Working Paper Series
ISSN 1170-487X

UNIFYING STATE AND PROCESS DETERMINISM

Steve Reeves and David Streader

Working Paper: 02/2004
February 2004

Department of Computer Science
The University of Waikato
Private Bag 3105
Hamilton, New Zealand

Unifying state and process determinism

Steve Reeves and David Streader
Department of Computer Science
University of Waikato
Hamilton, New Zealand
{dstr,stever}@cs.waikato.ac.nz

Abstract

If a coin is given to a deterministic robot that interacts with a deterministic vending machine then is the drink that the robot is delivered determined? Using process definitions of determinism from CSP, CCS or ACP the answer is "no", whereas state-based definitions of determinism can reasonably be construed as giving the answer "yes".

In order to unify what we see as discrepancies in state- and action-based notions of determinism we will consider process algebras over two sets of actions: the **active** *or* **causal** *actions of the robot and the* **passive** *or* **reactive** *actions of the vending machine. In addition we will add priority to the actions and when two τ actions are possible then the τ action with the highest priority will be executed.*

Keywords: process algebra, determinism, abstraction, hiding state

1 Introduction

Although it would be nice to believe that the formal definitions of real-world concepts are a complete and faithful rendering of our intuitions, it is likely that they are a compromise between these intuitions and the restrictions of the formalism within which they are set. When, further, we seek to mix two formalisms to synthesise something new, as is increasingly being done with state-based and action-based formalisms, there is a danger that the different compromises result in a single idea having two formal definitions with distinct properties. We believe such compromises have happened with determinism when modelled with state-based and action-based formalisms, so care is required.

Since we are interested in synthesising new formalisms from existing ones in the state-plus-action field, we want to consider ways of unifying the notion of determinism as understood from the state world and from the process world while avoiding any possible pitfalls that might arise from this synthesis.

As a running example we will consider vending machines and robots. By a robot we mean, as usual, a machine that has no external agent controlling it, but which is able to control a vending machine. Before we discuss any formalism we consider the world we want to model, so we pose a simple question about a real deterministic vending machine and a real deterministic robot. *If a coin is given to the robot and it interacts with the vending machine then is the drink that the robot is delivered determined?* It is our opinion that the answer has to be "yes, the drink is determined". Further, if we have an operation representing how to build an implementation by the composition of component implementations then we expect it to build a deterministic implementation.

Hoare's description of non-determinism of selection [12, p101] is "There is nothing mysterious about this kind of non-determinism: it arises from a deliberate decision to ignore the factors that influence the selection". As real, physical things (like implementations) are unable to "ignore the factors that influence" them, we regard them (and so we regard implementations) as deterministic; specifications are not physical things, so only they (not implementations) might possibly be non-deterministic.

We will illustrate a subtle difference between non-determinism defined, as we see it, in state-based languages like Z and non-determinism as defined in process algebras like CSP [12, 18], Algebraic Theory of Processes [11] , CCS [15], ACP [1], CFFD [?] and NDFD [13]. (We will use the terms *p-non-determinism* and *p-determinism* for the process-based notions in the sequel when we want to carefully distinguish the two views and where the context does not make this distinction clear.) Because process algebras do not formally distinguish implementation from specification, the fact that, for them, parallel composition does not preserve

p-determinism cannot be seen as a fault or shortcoming. Nonetheless, as we take the process algebra notion of parallel composition as accurately reflecting how to build real processes[1] from components, we believe the fact that it fails to preserve p-determinism may seem counter-intuitive and surprising to people more familiar with formalisms other than process algebras, and in particular to people from the state-based world.

2 Modelling Determinism

From the dictionary [7] we find:

Definition 1 Deterministic *Having the property that everything that happens is fixed by a necessary chain of causation*

which we might, in our context, paraphrase as "having the property that every response is fixed by a cause".

By common acceptance:

Real World Assumption 1 *The world is deterministic.*

By "world" here we can restrict ourselves to meaning the world of implementations of "computational systems" and the systems they control, especially if it is necessary to defuse arguments about reality in general.

We are interested in using refinement to formalise the steps from specification to implementation and so to know when to stop applying refinement we need to distinguish implementations from specifications. We assume our formalism can capture sufficient detail of processes so that non-determinism will be restricted to abstract specifications or descriptions and will not appear in any definition of an implementation.

Without going into details we assume that in our formalism all deterministic specifications are implementable. We could do this by restricting the "functional" component of a transition but in this paper we restrict ourselves to finite state processes.

Assumption 1 *Any deterministic process can be implemented.*

From Real World Assumption 1 and Assumption 1 we can conclude that processes are implementable iff they are deterministic (remember: the world includes implementations).

[1] We are not trying to fix the level of abstraction at which processes are represented. We are suggesting that at any level of abstraction we have a notion of implementability that is preserved by our operators. Elsewhere we formalise how to change levels of abstraction.

We assume that formalisms possess operators Op that model the composition of real (implemented) processes. That is to say if A and B represent implementations then, where $op \in Op$, so too does AopB. As an immediate consequence of this, Real World Assumption 1 and Assumption 1 we have:

Property 1 *If X (ADT) and context $[\_]_P$ (program) are deterministic and $[X]_P$ models their actual interaction then it must be deterministic.*

In a world of vending machines and robots this property would lead us to expect that: *if a coin is given to a deterministic robot that interacts with a deterministic vending machine then the drink that the robot is delivered is determined.*

2.1 State-based determinism

The standard definition of determinism we take as:

Definition 2 *An operation is deterministic when its response is determined by its cause. For "stateful" operations interpret cause as both initial state and any value input and interpret response as both final state and any value output.*

An ADT X is deterministic iff it is composed from a set of deterministic operations.

This definition of ADT determinism satisfies Property 1. Consequently it is our interpretation of state-based systems that the composition of two deterministic components must itself be deterministic. Hence *the drink that the robot is delivered is determined.*

2.2 Action-based determinism

Processes generalise ADTs in that:

1. processes can interact with contexts that are not sequences of operations but have a branching structure, *e.g.* if a stack was a process it would be able to interact with a "program" that offered it the ability to choose from a set of next actions;
2. processes may have "agency" or a thread of control independent of their context. Whereas ADTs are totally passive, only programs have a thread of control. Hence implicitly in a world of ADTs and programs we have a distinction between passive actions (those of an ADT) and active actions (those of a program).

We could treat active actions in the same way as passive actions and *simply lift* the definition of determinism from ADTs to processes, *i.e. deterministic if*

composed of a set of deterministic operations. But such a definition would not satisfy Property 1 and consequently we believe this not to give a satisfactory model of determinism. By a simple modification we have a definition that does satisfy Property 1.

Definition 3 *A process X is deterministic if both:*

1. *it is composed from elements of* **Act**, *a set of deterministic operations;*
2. *no more than one active action can ever be enabled.*

The second clause in this definition is always satisfied by ADTs as they have no active actions and by programs because they only ever have one action enabled.

2.2.1 Time and determinism

Real World Assumption 1 implies that all real machines are deterministic, even concurrent ones. Although this may be reasonable for timed models, when time is abstracted away concurrency can introduce non-determinism (and this is what we would expect from Hoare's statement quoted in section 1). In our work, although we abstract away (global) time, we have (as we shall see) a notion of global priority that can be used to resolve any non-determinism in place of time. An interesting alternative, which we do not explore here, is to restrict non-determinism to being between actions from different parallel processes.

2.2.2 Process algebras do not distinguish causal and response actions

Process algebras abstract away from notions of cause and response in as much as they do not treat the causal action *I push button one* differently to the response of a passive process, such as a vending machine, *button one is pushed.*

This cause/response distinction we believe to be natural in the state-based world in as much as a program can *call* an action of a data type whereas the data type normally cannot *cause* the program to call it. This difference is formalised in [6] where the actions of the programs must be sequential whereas the actions of the ADT can be branching.

The cause/response distinction can also be found in the world of abstract state machines: see the "pushing and pulling of information" [3, section 5] and the definition of "active agents" in [10, section 6.3.1].

In the world of broadcast communication a causal action is an output action to which a listener can respond by hearing or taking as input the message broadcast. Clearly listening cannot *cause* something to be broadcast, and this is formalised in IOA [14] where output actions (not input actions) are "enabled".

Not distinguishing causal and response actions makes VM and Rob essentially identical process algebra expressions in Figure 1.

VM is easy to understand as a machine that accepts a coin and then reacts to either button one (b1) or button two (b2) being pushed and subsequently dispensing drink d1 or d2. It is hopefully easy to see, and we assume anyone would agree, that VM can be implemented and is deterministic. The robot Rob first inserts a coin

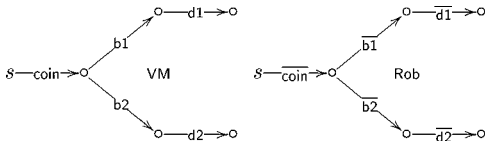

Figure 1. Vending machine and Robot.

in the vending machine then is prepared to push either button one or button two and subsequently take a drink.

Informally speaking all the actions of the robot cause the corresponding actions of the vending to occur, but $\overline{\text{coin}}$ is an output action and the $\overline{\text{di}}$ are input actions. So we can see that for processes the cause/response does not coincide with the input/output distinction. This is to be expected given the state-based notion of programs calling operations of a data type and the abstract state machine notion of "pushing and pulling of information".

2.2.3 Process algebras do not distinguish observing and choosing

The process algebra world is based on a set of actions that can both be *observed* and *chosen*/refused and a τ action that can neither be observed nor chosen. Separating observable from choosable is normal in the literature on the control of discrete event systems: see for example the events in [16] that are both "controllable" and "unobservable".

If the actions of a robot (recall by robot we mean a process, uncontrolled by any external agent, that controls the vending machine) cause the actions of a vending machine to occur, then simply observing the interaction will not affect the outcome of that interaction. For example: if a deterministic robot goes to a deterministic vending machine and fetches a drink then whether you "observe" the button the robot pushes, or not, you cannot influence the drink that is returned. In the usual process algebra formalisms it is not possible

to model being able to "observe" which button is being pushed without also being able to "choose" which button is being pushed.

2.2.4 Process algebras' interpretation of Rob

From the process algebraic point of view Rob in Fig 1 requires another process to choose the button it pushes on VM. If a process can perform one of two actions, there is no process algebra term that can say which of the two it will choose in a context that allows both to be performed.

We will write VM $\|_S$ Rob for parallel composition where actions in S must synchronise (in ACP terminology VM $\|_S$ Rob is (VM || Rob)$\delta_{S\cup\overline{S}}$ where $\gamma(\mathsf{a},\overline{\mathsf{a}}) = \widehat{\mathsf{a}}$ is assumed) and for the hiding of actions we use ACPs $\_\tau_S$ (which is $\_\backslash S$ in CSP).

A consequence of the identification of observing and choosing is that abstraction $\_\tau_{\{\mathsf{b1},\mathsf{b2}\}}$ introduces p-non-determinism because it prevents any other process from choosing between the abstracted actions b1 and b2 (again recall the quote from Hoare in Section 1).

2.2.5 Our interpretation of Rob

It is hard to see how we could interpret Rob as an implementation. What does it mean for a robot, that is not to be controlled by another process, to be prepared to push either button one or button two? We see no way to implement Rob so that VM $\|_{\{\mathsf{b1},\mathsf{b2}\}}$ Rob is non-deterministic [2].

We interpret Rob as satisfying the partial specification:

VM1 || Rob to return d1	VM1 $\stackrel{\text{def}}{=}$ coin; b1; d1,
VM2 || Rob to return d2	VM2 $\stackrel{\text{def}}{=}$ coin; b2; d2,

VM || Rob to be specified,

We want to be able to refine Rob by specifying what drink is to be returned from VM. Undoubtedly this *could* be done within process algebra, but only by redefining *both* the vending machine and the robot at a more detailed level of abstraction. However, VM is (as agreed) a perfectly good implementation of a vending machine, so we wish to keep the definition of the vending machine as it is.

2.2.6 Process algebra determinism

A CSP process is defined to be deterministic where it is always true that the next action to be executed *can be determined* by a choice made in its context. Recall that we call this **p-determinism** to distinguish it from the notion of determinism in Definition 3 and more formally, later, in Definition 5. Both VM and Rob are p-deterministic; so too is CSP's VM || Rob.

Because we wish the robot to be able to push a button without the help or hindrance of any outside agent (context) the actions b1 and b2 must be hidden in CSP as in (VM || Rob)$\backslash\{\mathsf{b1},\mathsf{b2}\}$. It is because $\_\backslash\{\mathsf{b1},\mathsf{b2}\}$ prevents the ability to **choose** a that p-non-determinism is introduced, not because $\_\backslash\{\mathsf{b1},\mathsf{b2}\}$ prevents the ability to **see** a.

2.2.7 Our determinism

We write $\widehat{\mathsf{a}}$ for the composition of an active action $\overline{\mathsf{a}}$ and a passive action a (in ACP terminology $\gamma(\mathsf{a},\overline{\mathsf{a}}) = \widehat{\mathsf{a}}$ but in CSP all three actions would be a) (see Fig. 2).

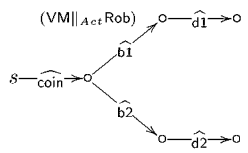

Figure 2. VM **with** Rob

Our passive actions need to be controlled and our active actions should control them: neither can be executed without the other. Hiding passive or active actions changes this interpretation of actions so that they may be executed independently of any other action. Synchronised actions represent the joint execution of a pair of actions and do not need the cooperation of any other action in order to be performed. Hence hiding a synchronised action only changes its ability to be observed.

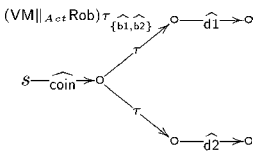

Figure 3. Hidden buttons

We are not the first to see the special relationship between hiding and synchronised pairs of actions; in CCS's definition of parallel composition [15][p.46] synchronised pairs of actions are modelled by τ actions. For us both Rob and VM $\|_{Act}$ Rob are non-deterministic. The hiding of actions cannot introduce non-determinism as for us hiding limits what you can see, not what you can choose. Only the action with the highest priority, from a set of actions, can be chosen or executed. Thus our model reflects the fact that simply seeing what button a real robot pushes cannot affect whether or not the choice was deterministic.

Because we wish the robot to be able to push a button without the help or hindrance of any outside agent (context) we do not allow $\hat{\mathsf{x}}$ to synchronise with

[2] Occam can be thought of as an implementation of a part of CSP and using Occam Rob can easily be coded and hence implemented. But, Occam's implementation of || has a hidden **arbiter** that decides what button is pushed in Rob || VM. This arbiter is not "truly non-deterministic" and as such Occam's || is not CSP's ||.

any other action. To model the actions $\widehat{\mathsf{b1}}$ and $\widehat{\mathsf{b2}}$ as unobservable we hide them, for which we write $(\mathsf{VM} \parallel_{Act} \mathsf{Rob})\tau_{\{\widehat{\mathsf{b1}},\widehat{\mathsf{b2}}\}}$ (see Fig 3).

Here $(\_ \parallel_S \_)\tau_{\widehat{R}}$ is being used to model the interaction between implementations, *i.e.* to build larger implementations from smaller implementations. Consequently it preserves determinism of finite state processes.

3 Causal Process algebra

Because we believe it natural to view Rob as nondeterministic but VM as deterministic we need to distinguish these, and similar, processes. This we do by separating actions into two classes: **active** actions represented by an over-lined name and **passive** actions with no over-line. In order that we can refine Rob into a deterministic process we introduce priority between actions.

We will define process operations that, when applied to deterministic processes, will always result in deterministic processes

3.1 Cause and effect actions

Most real world *events* can be described as the synchronisation of a **passive** action and an **active** action, *e.g.* the event "button 1 on a vending machine is pushed" is the synchronisation of the active action "I push button 1" and the passive action of the vending machine "button 1 is pushed".

We contend that it is easy to implement VM because it offers the choice between passive actions. More problematic is the choice between the "active" actions of the robot Rob.

The choice between b1 and b2 is to be made by the context in which VM finds itself. But the actions $\overline{\mathsf{b1}}$ and $\overline{\mathsf{b2}}$ are quite different as their attempted execution, like the methods called by a program, are under local control. Hence Rob should not be seen as an implementation. Note the essentially sequential nature of causal actions of programs has been used in [5] where singleton failure semantics was introduced. What we are doing that is different is that we do not split processes into ADTs with only passive actions and programs/contexts with only active actions. We will illustrate this in Section 3.3.

3.2 Priority

We are going to extend our model of actions to include priority. We then define a process as deterministic if it is both p-deterministic and all its actions must be passive or have a unique priority. Our operational semantics is such that out of two τ actions the one with the highest priority will always be taken.

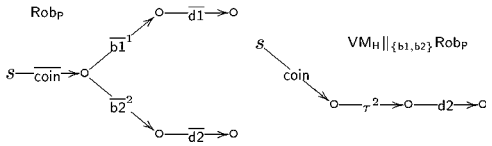

Figure 4. (*cf.* **Fig 1**) $Rob \sqsubseteq Rob_P$

In any implementation of our robot it must first put a coin in the vending machine then, after finding out what buttons are on offer, it must choose either button one or button two. Our allocation of priority to the actions "push button one" and "push button two" could be used as an abstract representation of what button it tries to push first.

Adding priority (which we do via superscripting) to the active actions of our robot (see Fig 4) results in a simpler observational semantics, and abstraction, the removal of τ actions, does not introduce nondeterminism.

What we are doing with priority is different to what is done in the process literature where active and passive actions are not distinguished. Normally priority is used to model the idea of treating one set of actions (such as interrupts) as having priority over another set of actions. For us it makes sense for refinement to change the priority of an action, whereas it does not when using priority to model high priority interrupts and low priority actions.

3.3 Choice between active and passive

If we add to our vending machine the action $\overline{\mathsf{push}}$, the ability to push the robot, and add to the robot the action push, a tilt detection, we get Fig 5 (where u and v are arbitrary priority levels).

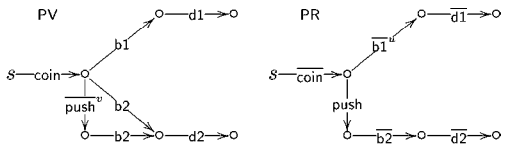

Figure 5. PV and PR

Our specifications PV and PR can be seen to be quite abstract by considering what must happen after the robot puts a coin in the vending machine. At this point both processes are going to attempt to perform

an active action but must jointly agree which action is actually going to be performed. How such a joint decision is reached is completely hidden.

We do not wish to formalise a particular mechanism for joint decision making; all we wish to do is record the decision.

Again both PV and PR are p-deterministic but $(\mathsf{PV} \parallel_{Act} \mathsf{PR})\tau_{\{b1,b2,push\}}$ is p-non-deterministic. The problem again stems from composing processes that do not decide which of two actions are to be performed, each preventing the other process from making the choice. The choice cannot be made by either process on its own, so the choice is in some sense an **inter-process** phenomenon. ('Time' is another example of such an inter-process phenomenon.)

Here we want PV and PR to be non-deterministic and we want to be able to refine them to deterministic processes so that the composition of deterministic processes is deterministic.

To do this where a decision must be made jointly we need to make refinement of a process sensitive to the other processes with which it can interact. Taking account of priorities, $(\mathsf{PV} \parallel_{Act} \mathsf{PR})\tau_{\{b1,b2,push\}}$ is deterministic only if $v \neq u$, but this cannot be known by considering one of PV or PR on its own.

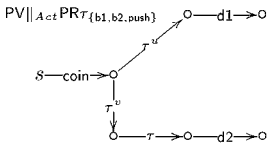

Figure 6. PV **with** PR

Because action synchronisation is an inter-process phenomenon it is usual [1] to define a global synchronisation function γ. Here, to model **inter-process choice**, we will define a global priority relation $Act_{Pri} \subseteq \ddot{Act} \times Pri$. Our definition of determinism of P will consider the priorities of actions that are either in P or can synchronise with actions of P.

3.4 Causal Process Algebra

We assume a universe containing: **passive** actions Act, **active** actions $\overline{Act} \stackrel{\text{def}}{=} \{\bar{a} \mid a \in Act\}$ and **synchronised** actions $\widehat{Act} \stackrel{\text{def}}{=} \{\hat{a} \mid a \in Act\}$, on which we define $Obs \stackrel{\text{def}}{=} (Act \cup \overline{Act} \cup \widehat{Act})$ and $\ddot{Act} \stackrel{\text{def}}{=} Obs \cup \tau$.

In addition our universe contains: a set of priorities Pri with an irreflexive priority relation $\angle \subseteq Pri \times Pri$ and a static priority function $Act_{Pri} : Obs \to Pri$ such that $Act_{Pri}(a) = Act_{Pri}(\bar{a}) = Act_{Pri}(\hat{a})$.

It would be more usual to define the priority relation $\angle \subseteq Pri \times Pri$ as a total order or a preorder, but we will need more flexibility when defining our weak semantics.

We use Act_{Pri} to model the inter process decision making discussed in Section 3.3 and use it to define both determinism and refinement.

Let $\ddot{a}, \ddot{b}, \ldots \in \ddot{Act}, \alpha, \beta, \ldots \in \ddot{Act} \times Pri$ and $\overline{Act}_\tau \stackrel{\text{def}}{=} \overline{Act} \cup \tau$

Definition 4 *Priority labelled transition systems (PLTS). Given a finite set of nodes* N_A *and* $s_\mathsf{A} \in N_\mathsf{A}, e_\mathsf{A} \subset N_\mathsf{A}$, $T_\mathsf{A} \subseteq \{(n,(x,p),m) \mid n,m \in N_\mathsf{A} \wedge (x,p) \in Act_{Pri} \wedge x \in Obs\} \cup \{(n,(\tau,p),m) \mid p \in Pri\}$

then $\mathsf{A} \stackrel{\text{def}}{=} (N_\mathsf{A}, T_\mathsf{A}, s_\mathsf{A}, e_\mathsf{A})$ *is a PLTS* •

We write $n \xrightarrow{\ddot{a}^p} m$ iff $(n,(\ddot{a},p),m) \in T_\mathsf{A}$ and $n \xrightarrow{\alpha}$ iff $\exists\, m.n \xrightarrow{\alpha} m$. We then define the ready set of node n as $\pi(n) \stackrel{\text{def}}{=} \{x \mid n \xrightarrow{(x,p)}\}$ and $\pi(\mathsf{A}) \stackrel{\text{def}}{=} \pi(s_\mathsf{A})$, $\pi_p(n) \stackrel{\text{def}}{=} \{p \mid n \xrightarrow{(x,p)}\}$ and $\pi_p(\mathsf{A}) \stackrel{\text{def}}{=} \pi_p(s_\mathsf{A})$.
We define the **alphabet** of process A as $\mathsf{A}_\alpha = \{x \mid (\_,(x,\_),\_) \in T_\mathsf{A}\}$ and the priorities in A as $Pri(\mathsf{A}) = \{p \mid (\_,(\_,p),\_) \in T_\mathsf{A}\}$.
We lift $\angle$: $n \xrightarrow{(\_,p)} \_ \angle n \xrightarrow{(\_,p')} \_ \Leftrightarrow p \angle p'$. •

A is p-deterministic if there is only one transition leaving any given node with a given name, whereas here A is deterministic if all actions leaving any given node are passive or have a unique priority

Definition 5 A *is deterministic iff* $\forall_{x,p,\ddot{a} \in \overline{Act} \cup \widehat{Act} \cup \{\tau\}}$

$$x \xrightarrow{(\ddot{a},p)} z \wedge x \xrightarrow{(\ddot{b},p)} y \Rightarrow z = y \wedge \ddot{a} = \ddot{b}$$ •

Definition 6 *We define* $\preceq$: $Act_{Pri} \times Act_{Pri}$ *by* $(x,p) \preceq (y,p') \stackrel{\text{def}}{=} p \angle p'$.
Define refinement of the priority relation $\sqsubseteq_p \subseteq Act_{Pri} \times Act_{Pri}$ *by* $Act_{Pri1} \sqsubseteq_p Act_{Pri2} \Leftrightarrow \preceq_1 \subseteq \preceq_2$.

Clearly $\sqsubseteq_p$ *will induce a relation between processes, which we call* $\sqsubseteq_p$ *too.*

Refinement $\sqsubseteq_{tp}$ *is the transitive closure of* $\sqsubseteq_p \cup \sqsubseteq_t$ *where* $\sqsubseteq_t$ *is taken from one of the many definition of refinement on labelled transition systems (see [17] for examples).* •

Note from definition of *PLTS* that changing Act_{Pri} induces a change of the priority on a *PLTS*. This change must be applied globally to all processes. Priority refinement decreases the number of actions with the same priority.

Definition 7 $+, ;$ τ_S δ_S *and* $\|_S$ *on PLTS* A *and* B

$N_{\mathsf{A}\|_S\mathsf{B}} = N_\mathsf{A} \times N_\mathsf{B}$,
$e_{\mathsf{A}\|_S\mathsf{B}} = e_\mathsf{A} \times e_\mathsf{B}$,
$s_{\mathsf{A}\|_S\mathsf{B}} = \langle s_\mathsf{A}, s_\mathsf{B} \rangle$ *and*
$T_{\mathsf{A}\|_S\mathsf{B}}$ *defined by:*

$$\frac{n \xrightarrow{(\ddot{a},p)}_\mathsf{A} l, \ddot{a} \notin S}{\forall_{x \in N_\mathsf{B}}\ (n,x) \xrightarrow{(\ddot{a},p)}_{\mathsf{A}\|_S\mathsf{B}} (l,x)}$$

$$\frac{m \xrightarrow{(\ddot{a},p)}_\mathsf{B} k, \ddot{a} \notin S}{\forall_{x \in N_\mathsf{A}}\ (x,m) \xrightarrow{(\ddot{a},p)}_{\mathsf{A}\|_S\mathsf{B}} (x,k)}$$

$$\frac{n \xrightarrow{(a,p)}_{\mathsf{A}} l,\ m \xrightarrow{(\bar{a},p)}_{\mathsf{B}} k,\ a \in S}{(n,m) \xrightarrow{(\hat{a},p)}_{\mathsf{A}\|_S\mathsf{B}} (l,k)} \qquad \frac{n \xrightarrow{(a,p)}_{\mathsf{B}} l,\ m \xrightarrow{(\bar{a},p)}_{\mathsf{A}} k,\ a \in S}{(n,m) \xrightarrow{(\hat{a},p)}_{\mathsf{A}\|_S\mathsf{B}} (l,k)}$$

Further we have:

$N_{\mathsf{A}\tau_S} = N_{\mathsf{A}}$, $s_{\mathsf{A}\tau_S} = s_{\mathsf{A}}$, $e_{\mathsf{A}\tau_S} = e_{\mathsf{A}}$, $T_{\mathsf{A}\tau_S}$ *defined by:*

$$\frac{n \xrightarrow{(a,p)}_{\mathsf{A}} l,\ a \notin S}{n \xrightarrow{(a,p)}_{\mathsf{A}\tau_S} l} \qquad \frac{n \xrightarrow{(a,p)}_{\mathsf{A}} l,\ a \in S}{n \xrightarrow{(\tau,p)}_{\mathsf{A}\tau_S} l}$$

$N_{\mathsf{A}\delta_S} = N_{\mathsf{A}}$, $s_{\mathsf{A}\delta_S} = s_{\mathsf{A}}$, $e_{\mathsf{A}\delta_S} = e_{\mathsf{A}}$, $T_{\mathsf{A}\delta_S}$ *defined by:*

$$\frac{n \xrightarrow{(a,p)}_{\mathsf{A}} l,\ a \notin S}{n \xrightarrow{(a,p)}_{\mathsf{A}\delta_S} l}$$

$\mathsf{A}+\mathsf{B} \stackrel{\text{def}}{=} (N_{\mathsf{A}} \cup N_{\mathsf{B}} - \{s_{\mathsf{B}}\}, s_{\mathsf{A}}, e_{\mathsf{A}} \cup e_{\mathsf{B}}, T_{\mathsf{A}} \cup T_{\mathsf{B}}\{s_{\mathsf{B}}/s_{\mathsf{A}}\})$

$\mathsf{A};\mathsf{B} \stackrel{\text{def}}{=} (N_{\mathsf{A}} \cup N_{\mathsf{B}}, s_{\mathsf{A}}, e_{\mathsf{B}}, T_{\mathsf{A}} \cup T_{\mathsf{B}} \cup \{e \xrightarrow{x} n \mid e \in e_{\mathsf{A}} \wedge s_{\mathsf{B}} \xrightarrow{x} n\})$ •

Definition 8 *If* $(x \xrightarrow{a^p} \vee\ x \xrightarrow{\tau^p})\ \wedge\ q \angle p$ *then* $x \xrightarrow{a^q} y$ *is* **priority unreachable** *(*i.e. *it can never be executed) written* $Un(x \xrightarrow{a^q} y)$.

Prune$(\mathsf{A}) \stackrel{\text{def}}{=} (N_{\mathsf{A}}, \{x \xrightarrow{a^q} y \mid x \xrightarrow{a^q} y \in T_{\mathsf{A}} \wedge \neg\ Un(x \xrightarrow{a^q} y)\}, s_{\mathsf{A}}, e_{\mathsf{A}})$

Lemma 1 *If* $\sqsubseteq_t$ *is monotonic then* $\sqsubseteq_{tp}$ *is monotonic.*

Proof Monotonicity of $\sqsubseteq_{tp}$ follows from assumption and monotonicity of $\sqsubseteq_p$. Monotonicity of $\sqsubseteq_p$ is a direct consequence of definition. •

4 Implementations

The operators Σ from Section 3.4 can be used to build, potentially non-deterministic, specifications. We define a set of process operators $\widehat{\Sigma}$, that are restrictions of Σ, that can be use to build process implementations $T_{\widehat{\Sigma}}$ (*i.e.* terms over $\widehat{\Sigma}$). All such implementations will be deterministic and all contexts built from $\widehat{\Sigma}$ will preserve determinism, *i.e.* P and $\widehat{[\_]}_X$ deterministic implies $\widehat{[\mathsf{P}]}_X$ deterministic.

We can view $\widehat{\Sigma}$ as defining an abstract programming language or as operators that describe how real processes might be combined.

Definition 9 $\widehat{\Sigma} \stackrel{\text{def}}{=} \{\widehat{\oplus}, \widehat{;}, \widehat{\tau}_A, \delta_A, \widehat{\|_S}, Act \cup \overline{Act}\}$. *Implementations are* $T_{\widehat{\Sigma}}$ *and contexts* $\widehat{[\_]}_X$ *are implementations with a slot for a process.*

$\mathsf{A}\widehat{\|_S}\mathsf{B} \stackrel{\text{def}}{=} \mathsf{A} \|_S \mathsf{B}$ *defined only if* $Pri(\mathsf{A}) \cap Pri(\mathsf{B}) = \varnothing$.

$\mathsf{A}\widehat{\oplus}\mathsf{B} \stackrel{\text{def}}{=} \mathsf{A} + \mathsf{B}$ *defined only if* $\pi_p(\mathsf{A}) \cap \pi_p(\mathsf{B}) = \varnothing$.

$\mathsf{A}\widehat{;}\mathsf{B} \stackrel{\text{def}}{=} \mathsf{A};\mathsf{B}$ *defined only if* $\bigcup_{e \in e_{\mathsf{A}}} \pi_p(e) \cap \pi_p(\mathsf{B}) = \varnothing$.

$\mathsf{P}\widehat{\tau}_A \stackrel{\text{def}}{=} \mathsf{P}\tau_A$ *defined only if* $A \subseteq \widehat{Act}$ •

In ACP [1] δ_S may introduce unreachable actions and τ_S may introduce p-non-determinism, whereas here both δ_S and $\widehat{\tau}_S$ may introduce priority unreachable actions and neither may introduce non-determinism.

Lemma 2 *Let* $\widehat{op} \in \{\widehat{\oplus}, \widehat{;}, \widehat{\tau}_S, \delta_S, \widehat{\|_S}\}$.

1. *Implementations* $T_{\widehat{\Sigma}}$ *are deterministic*
2. *If* P *is deterministic then* $\widehat{[\mathsf{P}]}_X$ *is deterministic*

Proof 1. By structural induction on terms.

$\mathsf{A}\widehat{\oplus}\mathsf{B}$: The only node that changes in the construction of $\mathsf{A}\widehat{\oplus}\mathsf{B}$ is the root node. From $\pi_p(\mathsf{A}) \cap \pi_p(\mathsf{B}) = \varnothing$ and the determinism of A and B we can see that $\forall x, p.x \xrightarrow{(a,p)} z \wedge x \xrightarrow{(b,p)} y \Rightarrow z = y \wedge b = a$.

$\mathsf{A}\widehat{\|_S}\mathsf{B}$: From $Pri(\mathsf{A}) \cap Pri(\mathsf{B}) = \varnothing$ and the determinism of A and B we can see that $\mathsf{A}\widehat{\|_S}\mathsf{B}$ must be deterministic.

$\mathsf{A}\widehat{\tau}_S$ and $\mathsf{A}\delta_S$: A is deterministic if from each node there is only one transition leaving it with any given priority. Renaming one of the transitions does not change this.

2. From 1. •

4.0.1 Pragmatics

Our language for building implementations is defined by restricting the operators of Section 3.4, but these restrictions are based on priority. In most circumstances the following simplification may suffice: build the function between actions and priority $Act_{Pri} : \overline{Act} \to Pri$ that is an injection. Subsequently processes are deterministic if and only if they are p-deterministic. The restrictions on implementations are more familiar:

$\mathsf{A}\widehat{\|_S}\mathsf{B} \stackrel{\text{def}}{=} \mathsf{A} \|_S \mathsf{B}$ defined only if $\alpha(\mathsf{A}) \cap \alpha(\mathsf{B}) = \varnothing$

$\mathsf{A}\widehat{\oplus}\mathsf{B} \stackrel{\text{def}}{=} \mathsf{A} \oplus \mathsf{B}$ defined only if $\pi(\mathsf{A}) \cap \pi(\mathsf{B}) = \varnothing$

$\mathsf{A}\widehat{;}\mathsf{B} \stackrel{\text{def}}{=} \mathsf{A};\mathsf{B}$ defined only if $\forall_{e \in e_{\mathsf{A}}} .(\pi(e) \cap \pi(\mathsf{B}) = \varnothing)$

5 Observational semantics

Weak semantics with a fixed set of priorities is difficult to define. Consider process A in Fig 7 placed in a context where both b^{l} and e^{p} are enabled after s, then b^{l} will never be executed. But b^{l} can be executed if b^{l} and c^{p} are enabled after s. The problem is that neither b^{p} nor b^{l} is acceptable for $\mathsf{b}^{?}$.

A solution to this problem that keeps the fixed set of priorities can be found in [2]. But because we are working in a more relaxed situation where the set of priorities may be expanded it is possible to construct (what we believe to be) a simpler observational semantics.

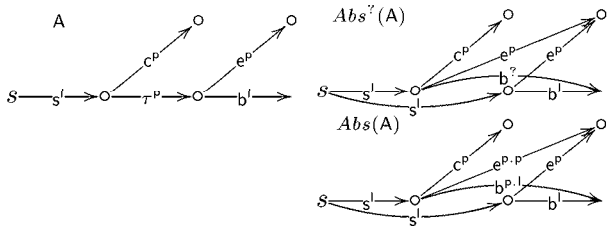

Figure 7. Actions following τ

By allowing priorities p, l and states n, m, k to be joined in a sequence and extending $\angle$ to these sequences so that *e.g.* $p.m.l \angle p.m.p$ but $p \not\angle p.m.l$ and $p.m.l \not\angle p.k.p$, we can give the actions of $Abs(\mathsf{A})$ the priorities we would expect from an operational understanding of A in Fig 7. The states as well as priorities were used in the sequence to cope with examples such as B in Fig 8.

Definition 10 *The abstraction of τ actions from PLTS* A *is written* $Abs(\mathsf{A})$*:*

$$Abs(\mathsf{A}) \stackrel{\text{def}}{=} (N_{\mathsf{A}}, T_{Abs(\mathsf{A})}, s_{\mathsf{A}}, e_{\mathsf{A}}) \text{ where}$$

$$T_{Abs(\mathsf{A})} \stackrel{\text{def}}{=} \{n_1 \xrightarrow{\alpha^{p_1 n_2 p_2 \cdots p_n y p m_1 q_1 \cdots q_m}} z \mid \forall_{i<n}\, n_i \neq n_{i-1} \wedge \forall_{j<m}\, m_j \neq m_{j-1} \wedge n_1 \xrightarrow{\tau^{p_1}} n_2, n_2 \xrightarrow{\tau^{p_2}} n_3 \ldots n_n \xrightarrow{\tau^{p_n}} y \wedge y \xrightarrow{\alpha^{p}} m_1 \wedge m_1 \xrightarrow{\tau^{q_1}} m_2, \ldots m_m \xrightarrow{\tau^{q_j}} z\} \quad \bullet$$

In the above definition the clause $\forall_{i<n}\, n_i \neq n_{i-1} \wedge \forall_{j<m}\, m_j \neq m_{j-1}$ results in τ loops being ignored. This is similar to the treatment in CCS and ACP but conflicts with the chaotic interpretation of divergence in CSP. An operational approach where τ abstraction splits states that may diverge into states that must diverge and states that cannot diverge can be seen in [19]. Here we avoid further discussion of divergence and restrict ourselves to the CCS and ACP interpretation of divergence.

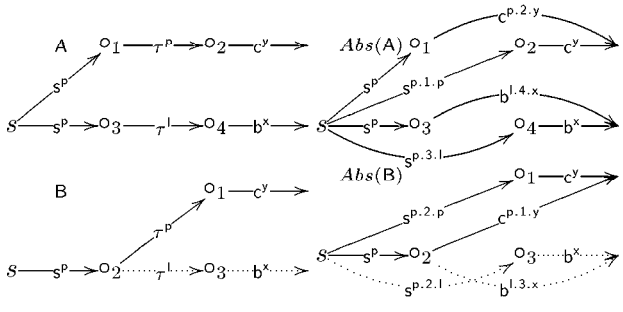

Figure 8. Actions preceding τ

Definition 11 *Let P and Q be sequences of priorities and let $p : P$ be a sequence with head p and tail P and let $\langle\rangle$ be the empty sequence. We define $\angle$ on sequences of priorities:*

$$p : P \angle q : Q \stackrel{\text{def}}{=} p \angle q \vee (p = q \wedge P \angle Q),$$
$$n.p : P \angle n.q : Q \stackrel{\text{def}}{=} p \angle q \vee (p = q \wedge P \angle Q), \quad \bullet$$

Definition 12 *The abstraction of $x \xrightarrow{\tau^p} y$ from PLTS* A*:*

$$Abs(x \xrightarrow{\tau^p} y, \mathsf{A}) \stackrel{\text{def}}{=} (N_{\mathsf{A}}, T_{\stackrel{\rightarrow}{Abs(\mathsf{A})}}, s_{\mathsf{A}}, e_{\mathsf{A}}) \text{ where}$$

$$T_{\stackrel{\rightarrow}{Abs(\mathsf{A})}} \stackrel{\text{def}}{=} \begin{cases} x = y, & T_{\mathsf{A}} - x \xrightarrow{\tau^p} y \\ x \neq y, & T_{\mathsf{A}} \cup \{w \xrightarrow{\alpha^{qxp}} y \mid w \xrightarrow{\alpha^q} x\} \cup \\ & \{x \xrightarrow{\alpha^{pyq}} z \mid y \xrightarrow{\alpha^q} z\} - x \xrightarrow{\tau^p} y \end{cases} \quad \bullet$$

Abstraction of set of τ actions can be computed by the abstraction of the τ actions one at a time.

Lemma 3 *Let $\{x_1 \xrightarrow{\tau^{p_1}} y_1 \ldots x_n \xrightarrow{\tau^{p_n}} y_n\}$ be the set of τ actions in PLTS* A*.*

$$Abs(\mathsf{A}) = Abs(x_1 \xrightarrow{\tau^{p_1}} y_1, \ldots Abs(x_n \xrightarrow{\tau^{p_n}} y_n, \mathsf{A}) \ldots)$$

Proof Define an order: the maximum length of τ chains within which order by the number of τ chains of maximum length. Then by induction on this order.

Base case $i = (1, 1)$ result obvious.

Assume true for $i < n$ then: Let $x_n \xrightarrow{\tau^{p_n}} y_n$ be in a τ chains of maximum length.

Let PLTS B be of order n, hence $Abs(x_n \xrightarrow{\tau^{p_n}} y_n, \mathsf{B})$ is of order less than n.

By inductive hypothesis $Abs(Abs(x_n \xrightarrow{\tau^{p_n}} y_n, \mathsf{B})) =$
$Abs(x_1 \xrightarrow{\tau^{p_1}} y_1, \ldots Abs(x_n \xrightarrow{\tau^{p_n}} y_n, \mathsf{B}) \ldots)$

We need to show that

$$Abs(\mathsf{B}) = Abs(Abs(x_n \xrightarrow{\tau^{p_n}} y_n, \mathsf{B}))$$

The difference between B and $Abs(x_n \xrightarrow{\tau^{p_n}} y_n, \mathsf{B})$ is restricted to transition using nodes x_n and y_n. Each action added is dependent on the existence of a single action connected to x_n or y_n. We need to prove that for each action in $Abs(x_n \xrightarrow{\tau^{p_n}} y_n, \mathsf{B})$ but not in B the actions added by applying $Abs(\_)$ to $Abs(x_n \xrightarrow{\tau^{p_n}} y_n, \mathsf{B})$ will also be added by applying $Abs(\_)$ to B. By case analysis on actions connected to x_n or y_n:

Case 1 $x_n = y_n$ add nothing

Case 2. $x_n \neq y_n$ and action connected to only one of x_n or y_n:

2.a $z \xrightarrow{\alpha^q} y_n$ add nothing

2.b $x_n \xrightarrow{\alpha^q} z$ add nothing

2.c $x_n \xrightarrow{\alpha^q} x_n$ add $x_n \xrightarrow{\alpha^{q r_n p_n}} y_n$, and clearly whenever this action is used in the construction of $Abs(\_)$ exactly the same effect could have been achieved using the two component actions $x_n \xrightarrow{\alpha^q} x_n$ and $x_n \xrightarrow{\tau^{p_n}} y_n$.

2.d $z \xrightarrow{\alpha^q} x_n$, add $z \xrightarrow{\alpha^{q x_n p_n}} y_n$

2.e $y_n \xrightarrow{\alpha^q} y_n$ add $x_n \xrightarrow{\alpha^{x_n p_n q}} y_n$, $y_n \xrightarrow{\alpha^q} z$, add $x_n \xrightarrow{\alpha^{x_n p_n q}} z$

Case 3. $x_n \neq y_n$ and action connected to both x_n and y_n:

3a. $x_n \xrightarrow{\alpha^q} y_n$ add nothing

3b. $y_n \xrightarrow{\alpha^q} x_n$, add $x_n \xrightarrow{\alpha^{p y q}} x_n$, $x_n \xrightarrow{\alpha^{p y q r p}} y_n$, $y_n \xrightarrow{\alpha^{q r p}} y_n$, $y_n \xrightarrow{\alpha^{q x p y q}} x_n$ •

Lemma 4 *If PLTS* A *is deterministic then* $Abs(\mathsf{A})$ *is deterministic.*

Proof We will show that if A is deterministic then $Abs(x \xrightarrow{\tau} y, \mathsf{A})$ is deterministic. The result then follows by induction and Lemma 3.

If A is deterministic then $Abs(x \xrightarrow{\tau} y, \mathsf{A})$ is deterministic follows directly from the construction. •

5.1 Pragmatics

A common mathematical methodology is to use syntax to represent things of interest and use semantics to justify only syntactic rules. An alternative methodology is to use the strong semantics in place of the "syntax" and the observational semantics in place of the "semantics".

It is possible to view the construction of an observational semantics as a change in the level of abstraction *i.e.* the construction of a more abstract level. But this can lead to unnecessarily complex operational semantics. With "annotated" transition systems such as here and [9] the observational semantics require more complex transition systems than are needed to represent the strong semantics. To avoid such complexity there is a common methodology:

1. build the strong operational semantics;

2. use the more abstract observational operational semantics to justify simplification rules that can be applied to the simpler strong operational semantics, such as identifying observationally bisimular nodes.

Our definition of priority on the strong semantics may be a total order while the priority relation of observational semantics cannot be transitive. By this methodology the non-transitive priority relation is no more than a computational step in the simplification of the strong semantics with its totally ordered priority relation.

6 Conclusion

The standard process algebras are very elegant and are a simple means of representing process that have proven to be very useful. An unfortunate consequence of process algebras' simplicity is that the interaction between two p-deterministic processes is not p-deterministic.

This is inconsistent with the way determinism for state-based systems works. The inconsistency will clearly make any synthesis between action-based and state-based formalisms problematic. The work here seeks to dissolve this inconsistency.

With our representation of processes we take as primitive the distinction between *causal* actions and *response* actions. For each causal action $\overline{\mathsf{a}}$ (response action a) there is a matching response action a (causal action $\overline{\mathsf{a}}$). A process can perform only one of these actions when placed in a context that is prepared to perform the matching action. Where a context is prepared to perform more than one of the actions the action with the highest priority is performed.

With our representation of processes the ability to *observe* is distinct from the ability to *choose*. Consequently we can model the ability to observe what a robot does while being unable to change the behaviour of the robot.

We define a finite state process to be implementable if and only if it is deterministic, and our process operators construct an implementation from two component implementations.

In [4] they consider coupling the approaches from process algebras and abstract state machines by extending abstract state machines with process algebraic ideas. We make one very small step the other way by extending process algebra with the notion from abstract state machines of one action causing another.

Priority between process actions has been well researched [8, 9] and our approach is limited to what is called *global priority* in [9]. Our work adds priorities to a semantics with causal and response actions. We can find no prior work to do this.

Although the reduction of non-determinism is an important method of refinement, the change of level of abstraction is also very important and needs to be integrated with our approach.

Finally, during the analysis of large systems, information hiding will now not introduce non-determinism, thus only the trace semantics will be needed which will make the analysis simpler. Using our model it may be possible to apply the language- (trace-) based analytic methods of discrete event systems [16] to situations where not all the events are observable.

Acknowledgements

Thanks to Eerke Boiten, Moshe Deutsch, Doug Goldson, Robi Malik, Greg Reeve and Mark Utting for many helpful discussions. Thanks to New Zealand Government's Foundation for Research, Science and Technology for funding to make this research possible.

References

[1] J. C. M. Baeten and W. P. Weijland. *Process Algebra.* Cambridge Tracts in Theoretical Computer Science 18, 1990.

[2] J. Bergstra, A. Ponse, and S. Smolka. *Handbook of Process Algebra.* Elsevier, 2001.

[3] A. Blass and Y. Gurevich. Abstract state machines capture parallel algorithms. *ACM Trans. Comput. Logic*, 4(4):578–651, 2003.

[4] T. Bolognesi and E. Borger. Abstract State Processes. In E. Börger, A. Gargantini, and E. Riccobene, editors, *Abstract State Machines, Advances in Theory and Practice, 10th International Workshop, ASM 2003, Taormina, Italy, March 3-7, Proceedings*, volume 2589 of *Lecture Notes in Computer Science*, pages 218–228, 2003.

[5] C. Bolton and J. Davies. A singleton failures semantics for Communicating Sequential Processes. Research Report PRG-RR-01-11, Oxford University Computing Laboratory, 2001.

[6] C. Bolton and J. Davies. A comparison of refinement orderings and their associated simulation rules. In J. Derrick, E. Boiten, J. Woodcock, and J. von Wright, editors, *Electronic Notes in Theoretical Computer Science*, volume 70. Elsevier, 2002.

[7] L. Brown, editor. *The New Shorter Oxford English Dictionary.* Oxford University Press, 1993.

[8] R. Cleaveland, G. Luttgen, and V. Natarajan. A process algebra with distributed priorities. In *International Conference on Concurrency Theory*, pages 34–49, 1996.

[9] R. Cleaveland, G. Luttgen, and V. Natarajan. Priority in Process Algebra. In J. Bergstra, A. Ponse, and S. Smolka, editors, *Handbook of Process Algebra.* Elsevier Science, Amsterdam, The Netherlands, 2001.

[10] Y. Gurevich. Evolving Algebras 1993: Lipari Guide. In E. Börger, editor, *Specification and Validation Methods*, pages 9–36. Oxford University Press, 1995.

[11] M. Hennessy. *Algebraic Theory of Processes.* The MIT Press, 1988.

[12] C. Hoare. *Communicating Sequential Processes.* Prentice Hall International Series in Computer Science, 1985.

[13] R. Kaivola and A. Valmari. The Weakest Compositional Semantic Equivalence Preserving Nexttime-less Linear temporal logic. In *International Conference on Concurrency Theory LNCS 630*, pages 207–221, 1992.

[14] N. Lynch and M. Tuttle. An introduction to input/output automata. *CWI-Quarterly*, pages 2(3):219–246, 1989.

[15] R. Milner. *Communication and Concurrency.* Prentice-Hall International, 1989.

[16] P. Ramadge and W. Wonham. The control of discrete event systems. *Proceedings of IEEE*, 77(1):81–98, January 1989.

[17] S. Reeves and D. Streader. Comparison of Data and Process Refinement. In J. S. Dong and J. C. P. Woodcock, editors, *ICFEM 2003*, LNCS 2885, pages 266–285. Springer-Verlag, 2003.

[18] A. Roscoe. *The Theory and Practice of Concurrency.* Prentice Hall International Series in Computer Science, 1997.

[19] S.Reeves and D.Streader. Atomic Components. Technical report, Universty of Waikato, 2004. Computer Science Technical Report 01/04 (submitted), http://www.cs.waikato.ac.nz/~dstr.

[20] A. Valmari and M. Tienari. An improved failure equivalence for finite-state systems with a reduction algorithm. In *Protocol Specification, Testing and Verification*, IFIP XI. North-Holland, 1991.