p \in P_{m_{wm}},\ m \in M,\ wm' \wedge m_{wm}, \\
& \qquad\qquad \lambda_{m_{wm}}(p) = io \in IO_{wm'}\}
\end{aligned}$$

$$\begin{aligned}
r^* \;=\; & (Rec \;\triangleleft\; r \;\triangleright\; R) \cup \\
& \{r^*_{p\text{-}wor \wedge^* m'_{wm'}}(rec) = m'_{wm'} \text{-}r\text{-}\rightarrow n'_{wm'} \mid \\
& \quad r_{wm \wedge m'_{wm'}}(rec) = m'_{wm'} \text{-}r\text{-}\rightarrow n'_{wm'}\} \cup \\
& \{r^*_{wm' \wedge^* m_{p\text{-}man})}(rec) = m_{p\text{-}man} \text{-}r\text{-}\rightarrow n_{p\text{-}man} \in R \mid \\
& \quad r_{wm' \wedge m_{wm}}(rec) = m_{wm} \text{-}r\text{-}\rightarrow n_{wm}\}
\end{aligned}$$

$$\begin{aligned}
Initial_{WM^*} \;=\; & (sts^*, qs^*) \\
& \text{where} \\
& \quad sts^* = \begin{cases} (sts - INIS) \cup \\ \qquad \{(Init_{p\text{-}wor}, m_{I,p\text{-}wor}), (Init_{p\text{-}man}, m_{I,p\text{-}man})\} \\ \qquad \text{if } sts \cap INIS \neq \emptyset \\ sts \text{ otherwise} \end{cases} \\
& \quad qs^* = \begin{cases} (qs - INIQ) \cup \{d{:}[\,] \mid d \in C_{m_{I,p\text{-}man}}\} \\ \qquad \text{if } sts \cap INIS \neq \emptyset \\ qs \text{ otherwise} \end{cases}
\end{aligned}$$

and where $INIS = \{(Init_{wm}, m_{\mathrm{I},wm})\}$,
$INIQ = \{d{:}[\,] \mid d \in C_{m_{\mathrm{I},wm}}\}$,
$Initial_{WM} = (sts, qs)$

is a well defined IWIM system.

Proof. This is straightforward when we realise that the trivial worker and manager facets introduced by this procedure are not above or below anything else. ☺

Proposition 4.36 (Worker-Manager Pull-Apart Reconstruction in Systems) Let *WM* be an IWIM system and let $wm = wor \otimes man = (I, O, A = (St, Init, Tr)) \otimes (M, m_{\mathrm{I}}, R)$ be a worker-manager automaton of *WM*. Let *WM** be obtained as described in Proposition 4.35. Let $wm^{\bullet}$ be a worker-manager automaton with trivial manager facet, trivial worker facet but with sets of input and output channels (I, O). Suppose $(f_{\mathrm{aw},1}{}^{\bullet}, f_{\mathrm{am},1}{}^{\bullet}) : wm^{\bullet} \to p\text{-}wor$ and $(f_{\mathrm{aw},2}{}^{\bullet}, f_{\mathrm{am},2}{}^{\bullet}) : wm^{\bullet} \to p\text{-}man$ are the obvious two asynchronous worker-manager homomorphisms that identify the initial states in corresponding facets in the expected way. Then *WM***, the asynchronous worker-manager pushout of $(f_{\mathrm{aw},1}{}^{\bullet}, f_{\mathrm{am},1}{}^{\bullet})$ and $(f_{\mathrm{aw},2}{}^{\bullet}, f_{\mathrm{am},2}{}^{\bullet})$ exists, and is set theoretically isomorphic to *WM*.

Proof. It is easy to check that conditions (1)-(3) for the applicability of the construction in Proposition 4.29 are satisfied so that the asynchronous worker-manager pushout exists. Furthermore, the claimed isomorphism is easy to see since the only nontrivial equivalence classes of states contain just an initial state, and the unique state from the other component. ☺

In this manner, an arbitrarily complicated IWIM system can be decomposed into what are effectively elementary IWIM subsystems, the reverse of this procedure giving us a recipe for rebuilding the desired system from such components. In turn such an elementary subsystem can be built up from trivial one-state or one-transition components. Since elementary subsystems are basically tree-structured, there will be a variety of ways to do this in a well founded way, so we will not go into details. This supports our claim that the techniques discussed here, with the addition of suitable lower level techniques for building elementary subsystems, are complete.

5 IWIM Systems with Delayed Reconfigurations

Now we tackle the problem of the asynchronous nature of true IWIM system event processing. As noted previously, this can be captured within our framework. The basic idea is simple. We introduce fresh pure worker automata, delay automata, whose job is to buffer the reconfiguration events generated by the worker facets of the automata of the original model on their way to the relevant destination manager facet. The way this is done is to change the *rec* events of the original model into *rec* messages to the delay automata, who then subsequently raise the required event. Since buffering is already implicit in the message queues used by worker facets, and further buffering can be achieved by retaining information in automaton states, there are a number of ways one can imagine of implementing such an idea. In the one we will follow, the workers each acquire an extra output port through which to send *rec* mes-

sages instead of raising *rec* events. Connected to these extra output ports, are channels leading to delay automata, one per manager facet in charge of the worker. This ensures that the *rec* messages are broadcast asynchronously towards each relevant manager. (Because event processing takes place simultaneously by all managers below a worker, we need to ensure that each delay automaton is above only one manager. To ensure the correct separation of concerns between automata it is easiest to introduce delay automata on a per per $wm'^\wedge m_{wm}$ tuple basis.) Upon receipt of the *rec* message, the delay automaton raises the corresponding event with the manager.

Assuming that some particular worker facet is above k manager facets, the behaviour of the original system can be recovered as long as there is always the possibility of performing the following $2k+1$ step sequence of the new system instead of a *rec* transition of the original system, in a manner uninterrupted by other system transitions:

(1) the worker facet transmits the relevant *rec* value through its extra output port onto the n delay channels leading to the n delay automata corresponding to the n manager facets above which it sits,
(2_i) delay automaton i receives the *rec* value from delay channel i, recording it in its state,
(3_i) delay automaton i performs a *rec* transition causing manager facet i to perform the required reconfiguration.

This sequence of steps preserves the property that all delay channels remain empty except between steps (1) and (2_i), which is correspondingly consistent with enabling them to be executed without interruptions.

On the other hand, if we consider that the execution of these steps can indeed be interrupted, as allowed by the asynchrony inherent in the fragmenting of a single transition into several, other outcomes become possible. Since the original system had only synchronous reconfigurations, it provides no definition of what might happen should a reconfiguration be attempted nonatomically, and any evolution consistent with the semantics is permissible. For example, a context dependent notion of reconfiguration can be created by having delay automata raise different reconfiguration actions in manager facets, depending on what reconfigurations intervened between the receiving of some particular *rec* value from a worker, and the raising of the corresponding reconfiguration event in the manager; the information to manage this being kept in a delay automaton's state, suitably managed through intervening reconfigurations. And depending on what policy is adopted for the introduction and behaviour of the delay automata, different policies for the handling of pending events become possible. Moreover being themselves workers, delay automata can be woken and suspended during reconfiguration transitions, further tuning this aspect.

One canonical possibility for dealing with reconfigurations that attempt to interleave other reconfiguration actions, is to enforce a strict sequentialisation policy. This can be done by ensuring that once a *rec* message arrives at a delay automaton, the only thing the delay automaton can then do is to raise the corresponding event, ignoring further inputs till it has done so. We call this arrangement the standard asynchronisation of an IWIM system, and we now present the technical details.

Suppose *WM* is an IWIM system with the usual notations, i.e. typical automaton name *wm* mapping to $(I, O, A = (St, Init, Tr))\otimes(M, m_I, R)$, with manager states *m* mapping to networks $(P_{m_{wm}}, C_{m_{wm}})$, and reconfigurations m_{wm} -*r*-› $n_{wm} = \chi_{m_{wm},n_{wm}} : C_{m_{wm}} \to C_{n_{wm}}$; and with ancillary data given by $wm'^{\wedge}m_{wm}$, $\lambda_{m_{wm}}$, $r_{wm'^{\wedge}m_{wm}}$.

The standard asynchronisation of *WM*, which we call here *WM**, has the set of automaton names $WM^* = WM \cup \{\Delta.wm'.m.wm \mid wm'^{\wedge}m_{wm}\}$. We assume all of these $\Delta.wm'.m.wm$ names are fresh, and introduce for each $\Delta.wm'.m.wm$ name, for future convenience, fresh port, channel, and input and output port names[1]:

$$\Delta.wm'.m.wm_s\ ,\ \Delta.wm'.m.wm_t\ ,\ \Delta.wm'.m.wm_{ch}\ ,\ \Delta.wm'.m.wm_i\ ,\ \Delta.wm'_o$$

If *wm* maps to $(I, O, A = (St, Init, Tr))\otimes(M, m_I, R)$ in *WM*, in *WM**, *wm* maps to $(I, O^*, A^* = (St, Init, Tr^*))\otimes(M, m_I, R^*)$.

The input ports *I* of the worker facet of *wm* remain unchanged. However for the output ports we have $O^* = O \cup \{\Delta.wm_0\}$. The worker facet automaton *wor*(*wm*) itself is given by the same state space *St*, initial state *Init*, and:

$$Tr^* = Tr_I \cup Tr_O \cup \{a \text{ -}\Delta.wm_0!rec\text{-› } b \mid a \text{ -}rec\text{-› } b \in Tr_R\}$$

This ensures that *rec* messages can be sent over $\Delta.wm_0$ to all delay automata $\Delta.wm.m'.wm'$. To ensure that these are handled properly, we examine the manager facet of *wm*.

In the manager facet *man*(*wm*), the state space *M* and initial state m_I remain unchanged. State *m* however maps to the communication network $(P^*_{m_{wm}}, C^*_{m_{wm}})$ where:

$$P^*_{m_{wm}} = P_{m_{wm}} \cup \{\Delta.wm'.m.wm_s, \Delta.wm'.m.wm_t \mid wm'^{\wedge}m_{wm}\}$$
$$C^*_{m_{wm}} = C_{m_{wm}} \cup \{\Delta.wm'.m.wm_{ch} \mid wm'^{\wedge}m_{wm}\})$$

$$s^*_{m_{wm}} = s_{m_{wm}} \cup \{\Delta.wm'.m.wm_{ch} \mapsto \Delta.wm'.m.wm_s \mid wm'^{\wedge}m_{wm}\}$$
$$t^*_{m_{wm}} = t_{m_{wm}} \cup \{\Delta.wm'.m.wm_{ch} \mapsto \Delta.wm'.m.wm_t \mid wm'^{\wedge}m_{wm}\}$$

Finally, if m_{wm} -*r*-› $n_{wm} = \chi_{m_{wm},n_{wm}} : C_{m_{wm}} \to C_{n_{wm}}$ is a reconfiguration transition of *R*, there is a corresponding transition of *R** given by $\chi^*_{m_{wm},n_{wm}} : C^*_{m_{wm}} \to C^*_{n_{wm}}$ where $\chi^*_{m_{wm},n_{wm}} = \chi_{m_{wm},n_{wm}}$ interpreted as a partial injection on $C^*_{m_{wm}}$.

Standing between the worker and manager facets of the preceding automata, are the delay automata themselves. A delay automaton name $\Delta.wm'.m.wm$ maps to a pure worker given by:

$$(I_{\Delta.wm'.m.wm}, O_{\Delta.wm'.m.wm}, A_{\Delta.wm'.m.wm} = (St_{\Delta.wm'.m.wm}, Init_{\Delta.wm'.m.wm}, Tr_{\Delta.wm'.m.wm}))\otimes(\{\bullet\}, \bullet, \varnothing)$$

Here:

$$I_{\Delta.wm'.m.wm} = \{\Delta.wm'.m.wm_i \mid wm'^{\wedge}m_{wm}\}$$

1. The last of these is not an error.

while $O_{\Delta.wm'.m.wm} = \emptyset$. The worker automaton $A_{\Delta.wm'.m.wm}$ is given by the state space:

$$St_{\Delta.wm'.m.wm} = Rec_{wm'} \uplus \{Init_{\Delta.wm'.m.wm}\}$$

and the initial state $Init_{\Delta.wm'.m.wm}$ is the one named as such. The transitions of $A_{\Delta.wm'.m.wm}$ are given by:

$$\begin{aligned} Tr_{\Delta.wm'.m.wm} = {} & \{Init_{\Delta.wm'.m.wm} \text{-}\Delta.wm'.m.wm_{\mathsf{i}}?rec\text{-}\rightarrow rec \mid rec \in Rec_{wm'}\} \cup \\ & \{rec \text{-}rec\text{-}\rightarrow Init_{\Delta.wm'.m.wm} \mid rec \in Rec_{wm'}\} \end{aligned}$$

where we have abused notation a little by allowing *rec* to name the state reached by inputting a *rec* message (not to mention its original use as event name), hopefully without causing confusion. It is now clear that the delay automaton inputs a *rec* message coming from the original worker, and then provokes a *rec* reconfiguration event in the manager at a later point.

To connect all this together, we give the above relation, which is:

$$\wedge^* = \wedge \cup \{\Delta.wm'.m.wm \wedge^* m_{wm} \mid wm' \wedge m_{wm}\}$$

and the $\lambda^*_{m_{wm}}$ bijections which are:

$$\begin{aligned} \lambda^*_{m_{wm}} = {} & \lambda_{m_{wm}} \cup \{\Delta.wm'.m.wm_{\mathsf{s}} \mapsto \Delta.wm'_0 \mid wm' \wedge m_{wm}\} \cup \\ & \{\Delta.wm'.m.wm_{\mathsf{t}} \mapsto \Delta.wm'.m.wm_{\mathsf{i}} \mid wm' \wedge m_{wm}\} \end{aligned}$$

Note how in the first line of the above the original worker's output port $\Delta.wm'_0$ is shared by as many managers as it has, each controlling an individual queue to a separate $\Delta.wm'.m.wm$ delay automaton.

Finally the $r^*_{\Delta.wm'.m.wm \wedge^* m_{wm}}$ functions are given by:

$$r^*_{\Delta.wm'.m.wm \wedge^* m_{wm}}(rec) = m_{wm} \text{-}r\text{-}\rightarrow n_{wm}$$

iff $r_{wm' \wedge m_{wm}}(rec) = m_{wm} \text{-}r\text{-}\rightarrow n_{wm}$.

It is now clear that this construction has the properties indicated informally above. Thus whereas in *WM*, a worker wm' above a manger state m_{wm} can perform the step $a \text{-}rec\text{-}\rightarrow b$ simultaneously with each implicated manager's performing the appropriate $m_{wm} \text{-}r\text{-}\rightarrow n_{wm}$ (because $r_{wm' \wedge m_{wm}}$ maps *rec* to $m_{wm} \text{-}r\text{-}\rightarrow n_{wm}$), in *WM**, wm' can no longer do this directly. Instead it passes a *rec* message to $\Delta.wm'.m.wm$ via a single $a \text{-}\Delta.wm'_0!rec\text{-}\rightarrow b$ action which causes *rec* messages to be broadcast onto all relevant channels $\Delta.wm'.m.wm_{\mathsf{ch}}$. If such a channel was previously empty, then $\Delta.wm'.m.wm$ can swallow the *rec* message by performing an $Init_{\Delta.wm'.m.wm} \text{-}\Delta.wm'.m.wm_{\mathsf{i}}?rec\text{-}\rightarrow rec$ input from the same channel. This obtains by the fact that ports $\Delta.wm'_0$ and $\Delta.wm'.m.wm_{\mathsf{i}}$ are connected via $\Delta.wm'.m.wm_{\mathsf{ch}}$, since $\lambda^*_{m_{wm}}$ connects $\Delta.wm'_0$ to $\Delta.wm'.m.wm_{\mathsf{s}} = s^*_{m_{wm}}(\Delta.wm'.m.wm_{\mathsf{ch}})$, and also connects $t^*_{m_{wm}}(\Delta.wm'.m.wm_{\mathsf{ch}}) = \Delta.wm'.m.wm_{\mathsf{t}}$ to $\Delta.wm'.m.wm_{\mathsf{i}}$. Since $r^*_{\Delta.wm'.m.wm \wedge^* m_{wm}}$ maps the only available $\Delta.wm'.m.wm$ transition $rec \text{-}rec\text{-}\rightarrow Init_{\Delta.wm'.m.wm}$ to the reconfiguration $m_{wm} \text{-}r\text{-}\rightarrow n_{wm}$, it follows that when $\Delta.wm'.m.wm$ performs $rec \text{-}rec\text{-}\rightarrow Init_{\Delta.wm'.m.wm}$, it provokes the desired reconfiguration $m_{wm} \text{-}r\text{-}\rightarrow n_{wm}$. Thus if $\Delta.wm'.m.wm_{\mathsf{ch}}$ was empty at the outset, the simulation of one manager's reconfiguration by a delayed but uninterrupted se-

quence of steps is available. Evidently when several managers need to react, consequent on the same original atomic reconfiguration, similar simulations can also be constructed. These simulations may also be interleaved with other actions, provided none of the other actions 'beat the sequence to the tape', where the 'tape' is the invocation of a *rec* step mapped by a $r^*_{\Delta.wm'.m.wm^\wedge * m_{wm}}$ to a change of configuration of the manager *wm*, while the manager remains in the original state *m*. Examples of other actions that can safely be interleaved in this manner are ordinary I/O actions, and reconfigurations not involving any of the automata involved.

Proposition 5.1 The construction just given is idempotent, in the sense that applying it *n* more times to *WM** results in a system $WM^{*\cdots *}_{\overline{n+1}}$ which can simulate an atomic reconfiguration of *WM* that involves *k* managers in $2k(n+1)+1$ uninterrupted steps.

The straightforward if tedious proof rests on the observation that in *WM**, the only worker above m_{wm} capable of provoking a reconfiguration is a $\Delta.wm'.m.wm$, so that the next application of the construction replaces each $\Delta.wm'.m.wm$'s *rec* steps by a three step sequence etc. Thus iterated application of the construction exemplifies the fact that a chain of buffers is behaviourally equivalent to a single buffer.

6 The Arbab, de Boer, Bonsangue Model

In this section we show how the model proposed by Arbab, de Boer and Bonsangue in [Arbab et al. (2000a)] (see also [Arbab et al. (2000b)]), henceforth the ABB model, can be subsumed within our framework. In the ABB model, there is a family of *components*. Each component is a transition system similar to one of our worker automata, and it has access to a set of channel ends to which it is connected. A component may output values along channel source ends (eg. $\overline{c}$) to which it is connected, and may input values from channel sink ends (eg. c) to which it is connected. The state transitions for these actions are of the form $a \text{ -}\overline{c}!v\text{-}\!\rightarrow b$ and $a \text{ -}c?v\text{-}\!\rightarrow b$ respectively, and these are the only kinds of action that components may perform. The dynamic reconfigurability of ABB systems comes from the fact that they can alter their set of connected channel ends by sending and receiving channel end identities along the channels themselves. Thus if a component possesses channel ends $\overline{c}$, d, it may relinquish possession of d by a transition like $a \text{ -}\overline{c}!d\text{-}\!\rightarrow b$; likewise $a \text{ -}\overline{c}!\overline{d}\text{-}\!\rightarrow b$ relinquishes possession of $\overline{d}$. Likewise possession of d or $\overline{d}$ can be gained by $a \text{ -}c?d\text{-}\!\rightarrow b$ or $a \text{ -}c?\overline{d}\text{-}\!\rightarrow b$. It is tacitly assumed that since channels are point to point connections, once a component has relinquished possession of a channel end, it will no longer attempt to use it until it has received it once again from some other component. Channels themselves are queues in the ABB model, just as they are in ours, and when a channel end, d (resp. $\overline{d}$) say, becomes detached from the component to which it was previously connected by being output along channel c say, no inputs over d (resp. outputs over $\overline{d}$) can take place until the relevant message has been consumed by the component connected to the sink end of c, whereupon d (resp. $\overline{d}$) becomes available to that component for communication purposes. Output and input transitions in which a channel end is respectively transmitted or received are called reconfiguring output and input transitions.

We will now describe the mapping of a family of ABB components to a corresponding IWIM system. Note that since channels are not created dynamically in the ABB model, the complete set of channels that figure in an execution of an ABB system is known at initialisation time, and given an ABB system, we call this complete set of channels *CH*. From this we create the five disjoint alphabets:

$CH_{\mathsf{i}} = \{ch_{\mathsf{i}} \mid ch \in CH\}$
$CH_{\mathsf{0}} = \{ch_{\mathsf{0}} \mid ch \in CH\}$
$CH_{\mathsf{s}} = \{ch_{\mathsf{s}} \mid ch \in CH\}$
$CH_{\mathsf{t}} = \{ch_{\mathsf{t}} \mid ch \in CH\}$
$CH_{\mathsf{ch}} = \{ch_{\mathsf{ch}} \mid ch \in CH\}$

Let $C_1 \ldots C_n$ be a family of ABB components. For each C_i we construct a transition system K_i as follows. Let C_i be (St_i, $Init_i$, Tr_i, r_i) where St_i is a set of states of which $Init_i$ is an initial state, Tr_i is a transition relation containing transitions a -$\overline{out}!v$-› b or a -$in?v$-› b (with $in, out \in CH$), and r_i is the initial value of the dynamically changing set of channel ends possessed by C_i. By the remarks above we can assume that $CH = \{ch \mid$ for some i, $ch \in r_i$ or $\overline{ch} \in r_i\}$. For simplicity we will assume that each end of each channel in CH is in some r_i.

Now we set K_i to be the transition system given by (St_i*, $Init_i$*, Tr_i*), where the set of states is St_i* = $St_i \cup$ *new*St_i, with $Init_i$* = $Init_i$, and Tr_i* is given as follows (also implicitly defining the fresh states *new*St_i). Each transition a -$\overline{out}!v$-› b or a -$in?v$-› b of C_i where v is not a channel end yields a transition a -$out_0!v$-› b or a -$in_{\mathsf{i}}?v$-› b of K_i. Moreover each reconfiguring output a -$\overline{out}!\overline{ch}$-› b of C_i is replaced by two transitions a -$out_0!ch_0$-› ab -$rec(out_0!ch_0)$-› b, where ab is a fresh state in *new*St_i and $rec(out_0!ch_0)$ is a reconfiguration action where the intention is to simulate the detaching of the channel end ch_0 from the component in a manner that will be made clear below. Likewise if the channel end being detached is ch rather than $\overline{ch}$, K_i will contain the sequence a -$out_0!ch_{\mathsf{i}}$-› ab -$rec(out_0!ch_{\mathsf{i}})$-› b. A similar arrangement holds for reconfiguring input transitions a -$in?\overline{ch}$-› b or a -$in?ch$-› b. We have respectively a -$in_{\mathsf{i}}?ch_0$-› ab -$rec(in_{\mathsf{i}}?ch_0)$-› b and a -$in_{\mathsf{i}}?ch_{\mathsf{i}}$-› ab -$rec(in_{\mathsf{i}}?ch_{\mathsf{i}})$-› b.

For technical reasons, it is not sufficient to work with just the K_i. Given K_i, let ${\theta_i}^{+a}$ be a finite directed path through the transition system of K_i (i.e. a finite sequence of contiguous transitions of K_i), starting at state a. Let ${K_i}^a$ be the transition system determined by the set of paths: $\{{\theta_i}^{+a} \mid {\theta_i}^{+a}$ is a path through the transition system of K_i starting at a, and if ${\theta_i}^{+a}$ contains a *rec* transition, there is only one and it is the last transition of ${\theta_i}^{+a}\}$.

Given a ${\theta_i}^{+a}$, let ${\theta_i}^a$ be the result of erasing from ${\theta_i}^{+a}$ all non-*rec* transitions (so the transitions listed in ${\theta_i}^a$ will not be contiguous, neither will they necessarily mention a). Let $\phi({\theta_i}^{+a})$, $\phi({\theta_i}^a)$ denote the final state reached by such a ${\theta_i}^{+a}$ or ${\theta_i}^a$. Define ${\Theta_i}^a = \{{\theta_i}^a \mid {\theta_i}^{+a}$ is a path through the transition system of K_i starting at $a\}$; consequently ${\Theta_i}^a$ is partially ordered by the prefix relation. We write ${\theta_i}^+$, θ_i, Θ_i to denote ${\theta_i}^{+Init_i}$, ${\theta_i}^{Init_i}$, ${\Theta_i}^{Init_i}$. Let:

$M = \prod\{\Theta_i \mid i \in \{1 \ldots n\}\}$

The rest of the construction will proceed by recursion on the structure of M, which is again partially ordered by the prefix relation. We construct a pure manger automaton pm, whose space of states is M, and above each $m \in M$, there will be a collection of pure worker automata crafted from the K_i^a transition systems[2].

The base case is $m = [\,]\times[\,]\times\ldots\times[\,]$. Above this m we have the collection of pure workers $pw_i^{[\,]}$ for $i \in \{1 \ldots n\}$, where $pw_i^{[\,]}$ is given by $(CH_{\mathsf{i}i}^{[\,]}, CH_{0i}^{[\,]}, K_i^{Init_i})$, with $CH_{\mathsf{i}i}^{[\,]} = \{ch_{\mathsf{i}} \mid ch_{\mathsf{i}} \in CH_{\mathsf{i}}, ch \in r_i\}$ and $CH_{0i}^{[\,]} = \{ch_0 \mid ch_0 \in CH_0, \overline{ch} \in r_i\}$. Note that $Init_i = \phi([\,])$ (with the understanding that $[\,]$ is the empty path through K_i).

The manager state m maps to (P_m, C_m) where:

$$P_m = \{ch_{\mathsf{s}} \mid ch_{\mathsf{s}} \in CH_{\mathsf{s}}, \overline{ch} \in r_i\} \cup \{ch_{\mathsf{t}} \mid ch_{\mathsf{t}} \in CH_{\mathsf{t}}, ch \in r_i\}$$
$$C_m = \{ch_{\mathsf{ch}} \mid \{ch_{\mathsf{s}}, ch_{\mathsf{t}}\} \cap P_m \neq \emptyset\}$$

and the s_m, t_m maps function in the way we would expect, i.e. $s_m(ch_{\mathsf{ch}}) = ch_{\mathsf{s}}$ and $t_m(ch_{\mathsf{ch}}) = ch_{\mathsf{t}}$. The link between the manager and the workers is also unsurprising:

$$\lambda_m = \{ch_{\mathsf{t}} \mapsto ch_{\mathsf{i}} \mid ch_{\mathsf{i}} \in CH_{\mathsf{i}i}^{[\,]}\} \cup \{ch_{\mathsf{s}} \mapsto ch_0 \mid ch_0 \in CH_{0i}^{[\,]}\}$$

$$pw_i^{[\,]\wedge m}$$

completing the base case.

Now suppose that $m = (\theta_1 \ldots \theta_n)$ and suppose $m' = (\theta_1 \ldots \theta_i' \ldots \theta_n)$ where $\theta_i' = \theta_i@[a_i \text{ -}rec(out_0!ch_0)\text{-}\rightarrow b_i]$, and where the transition $a_i \text{ -}rec(out_0!ch_0)\text{-}\rightarrow b_i$ is a K_i- immediate successor reconfiguring transition to the last one in θ_i. The manager state m which maps to (P_m, C_m) is transformed to m' which maps to $(P_{m'}, C_{m'})$ where:

$$P_{m'} = P_m - \{ch_{\mathsf{s}}\}$$
$$C_{m'} = \{ch_{\mathsf{ch}} \mid \{ch_{\mathsf{s}}, ch_{\mathsf{t}}\} \cap P_{m'} \neq \emptyset\}$$

and the $s_{m'}$, $t_{m'}$ maps work as expected, i.e. $s_{m'}(ch_{\mathsf{ch}}) = ch_{\mathsf{s}}$ and $t_{m'}(ch_{\mathsf{ch}}) = ch_{\mathsf{t}}$. It now makes sense to define the manager reconfiguration transition $m \text{ -}r\text{-}\rightarrow m'$ as the partial injection

$$\chi_{m,m'} : C_m \rightarrow C_{m'}$$

which is the maximal identity function on $C_m \cap C_{m'}$.

Suppose that above m we had the n pure workers $\{pw_j^{\theta_j} \mid j \in \{1 \ldots n\}\}$. Then above m' we will also have n pure workers. For $j \neq i$, $pw_j^{\theta_j}$ will continue to be above m' and the reconfiguration transition $m \text{ -}r\text{-}\rightarrow m'$ will leave it in the same state as it was. For the case $j = i$ we have instead the pure worker $pw_i^{\theta_i'} = (CH_{\mathsf{i}i}^{\theta_i'}, CH_{0i}^{\theta_i'}, K_i^{\phi(\theta_i')})$ where:

$$CH_{\mathsf{i}i}^{\theta_i'} = CH_{\mathsf{i}i}^{\theta_i}$$
$$CH_{0i}^{\theta_i'} = CH_{0i}^{\theta_i} - \{ch_0\}$$

and so we can summarise the above map for m' as:

$$\{pw_i^{\theta_j\wedge m'} \mid pw_i^{\theta_j\wedge m}, j \in \{1 \ldots n\} - \{i\}\} \cup \{pw_i^{\theta_i'\wedge m'}\}$$

2. Since there is only one nontrivial manager, we suppress the '*pm*' tags for convenience.

The $\lambda_{m'}$ map is:

$$\lambda_{m'} = \lambda_m - \{ch_s \mapsto ch_0\}$$

and we have that:

$$r_{pw_i^{\theta_i \wedge} m}(rec(out_0!ch_0)) = m \text{ -}r\text{-}\rightarrow m'$$

which completes the piece of the recursion for the case of a $rec(out_0!ch_0)$ reconfiguration. If we consider instead $rec(out_0!ch_i)$, $rec(in_i?ch_0)$, $rec(in_i?ch_i)$ reconfigurations, the above is modified respectively by:

$$CH_{ii}^{\theta_i'} = CH_{ii}^{\theta_i} - \{ch_i\} \;;\; CH_{0i}^{\theta_i'} = CH_{0i}^{\theta_i} \;;$$
$$P_{m'} = P_m - \{ch_t\} \;;\; C_{m'} = \{ch_{ch} \mid \{ch_s, ch_t\} \cap P_{m'} \neq \emptyset\} \;;$$
$$\lambda_{m'} = \lambda_m - \{ch_t \mapsto ch_i\}$$

$$CH_{ii}^{\theta_i'} = CH_{ii}^{\theta_i} \;;\; CH_{0i}^{\theta_i'} = CH_{0i}^{\theta_i} \cup \{ch_0\} \;;$$
$$P_{m'} = P_m \cup \{ch_s\} \;;\; C_{m'} = \{ch_{ch} \mid \{ch_s, ch_t\} \cap P_{m'} \neq \emptyset\} \;;$$
$$\lambda_{m'} = \lambda_m \cup \{ch_s \mapsto ch_0\}$$

$$CH_{ii}^{\theta_i'} = CH_{ii}^{\theta_i} \cup \{ch_i\} \;;\; CH_{0i}^{\theta_i'} = CH_{0i}^{\theta_i} \;;$$
$$P_{m'} = P_m \cup \{ch_t\} \;;\; C_{m'} = \{ch_{ch} \mid \{ch_s, ch_t\} \cap P_{m'} \neq \emptyset\} \;;$$
$$\lambda_{m'} = \lambda_m \cup \{ch_t \mapsto ch_i\}$$

together with the obvious consequences. Since the ABB system enjoys the property that a component cannot give away a channel end that it is not connected to and neither does it ever receive a channel end that it already possesses, it readily follows that the set operations above are nonnull.

Beyond these there are the expected identity transitions on states of M of course, which completes the construction. Thus we have cut up the original ABB system into a collection of pieces that can be reassembled as an IWIM system, in order that the latter is able to achieve the same effect as the original system. In fact it is easy to convince onself that the IWIM system constructed from a given ABB system by the above technique is able to simulate it in the sense that non-reconfiguring inputs and outputs correspond bijectively, while reconfiguring inputs and outputs correspond to sequences of two steps in the IWIM system, the first to receive or transmit the channel end identifier, the second to provoke the desired reconfiguration via the manager.

7 The Katis, Sabadini, Walters Model

In this section we consider a model proposed by Katis, Sabadini and Walters in [Katis et al. (2000)], henceforth the KSW model, and show how it too can be subsumed within our framework. In the KSW model, the main entity of interest is the CP automaton. A CP automaton $\mathsf{G} = (\mathbf{G}, X, Y, A, B, \partial_0, \partial_1, \gamma_0, \gamma_1)$, consists of a directed graph $\mathbf{G} = (G_0, G_1)$ where G_0 is the set of nodes and G_1 is the set of arcs, together with four maps:

$$\partial_0 : G_1 \to X \;;\; \partial_1 : G_1 \to Y \;;\; \gamma_0 : A \to G_0 \;;\; \gamma_1 : B \to G_0$$

These work as follows. The arcs of the graph represent transitions of the automaton, whose states are the nodes. The sets X and Y are input and output alphabets respectively. Thus the maps $\partial_0 : G_1 \to X$ and $\partial_1 : G_1 \to Y$ describe which input letter a transition of the graph consumes, and which output letter it produces. Since both maps are total, each transition involves both input and output. We will write a CP automaton transition as:

s -(*ind*, *arc*, *outd*)-› t

where s and t are states, *arc* is the arc carrying the transition, and *ind*, *outd* are the input and output data. (In [Katis et al. (2000)], the authors also admit null elements in both X and Y alphabets, to aid abstraction and to represent the absence of genuine communication during a step.) Communication is synchronous, thus when two CP automata communicate, the symbol output by the producer of the communication, is simultaneously input by the consumer of the communication. Most emphatically, there are no queues in the model: communication in this model is above all a synchronisation mechanism.

The sets A and B (called the in-condition and out-condition respectively in [Katis et al. (2000)]), are to do with initialisation and finalisation, though in a slightly non-standard manner. Specifically, the γ_0-image of A is the set of entry points into the CP automaton, i.e. initial states, and the γ_1-image of B is the set of exit points, i.e. final states, of the automaton — except that when CP automata are combined in the appropriate way, then subsets of entry or exit points may be identified, leading to a richer gamut of possibilities parameterised by partitions of $\gamma_0(A)$ and $\gamma_1(B)$.

CP automata are endowed with a number of algebraic operations, which construct more complex CP automata out of simpler ones. We will model the KSW formalism by mapping CP automata to IWIM systems, and then showing how the CP automaton algebraic operations can be reflected in constructions on the corresponding IWIM systems.

Let $\mathsf{G} = (\mathbf{G} = (G_0, G_1), X, Y, A, B, \partial_0, \partial_1, \gamma_0, \gamma_1)$ be a CP automaton. We build an IWIM system corresponding to G, and consisting of a pure manager and a pure worker. The pure manager *pm* has one-state $\blacklozenge$ which maps to $(\{p_s, p_t\}, \{ch_s, ch_t\})$ with $s_\blacklozenge(ch_s) = p_s$ and $t_\blacklozenge(ch_t) = p_t$ (and with $s_\blacklozenge(ch_t)$ and $t_\blacklozenge(ch_s)$ undefined). The state $\blacklozenge$ is initial and the only transition of the manager is the identity. Clearly the manager's structure is independent of G.

The pure worker *pw* is $(\{p_i\}, \{p_0\}, (St, Init, Tr))$ where the transition system *Tr* is constructed thus. For each G transition s -(*ind*, *arc*, *outd*)-› t, *Tr* contains the two step sequence s -p_i?*ind*-› *arc* -p_0!*outd*-› t ; this makes it clear that $St = G_0 \cup G_1$ (we will tacitly assume that this union is disjoint). Regarding *Init*, we can choose *any* state s_0 in $\gamma_0(A)$ to be *Init*. Thus the mapping from CP automata to IWIM systems is in general one to many. In reality of course, examples of CP automata that represent complete systems typically have unique initial states, reflecting the often observed fact that most real systems start in a well defined condition. The plurality comes in useful when component CP automata are combined to form the a larger system. We will

comment on this further below. More generally, $\gamma_0(A)$ and $\gamma_1(B)$ are sets of states of the pure worker *pw*.

Our basic construction is nearly complete. All that remains is to note that the λ mapping is given by:

$$\lambda_{\blacklozenge}(p_s) = p_0 \ ; \ \lambda_{\blacklozenge}(p_t) = p_i$$

that the above mapping is given by:

$$pw^{\wedge}{}_{\blacklozenge pm}$$

and that since there are no *rec* actions in the worker, the *r* map is empty.

Note the following invariant of the generated IWIM system: regardless of G, there is exactly one pure worker, one one-state pure manager, one external input channel, one external output channel, and $\gamma_0(A)$ and $\gamma_1(B)$ can be identified with sets of configurations of the pure worker.

We can easily see that whatever the initial state of the given CP automaton, we can find an IWIM system from among the possibilities constructed, with the same initial state; and which furthermore simulates it in the sense that the execution of a CP automaton transition inputting *x* and outputting *y*, corresponds in the IWIM system to the input from the input queue of *x* and the output onto the output queue of *y*, in that order. (The alternative order leads to an equally acceptable construction.) Note that in the IWIM system these are comunications with the environment.

We now move on to constructions on CP automata and how these are reflected in the corresponding IWIM systems; the principal ones that we must consider are binary combinators. We will subscript with the name of the relevant automaton to disambiguate when notations would otherwise clash.

Communicating Parallel Composition. Let $G = (\mathbf{G} = (G_0, G_1), X, Y, A, B, \partial_{0,G}, \partial_{1,G}, \gamma_{0,G}, \gamma_{1,G})$ and $H = (\mathbf{H} = (H_0, H_1), Y, Z, C, D, \partial_{0,H}, \partial_{1,H}, \gamma_{0,H}, \gamma_{1,H})$ be CP automata. Then the communicating parallel composition of G and H, written $G \cdot H$, is the CP automaton:

$$\begin{aligned} G \cdot H = (&\mathbf{G}\cdot\mathbf{H} = (G_0 \times H_0, G_1 \cdot H_1 = \{(g, h) \mid g \in G_1, h \in H_1, \partial_{1,G}(g) = \partial_{0,H}(h)\}),\\ &X, Z,\\ &A \times C, B \times D,\\ &\partial_{0,G\cdot H}(g, h) = \partial_{0,G}(g),\ \partial_{1,G\cdot H}(g, h) = \partial_{1,H}(h),\\ &\gamma_{0,G\cdot H} = \gamma_{0,G} \times \gamma_{0,H},\ \gamma_{1,G\cdot H} = \gamma_{1,G} \times \gamma_{1,H}) \end{aligned}$$

This definition makes clear the statement above that communication is synchronous in the KSW model. The input and output labels on an arc (g, h) of the combined system are $\partial_{0,G}(g)$ and $\partial_{1,H}(h)$ respectively, while the very existence of the arc is predicated on the condition $\partial_{1,G}(g) = \partial_{0,H}(h)$, which supports the interpretation that arc *g* output and arc *h* input the same symbol. This is the only notion of communication in the KSW model.

We model the communicating parallel composition of G and H at the IWIM system level as follows. Suppose WM_G is an IWIM system representing G, and WM_H is an IWIM system representing H. We assume that both WM_G and WM_H each have a pure worker, pw_G and pw_H respectively, a one-state pure manager, pm_G and pm_H respectively, an external input channel $ch_{t,G}$ and $ch_{t,H}$ respectively, an external output channel $ch_{s,G}$ and $ch_{s,H}$ respectively, that $\gamma_{0,G}(A)$ and $\gamma_{1,G}(B)$ can be identified with a set of states of pw_G, and that $\gamma_{0,H}(C)$ and $\gamma_{1,H}(D)$ can be identified with a set of states of pw_H. The IWIM system $WM_{G\cdot H}$ we seek can be generated from WM_G and WM_H as follows.

There is the usual one-state pure manager $pm_{G\cdot H}$ as above. The corresponding pure worker $pw_{G\cdot H} = (\{p_i\}, \{p_0\}, (St_{G\cdot H}, Init_{G\cdot H}, Tr_{G\cdot H}))$ is built from pw_G and pw_H by defining $St_{G\cdot H} = St_G \times St_H$, $Init_{G\cdot H} = (Init_G, Init_H)$, and for $Tr_{G\cdot H}$, whenever we have a pair of transitions in Tr_G of the form s_G -p_i?*ind*-› $arc_{st,G}$ -p_0!*val*-› t_G, and a pair of transitions in Tr_H of the form s_H -p_i?*val*-› $arc_{st,H}$ -p_0!*outd*-› t_H, we form the $Tr_{G\cdot H}$ transitions (s_G, s_H) -p_i?*ind*-› $(arc_{st,G}, arc_{st,H})$ -p_0!*outd*-› (t_G, t_H). It is clear that this procedure only succeeds because of the special structure of the transition systems Tr_G and Tr_H. We can now identify $\gamma_{0,G\cdot H}(A \times C)$ with states corresponding to $\gamma_{0,G}(A) \times \gamma_{0,H}(C)$, and $\gamma_{1,G\cdot H}(B \times D)$ with states corresponding to $\gamma_{1,G}(B) \times \gamma_{1,H}(D)$; and the rest of the data for the IWIM system $WM_{G\cdot H}$ is routine.

It is obvious that $WM_{G\cdot H}$ is able to simulate $G \cdot H$ in a straightforward manner provided WM_G can simulate G and WM_H can simulate H.

Parallel Composition without Communication. Let $G = (\mathbf{G} = (G_0, G_1), X, Y, A, B, \partial_{0,G}, \partial_{1,G}, \gamma_{0,G}, \gamma_{1,G})$ and $H = (\mathbf{H} = (H_0, H_1), Z, W, C, D, \partial_{0,H}, \partial_{1,H}, \gamma_{0,H}, \gamma_{1,H})$ be CP automata. Then the noncommunicating parallel composition of G and H, written $G \times H$, is the CP automaton:

$$\begin{aligned} G \times H = (&\mathbf{G} \times \mathbf{H} = (G_0 \times H_0, G_1 \times H_1), X \times Z, Y \times W, A \times C, B \times D, \\ &\partial_{0,G\times H}(g, h) = \partial_{0,G}(g) \times \partial_{0,H}(h), \partial_{1,G\times H}(g, h) = \partial_{1,G}(g) \times \partial_{1,H}(h), \\ &\gamma_{0,G\times H} = \gamma_{0,G} \times \gamma_{0,H}, \ \gamma_{1,G\times H} = \gamma_{1,G} \times \gamma_{1,H}) \end{aligned}$$

This noncommunicating parallel composition still features synchronous communication, but this time of pairs of data values.

We model the noncommunicating parallel composition of G and H at the IWIM system level thus. Let WM_G and WM_H be IWIM systems representing G and H respectively. We assume that WM_G and WM_H have pure workers, pw_G and pw_H, one-state pure managers, pm_G and pm_H, external input channels $ch_{t,G}$ and $ch_{t,H}$, external output channels $ch_{s,G}$ and $ch_{s,H}$, that $\gamma_{0,G}(A)$ and $\gamma_{1,G}(B)$ can be identified with a set of states of pw_G, and that $\gamma_{0,H}(C)$ and $\gamma_{1,H}(D)$ can be identified with a set of states of pw_H. Then we proceed as follows to construct $WM_{G\times H}$.

There is the usual one-state pure manager $pm_{G\times H}$ as above. We build a corresponding pure worker $pw_{G\times H} = (\{p_i\}, \{p_0\}, (St_{G\times H}, Init_{G\times H}, Tr_{G\times H}))$ from pw_G and pw_H by defining $St_{G\times H} = St_G \times St_H$, $Init_{G\times H} = (Init_G, Init_H)$, and for $Tr_{G\times H}$, whenever we have a pair of transitions in Tr_G of the form s_G -p_i?ind_G-› $arc_{st,G}$ -p_0!$outd_G$-› t_G, and a

pair of transitions in Tr_H of the form s_H -p_i?ind_H-› $arc_{st,H}$ -p_0!$outd_H$-› t_H, we form the $Tr_{G\times H}$ transition pair:

(s_G, s_H) -p_i?(ind_G, ind_H)-› $(arc_{st,G}, arc_{st,H})$ -p_0!$(outd_G, outd_H)$-› (t_G, t_H).

We can now identify $\gamma_{0,G\times H}(A \times C)$ with states corresponding to $\gamma_{0,G}(A) \times \gamma_{0,H}(C)$, and $\gamma_{1,G\times H}(B \times D)$ with states corresponding to $\gamma_{1,G}(B) \times \gamma_{1,H}(D)$; and the rest of the data for $WM_{G\times H}$ is routine.

It is obvious that $WM_{G\times H}$ is able to simulate $G \times H$ in a straightforward manner provided WM_G can simulate G and WM_H can simulate H.

Up to now, the in-conditions and out-conditions of the component CP automata have played a passive role; the next construction remedies this.

Restricted Sum. Let $G = (\mathbf{G} = (G_0, G_1), X, Y, A, B, \partial_{0,G}, \partial_{1,G}, \gamma_{0,G}, \gamma_{1,G})$ and $H = (\mathbf{H} = (H_0, H_1), X, Y, B, C, \partial_{0,H}, \partial_{1,H}, \gamma_{0,H}, \gamma_{1,H})$ be CP automata. Then the restricted sum of G and H, written $G + H$, is the CP automaton:

$$\begin{aligned} G + H = (&\mathbf{G} + \mathbf{H} = (G_0 + H_0 \,/\sim_B \text{ where } \sim_B \text{ is the finest equivalence} \\ &\qquad \text{relation generated by } \gamma_{1,G}(b) \sim_B \gamma_{0,H}(b) \text{ (and we write} \\ &\qquad [g]_B \text{ for the equivalence class containing } g),\ G_1 + H_1), \\ &X, Y, A, C, \\ &\partial_{0,G+H} = \partial_{0,G} + \partial_{0,H},\ \partial_{1,G+H} = \partial_{1,G} + \partial_{1,H}, \\ &\gamma_{0,G+H} = \gamma_{0,G},\ \gamma_{1,G+H} = \gamma_{1,H}) \end{aligned}$$

(As expected, the sources and targets of the arcs in $G_1 + H_1$ are the equivalence classes of the corresponding sources and targets in G_0 and H_0.)

Let WM_G and WM_H be IWIM systems representing G and H respectively. We assume that WM_G and WM_H have pure workers, pw_G and pw_H, one-state pure managers, pm_G and pm_H, external input channels $ch_{t,G}$ and $ch_{t,H}$, external output channels $ch_{s,G}$ and $ch_{s,H}$, that $\gamma_{0,G}(A)$ and $\gamma_{1,G}(B)$ can be identified with a set of states of pw_G via maps $\gamma_{w0,G} : A \to St_G$, $\gamma_{w1,G} : B \to St_G$, and that $\gamma_{0,H}(B)$ and $\gamma_{1,H}(C)$ can be identified with a set of states of pw_H via maps $\gamma_{w0,H} : B \to St_H$, $\gamma_{w1,H} : C \to St_H$. We proceed as follows to construct WM_{G+H}.

There is the usual one-state pure manager pm_{G+H} as above. We build a corresponding pure worker $pw_{G+H} = (\{p_i\}, \{p_0\}, (St_{G+H}, Init_{G+H}, Tr_{G+H}))$ from pw_G and pw_H by defining:

$$\begin{aligned} St_{G+H} &= St_G + St_H \,/\sim_B \text{ where } \sim_B \text{ is the finest equivalence relation} \\ &\qquad \text{generated by } \gamma_{w1,G}(b) \sim_B \gamma_{w0,H}(b) \text{ (and we write} \\ &\qquad [s]_B \text{ for the equivalence class containing } s) \end{aligned}$$

$$Init_{G+H} = [Init_G]_B$$

$$\begin{aligned} Tr_{G+H} = \{&[s]_B \text{ -}p_i\text{?}v\text{-› } [t]_B \mid [s]_B, [t]_B \in St,\ s \text{ -}p_i\text{?}v\text{-› } t \in Tr_{G,I} \cup Tr_{H,I}\} \cup \\ \{&[s]_B \text{ -}p_0\text{!}v\text{-› } [t]_B \mid [s]_B, [t]_B \in St,\ s \text{ -}p_0\text{!}v\text{-› } t \in Tr_{G,O} \cup Tr_{H,O}\} \end{aligned}$$

That this works as desired is conditional on the observation that in both pw_G and pw_H, the states picked out by $\gamma_{w0,G}$, $\gamma_{w1,G}$, $\gamma_{w0,H}$, $\gamma_{w1,H}$ are, so to speak, 'G_0-states' and not

'*arc*-states'. This can be assured by choosing $\gamma_{w0,G}$, $\gamma_{w1,G}$, $\gamma_{w0,H}$, $\gamma_{w1,H}$ to be $\gamma_{0,G}$, $\gamma_{1,G}$, $\gamma_{0,H}$, $\gamma_{1,H}$ in the base case construction, whereupon it evidently persists through the binary combinator simulations we have described, and enables us to formally identify $\gamma_{0,G+H} = \gamma_{0,G}$ with a set of states of pw_{G+H} via $\gamma_{w0,G+H} : A \to St_{G+H} = \gamma_{w0,G} / \sim_B$ and to identify $\gamma_{1,G+H} = \gamma_{1,H}$ with a set of states of pw_{G+H} via $\gamma_{w1,G+H} : C \to St_{G+H} = \gamma_{w1,H} / \sim_B$. With this confirmed, the construction of $St_G + St_H / \sim_B$ results in a gluing of s -p_i?*ind*-› *arc* -p_0!*outd*-› t sequences only at their ends, and it then becomes easy to see that the given recipe gives us an IWIM system WM_{G+H} capable of simulating the CP automaton $G + H$, if WM_G simulates G and WM_H simulates H.

Two points deserve comment. Firstly, [Katis et al. (2000)] speak of the need to 'adjust' the in-conditions or out-conditions of a CP automaton in order to make it fit for some particular purpose. More than anything else this is an indication that these interconnection aspects of the automaton are really properties that belong more to the interconnection mechanism itself, than to the automata involved.

Refering back to our IWIM system scenario, we have recognised this, and reflected it in the design of our various IWIM system pullback and pushout operations, in which the intermediate worker-manager $wm^\bullet$ was outside the system being manipulated, i.e. $wm^\bullet$ (and its attendant homomorphisms) parameterise the operation itself, and do not form part of the entities being operated on.

Secondly if, following [Katis et al. (2000)], we intend the restricted sum to model sequential composition, the construction of WM_{G+H}, though faithful to the CP automaton $G + H$, suffers from the weakness pointed out in Section 4, namely that if a final state of G has out-transitions, and a corresponding initial state of H has in-transitions, then a run may wander from G to H and then back in to G. The IWIM system paradigm offers more flexibility here, allowing the expression of an irreversible transition from G to H. We describe the details, resulting in the construction of an IWIM system WM^*_{G+H} that simulates $G + H$ in a different way.

Suppose in $G_0 + H_0 / \sim_B$ above, there are k of the equivalence classes that are non-singletons, i.e. there are k classes that glue at least one element of G_0 to at least one element of H_0 (the remaining classes just containing individual elements outside the ranges of $\gamma_{1,G}(B)$ and $\gamma_{0,H}(B)$). Call them:

$$[\gamma_{w1,G}(b)_1], [\gamma_{w1,G}(b)_2] \ldots [\gamma_{w1,G}(b)_k]$$

Now partition each of $[\gamma_{w1,G}(b)_1] \ldots [\gamma_{w1,G}(b)_k]$ into two subsets each:

$$[\gamma_{w1,G}(b)_1]_G = [\gamma_{w1,G}(b)_1] \cap G_0 \quad \text{and} \quad [\gamma_{w1,G}(b)_1]_H = [\gamma_{w1,G}(b)_1] \cap H_0$$
$$\ldots$$
$$[\gamma_{w1,G}(b)_k]_G = [\gamma_{w1,G}(b)_k] \cap G_0 \quad \text{and} \quad [\gamma_{w1,G}(b)_k]_H = [\gamma_{w1,G}(b)_k] \cap H_0$$

all nonempty by our assumptions. Replacing in St_{G+H} the $[\gamma_{w1,G}(b)_1] \ldots [\gamma_{w1,G}(b)_k]$ by the $[\gamma_{w1,G}(b)_1]_G, [\gamma_{w1,G}(b)_1]_H \ldots [\gamma_{w1,G}(b)_k]_G, [\gamma_{w1,G}(b)_k]_H$ is tantamount to generating a new equivalence relation, which we call B^*, on the state space $St_G + St_H$. This is the finest relation generated by the two families of clauses:

$$(\gamma_{w1,G}(b) \sim_B \gamma_{w0,H}(b) = \gamma_{w0,H}(c) \sim_B \gamma_{w1,G}(c)) \Rightarrow \gamma_{w1,G}(b) \sim_{B^*} \gamma_{w1,G}(c))$$

$$(\gamma_{w0,H}(b) \sim_B \gamma_{w1,G}(b) = \gamma_{w1,G}(c) \sim_B \gamma_{w0,H}(c)) \Rightarrow \gamma_{w0,H}(b) \sim_{B^*} \gamma_{w0,H}(c))$$

Now we define:

$$St^*_{G+H} = (St_{G+H} - \{[\gamma_{w1,G}(b)_1] \ldots [\gamma_{w1,G}(b)_k]\}) \cup \{[\gamma_{w1,G}(b)_1]_G, [\gamma_{w1,G}(b)_1]_H \ldots [\gamma_{w1,G}(b)_k]_G, [\gamma_{w1,G}(b)_k]_H\}$$

$$Init^*_{G+H} = [Init_G]_{B^*}$$

$$Tr^*_{G+H} = \{[s]_{B^*} \text{-}p_i?v\text{-}\rightarrow [t]_{B^*} \mid [s]_{B^*}, [t]_{B^*} \in St, s \text{-}p_i?v\text{-}\rightarrow t \in Tr_{G,I} \cup Tr_{H,I}\} \cup \{[s]_{B^*} \text{-}p_0!v\text{-}\rightarrow [t]_{B^*} \mid [s]_{B^*}, [t]_{B^*} \in St, s \text{-}p_0!v\text{-}\rightarrow t \in Tr_{G,O} \cup Tr_{H,O}\} \cup \{[s]_{B^*} \text{-}rec\text{-}\rightarrow [t]_{B^*} \mid s = \gamma_{w1,G}(b) = \gamma_{w0,H}(b) = t\ ,\ b \in B\}$$

By distinguishing the G from the H components of the glueing states, we are able to introduce *rec* transitions from one to the other. All of these *rec* transitions are above the unique state of the pure manager, and all map to the identity reconfiguration on the corresponding port/channel network ($\{p_s, p_t\}$, $\{ch_s, ch_t\}$). Since the pure worker remains above this state when such a *rec* transition is executed, its *rec* transition completes and the run continues in the H component; however this time there is no way back to the G component, even if there are in-transitions to the initial state of H used, and out-transitions from the final state of G reached.

This all works adequately, but is still open to the criticism that pure worker pw_G, its useful life over when the locus of control moves into the pw_H part of the system, remains alive, though defunct, preventing its resources from being reused. In a real system, it would be garbage collected releasing its resources for other activities. Equally, a demand driven implementation might well not create the pw_H part of the system until it was needed. Our IWIM system model enables us to express these aspects though we will not go into all the formal details. Here is the general idea.

We split the state of the pure manager into two; and (a modified) pw_G is above the new initial state, while pw_H is above the other state. There is a reconfiguration transition from the former to the latter, whose data is the identity reconfiguration on the port/channel network ($\{p_s, p_t\}$, $\{ch_s, ch_t\}$). The modification to pw_G entails adding the $[\gamma_{w1,G}(b)_1]_G \ldots [\gamma_{w1,G}(b)_k]_G$ states described previously to its state space, and then adding *rec* transitions to a typical $[\gamma_{w1,G}(b)_j]_G$ state from each of its comprising $\gamma_{w1,G}(b)_j$ states. These *rec* transitions map to the reconfiguration mentioned above.

It is clear that the behaviours of the resulting system are as follows. The manager starts in its initial state; consequently the modified pw_G is active. It executes until it reaches a $\gamma_{w1,G}(b)_j$ state and proceeds to perform the $\gamma_{w1,G}(b)_j$ *-rec-*$\rightarrow [\gamma_{w1,G}(b)_j]_G$ transition. This maps to the reconfiguration step of the manager, and because pw_H is above the new manager state, the modified pw_G leaves the system configuration and pw_H joins it, starting in its initial state.

This story holds up if H has a unique initial state. If not, an unwinding technique similar to that used in our ABB system simulation must be employed.

Furthermore, the nontrivial state space now introduced for the manager has consequences for all the combinators. A product-like construction must be used on the manager states for the communicating and noncommunicating parallel compositions, while a sum-like construction, involving the introduction of reconfiguration transitions must be used for the restricted sum. We leave the fascinating details for the motivated reader.

8 Conclusions

In the preceding sections we have introduced a formal model for capturing some of the essence of the IWIM concept in an automata based framework. Since the key idea in IWIM is that manger processes exercise some degree of control over their subordinate workers, expressing this in a theoretical framework inevitably leads to some complexity, and we have seen this reflected in the constructions we have described. Despite this, the model that emerges enjoys a selection of appealing properties, ranging from the projection results of Section 3, to the various algebraic constructions presented in Section 4, which as we said, contains a by no means exhaustive list of such possibilities.

Part of the reason for these appealing phenomena rests in the fact that the design of the model was tacitly undertaken in a manner in sympathy with categorical imperatives — though no explicit mention was made of categorical concepts aside from the naming of constructions in Section 4 — a strategy which was conducive to the fostering of relatively elegant structural properties. Still it is by no means the case that such categorical properties are the only ones of practical interest, as the more ad hoc constructions of Sections 5, 6 and 7 made abundantly clear. Regarding the latter it is noteworthy that despite the emphasis on algebraic structures in the KSW model, to capture the KSW ideas in our own model, we were not able to make use of the algebraic combinators we spent time describing in Section 4. One observation suffices to make clear why this is not in hindsight unexpected.

Consider communicating parallel composition. The most appealing way to model this using the techniques from Section 4, is to pipe the output of the first worker into the input of the second. This idea gives a system that behaves as expected. Nevertheless there is a problem when one wishes to form the restricted sum of such a communicating parallel composition with another system. What are the final states of the parallel composition that one can glue to the other system? They are, unfortunately, pairs of final states of the communicating components, implicit in configurations of the system, but not explicit in the static description of the system without unwinding it (essentially this unwinding is what the ad hoc construction given for communicating parallel composition accomplishes). So the obvious way of modelling the algebraic operators of the KSW theory (which are combinators on the static descriptions of KSW systems), as combinators on the static descriptions of the translated components, does not succeed. In particular we cannot translate communicating parallel compositions of KSW systems into networks of communicating IWIM subsystems.

This is a consequence of the fact that the KSW model is a global state model, i.e any state of a configuration is precisely one of the states occurring in the static description of the system. This does not happen in the ABB model, nor in ours, because in both cases the states of runtime configurations, are more complicated structures built out of the states mentioned in the static description of the system. For such systems, which (let us face it) give a more natural account of typical distributed systems with their de facto distributed global state, the notion of sequential composition, one of the objectives of the sum construction, is a non-trivial issue. Concerning such systems, sequential composition is: either ignored completely; or is a feature that becomes available only after a substantial investment of theoretical effort (to perform the required unwinding); or in practical scenarios, requires the use of a serviceable distributed termination algorithm. Petri nets (see eg. [Best et al. (2000)]) is a well known formalism that exhibits the same characteristics.

The fact that we were able to simulate other formal approaches to IWIM in our model, means that we gain the capability of inheriting results obtained in these models, in ours. One particular instance that comes to mind concerns the deadlock avoidance results proved under suitable conditions for the ABB model in [Arbab et al. (2000a)]. Another concerns the algebraic operations considered in the context of the KSW model in [Katis et al. (2000)], which helped to stimulate the development of the algebraic properties of ours. Regarding the latter, we have not confronted the normal questions that arise concerning the coherence of combinations the various operations, commutativity, associativity, and so on. However, recognising that our models are built using elementary set theoretic machinery, we do not anticipate problems provided we are prepared to take results up to set theoretic isomorphism.

The juxtaposition of conventionally inspired algebraic properties with the more ad hoc constructions appearing directly afterwards, illustrates that the agendas of algebra and of system design cannot always be relied upon to coincide. While the former can give a useful perspective at a high level of abstraction, more specialised 'bricollage' is often needed to accomplish desired lower level goals while expending no more than a reasonable amount of effort. To put it another way, perhaps more clearly, the way a system can be decomposed as recommended by a particular suite of algebraic primitives, may well not coincide with the way that the same system can be decomposed respecting 'application level concerns'. The former are normally designed with genericity in mind, while the latter can exploit specific (and usually crucial) features of the application to achieve a much more natural account for the system in question, even if the techniques utilised do not generalise to arbitrary systems. It is no more than a little ironic that in this paper, this point has been illustrated by considering the naturally arising generic algebraic primitives of one model, and considering the question of how these might best be expressed using the naturally arising generic algebraic primitives of another model. More generally it illustrates that relying on some fixed set of algebraic or other tools, and ignoring the tighter properties that specific systems enjoy, restricts expressivity.

Finally we observe that coordination models different from the IWIM one, and in particular the global state tuple based approaches, must nevertheless embody the ca-

pacity for disentangling management from worker aspects, so readily done for IWIM, even if only implicitly. The challenge of extracting this structure from so different looking starting points remains an intriguing issue to explore in future publications.

Acknowledgement

The work described in this paper was partially supported by the EU in the course of the KIT-INCO Project SEEDIS (Contract No. 962114).

References

Agha G. (1986); Actors: A Model of Concurrent Computation in Distributed Systems. MIT Press.

Arbab F. (1995); Coordination of Massively Concurrent Activities. CWI Tech. Rep. CS-R9565.

Arbab F. (1996); The IWIM Model for Coordination of Concurrent Activities. *in*: Proc. COORD-96, Ciancarini, Hankin (eds.), LNCS **1061**, 34-56, Springer.

Arbab F., Herman I., Spilling P. (1993); An overview of Manifold and its Implementation. Concurrency: Practice and Experience **5**, 23-70.

Arbab F., Blom C. L., Burger F. J., Everaars C. T. H. (1998); Rusable Coordination Modules for Massively Concurrent Applications. Software: Practice and Experience **28**, 703-735.

Arbab F., de Boer F. S., Bonsangue M. M. (2000a); A Logical Interface Description Language for Components. *in*: Proc. COORD-00, Porto, Roman (eds.), LNCS **1906**, 249-266, Springer.

Arbab F., de Boer F. S., Bonsangue M. M. (2000b); A Coordination Language for Mobile Components. *in*: Proc. ACM SAC-00, 166-173.

Best E., Devillers R., Koutny M. (2000); Petri Net Algebra. Springer.

Bonsangue M. M., Arbab F., de Bakker J. W., Rutten J. J. M. M., Scutellà A., Zavattaro G. (2000); A Transition System Semantics for the Control-Driven Coordination Language MANIFOLD. Theor. Comp. Sci. **240**, 3-47.

Carriero N., Gelernter D. (1989); LINDA in Context. Comm. ACM **32**, 444-458.

Ciancarini P., Hankin C. H. L. (eds.) (1996); Coordination Languages and Models 1996 (Proc. COORD-96). LNCS **1061**, Springer.

Ciancarini P., Wolf A. L. (eds.) (1999); Coordination Languages and Models 1999 (Proc. COORD-99). LNCS **1594**, Springer.

Garlan D., Le Metayer D. (eds.) (1997); Coordination Languages and Models 1997 (Proc. COORD-97). LNCS **1282**, Springer.

Gelernter D. (1985); Generative Communication in Linda. ACM Trans. Prog. Lang. Sys. **7**, 80-112.

Katis P., Sabadini N., Walters R. F. C. (2000); A Formalisation of the IWIM Model. *in*: Proc. COORD-00, Porto, Roman (eds.), LNCS **1906**, 267-283, Springer.

Malone T., Crowston K. (1994); The Interdisciplinary Study of Coordination. ACM Comp. Surv. **26**, 87-119.

Omicini A., Zambonelli F., Klusch M., Tolksdorf R. (2002); Coordination of Internet Agents: Models, Technologies, and Applications. Springer.

Papadopoulos G. A., Arbab F. (1998); Coordination Models and Languages. *in*: Advances in Computers — The Engineering of Large Systems, Zelkowitz (ed.), 329-400, Academic.

Porto A., Roman G-C. (eds.) (2000); Coordination Languages and Models 2000 (Proc. COORD-00). LNCS **1906**, Springer.

Shapiro E. (1989); The Family of Concurrent Logic Languages. ACM Comp. Surv. **21**, 412-510.