An Automata Model of Distributed Systems

P. Katis[1], R. Rosebrugh[2], N. Sabadini[1], and R.F.C. Walters[1]

[1] Dipartimento di Scienze CC. FF. MM., Università degli Studi dell'Insubria, Como, Italy

[2] Department of Mathematics and Computer Science, Mount Allison University, Sackville, New Brunswick, Canada

Abstract. Three examples of the use of case-place automata are given: (i) a compositional model checking algorithm, (ii) a formalization of the IWIM coordination model, and (iii) an example of the representation of mobile processes.

1 Introduction

There are a variety of automata models of concurrent systems based on products of automata, the most prominent being asynchronous automata [16], and the synchronous product of automata of Arnold and Nivat [6]. These theories however have limitations in the modelling of distributed systems. The first is that they have a fixed geometry. The fact that asynchronous automata have a fixed dependence relation has been regarded in the concurrency community as a serious drawback. Secondly the theories lack compositionality. To achieve compositionality and variability of geometry is precisely the purpose of the algebra of case-place automata developed in this paper.

In previous articles [10], [11], the authors have introduced automata suitable for describing the normal communicating-parallel activity of workers, which automata we called *spans of graphs* from their categorical origin, but which might be more appropriately called *place automata*. In section 2 of this paper we show how place automata may be used to do compositional model checking in the style of (but with important differences from) [8], [9].

In section 3 we give the formal definition of *case place* (or CP) *automaton*. We introduce the operations of parallel composition and restricted sum. We then show how CP automata may be used to formalize the IWIM ('idealized workers and idealized managers') coordination model, doing a case study of a distributed bucket sort. This work has also been reported in [13]. In section 4 we show how mobility of processes may be modelled using CP automata, introducing at the same time two extra feedback operations on CP automata.

In the remainder of the introduction we describe the notation we use for sets and functions and their operations.

Notation By a *graph* **G** we mean a set G_0 of vertices and a set G_1 of (directed) arcs, together with two functions d_0, $d_1 : G_1 \to G_0$ which specify the source and target, respectively, of each arc. A *morphism* from **G** to **H** consists of a function from vertices to vertices, and a function from arcs to arcs which respects the source and target functions; an isomorphism is a morphism for which both functions are bijections.

We shall frequently use the following simple algebra of sets and functions in our constructions below. Given two sets X and Y the cartesian product of X and Y will be denoted $X \times Y$, and the sum (disjoint union) of X and Y will be denoted $X + Y$. Associated with the cartesian product and sum constructions there are a variety of special functions: the projections $pr_X : X \times Y \to X$, $pr_Y : X \times Y \to Y$, the diagonal $\Delta_X : X \to X \times X$, the product twist $X \times Y \to Y \times X$; the injections of the sum $in_X : X \to X + Y$, $in_Y : Y \to X + Y$, the codiagonal $\nabla_X : X + X \to X$, and the sum twist $X + Y \to Y + X$. We denote the composition of function $\varphi : X \to Y$ with $\psi : Y \to Z$ by $\psi \circ \varphi : X \to Z$. Given two functions $\varphi : X \to Y$, $\psi : Z \to W$, there are functions $\varphi + \psi : X + Z \to Y + W$ and $\varphi \times \psi : X \times Z \to Y \times W$. Given two functions $\varphi : X \to Z$, $\psi : Y \to Z$, an abbreviation for the composite function $\nabla_Z \circ (\varphi + \psi) : X + Y \to Z$ is $(\varphi | \psi)$. Given sets X, Y, Z, there is the distributive law bijection $\delta_{X,Y,Z} : X \times Y + X \times Z \to X \times (Y + Z)$. If T is a one point set then for any X, there are isomorphisms $T \times X \cong X \cong X \times T$. Finally there is a unique function from the empty set $\emptyset$ to any other set X, and further $\emptyset \times X \cong \emptyset$. For further details of this algebra see [15].

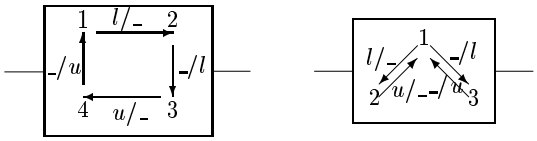
2 Place automata and model checking

The model of concurrent system we use is that of a span of graphs [11]. Composition of systems on interfaces is as considered in [11]. For the purposes of this paper a somewhat simplified version of the model is sufficient, which we describe briefly.

Definition 1. *Consider an alphabet A which contains a special null symbol $\_$. Consider graphs for which at every vertex there is a specified null loop. Then a place automaton is a tuple $\mathcal{X} = (X, m, n, \partial_0, \partial_1)$ where X is such a graph, m, n are natural numbers, and $\partial_0 : arcs(X) \to A^m$, $\partial_1 : arcs(X) \to A^n$, are labellings of the arcs of X by tuples of letters in the alphabet A with the property that null loops are alway labelled by tuples of null symbols.*
The graph X is called the graph of states *(vertices) and* transitions *(arcs) of the automaton. The number m is the number of left interfaces of the system; the number n that of the right interfaces. The alphabet A is the set of possible* transitions on the interfaces *of the system. If α is a transition of X then $\partial_0(\alpha)$ is the m-tuple of transitions on the left interfaces which occurs when the transition α occurs; $\partial_1(\alpha)$ is the corresponding n-tuple of transitions on the right interface. Where there is no confusion we will denote the labelling functions for any system by the same letters ∂_0, ∂_1. We represent a system graphically by a rectangle containing the labelled graph, with m wires on the left and n on the right.*

A behaviour of the place automaton $\mathcal{X}$ is a path in the graph X. A reachable place automaton *is a place automaton with a specified initial state $x_0 \in X$, with the property that each state of X is reachable by a behaviour commencing at x_0. A* deadlock *state of $\mathcal{X}$ is a state out of which there is only one transition, the null transition.*
A morphism φ from automaton $\mathcal{X}$ to automaton $\mathcal{Y}$ (both with the same number of left and right interfaces) we mean a graph morphism which respects the left and right labellings. Morphism φ is path-lifting if given vertex $x \in X$ and transition $\beta : \varphi(x) \to y \in \mathcal{Y}$ there is a path $x \to x_1 \to x_2 \to ... \to x_n$ such that $\varphi(x_n) = y$, all except the last edge in the path are taken by φ to the null loop on $\varphi(x)$, and $\varphi(x_{n-1} \to x_n) = \beta$. For reachable automata we require also that the intial state is preserved by morphisms.

Example 1. The two place automata we need for the example are the philosopher $\mathcal{P}$ and the fork $\mathcal{F}$ described diagrammatically as follows - except that we have *omitted the null-loop* at each vertex:



Philosopher Fork

For each automaton 1 is the initial state. The state 1 of the fork means free, 2 means locked to the left, 3 means locked to the right. The symbols l and u mean lock and unlock respectively. The transition $l/\_ : 1 \to 2$ of the philosopher means lock the left fork, and do nothing on the right.

For the purposes of this section we need only describe two operations on systems (further operations may be found in [11]).

Definition 2. *(Communicating parallel) The communicating parallel composite of systems $\mathcal{X} = (X, m, n, \partial_0, \partial_1)$ and $\mathcal{Y} = (Y, n, p, \partial_0, \partial_1)$ is $\mathcal{X} \cdot \mathcal{Y} = (X \cdot Y, m, p, \partial_0, \partial_1)$ where $X \cdot Y$ has vertices all pairs of vertices (x, y) $(x \in X,\, y \in Y)$; and arcs all pairs (α, β) $(\alpha \in X,\, \beta \in Y)$ such that the right labelling of α equals the left labelling of β. The left and right labellings of (α, β) are the left labelling of α and the right labelling of β respectively. For reachable automata $\mathcal{X}$, $\mathcal{Y}$ we denote by $\mathcal{X} \cdot_R \mathcal{Y}$ the part of $\mathcal{X} \cdot \mathcal{Y}$ reachable from state (x_0, y_0).*

Definition 3. *(Feedback) The feedback of system $\mathcal{X} = (X, m+1, n+1, \partial_0, \partial_1)$ is $\mathsf{Fb}(\mathcal{X}) = (Fb(X), m, n, \partial_0, \partial_1)$ where $Fb(X)$ has vertices all vertices of X and arcs all arcs $\alpha \in X$ such that the $n+1$ st right label equals the $m+1$ st left labelling of α. The left and right labellings of $\alpha \in Fb(X)$ are the first m left labels and the first n right labels of α respectively. For a reachable automaton $\mathcal{X}$ we denote by $\mathsf{Fb}_R(\mathcal{X})$ the part of $\mathsf{Fb}(\mathcal{X})$ reachable from x_0.*

Example 2. The classical dining philosopher problem is $\mathsf{Fb}(\mathcal{P} \cdot \mathcal{F} \cdot \mathcal{P} \cdot \mathcal{F} \cdot ... \cdot \mathcal{P} \cdot \mathcal{F})$.
The reachable part of the dining philosopher problem is $\mathsf{Fb}_R(\mathcal{P} \cdot_R \mathcal{F} \cdot_R \mathcal{P} \cdot_R \mathcal{F} \cdot_R \cdots \cdot_R \mathcal{P} \cdot_R \mathcal{F})$.

Minimization Given a reachable place automaton $\mathcal{X}$ we will now define a construction $min(\mathcal{X})$ which is often much smaller than $\mathcal{X}$ and which has the property that there is a path lifting morphism $\varepsilon_{\mathcal{X}} : \mathcal{X} \to min(\mathcal{X})$. As a simple consequence of the path lifting property is that deadlocks are preserved, so that finding (reachable) deadlocks in $\mathcal{X}$ reduces to finding deadlocks in $min(\mathcal{X})$, then finding the inverse images of deadlocks in $\mathcal{X}$ and checking which of these states are deadlock states of $\mathcal{X}$. Notice that path lifting morphisms φ do not have the property that $\varphi(x)$ is deadlock implies the x is deadlock.

Definition 4. *Given a relation R on the states of $\mathcal{X}$ define a new relation $E(R)$ as follows: $(x, y) \in E(R)$ if (i) for any labelled transition $a/b : x \to x'$ in $\mathcal{X}$ there is a path $y = y_0 \to y_1 \to y_2 \to ... \to y_n$ with the first $n-1$ steps labelled $\_/\_$, with $(x, y_i) \in R$ for $i = 0, 1, .., n-1$, and with $(x', y_n) \in R$,* and *(ii) the symmetric property for y in terms of x. Commence with the total relation T - any x is related to any y. Then $E^k(T)$ $(k = 1, 2, 3, ...)$ eventually stabilizes as an equivalence relation on the states of $\mathcal{X}$ (a notion closely related to the maximal autobisimulation as described in [6]). Define the set of states of $min(\mathcal{X})$ to be the equivalence classes of states of $\mathcal{X}$ under this relation. There is a transition $a/b : [x] \to [y]$ in $min(\mathcal{X})$ if there is such a labelled transition for any representatives of the classes.*

Remark 1. The minimization of any closed system, for example the dining philosophers, has one state and one null loop.

Proposition 1. *The morphism $\varepsilon_{\mathcal{X}} : \mathcal{X} \to min(\mathcal{X})$ is surjective and path lifting, and has the property that given any path lifting morphism $\varphi : \mathcal{X} \to \mathcal{Y}$ there is a unique path lifting morphism $\mu : \mathcal{Y} \to min(\mathcal{X})$ such that $\mu \circ \varphi = \varepsilon_{\mathcal{X}}$.*

The other crucial property of $\varepsilon_{\mathcal{X}} : \mathcal{X} \to min(\mathcal{X})$ is that it has the following compositional properties (which means that it can be calculated compositionally rather than globally):

Proposition 2.

$$\begin{aligned} min(\mathcal{X} \cdot_R \mathcal{Y}) &= min(min(\mathcal{X}) \cdot_R min(\mathcal{Y})), \\ min(\mathsf{Fb}_R(\mathcal{X})) &= min(\mathsf{Fb}_R(min(\mathcal{X})), \\ \varepsilon_{min(\mathcal{X}) \cdot_R min(\mathcal{Y})} \circ (\varepsilon_{\mathcal{X}} \cdot_R \varepsilon_{\mathcal{Y}}) &= \varepsilon_{\mathcal{X} \cdot_R \mathcal{Y}}, \\ \varepsilon_{\mathsf{Fb}_R(min(\mathcal{X})} \circ \mathsf{Fb}_R(\varepsilon_{\mathcal{X}}) &= \varepsilon_{\mathsf{Fb}_R(\mathcal{X})}. \end{aligned}$$

The algorithm based on these ideas is as follows: given a system which is presented as an expression in the operations of communicating parallel and feedback, calculate *min* of the system by successively evaluating an operation, and then calculating *min* of the result. Record the morphisms ε which arise in the calculation. When *min* of the system is obtained, succesively find all the deadlock states, and then inverse images along the calculated ε 's.

The dining philosophers The result of applying this algorithm to the dining philosopher problem is as follows:
$min(\mathcal{P}\cdot_R\mathcal{F})$ is a five vertex graph with 15 arcs. The expression $min(\mathcal{P}\cdot_R\mathcal{F}\cdot_R\mathcal{P}\cdot_R\mathcal{F})$ may then be evaluates as $min(min(\mathcal{P}\cdot_R\mathcal{F})\cdot_R min(\mathcal{P}\cdot_R\mathcal{F}))$. Calculating

$$min(\mathcal{P}\cdot_R\mathcal{F}\cdot_R\mathcal{P}\cdot_R\mathcal{F}\cdot_R\mathcal{P}\cdot_R\mathcal{F})$$

in a similar way we obtain the surprizing result that

$$min(\mathcal{P}\cdot_R\mathcal{F}\cdot_R\mathcal{P}\cdot_R\mathcal{F}\cdot_R\mathcal{P}\cdot_R\mathcal{F}) \cong min(\mathcal{P}\cdot_R\mathcal{F}\cdot_R\mathcal{P}\cdot_R\mathcal{F}).$$

That is, $min((\mathcal{P}\cdot_R\mathcal{F})^n) \cong min((\mathcal{P}\cdot_R\mathcal{F})^2)$ for all $n \geq 2$. This means that the calculation of the minimization of $(\mathcal{P}\cdot_R\mathcal{F})^n$ has linear time complexity. If the extra function is added to the algorithm of checking when $min((\mathcal{P}\cdot_R\mathcal{F})^k) \cong min((\mathcal{P}\cdot_R\mathcal{F})^l)$ for two different stages k, l of the algorithm then the time complexity of calculating $min((\mathcal{P}\cdot_R\mathcal{F})^n)$ becomes constant.

The final stage of calculating *min* of the dining philosophers is to calculate $\mathsf{Fb}_R(min((\mathcal{P}\cdot_R\mathcal{F})^n))$ and then minimize the result, yielding one state and one null edge as expected.

The process of retrieving the deadlocks involves tracing back through the ε's checking considering only deadlock states as you go. The first step of the process involves noting that the only state of the final minimization is a deadlock. However among its images in $\mathsf{Fb}_R(min((\mathcal{P}\cdot_R\mathcal{F})^n))$ only one is a deadlock state. Repeating, the inverse image of the deadlock in $\mathsf{Fb}_R(min((\mathcal{P}\cdot_R\mathcal{F})^{n-1}\cdot_R(\mathcal{P}\cdot_R\mathcal{F})))$ consists of one state which is a deadlock. And so on. The information used in doing this calculation is the expression for the dining philosophers and the minimization functions ε for the open systems consisting of one, two and three philosophers with forks.

3 CP automata and the IWIM model of coordination

Arbab introduced in [1] a model of coordination he called 'idealized workers and idealized managers' (IWIM) on which the language MANIFOLD is based. In this model workers are rather like circuit components which process their input and produce output without knowledge as to which other components they are connected. In normal activity a group of workers is rather like a circuit. The manager of a group of workers on the other hand knows nothing of the detailed activities of its workers but is able to reconfigure them in response to certain events.

In this section we extend the notion of place automata defined above to obtain a new kind of automaton, and operations on these automata, to formalize the IWIM model. The novelty is that the extension permits the description of the manager's activity in changing the configuration of the workers.

The automata we introduce, which we call *case place automata* (CP automata), are in the first place graphs - the vertices represent states and the

arcs transitions. Each CP automata has in addition *interfaces* (as for place automata) which allow parallel communication. The interfaces are distinguished into *left* interfaces (often *input)* and *right* interfaces (often *output*). An interface is represented mathematically by a labelling of the transitions of the automaton in an alphabet. The idea is that when a transition occurs in the automaton the corresponding label occurs on the interface. Having such interfaces allows us to describe communicating parallel composition (*restricted product* denoted $\cdot$, and *free product* denoted $\times$) of automata; in such a composite a state is a pair of states, and a transition is a pair of transitions, one of each automaton, which agree or synchronize on the common interface. An expression of automata using these compositions is a circuit.

The new aspect of CP automata is that they have in addition to interfaces also *conditions*. Very often the state space of an automaton decomposes naturally into a disjoint sum of *cases*. For example, in representing a system consisting of a producer, a buffer and a consumer, the actions of the producer and the consumer are determined by three relevant cases of the buffer *empty*, *full*, and *partly full*. We are concerned here with cases relevant to activating (creating) or disactivating (destroying) automata (including channels between automata), what we call *in-conditions* or *out-conditions* respectively. A condition is a subset of the set of states and it represents states in which the configuration may change in a particular way. However it is *crucial* not to think of conditions just as initial and terminal states. In purely sequential programming it is reasonable to think in this way, but when there are several active processes one of the active processes may die in a particular terminal state while the others are in general activity - that is, the global state of the system in which a change of configuration occurs is a terminal state in only one component. To permit a change of configuration in only one component of a system it is crucial to allow the whole state space among the in- and out-conditions. Formally, we gather the various in-conditions into a single *function* (usually not the inclusion of a subset) which lands in the set of states of the automaton. Similarly the out-conditions consist of a function into the state space of the automaton.

With the structure of in- and out-conditions we define the *restricted sum* (denoted $+$) of CP automata which expresses the deactivation of the first automaton in one of its out-conditions followed be the activation of the second in one of its in-conditions.

As a case study of the use of CP automata, we give a detailed description the distributed sort/numerical optimization of [4], showing that the manager process in these algorithms is the recursive equation

$$\mathcal{S} = \mathcal{A} + \mathcal{D} \cdot (\mathcal{A} \times \mathcal{S}) \cdot \mathcal{M},$$

which when instantiated to yield the sort algorithm, for example, yields

$$Sort = Atomsort + Divert \cdot (Atomsort \times Sort) \cdot Merge.$$

We prove distributive laws between the parallel and restricted sum operations which allow us to solve this recursive equation.

As an example of the use of CP automata in expressing mobility we describe another example, a variant of the Dining Philosopher problem in which diners may move, and which requires two further operations on CP automata *case feedback*, and *place feedback*.

3.1 The IWIM coordination model

Arbab in a series of papers [1], [2], [5], has described a conceptual model of coordination, idealized workers and idealized managers (IWIM), on which his language MANIFOLD is based. The essential features of IWIM are as follows:

- The elements of the model are processes; each process may partake of two roles - a worker role and a manager role. There is only one type of process however, a fact which leads to the possibility of describing hierarchical systems, in which the role of a process may be managerial at one level while that of a worker at the next.
- In the role of worker a process has no knowledge of the processes with which it may be communicating. In this sense it is like a component in an electrical circuit. Its role is simply to process what arrives at its ports and dispatch results to its ports. It can also broadcast events which allow the manager to make decisions about rearrangement of its workers.
  Its communications are anonymous, a feature that characterizes exogenous models.
- In the role of a manager a process has no knowledge of the precise activities of its workers. It knows simply their configuration and certain events, as a result of which it is able to decide on a change of topology of the workers.
- The manager may be useful for a variety of different jobs, in which workers of one type are substituted by workers of another type. This leads to reusability of management structures.

3.2 CP automata

The algebra we use to formalize the IWIM model has as elements certain automata, which we call *case-place automata* (or CP automata).

Definition 5. *A* CP automaton $\mathcal{G}$ *consists of a graph* $\mathbf{G}$, *four sets* X, Y, A, B *and four functions*

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

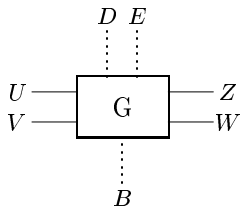
A behaviour *of* $\mathcal{G}$ *is a path in the graph* $\mathbf{G}$.

The graph $\mathbf{G}$ is called the *centre* of the automaton. We remark that ∂_0, ∂_1 may be thought of as *labellings* of the arcs of $\mathbf{G}$ in the alphabets X, Y, respectively. These labellings will be used in the restricted product of two CP automata, the operation which expresses communicating parallel processes. Alternatively,

one may think of the vertices and arcs of $\mathbf{G}$ as the *states* and *transitions* of the system, whereas the elements of X, Y are the transitions of the interfaces. We call X the *left interface* of the automaton, and Y the *right interface* - automata communicate through these interfaces. Often the interface sets will be products of sets; for example the left interface of $\mathbf{G}$ may be $X = U \times V$, and the right interface may be $Y = Z \times W$, and we then speak of U and V as the left interfaces, and Z and W as the right interfaces. If we ignore the functions γ_0, γ_1 a CP automaton is a (particular type of) *span of graphs* as defined in [11], where the reader may find more details and examples.

The set A represents a *condition* on the states in which the automaton may come into existence, and the set B a condition in which it may cease to exist. We call A the *in-condition* of the automaton, and B the *out-condition*. The functions γ_0, γ_1 of a CP automaton will be used in the restricted sum of CP automata - an operation which expresses change of configuration of processes. The meaning of the in- and out-conditions will become clearer in the section on the restricted sum of automata, and in the example of the distributed sort. Often the condition sets will be sums of sets; for example the in-condition may be $A = D + E$, and we then speak of D and E as in-conditions.

There is a useful graphical representation of CP automata (ignoring the conditions this is described in [11]). For example, we will represent a CP automaton with left interface $U \times V$, right interface $Z \times W$, in-condition $D + E$, and out-condition B, by a diagram of the following sort:



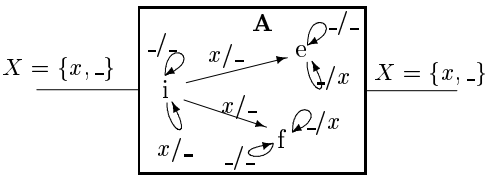
For simplicity we use the same names ∂_0, ∂_1, γ_0, γ_1 for the four functions of any CP automaton where there is no risk of confusion, introducing further suffixes when clarification is needed. We use symbols $X, Y, Z, U, V, W...$ for the (left and right) interfaces, and symbols $A, B, C, D, E, F, I, ...$ for the (in- and out-) conditions.

Example 3. An example of a CP automaton is provided by a Mealy automaton with input set X, output set Y, internal states G_0, initial states A and final states B. The arrows of the graph $\mathbf{G}$ are provided by the transitions of the automaton. The functions γ_0, γ_1 are inclusions. The reader is warned that this example gives a false impression of the strength of the model we are describing. It is essential, for example, in expressing the changing geometry of a system that the functions γ_0, γ_1 not be restricted solely to inclusions. The sets A and B are not to be thought of as initial and final states, but rather as conditions under which a change of geometry might occur. Another difference with Mealy

automata is that the sets X and Y need not be input and output but rather interfaces on which synchronization occurs with connected components.

Remark 2. We want to describe a program for a manager that will co-ordinate a family of workers in order to produce a distributed bucket sort algorithm. There are three types of workers: an atomic sort, a divert and a merge. In terms of the model presented here, the manager will be a recursive equation built out of the operations of CP automata. There will be three variables in the expression, one variable $\mathcal{A}$ is to play the role of the atomic sort and will require the following structure: it has a single input line and a single output line; the in-condition has the form $A_0 + I + F$, where I is a one element set that represents the initial state of the atomic sort, F represents states in which the atomic sort is full, and A_0 denotes the set of all states of the atomic sort; the out-condition of $\mathcal{A}$ is $A_0 + F$. The second variable $\mathcal{D}$ is to play the role of the divert (input to second output line) which has the following structure: it has one input line and two output line; the in-condition is $I + D_0$, and out-condition D_0, where I is a one element set that represents the initial state and D_0 denotes the set of all states of the divert. The third variable $\mathcal{M}$ is to play the role of the merge (of two sorted lists) which has the following structure: it has two input lines and one output line; the in-condition is $I + M_0$, and out-condition M_0, where I is a one element set that represents the initial state and M_0 denotes the set of all states of the merge. In order to obtain from the manager program an actual sort it is necessary to instantiate the variables as CP automata with appropriate functionality. In this paper we will not describe such CP automata, but rather give some illustrative finite state abstractions. The same manager program can be used with different instantiation of the variables to produce other algorithms. This fact is in keeping with the spirit of the IWIM model; in particular, the separation of coordination from computation. The manager is not concerned with inner workings of its workers; the manager's only concern is the coordination of its workers. The manager's decision to create or destroy its workers, or to reconfigure its network of workers, is based on events it receives from the workers. In our framework, these events are represented by conditions.

Example 4. Consider the set $X = \{x, \_\}$ (the underscore symbol is intended to represent the null action). The following diagram specifies the centre **A** of a CP automaton $\mathcal{A}$, and its left and right interfaces, both being the set X. Note that we have indicated the functions ∂_0, ∂_1 by labelling the arcs of the graph.



To complete the definition of $\mathcal{A}$ we must also describe the in-conditions and out-conditions. The vertex set A_0 of the centre **A** of $\mathcal{A}$ will be one of the in- and

also one of the out-conditions. Consider two further sets $I = \{i\}$ and $F = \{f\}$ (subsets of A_0); denote the inclusion of I in A_0 by inc_I, and the inclusion of F in A_0 by inc_F. Then the in-condition set of $\mathcal{A}$ is $A_0 + I + F$ and $\gamma_0 = (1_{A_0} \mid inc_I \mid inc_F)$. The out-condition set of $\mathcal{A}$ is $A_0 + F$ and $\gamma_1 = (1_{A_0} \mid inc_F)$.

Example 5. A behaviour of $\mathcal{A}$ (a path in the centre of $\mathcal{A}$) commencing in state i consists in repeatedly accepting symbols x in state i until perhaps a change either to state e or to state f when it can output symbols x. In any state it can idle by accepting and outputting null symbols. This CP automaton is quite abstract and has many possible meanings. One particular concrete realization is a bounded sorting program - what we will call an atomic sort. Symbol x is an abstraction of the symbols to be sorted. State i is an abstraction of those states in which the sorter is receiving symbols. State e is the abstraction of those states in which the sorter has received an end of list symbol. State f is an abstraction of those states in which the sorter has reached its capacity (full states). In either e or f the sorter may output symbols in sorted order.

Two more examples useful for the analysis of the distributed sort algorithm:

Example 6. Let $X = \{x, \_\}$. Consider the graph $\mathbf{D}$ with vertex set $D_0 = \{0\}$ and three labelled edges $x/\_,\_ : 0 \to 0$, $\_/\_,\_ : 0 \to 0$, $\_/\_,x : 0 \to 0$. This forms the centre of a CP automaton $\mathcal{D}$ with left interface X and right interface $X \times X$. Take the in-condition to be $I + D_0$, where I is a one element set, and out-condition D_0. Since the CP automaton has only one state the functions γ_0, γ_1 are uniquely defined.

Again $\mathcal{D}$ is an abstract automaton. One of its concrete realizations is a program which diverts input to the second line of its output only.

Example 7. Let $X = \{x, \_\}$. Consider the graph $\mathbf{M}$ with vertex set $M_0 = \{0\}$ and four labelled edges $x, \_/\_ : 0 \to 0$, $\_,\_/\_ : 0 \to 0$, $\_,x/\_ : 0 \to 0$, $\_,\_/x$. This forms the centre of a CP automaton $\mathcal{M}$ with left interface $X \times X$ and right interface X. Take the in-condition to be $I + M_0$, where I is a one element set, and out-condition M_0. Since the CP automaton has only one state the functions γ_0, γ_1 are uniquely defined.

One of the concrete realizations of $\mathcal{M}$ is the merge of two queues, which produces a sorted output if the two lists are sorted.

We will need in section 4.1 the notion of *isomorphism* of CP automata.

Definition 6. *Given two CP automata* $\mathcal{G} = (\mathbf{G}, X, Y, A, B, \partial_{0,\mathcal{G}}, \partial_{1,\mathcal{G}}, \gamma_{0,\mathcal{G}}, \gamma_{1,\mathcal{G}})$ *and* $\mathcal{H} = (\mathbf{H}, X, Y, A, B, \partial_{0,\mathcal{H}}, \partial_{1,\mathcal{H}}, \gamma_{0,\mathcal{H}}, \gamma_{1,\mathcal{H}})$ *an isomorphism from* $\mathcal{G}$ *to* $\mathcal{H}$ *is a graph isomorphism* $\varphi : \mathbf{G} \to \mathbf{H}$ *such that* $\partial_{0,\mathcal{H}} \circ \varphi = \partial_{0,\mathcal{G}}$, $\partial_{1,\mathcal{H}} \circ \varphi = \partial_{1,\mathcal{G}}$, $\varphi \circ \gamma_{0,\mathcal{G}} = \gamma_{0,\mathcal{H}}$, $\varphi \circ \gamma_{1,\mathcal{G}} = \gamma_{1,\mathcal{H}}$.

3.3 Operations

Parallel composition

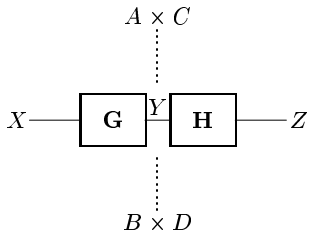
Definition 7. *Given two CP automata*

$$\mathcal{G} = (\mathbf{G}, X, Y, A, B, \partial_0, \partial_1, \gamma_0, \gamma_1),$$
$$\mathcal{H} = (\mathbf{H}, Y, Z, C, D, \partial_0, \partial_1, \gamma_0, \gamma_1)$$

the restricted product *(communicating parallel composition) of $\mathcal{G}$ and $\mathcal{H}$, denoted $\mathcal{G} \cdot \mathcal{H}$ is the CP automaton whose set of vertices is $G_0 \times H_0$ and whose set of arcs is that subset of $G_1 \times H_1$ consisting of pairs of arcs (g, h) such that $\partial_1(g) = \partial_0(h)$. The interfaces and conditions of $\mathcal{G} \cdot \mathcal{H}$ are $X, Z, A \times C, B \times D$, and the four functions are*

$$\partial_{0,\mathcal{G}\cdot\mathcal{H}}(g,h) = \partial_{0,\mathcal{G}}(g),\ \partial_{1,\mathcal{G}\cdot\mathcal{H}}(g,h) = \partial_{1,\mathcal{H}}(h),$$
$$\gamma_{0,\mathcal{G}\cdot\mathcal{H}} = \gamma_{0,\mathcal{G}} \times \gamma_{0,\mathcal{H}}\ ,\ \gamma_{1,\mathcal{G}\cdot\mathcal{H}} = \gamma_{1,\mathcal{G}} \times \gamma_{1,\mathcal{H}}\ .$$

Diagrammatically we represent the restricted product as follows:



Closely related to the restricted product is the free product of CP automata.

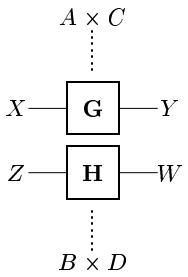
Definition 8. *Given two CP automata*

$$\mathcal{G} = (\mathbf{G}, X, Y, A, B, \partial_0, \partial_1, \gamma_0, \gamma_1),$$
$$\mathcal{H} = (\mathbf{H}, Z, W, C, D, \partial_0, \partial_1, \gamma_0, \gamma_1)$$

the free product *(parallel composition with no communication) of $\mathcal{G}$ and $\mathcal{H}$, denoted $\mathcal{G} \times \mathcal{H}$ is the CP automaton whose set of vertices is $G_0 \times H_0$ and whose set of arcs is $G_1 \times H_1$. The interfaces and conditions of $\mathcal{G} \times \mathcal{H}$ are $X \times Z, Y \times W, A \times C, B \times D$, and the four functions are*

$$\partial_{0,\mathcal{G}\times\mathcal{H}} = \partial_{0,\mathcal{G}} \times \partial_{0,\mathcal{H}},\ \partial_{1,\mathcal{G}\times\mathcal{H}} = \partial_{1,\mathcal{G}} \times \partial_{1,\mathcal{H}},$$
$$\gamma_{0,\mathcal{G}\times\mathcal{H}} = \gamma_{0,\mathcal{G}} \times \gamma_{0,\mathcal{H}}\ ,\ \gamma_{1,\mathcal{G}\times\mathcal{H}} = \gamma_{1,\mathcal{G}} \times \gamma_{1,\mathcal{H}}\ .$$

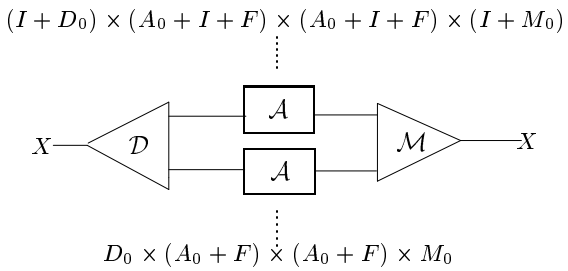
Diagrammatically we represent the free product as follows:



Example 8. Consider the following expression in three types of components described in Remark 1:

$$\mathcal{D} \cdot (\mathcal{A} \times \mathcal{A}) \cdot \mathcal{M}.$$

The diagrammatic representation of this is



When the variables are instantiated with the particular automata described in examples 2, 3, and 4, a behaviour of the resulting CP automaton is (an abstracted version of) the following: the symbols received by $\mathcal{D}$ are passed to the second sorter $\mathcal{A}$, while both the first and second sorters continue to work in parallel eventually outputting their results to the merge $\mathcal{M}$. If the second sorter becomes full it can accept no further symbols.

If we ignore the functions γ_0 , γ_1 of the CP automata the restricted product of CP automata is the *span composition* of [11] and the free product is the *tensor product* of the corresponding spans of graphs. For some examples of how these operations may be used to model concurrent systems see that paper. We will see further examples in the section on the distributed sort below. From a circuit theory point of view these operations correspond, respectively, to the series and parallel operations of circuit components. In the context of this paper the two operations correspond to the *communicating parallel composition of workers in their normal activity.* The communication of the parts of a product is anonymous - each automaton has a precisely defined interface, and has no knowledge of the automata with which it is communicating.

Restricted sum

Definition 9. *Given two CP automata*

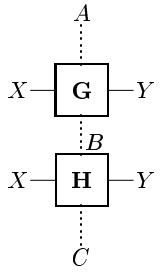
$$\mathcal{G} = (\mathbf{G}, X, Y, A, B, \partial_0, \partial_1, \gamma_0, \gamma_1),$$
$$\mathcal{H} = (\mathbf{H}, X, Y, B, C, \partial_0, \partial_1, \gamma_0, \gamma_1)$$

the restricted sum *(change of configuration) of $\mathcal{G}$ and $\mathcal{H}$, denoted $\mathcal{G} + \mathcal{H}$ is the CP automaton whose set of arcs is $G_1 + H_1$ and whose set of vertices is $(G_0 + H_0)/\sim$; that is $(G_0 + H_0)$ quotiented by the relation $\gamma_{1,\mathcal{G}}(b) \sim \gamma_{0,\mathcal{H}}(b)$ (for all $b \in B$). The interfaces and conditions of $\mathcal{G} + \mathcal{H}$ are X, Y, A and C, and the four functions are*

$$\partial_{0,\mathcal{G}+\mathcal{H}} = (\partial_{0,\mathcal{G}} \,|\, \partial_{0,\mathcal{H}}),\ \partial_{1,\mathcal{G}+\mathcal{H}} = (\partial_{1,\mathcal{G}} \,|\, \partial_{1,\mathcal{H}}),$$
$$\gamma_{0,\mathcal{G}+\mathcal{H}} = in_{G_0} \circ \gamma_{0,\mathcal{G}},\ \ \gamma_{1,\mathcal{G}+\mathcal{H}} = in_{H_0} \circ \gamma_{1,\mathcal{H}}\,.$$

A behaviour of $\mathcal{G} + \mathcal{H}$ is initially a behaviour of $\mathcal{G}$, and then, if a state in the image of B is reached, the behaviour may change to a behaviour of $\mathcal{H}$. The intended interpretation is that initially only the process $\mathcal{G}$ exists; when a state in B is reached the process $\mathcal{G}$ may die and the process $\mathcal{H}$ be created. We will see in the next section that this is the correct interpretation in a dynamic algorithm like the distributed sort.

The diagrammatic representation of the restricted sum is as follows:



Adjusting the conditions Given two CP automata we wish to compose, it may happen that the conditions are not appropriate to allow the composition, but that a modification is necessary. To this end we allow adjustment of conditions by composition with any of the special functions described in section 1.2. For example, the atomic sort $\mathcal{A}$ is defined with in-condition $A_0 + I + F$ and out condition $A_0 + F$. Composing $\gamma_0 : A_0 + I + F \to A_0$ with $in_I : I \to A_0 + I + F$, and composing $\gamma_1 : A_0 + F \to A_0$ with $in_F : F \to A_0 + F$ we obtain a new CP automaton with the same centre and interfaces but new in-condition I and new out-condition F. This new automaton we denote $\mathcal{A}^I_F$. In general an adjustment of in-condition we denote by adding a superscript, and adjusting an out-condition by adding a subscript. This notation is very efficient, though there is clearly in some cases the possibility of ambiguity.

Remark 3. Notice that the restricted sum seems very close to sequential composition. However this is deceptive; consider the parallel composite $\mathcal{G} \times \mathcal{H}$ of two processes, where $\mathcal{G}$ has out-condition T (a one element terminal state), and $\mathcal{H}$ has in- and out-condition H_0. Notice that $\mathcal{G} \times \mathcal{H}$ has out-condition $T \times H_0$ which may be adjusted by composing with the isomorphism $T \times H_0 \cong H_0$. The interpretation of the restricted sum $(\mathcal{G} \times \mathcal{H})_{H_0} + \mathcal{H}$ is that the process $\mathcal{G}$ dies upon reaching its terminal state leaving the process $\mathcal{H}$ still running.

3.4 Distributed sort

We will now describe the distributed sort algorithm using the components and operations described above. The idea is that the sort $\mathcal{S}$ begins with an atomic sort $\mathcal{A}$ which completes its activity *if* it receives an end of list before becoming full. If however the atomic sort becomes full a diversion $\mathcal{D}$, a new atomic sort $\mathcal{A}$, and a merge $\mathcal{M}$ are created and configured as $\mathcal{D} \cdot (\mathcal{A} \times \mathcal{A}) \cdot \mathcal{M}$ (see the diagram in section 3.1). The sort process is recursive; that is, if the second sort becomes full new diversions, atomic sorts, and merges are created, and so on.

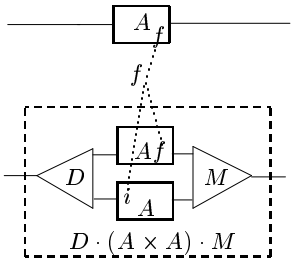
First we describe how the transition from the atomic sort $\mathcal{A}$ to two parallel sorters $\mathcal{D} \cdot (\mathcal{A} \times \mathcal{A}) \cdot \mathcal{M}$ can be described using the restricted sum. Recall that $\mathcal{D} \cdot (\mathcal{A} \times \mathcal{A}) \cdot \mathcal{M}$ has in-condition

$$(I + D_0) \times (A_0 + I + F) \times (A_0 + I + F) \times (I + M_0).$$

We compose the in-condition function of $\mathcal{D} \cdot (\mathcal{A} \times \mathcal{A}) \cdot \mathcal{M}$ with the composite of $F \cong I \times F \times I \times I$ and $in_I \times in_F \times in_I \times in_I$ to obtain a new in-condition for $\mathcal{D} \cdot (\mathcal{A} \times \mathcal{A}) \cdot \mathcal{M}$, namely F. Similarly we can adjust the out-condition of $\mathcal{D} \cdot (\mathcal{A} \times \mathcal{A}) \cdot \mathcal{M}$ to be $D_0 \times A_0 \times F \times M_0$. We can obviously adjust the out-condition of $\mathcal{A}$ to be F, so now the restricted sum

$$\mathcal{A}^I_F + (\mathcal{D} \cdot (\mathcal{A} \times \mathcal{A}) \cdot \mathcal{M})^F_{D_0 \times A_0 \times F \times M_0}$$

may be formed. In the specific instantiation we have previously considered, the effect is to equate the full state f of single atomic sorter $\mathcal{A}$ to the quadruple of states, the initial state of $\mathcal{D}$, the state f of the first atomic sorter, the state i of the second atomic sorter of $\mathcal{D} \cdot (\mathcal{A} \times \mathcal{A}) \cdot \mathcal{M}$, and the initial state of the $\mathcal{M}$. That is, the first atomic sorter is just the continuation of the single sorter, whereas the second atomic sorter is newly created in its initial state. A diagram of the result is:



In a similar way the expression $\mathcal{A} + \mathcal{D} \cdot (\mathcal{A} \times \mathcal{S}) \cdot \mathcal{M}$ may be formed for any process $\mathcal{S}$ with appropriate in-condition of an initial state. The manager process of the recursive sort is then the *unevaluated equation*

$$\mathcal{S} = \mathcal{A} + \mathcal{D} \cdot (\mathcal{A} \times \mathcal{S}) \cdot \mathcal{M}.$$

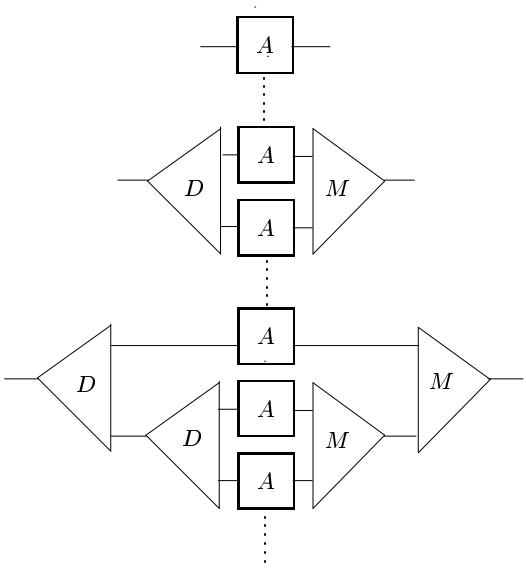
For $\mathcal{A}, \mathcal{D}, \mathcal{M}$ as given above the solution of the evaluated expression is a distributed sort, but in [4] the authors show how the same protocol may be used to perform parallel/distributed numerical optimization of a complex function by supplying a different $\mathcal{A}$, an evaluator, each instance of which takes in an input unit describing a subdomain of a function, and produces its best estimate of the optimum value of the function in that subdomain; and a different $\mathcal{M}$, a selector, each instance of which produces as its output the best optimum value it receives as its input.

We will now do some *rough calculations* with this equation ignoring for the moment the important aspect of in- and out-conditions. The calculation will be made more precise and justified in the next section.

The equation $\mathcal{S} = \mathcal{A} + \mathcal{D} \cdot (\mathcal{A} \times \mathcal{S}) \cdot \mathcal{M}$ may be formally expanded using the distributive laws (section 4.1) which exist between the products and the restricted sum.

$$\begin{aligned}
\mathcal{S} &= \mathcal{A} + \mathcal{D} \cdot (\mathcal{A} \times \mathcal{S}) \cdot \mathcal{M} \\
&= \mathcal{A} + \mathcal{D} \cdot (\mathcal{A} \times (\mathcal{A} + \mathcal{D} \cdot (\mathcal{A} \times \mathcal{S}) \cdot \mathcal{M})) \cdot \mathcal{M} \\
&= \mathcal{A} + \mathcal{D} \cdot (\mathcal{A} \times \mathcal{A}) \cdot \mathcal{M} + \mathcal{D} \cdot (\mathcal{A} \times (\mathcal{D} \cdot (\mathcal{A} \times \mathcal{S}) \cdot \mathcal{M})) \cdot \mathcal{M} \\
&= \ldots \\
&= \mathcal{A} + \mathcal{D} \cdot (\mathcal{A} \times \mathcal{A}) \cdot \mathcal{M} + \mathcal{D} \cdot (\mathcal{A} \times (\mathcal{D} \cdot (\mathcal{A} \times \mathcal{A}) \cdot \mathcal{M})) \cdot \mathcal{M} + \ldots.
\end{aligned}$$

The terms in the final (infinite) expansion correspond precisely to the sequence of geometries possible for the distributed sort:



Distributive laws We would now like to make more precise the distributive laws we have used in expanding the expression for the distributed sort. Consider CP automata $\mathcal{G}$, $\mathcal{H}$, $\mathcal{K}$.

Proposition 3. *The following distributive laws hold provided the in- and out-conditions indicated are among those of the associated CP automaton, and the interfaces of the automata are appropriate for the operations:*

$$\mathcal{G}_B^A \cdot (\mathcal{H}_D^C + \mathcal{K}_E^D) \cong \mathcal{G}_{G_0}^A \cdot \mathcal{H}_D^C + \mathcal{G}_B^{G_0} \cdot \mathcal{K}_E^D, \tag{1}$$

$$(\mathcal{H}_D^C + \mathcal{K}_E^D) \cdot \mathcal{G}_B^A \cong \mathcal{H}_D^C \cdot \mathcal{G}_{G_0}^A + \mathcal{K}_E^D \cdot \mathcal{G}_B^{G_0}, \tag{2}$$

$$\mathcal{G}_B^A \times (\mathcal{H}_D^C + \mathcal{K}_E^D) \cong \mathcal{G}_{G_0}^A \times \mathcal{H}_D^C + \mathcal{G}_B^{G_0} \times \mathcal{K}_E^D, \tag{3}$$

$$(\mathcal{H}_D^C + \mathcal{K}_E^D) \times \mathcal{G}_B^A \cong \mathcal{H}_D^C \times \mathcal{G}_{G_0}^A + \mathcal{K}_E^D \times \mathcal{G}_B^{G_0}. \tag{4}$$

Proof. We give the proof of the first only. The arcs in the left-hand side are either pairs (g,h) $(g \in G_1, h \in H_1)$ such that $\partial_1(g) = \partial_0(h)$ or pairs (g,h) $(g \in G_1, h \in K_1)$ such that $\partial_1(g) = \partial_0(k)$. But this is precisely what arcs in the right-hand side are. The vertices on the left-hand side are pairs of the form $(g,[l])$ $(g \in G_0, l \in H_0 + K_0)$ where $[l]$ denotes the equivalence class of l with respect to the equivalence generated by $\gamma_1(d) \sim \gamma_0(d)$ $(d \in D)$. The vertices on the right-hand side are equivalence classes of pairs $[(g,l)]$ $(l \in H_0 + K_0)$ with respect to the equivalence relation generated by $\gamma_1(g,d) \sim \gamma_0(g,d)$, that is, $(g,\gamma_1(d)) \sim (g,\gamma_0(d))$ $(g \in G_0, d \in D)$. There is a clear bijection between the vertices of the left hand and right hand sides. This defines an isomorphism of graphs which clearly respects the interfaces and conditions.

In making a precise expansion of the equation

$$\mathcal{S} = \mathcal{A} + \mathcal{D} \cdot (\mathcal{A} \times \mathcal{S}) \cdot \mathcal{M}$$

we will see that there are some distinctions that are worth making which we have neglected until now. First the equation is not really an equation but an isomorphism. Second we must specify in- and out conditions for S - we take the in-condition to be a one point set I (the initial state) and the out-condition to be $\emptyset$ (the empty set). The equation now takes the precise form

$$\mathcal{S}^I_\emptyset \cong \mathcal{A}^I_F + \mathcal{D}^I_{D_0} \cdot (\mathcal{A}^F_{A_0} \times \mathcal{S}^I_\emptyset) \cdot \mathcal{M}^I_{M_0}.$$

Let us now expand this isomorphism using the distributive laws. Substituting for $\mathcal{S}^I_\emptyset$ in the right hand side we get

$$\begin{aligned}
\mathcal{S}^I_\emptyset &\cong \mathcal{A}^I_F + \mathcal{D}^I_{D_0} \cdot (\mathcal{A}^F_{A_0} \times (\mathcal{A}^I_F + \mathcal{D}^I_{D_0} \cdot (\mathcal{A}^F_{A_0} \times \mathcal{S}^I_\emptyset) \cdot \mathcal{M}^I_{M_0})) \cdot \mathcal{M}^I_{M_0} \\
&\cong \mathcal{A}^I_F + \mathcal{D}^I_{D_0} \cdot (\mathcal{A}^F_{A_0} \times \mathcal{A}^I_F + \mathcal{A}^{A_0}_{A_0} \times (\mathcal{D}^I_{D_0} \cdot (\mathcal{A}^F_{A_0} \times \mathcal{S}^I_\emptyset) \cdot \mathcal{M}^I_{M_0})) \cdot \mathcal{M}^I_{M_0} \\
&\cong \mathcal{A}^I_F + \mathcal{D}^I_{D_0} \cdot (\mathcal{A}^F_{A_0} \times \mathcal{A}^I_F) \cdot \mathcal{M}^I_{M_0} + \\
&\quad \mathcal{D}^{D_0}_{D_0} \cdot (\mathcal{A}^{A_0}_{A_0} \times (\mathcal{D}^I_{D_0} \cdot (\mathcal{A}^F_{A_0} \times \mathcal{S}^I_\emptyset) \cdot \mathcal{M}^I_{M_0})) \cdot \mathcal{M}^{M_0}_{M_0}.
\end{aligned}$$

Careful examination of this expansion reveals that the in- and out-conditions of the various parts are exactly the appropriate ones.

4 CP automata and mobility

We illustrate the use of CP automata in describing mobility of processes, introducing also two further operations, *case* and *place feedback*.

Definition 10. *Given a CP automaton*

$$\mathcal{G} = (\mathbf{G}, X \times Y, Z \times Y, A, B, \partial_0, \partial_1, \gamma_0, \gamma_1),$$

the place feedback of $\mathcal{G}$ *with respect to* Y*, denoted* $\mathsf{Pfb}_Y(\mathcal{G})$ *is the CP automaton whose set of vertices is* G_0 *and whose set of arcs is that subset of* G_1 *consisting of arcs* g *such that* $(pr_Y \circ \partial_1)(g) = (pr_Y \circ \partial_0)(g)$*. The interfaces and conditions of* $\mathsf{Pfb}_Y(\mathcal{G})$ *are* X, Z, A, B*,with the four functions defined as follows:*

$$\begin{aligned}
\partial_{0,\mathsf{Pfb}_Y(\mathcal{G})} &= pr_X \circ \partial_{0,\mathcal{G}},\ \partial_{1,\mathsf{Pfb}_Y(\mathcal{G})} = pr_Z \circ \partial_{1,\mathcal{G}}, \\
\gamma_{0,\mathsf{Pfb}_Y(\mathcal{G})} &= \gamma_{0,\mathcal{G}},\ \ \gamma_{1,\mathsf{Pfb}_Y(\mathcal{G})} = \gamma_{1,\mathcal{G}}.
\end{aligned}$$

The diagrammatic representation of $\mathsf{Pfb}_Y(\mathcal{G})$ involves joining the right interface Y to the left interface Y.

Definition 11. *Given a CP automaton*

$$\mathcal{G} = (\mathbf{G}, X, Y, A + B, C + B, \partial_0, \partial_1, \gamma_0, \gamma_1),$$

the case feedback *of $\mathcal{G}$ with respect to $\mathcal{B}$, denoted $\mathsf{Cfb}_B(\mathcal{G})$ is the CP automaton whose set of arcs is G_1 and whose set of vertices is $G_0/\sim$; that is G_0 quotiented by the relation $(\gamma_1 \circ in_B)(b) \sim (\gamma_0 \circ in_B)(b)$ (for all $b \in B$). The interfaces and conditions of $\mathsf{Cfb}_B(\mathcal{G})$ are X, Y, A and C, and the four functions are defined as follows:*

$$\begin{aligned} \partial_{0,\mathsf{Cfb}_B(\mathcal{G})} &= \partial_{0,\mathcal{G}},\ \partial_{1,\mathsf{Cfb}_B(\mathcal{G})} = \partial_{1,\mathcal{G}}, \\ \gamma_{0,\mathsf{Cfb}_B(\mathcal{G})} &= \gamma_{0,\mathcal{G}} \circ in_A,\ \ \gamma_{1,\mathsf{Cfb}_B(\mathcal{G})} = \gamma_{1,\mathcal{G}} \circ in_C. \end{aligned}$$

The diagrammatic representation of $\mathsf{Cfb}_B(\mathcal{G})$ involves joining the out-condition B to the in-condition B.

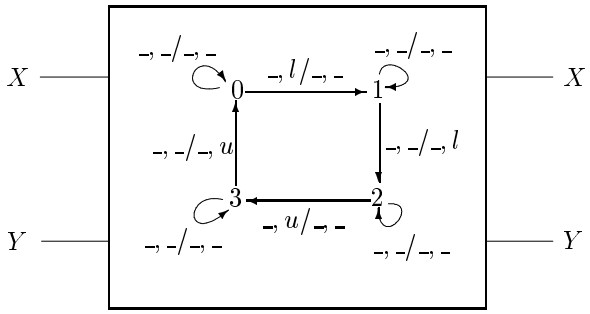
The example we would like to describe is a variant of the Dining Philosopher Problem which we call Sofia's Birthday Party. Instead of a circle of philosophers around a table with as many forks, we consider a circle of seats around a table separated by forks on the table. Then there are a number of children (not greater than the number of seats). The protocol of each child is the same as that of a philosopher. However in addition, if a child is not holding a fork, and the seat to the right is empty, the child may change seats - the food may be better there. To describe this we need six CP automata – a child $\mathcal{C}$, an empty seat $\mathcal{E}$, a fork $\mathcal{F}$, two transition elements $\mathcal{L}$ and $\mathcal{R}$, and an identity 1_X of X (a wire). The interface sets involved are $X = \{x, \_\}$ and $Y = \{\_, u, l\}$.

The transition elements have left and right interfaces $X \times Y$. The graph $\mathbf{L}$ of the transition element $\mathcal{L}$ has two vertices p and q and three labelled edges $\_, \_/\_, \_ : q \to q$; $x, \_/\_, \_ : q \to p$; $\_, \_/\_, \_ : p \to p$. Its in-condition is $Q = \{q\}$, and its out-condition is $P = \{p\}$. The graph $\mathbf{R}$ of the of the transition element $\mathcal{R}$ also has two vertices p and q, and has three labelled edges $\_, \_/\_, \_ : q \to q$; $\_, \_/x, \_ : q \to p$; $\_, \_/\_, \_ : p \to p$. Its in-condition is also $Q = \{q\}$, and its out-condition is $P = \{p\}$.

The empty seat $\mathcal{E}$ has left and right interfaces $X \times Y$. The graph $\mathbf{E}$ of the empty seat has one vertex e and one labelled edge $\_, \_/\_, \_ : e \to e$. Its in-condition is P and its out-condition is Q. The functions γ_0, γ_1 are uniquely defined.

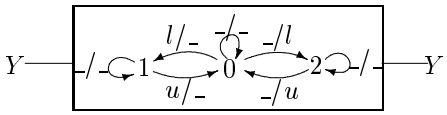
The identity 1_X has left and right interface X. Its graph has one vertex $*$ and two labelled edges $\_/\_ : * \to *$ and $x/x : * \to *$. Its in- and out-conditions are the empty set $\emptyset$.

The child $\mathcal{C}$ has labelled graph as follows:



The states have the following interpretation: in state 0 the child has no forks; in state 1 it has a fork in its left hand; in state 2 it has both forks; in state 3 it has returned it left fork. The action l is to be interpreted as *locking a fork*, and the action u as *unlocking a fork*. The child's in-condition is P and its out-condition is Q. The function γ_0 takes p to 0; the function γ_1 takes q to 0.

The fork $\mathcal{F}$ has labelled graph as follows:



The fork's in-condition and out-condition are $\emptyset$. Its states are to be interpreted as follows: in state 0 the fork is free; in state 1 it is locked on the left; in state 2 it is locked on the right.

Let $\mathcal{S} = \mathsf{Cfb}_P(\mathcal{C} + \mathcal{R} + \mathcal{E} + \mathcal{L})$. This CP automaton has the following interpretation – it can either be a child (on a seat) or an empty seat. The transition elements $\mathcal{R}$ and $\mathcal{L}$ allow the seat to become occupied or vacated. Then Sofia's Birthday Party is given by the expression

$$\mathsf{Pfb}_{X\times Y}(\mathcal{S}\cdot(1_X\times\mathcal{F})\cdot\mathcal{S}\cdot(1_X\times\mathcal{F})\cdot\ldots\cdot\mathcal{S}\cdot(1_X\times\mathcal{F})).$$

This CP automaton has the behaviour as informally described above.

We can reobtain the classical Dining Philosopher problem as follows: modify the interfaces of $\mathcal{C}$ and $\mathcal{F}$ by composing with the projection $X\times Y\to Y$ to obtain new automata $\mathcal{C}'$ and $\mathcal{F}'$. Take all in- and out-conditions to be the empty set $\emptyset$. Then the Dining Philosopher (as formulated in [12]) is given by the expression

$$\mathsf{Pfb}_Y(\mathcal{C}'\cdot\mathcal{F}'\cdot\mathcal{C}'\cdot\mathcal{F}'\cdot\ldots\cdot\mathcal{C}'\cdot\mathcal{F}').$$

5 Acknowledgments

The authors gratefully acknowledge stimulating conversations with Fahrad Arbab. In fact, the research of section 3 of this paper was carried out in response to a

remark of Arbab in [5], where he says: "An alternative approach to a mathematical foundation for the semantics of MANIFOLD can be sought in other category–theoretical models. The essence of the semantics of a coordinator process in MANIFOLD can be described as transitions between states, each of which defines a different topology of information–carrying streams among various sets of processes. What is defined in each such state is reminiscent of an (asynchronous) electronic circuit. Category theoretical models have been used to describe simple circuit diagrams [10]. Extensions of such models to account for the dynamic topological reconfiguration of MANIFOLD is a non–trivial challenge which, nevertheless, points to an interesting model of computation". We thank also Adriano Scutellà for interesting conversations. The research has been financially supported by the Dipartimento di Scienze Chimiche, Fisiche e Matematiche of the University of Insubria, Como, Italy, and by the Italian Progetto Cofinanziato MURST *Tipi di Ordine Superiore e Concorrenza* (TOSCA).

References

[1] F. Arbab, Coordination of massively concurrent activities, Technical Report CS–R9565, Centrum voor Wiskunde en Informatica, Amsterdam, The Netherlands, 1995, Available on-line at http://www.cwi.nl/ftp/CWIreports/IS/CS-R9565.ps.Z.

[2] F. Arbab, The IWIM model for coordination of concurrent activities. In Paolo Ciancarini and Chris Hankin, editors, Coordination Languages and Models, volume 1061 of Lecture Notes in Computer Science, 34–56, Springer–Verlag, 1996.

[3] F. Arbab, Manifold version 2: Language reference manual, Technical report, Centrum voor Wiskunde en Informatica, Amsterdam, The Netherlands, pages : 1–162, 1996, Available on-line at http://www.cwi.nl/ftp/manifold/refman.ps.Z .

[4] F. Arbab, C. L. Blom, F. J. Burger, and C. T. H. Everaars, Reusable Coordination Modules for Massively Concurrent Applications, Software: Practice and Experience, vol. 28, No. 7: 703–735, 1998.

[5] F. Arbab, What Do You Mean, Coordination?, March'98 Issue of the Bulletin of the Dutch Association for Theoretical Computer Science (NVTI)

[6] A. Arnold, Finite transition systems, Prentice Hall, 1994.

[7] M.M. Bonsangue, F. Arbab, J.W. de Bakker, J.J.M.M. Rutten, A. Scutellà, G. Zavattaro, A transition system semantics for the control-driven coordination language MANIFOLD, Technical Report SEN-R9829, Centrum voor Wiskunde en Informatica, Amsterdam, The Netherlands, 1998.

[8] J-C. Fernandez, An implementation of an efficient algorithm for bisimulation equivalence, 1989.

[9] S. Graf, B. Steffen, G. Lüttgen, Compositional minimization of finite state systems using interface specifications, Formal aspects of computing 1996.

[10] P. Katis, N. Sabadini, R.F.C. Walters, Bicategories of processes, Journal of Pure and Applied Algebra 115: 141–178, 1997.

[11] P. Katis, N. Sabadini, R.F.C. Walters, Span(Graph): A categorical algebra of transition systems, Proceedings Algebraic Methodology and Software Technology, volume 1349 of Lecture Notes in Computer Science, 307–321, Springer Verlag, 1997.

[12] P. Katis, N. Sabadini, R.F.C. Walters, On the algebra of systems with feedback and boundary, Rendiconti del Circolo Matematico di Palermo Serie II, Suppl. 63: 123-156, 2000.

[13] P. Katis, N. Sabadini, R.F.C. Walters, A formalization of the IWIM model, Coordination Languages and Models, volume 1906 Lecture Notes in Computer Science, 267–283, Springer Verlag, 2000.
[14] A. Valmari, State of the art report: Stubborn sets, Petri nets newsletter, 46:6–14, April 1994.
[15] R.F.C. Walters, Categories and Computer Science, Cambridge University Press, 1992.
[16] W. Zielonka, Note on asynchronous automata, RAIRO, 1987.