Modelling Shared Variables in Stochastic Petri Nets and Process Algebras

Richard Hopkins*

June 5, 2003

1 Introduction

This paper addresses the question of the relative adequacy of standard Process Algebras (PAs) and composable Petri net (PN) classes to provide compositional stochastic semantics for programming/design languages which allow shared variable updating by parallel components. We will consider both "High-level (HL)" and plain or "Low-level (LL)" versions of PNs and PAs. For PNs HL means a token carries a value, and for PAs it means what is normally known as value-passing. Consideration of LL semantics is important because in both PN and PA worlds there are (and probably always will be) analysis tools for LL models which are not yet extended for HL models.

First we make some general comments, including introducing a novel categorisation of resume policies, and explaining an example to be used in the rest of the paper which discusses approaches to its semantics using: composable PNs; standard PAs; non-standard PAs.

2 General Considerations

2.1 Process Algebras and Composable Petri Nets

By a "standard" PA we mean one having a compositional behavioural model in which an atom of functional behaviour (something which happens in its entirety or not at all) has no sub-structure. This includes both CCS-based and TCSP-based PAs. In the CCS case the behavioural model is a labelled transition system (LTS) where a transition label is the atom of behaviour. The atom comprises a functional part, a visible action or a τ, and possibly a non-functional part such as a duration distribution. The

*School of Mathematical and Computer Sciences, Heriot-Watt University, rph@macs.hw.ac.uk

TCSP failure-sets behavioural model is based on a trace as a sequence of atoms, for each of which the behaviour part is again un-structured. We will argue that standard PAs have some inadequacy with regard to modelling shared updatable variables.

By a composable PN class (e.g. Hierarchic Coloured PNs [16], Modular Nets [9], Labelled GSPNs [3, 10], STOCON [15]) we refer to a class with composition operators, or a hierarchic structure, which provides place fusion and transition fusion as the means of interaction between component nets.

2.2 Stochastic Features

We will consider shared variable languages with stochastic features, requiring a stochastic formalism for their semantics, for which we will consider the use of general distributions rather than just exponential. This is because, as we will see, the introduction of durations poses additional challenges to the semantics of shared variables, particularly so for non-exponential durations, and it seems reasonable to expect of an approach to semantics that it should not inherently preclude such capabilities.

Within stochastic PA/PN models there is a strong distinction between Markovian in which all durations have exponential distributions, versus non-Markovian in which more general distributions are allowed. For stochastic PAs there is a dichotomy in that, as a general rule, non-Markovian PAs are not generalisations of Markovian PAs — in a notional partial order of expressivity, they are incomparable. All Markovian PAs (e.g. PEPA[13], MTIPP[12], EMPA[4]) provide timed synchronisation, i.e. an atom can consist of an action defining its potential synchronisation and a rate defining its duration, whereas (with the exception of [8]) non-Markovian PAs (IGSMP[7], ♠[17], timed FSP [1]) do not. This is because they employ timers for durations — a duration has one atom for starting its associated timer and one atom for the completion. This may be: explicit [17, 1] where the PA syntax has start-timer and wait-timer actions; or implicit [7] where the PA provides for the direct specification of a duration, but in the behavioural model this is split into a start and end atom. The explicit-timer approach clearly precludes timed synchronisations. The implicit-timer approach probably does not in principle preclude timed synchronisations, but to include them would make more complex the already very complex rules for compositional construction of the behavioural model. The reason for using timers is that otherwise, as in [8], a general true-concurrency behavioural model is required, and the expansion law is lost. The effect of using timers is that the true-concurrency is restricted to the timers all being concurrent with each other and with the interleaved functional behaviour, and that concurrency is made implicit.

A general solution to the modelling of shared variables should be able to accommodate non-exponential timed synchronisations; if timed synchronisations are a useful modelling device in the exponential case, they are presumably also useful in the non-exponential case.

2.3 Examples

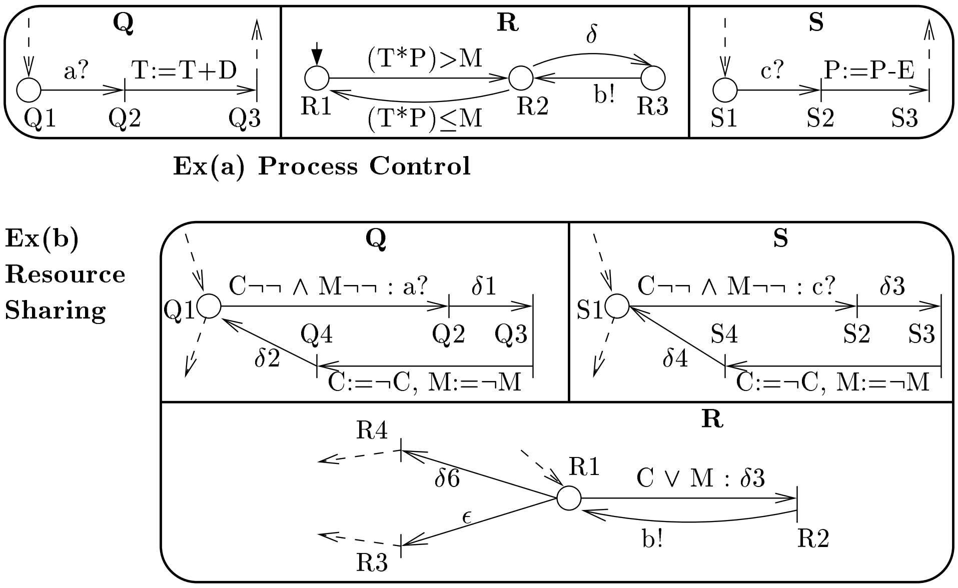


Figure 1: Shared Variable Examples

In Fig.1 we show two examples, represented as pseudo-UML state diagrams. Ex(a) is intended to motivate the need for shared updatable variables, whereas Ex(b) is a related example to be used as the guinea pig in the rest of the paper. Referring first to Ex(a). This is a state with three parallel sub-states for three design components. Both a circle and a bar are states and an arrow is an atomic transition between states. We have the following kinds of transitions: input, e.g. $a?$; output, e.g. $b!$; assignment, e.g. $T := T + D$; boolean guard, e.g. $(T * P) > M$; duration, e.g. δ (for exponentially distributed durations we will use ϵ). A circle is a general state having possibly multiple in-comming and out-going transitions, whereas as a bar is an intermediate state in a sequence, and so has exactly one of each. The initial state, $R1$ for R, is indicated by an arrow head.

The example is modelling some process control application with two variables for the process state - T its temperature, and P its pressure. The left and right components, Q and S, are responsible for tracking and updating T and P respectively. The central component, R, is responsible for monitoring a safety condition, that $T * P$ does not exceed some limit M. Whenever that limit has been exceeded for some time period, duration δ, an alarm is sounded by the output $b!$. The use of parallelism in the design is not necessarily to do with physical parallelism in the implementation, but may be just to do with factoring the design. Particularly, if R were not a parallel component then its behaviour would have to be replicated at every point in the rest of the design where T or P is changed.

Ex(b) is concerned with two shared resources, represented by boolean variables C and M, each of which is true if its resource is in use (reserved). The additional notation in this example is as follows. The $\neg\neg$ operator, e.g $C\neg\neg$ negates C and returns the negated value. The : is for a multi-part guard with any combination of: a boolean guard; a communication; a duration. If all three are present, that gives a timed synchronisation requiring the boolean to hold throughout its duration. Considering first component Q, from state $Q1$: first it does input $a?$ if the two resources are available and in doing so reserves them; it holds those resources for a duration $\delta 1$; it releases them; after a duration $\delta 2$ it returns to $Q1$. Component S has the same structure, and thus there is resource competition between Q and S. In component R the loop $R1 - R2 - R1$ monitors resource usage by sending a message $b!$ whenever there has been a period, defined by $\delta 5$, during which one or other resource is in use. The other two paths out of $R1$ are to introduce challenge to the semantics: a race involving exponential and non-exponential durations.

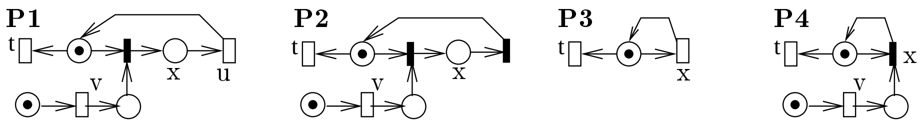
2.4 Preemption Policies

We will find that a particular semantics for Ex(b) may give different behaviour depending on the preemption policy adopted by the target formalism. A timed transition can be considered to have a count-down timer, which is set to some value when it is first enabled. If the transition is "preempted", then its timer value is discarded and the timer starts again (re-sampled) from whenever the transition is enabled. By preemption policy we here mean the rules that govern when a transition is preempted. Fig.2 shows: The possibles rules, P1–P4; A diagram of their relationships where an arrow is "implies"; A PN example of each rule, t being preempted if x is marked or fires, using a box for a timed transition and a bar for an immediate transition; Equivalent examples in a PA where a $\square$ is inaction, a δ is a timed (τ) action, and others are immediate. P1 is necessary for there to be any notion of preemption. In terms of the Resampling/Enabling-Memory/Age-Memory (REA) classification of [2, 18], excluding even P1 is Age-Memory; and the other combinations are variants of Enabling-Memory.

In a PA one would normally have rule P4 (and thus all the other rules) because conflict is given by choice which is normally seen as discarding everything to do with the non-taken branch. Also, one would not have rule P2 without rule P4, because using the agent definitions in the Fig., one would want the stochastic equivalence $P2 \approx P4$ (or more clearly, $a.\tau.X$

An enabled timed transition is preempted by
P1. The system reaching a tangible state in which it is not enabled.
P2. The system reaching any state in which it is not enabled.
P3. (P1 or) The occurrence of a conflicting timed transition.
P4. The occurrence of any conflicting transition.

	disabling state		conflicting transition
substantial	P1:tangible	$\leftarrow$	P3: timed
	$\uparrow$		$\uparrow$
(substantial and) ephemeral	P2: any	$\leftarrow$	P4: any



$P1 =$	$P2 =$	$P3 =$	$P4 =$
$(\text{fix}(X = (\delta_t.\square)$	$(\text{fix}(X = (\delta_t.\square)$	$(\text{fix}(X = (\delta_t.\square)$	$(\text{fix}(X = (\delta_t.\square)$
$+(a.\delta_u.X))$	$+(a.\tau.X))$	$+(\delta_x.X))$	$+(a.X))$
$\Vert(\delta_v.\widehat{a}.\square)) \setminus \{a\}$	$\Vert(\delta_v.\widehat{a}.\square)) \setminus \{a\}$	$)$	$\Vert(\delta_v.\widehat{a}.\square)) \setminus \{a\}$

Figure 2: Preemption Policies

congruent with $a.X$, since only there do $P2$ and $P4$ differ.) In contrast, in PNs preemption is more usually formulated in terms of disabling states. The Enabling-Memory of REA in [2] is rule P2, making no distinction between tangible and vanishing states. However, we would argue that: P2 without P4 is unreasonable, on the grounds that PA equivalence $P2 \approx P4$ should equally hold for nets; perhaps P2 is anyway unreasonable, on the grounds that vanishing states are not really real - in the net P2, t has 0 probability of noticing that its enabling token has been moved to x.

3 Using Composable Nets

3.1 A first Definition of Ex(b) Semantics

In Fig.3(a) we show how the semantics of Ex(b) would be obtained using a form of stochastic high-level composable nets. Referring to the left net, the semantics of Q. A place is either coloured, e.g. C, in which case a token on it carries a value (here a boolean value), or plain, e.g $Q1$ in which case a token has no value. Also it is either an interface place, e.g. C, or an internal place, e.g. $Q1$. An interface place can participate in a fusion

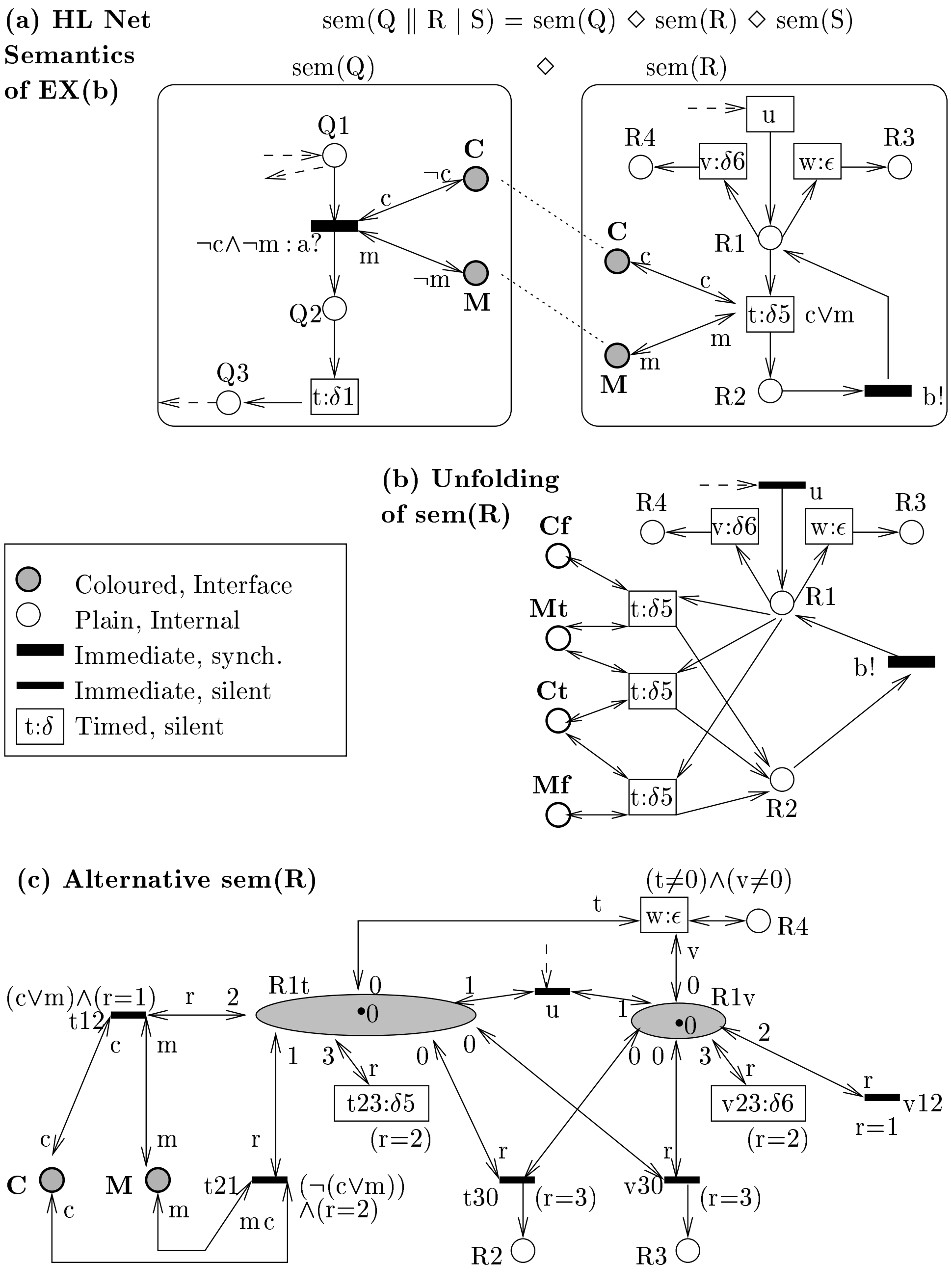


Figure 3: Semantics using Composable Nets

with interface places of other nets, whereas an internal place cannot. An arc from a coloured place to a transition is annotated (at the arrow) with a variable, e.g. c, which in any firing of the transition is bound to the value for the token from that place used for that firing. An arc from a transition to a coloured place is annotated with an expression, e.g. $\neg c$ which defines the value for the token output to that place. A transition may be annotated with a boolean guard, e.g. $\neg c \wedge \neg m$, in which case the guard must evaluate to True for the transition to fire. A transition may be silent or synchronising. If synchronising, it is labelled with a communication action, e.g. $a?$. It may be immediate or timed. If timed, it is decorated with a tag and a duration (distribution), e.g. $t : \delta 1$ (the role of the tag will be discussed later). We show one net combination operator, $\diamond$, which does disjoint net union, and then place and transition fusions. The place fusions are shown by dotted lines - the Cs are fused and the M's are fused. Transition fusions are used for synchronised communication - in a further composition there would be fusion of the $a?$ labelled transition with every $a!$ labelled transition from other nets.

The semantic mapping from Ex(b) to Fig.3(a) is straightforward, indeed almost trivial. The translation is compositional at the level of individual transitions. For the $Q1 - Q2$ transition of Ex(b) the semantics is the left net with only $Q1$, $Q2$, C, M, and the connecting transition, with all those places being interface places. For the complete compositional semantics, in addition to net combination operator used here, we would need at least: a net operator corresponding to CCS restriction on transition labels; a place hiding operator to internalise an interface place and assign its initial marking.

In Fig.3(b) we show the "unfolding" of part of the HL net semantics of R to give the corresponding LL net semantics. The LL net is of the same class as the HL net, but with the property that there are no coloured places and (thus) no transition guards. For each coloured place of the HL net, e.g. C, the LL net has a set of plain places, Cf and Ct, one for each possible value, f(alse) and t(rue). An f-valued token on C becomes a plain token on Cf, and similarly for Ct. There is one transition of the LL net for each HL net transition and each binding of its variables which gives its guard the value True. The upper transition in (b) is for $c = f$ and $m = t$, and so the arcs with C and M become respectively arcs with Cf and Mt.

3.2 Difficulties and Alternatives

The following scenarios illustrate various issues for this semantics of R when $\delta 5$ is non-exponential.

1. Preempted Durations. Suppose: $\delta 5$ is a duration of 4 units; At time T, we have the $Q1 - Q2$ transition of Q, enabling the start of that duration; At time $T + 2$ we have the two immediate transitions $Q3 - Q4$ and (then) $S1 - S2$, i.e. the release of the resources by one component and their immediate acquisition by another waiting component. This instantaneously disables and then re-enables the $\delta 5$ transition. So the behaviour is different depending on whether the preemption policy includes rule P2 (see Fig.2).

2. Alternative LL Transitions. Consider the above scenario, but at time $T+1$ some other parallel component does an immediate transition which changes $(C = t,\ M = t)$ to $(C = t,\ M = f)$. After that (in the HL semantics) the $\delta 5$ transition is still enabled. Thus if we don't have (preemption rule) P4, $\delta 5$'s duration is not re-started. However in the LL semantics: between T and $T+1$ only the central transition is enabled, whereas after that change the central transition is disabled and the lower transition becomes enabled. In effect the $\delta 5$ duration is re-started. One solution to this HL/LL inconsistency is to associate with each transition a tag, the t's in the figures, with the interpretation that multiple transitions with the same tag are "really" the same transition.
3. Concurrent Enabling. Consider the situation where we have a further component R' which is a copy of R. In these two components the $\delta 5$ transitions are in conflict on the C and M fused places (but leave each other enabled). Thus we get different behaviour depending on whether we have P3. This dependency can be removed by using enabler arcs — as the converse of inhibitor arcs, a transition requires a token to be on a place with which it has an enabler arc, but is not considered to be in conflict with other such transitions.

Note: none of the above issues arise if only exponential durations are used, because their memory-less-ness means it is irrelevant whether or not they are preempted.

The semantics of Fig.3(a) might now seem somewhat fragile in its requirements on the details of the target net class. Fig.3(a) gives a an alternative semantics for R. This is more complex, but more robust. Instead of a single plain place for $R1$ we have multiple coloured places, $R1t$ and $R1v$, one for each out-going non-exponentially timed transition, t and v. For each the semantics implements a timer. The meaning of the value on place $R1x$ is: 0, x is not enabled; 1-3, it is enabled; 2, its timer has started; 3, its timer has completed. Considering the transitions around $R1t$: $t12$ is for starting the timer; $t21$ is for cancelling the timer; $t23$ is the timer; $t30$ is for completion of t, setting both t and u as not enabled and setting the new state as $R2$. The only remaining fragility is from *1.* above - the behaviour is different depending on whether we have P2. And if we have P2, it is different from the original in that now it is non-deterministic as to whether $t21$ fires between the release of resources by Q and their immediate acquisition by S.

If, as argued above, P2 is excluded, then Fig.3(a) provides an ideally straightforward semantics if we have tags and either the absence of P3 or enabler arcs; and (b) is a satisfactory solution otherwise.

4 Using Standard PAs

As evidence of our claim that standard PAs are inadequate for semantics of shared variables we call on two examples of standard PAs being used for semantics of shared variable programming languages. Firstly, the TCSP semantics of OCCAM [20], where the validity of the semantics rests on the restriction that if one component updates a variable, no parallel component may access it. The second example is Milner's CCS semantics of a parallel programming language $\mathcal{M}$ [19]. In that, parallel updated variables are allowed, but the semantics give effects that are not what one would expect. First we consider that approach in detail, and then two other possible approaches.

4.1 Locking of Shared Variables

An example of the unexpected effects mentioned above is

$x := 0 \; ; \; y := 0 \; ; \; z := 0 \; ;$

$(\; (x := x + 1 \; ; \; y := y + 1) \; PAR \; (if \; y > x \; then \; z := z + 1 \; else \; skip \;)$

one would expect the $y > x$ should always fail, and thus z should always have final value 0, whereas in the semantics of [19] a final value of 1 for z is possible.

In that (HL) CCS semantics a variable x is an agent in parallel with the agents for the program components accessing it, providing actions for writing and reading its value ($put_x(V)$ and $get_x(V)$ where V is the value), and lock and un-lock actions (up_x and $down_x$). The semantics for $x := x+1$ is given as

$\widehat{up_x}.\widehat{get_x}(x).\widehat{put_x}(x+1).\widehat{down_x}$

with x being locked to prevent interference between simultaneous such updates (and similarly for $y := y + 1$). The semantics for

$if \; y > x \; then \; z := z + 1 \; else \; skip$

is equivalent to

$$
\begin{array}{ll}
 & (\widehat{get_x}(x).\widehat{get_y}(y). \\
 & \quad ((if \;\; y > x \; then(\ldots \text{“}z := z + 1\text{”} \ldots)) + (if \;\; y \leq x \; then(\ldots \text{“}skip\text{”} \ldots))) \\
+ & (\widehat{get_y}(y).\widehat{get_x}(x). \\
 & \quad ((if \;\; y > x \; then(\ldots \text{“}z := z + 1\text{”} \ldots)) + (if \;\; y \leq x \; then(\ldots \text{“}skip\text{”} \ldots)))
\end{array}
$$

That is, the values of x and y are accessed independently in either order, and then used for the conditional test. The semantics of PAR gives parallel composition. In the interleaving LTS semantics for this we can clearly get a transition sequence in which x is accessed as 0, then x is updated to 1, then y is updated to 1, and then y is accessed as 1, with the result that $y > x$ is found true, and so z is assigned 1.

The obvious solution is for the $y > x$ to lock x and y before doing the test. If the test fails the locks must be released, to allow other components accessing them to progress, and then that cycle must repeat. In the absence of any consideration of time, this might be considered a satisfactory solution. However if the language includes time, with the semantics being a stochastic PA, there is a problem in that: If the repeating cycle of lock acquisition/release comprises all immediate actions, then that prevents any progress of time; If that cycle involves timed actions, e.g. some duration between releasing locks and re-acquiring them, then this duration may distort the time behaviour of the program. (We emphasise that if the intended implementation does not involve physical parallelism, then the locking structure is entirely an artefact of the semantics.)

This "locking" solution for a standard PA seems inadequate both for the above reason and the complexity and non-transparency of the semantics.

4.2 Sequentialisation

$$
\begin{array}{rcl}
sem(QRS) & = & P(0,0,0,f,f) \\
\text{where} & & \\
P(q,r,s,c,m) & = & sem_{E_Q}(Q) + sem_{E_R}(R) + sem_{E_S}(S) \\
sem_{E_Q}(Q) & = & \ldots \\
& + & \langle (q=1) \wedge (\neg c) \wedge (\neg m) : a? \rangle \ . \ P(2,r,s,\neg c,\neg m) \\
& + & \langle (q=2) : \delta 1 \rangle \ . \ P(3,r,s,c,m) \\
& + & \langle (q=3) : \tau \rangle \ . \ P(4,r,s,\neg c,\neg m) \\
& + & \ldots \\
sem_{E_R}(R) & = & \ldots \\
& + & \langle (r=1) \wedge (c \vee m) : \delta 5 \rangle \ . \ P(q,2,s,c,m) \\
& + & \ldots \\
\ldots & &
\end{array}
$$

Figure 4: Semantics using Sequentialisation

Returning to the example of Ex(b) and employing a HL standard PA, we can obtain a semantics as shown in Fig.4. Here we have one parameterised agent, $P(q,r,s,c,m)$. The parameters c and m are for the values of the variables C and M. The parameters q, r and s are for the control states of the Q, R and S components, with $q=1$ meaning Q is in state $Q1$ etc. In the PA notation we use $\langle \ldots \rangle$ brackets to emphasise atomicity. Whatever is in the brackets occurs indivisibly. Our observations on this solution are:

1. Particularly to accommodate complex sub-structures of parallel components, the semantic definitions become complex and non-transparent.

2. Although the $sem_{E_Q}(Q) + sem_{E_R}(R) + sem_{E_S}(S)$ gives an appearance of compositionality, the semantics are essentially non-compositional; all the following points except 6 identify some compromise of compositionality.

3. The semantics of one component depends on details of other components, as represented by the environments E_X used in the semantic definitions. If say R has a sub-structure of parallel components, there has to be a parameter of the P for each component, and the semantic definition of Q needs to know about this.

4. The semantics does not capture compositional structure, such as parallel components, which may be relevant to application of compositional reasoning/analysis.

5. The semantics of an individual component does not yield a behaviour because it involves an agent identifier, P, which is not defined at that level.

6. The unfolding increases the size of the semantics inordinately. We obtain a family of agent definitions, one for every possible global state, e.g $P_{1,1,1,f,f} = a? \,.\, P_{2,1,1,t,t}$ as one part of the semantics for the $Q1$ to $Q2$ transition.

7. If we can have message communication between parallel components which need to be sequentialised due to this way of giving shared variable semantics, then the semantics has to explicitly (and non-compositionally) give every possible synchronisation.

4.3 State Replication with Prioritised Update

Fig.5 shows a semantics of Ex(b) using a standard PA enhanced with prioritised choice — $P <\!+ Q$ is choice in which Q takes priority over P. This is a compositional solution in that the semantics is the parallel composition of the semantics of each component. Each component involves agent definitions e.g. $Q1(c, m)$, which are parameterised with that components own copy of shared variables. When any component updates any variable, it outputs a message with the new value, as in the $C(\neg c)$, in the definition of $Q1$, using channel C to inform other components of the new value for variable C. At every state every component has the possibility to receive variable updates, as in $\widehat{C}(c)$, and these have priority over the other possibilities from that state. We are assuming a broadcast communication discipline where synchronisation is between one sender sending a specific value, and any number of receivers which receive that value into a local variable — the $C(c)$ is a send where the c is an expression, whereas $\widehat{C}(c)$ is a receive where the c is a variable.

Our observations on this solution are

1. There is a requirement (not inherent in the example) that the target PA provide prioritisation between immediate transitions, and this

$$
\begin{array}{rcl}
sem(QRS) & = & sem(Q) \| sem(R) \| sem(S), \text{ where} \\
sem(Q) & = & Q0(f,f), \text{ where} \\
Q1(c,m) & = & \langle \neg c \wedge \neg m : a? \rangle \ . \ C(\neg c) \ . \ M(\neg m) \ . \ Q2(\neg c, \neg m) \\
& <+ & (\ \widehat{C}(c).Q1(c,m) \ + \ \widehat{M}(m).Q1(c,m) \ ) \\
Q2(c,m) & = & \delta 1 \ . \ Q3(c,m) \\
& <+ & (\ \widehat{C}(c).Q2(c,m) \ + \ \widehat{M}(m).Q2(c,m) \ ) \\
\ldots & & \\
sem(R) & = & R0(f,f), \text{ where} \\
R1(c,m) & = & \langle c \vee m : \delta 5 \rangle \ .R2(c,m) \\
& <+ & (\ \widehat{C}(c).R1(c,m) \ + \ \widehat{M}(m).R1(c,m) \ ) \\
\ldots & & \\
sem(S) & = & S0(f,f), \text{ where} \\
S1(c,m) & = & \langle \neg c \wedge \neg m : c? \rangle \ . \ C(\neg c) \ . \ M(\neg m) \ . \ S2(\neg c, \neg m) \\
& <+ & (\ \widehat{C}(c).S1(c,m) \ + \ \widehat{M}(m).S1(c,m) \ ) \\
\ldots & &
\end{array}
$$

Figure 5: Semantics using State Replication with Prioritised Update

gives rise to an inconsistency between true-concurrency and interleaving semantics. In this example, when $c = f$ and $m = f$, the $\langle \neg c \wedge \neg m : a? \rangle$ action of $Q1(c,m)$ can execute concurrently with the $\langle \neg c \wedge \neg m : c? \rangle$ action of $S1(c,m)$, leading to state in which both Q and S believe they have acquired the resources. That state cannot be reached in the interleaving semantics, and is incorrect for this example. This inconsistency means that a true concurrency PA such as [8], which most cleanly supports non-exponential timed synchronisation, could not be coherently enhanced with the prioritisations needed for this solution.

2. Considering the scenario *2.* from section 3.2. A general-distribution stochastic PA would naturally have preemption rule R4, Thus in the $R1(c,m) =$
$$\langle c \vee m : \delta 5 \rangle \ .R2(c,m) <+ (\ \widehat{C}(c).R1(c,m) \ + \ \widehat{M}(m).R1(c,m) \ )$$
of this semantics, a parallel component making the change from $(C = t, \ M = t)$ to $(C = t, \ M = f)$ will preempt the timed transition. This is not the behaviour we would expect for this program. If the behavioural model does not have R4, then we have the same HL/LL inconsistency discussed previously, requiring the use of "tags" in the behavioural model. One could try to circumvent these difficulties by the kind of structure used in Fig.3(c), but (we believe) defining a reasonable compositional construction of that using standard PA operators is not easy (left as an exercise for the reader).

3. The LL PA semantics is big compared with the LL net semantics. The LL PA semantics involves one agent definition for every combination of: C control states ($Q1$, $P1$, etc.), and N possible values for D variables. I.e. of order $C \cdot (D^N)$. The analogous measure for the PN semantics is number of places, which is of order $C + (D \cdot n)$. For 10 components with 10 control states each, and 10 boolean variables this is 102,400 vs 120.

5 Sub-structured Functional Behaviour Atoms

5.1 Multi-actions

$$
\begin{aligned}
sem(CMQRS) &= sem(C) \| sem(M) \| sem(Q) \| sem(R) \| sem(S), \text{ where} \\
sem(C) &= C(f), \text{ where} \\
C(c) &= (\langle \{v_C('c, c')\}['c = c], * \rangle . C(c')) \\
&\ldots \\
sem(Q) &= Q0, \text{ where} \\
Q1 &= \langle \{\widehat{v_C}('c, c'),\ \widehat{v_M}('m, m'),\ a?\} \\
&\qquad [\neg' c \wedge \neg' m \wedge (c' = \neg' c) \wedge (m' = \neg' m)], 0 \rangle\ .Q2 \\
Q2 &= \langle \{\}, \delta 1 \rangle\ .\ Q3 \\
&\ldots \\
sem(R) &= R0, \text{ where} \\
R1 &= \langle \{\widehat{v_C}('c, c'),\ \widehat{v_M}('m, m')\} \\
&\qquad [(m \vee c) \wedge ('c = c') \wedge ('m = m')], \delta 5 \rangle\ .R2 \\
&\ldots
\end{aligned}
$$

Figure 6: Semantics using Multi-actions

The fundamental problem with attempting shared variable semantics in standard PAs is that the example we have been looking at involves atomic dependency on multiple independent external conditions, and having a behavioural model with un-structured behavioural atoms is unable to represent this directly. There are two classes of PAs where the behavioural atom is structured as a multi-set of independently-synchronisable actions, which gives them a natural capacity to model multi-dependency and thus updatable shared variables. The more relevant of these are PBC [5], its HL version [6], and its stochastic LL version sPBC [6]. The former two have both been used to give fully general semantics of a (non-stochastic) shared variable programming language.

The approach, as in Section 4, is for each variable to be an agent in parallel with the components accessing it. Fig.6 shows a semantics for the example using a HL PA with the PBC style of multi-actions, but with more standard composition operators than in PBC itself. The atom of behaviour is of general form $\langle \{M\}[G],\ D\rangle$ where M is a (multi-)set of synchronisable actions each possibly parameterised by action parameters, G is a guard using those parameters and agent definition parameters, and D is the duration, either: 0 for immediate, * for passive, or δ for timed. The semantics for variable X (C or M) is parameterised agent definition $X(x)$ offering communications of form $v_X('x, x')$ where $'x$ is X's pre-value and x' is its post-value. The semantics of Q etc. involve synchronisations with the actions offered by the variables, again with $'x$ and x' for a variable's pre- and post-values, and the guard defining both the execution conditions and the variable modification effects.

This semantics definition is compositional and is as adequate as the first net semantics of Fig.3(a) in that we can define a (compositional) net translation of this PA which gives the same net as that when the place fusions have been done. However:

1. The usual PBC operators of choice, parallel, sequence, iteration, synchronisation, restriction and refinement, are not very adequate for defining a variable agent, nor indeed for giving semantics for the unstructured "flow chart" style of definition used in UML state diagrams and activity diagrams. For that reason we are considering the PBC multi-actions within more traditional PA compositions.

2. In this example, the semantics has to include non-exponential timed synchronisation and thus the multi-action PA has to address the associated issues even though the example does not inherently require that.

3. All the issues raised for the net semantics arise here also, but the various solutions proposed there are less viable. It is difficult to realise the equivalent of either enabler arcs or the structure of Fig.3(c).

5.2 PAs with Places

The compositional behavioural model for composable PNs in [15] is an LTS with structured behaviour atoms which represent the input and output of tokens from and to interface places. Considering the LL semantics of R in Fig.1, and using $N_R.\{R1\}$ for that net with a token on $R1$, the firing of the upper $t;\delta5$ transition is: $N_R.\{R1\} \stackrel{\downarrow\{C_f, M_t\}/\tau,\delta5/\uparrow\{C_f, M_t\}}{\longrightarrow} N_R.\{R2\}$. The $\tau, \delta5$ is as in the usual stochastic PA behaviour atom. The $\downarrow \{\ldots\}$ identifies a multi-set of tokens input from the environment and the $\uparrow \{\ldots\}$ identifies a multi-set of tokens output to the environment. The net composition operators, $\diamond$ etc., can be characterised by inference rules for this behavioural model. Thus a composable net class can be seen as a PA with non-standard behaviour atom and non-standard composition operators. It is having the appropriate compositional behaviour model, with structured behaviour atoms, which makes composable nets more adequate than standard PAs for representing shared variables. Having identified that behavioural model, we could consider the benefit of using more standard PA operators, an approach explored in [14].

6 Conclusions

We have explored a number of ways of using standard stochastic PAs, those where the functional part of a behaviour atom is unstructured, to obtain semantics of shared variable programming languages, and have shown that, given sufficient generality in the source language, all these approaches encounter difficulties -

"locking of shared variables" becomes inadequate if time is introduced in any way, because the management of locks either distorts the time behaviour or prevents the progress of time.

"sequentialisation" is inadequate even without the introduction of time, due to its non-compositionality.

"state replication with prioritised update" is less than fully adequate in that: the LL semantics grows exponentially with the number of shared variables; if generally distributed durations are introduced there are difficulties with prioritisation being incoherent in a true concurrency model and the spurious preemption of timed actions which may result from one component tracking variable updates of other components.

For composable HL stochastic PNs we have given a semantics which is a trivial transformation of the original state diagram. This however raises issues about preemption policies. Depending on the policy of the target net class, we may need the notion of "tags" in order to obtain HL/LL consistency and enabler arcs, or to use a much more complex formulation of the semantics.

Finally we have considered non-standard PAs, having the functional part of a behaviour atom structured as either (a) a multi-set of actions; (b) an action and input/output multi-sets of places. Both of these are more adequate for shared variable semantics, but in need of more thorough investigation in this context, particularly with regard to preemption policies.

References

[1] T. Ayles. Performance modelling using stochastic transition systems. Technical report, Imperial College, London, 2002.

[2] G. Balbo. Introduction to stochastic Petri nets. In E.Brinksma et al. [11], pages 84–155.

[3] S. Bernardi, S. Donatelli, and J. Merseguer. From UML sequence diagrams and statecharts to analysable Petri net models. In *Proceedings Third International Workshop on Software and Performance - WOSP2002*, pages 35–45, 2002.

[4] M. Bernardo. *Theory and Application of Extended Markovian Process Algebra*. PhD thesis, Bologna Univ., 1999.

[5] E. Best, R. Devillers, and J. Hall. The Petri Box Calculus: A new causal algebra with multi-label communication. In G. Rozenberg, editor, *Advances in Petri Nets '92*, volume 609 of *Lecture Notes in Computer Science*, pages 21–69. Springer-Verlag, 1992.

[6] E. Best, W. Fraczak, R. Hopkins, H. Klaudel, and E. Pelz. M-nets: An algebra of high-level Petri nets, with an application to the semantics of concurrent programming languages. *Acta Informatica*, 35(10):813–857, 1998.

[7] M. Bravetti and R. Gorrieri. The theory of interactive generalised semi-Markov processes. *Theoretical Computer Science*, 282(1):5–32, June 2002.

[8] E. Brinksma et al. A stochastic causality-based process algebra. *Computer Journal (Special Issue - Process Algebra and Performance Modelling)*, 38(7), 1995.

[9] S. Christensen and L. Petrucci. Towards a modular analysis of coloured Petri nets. In *Proc. 13th Int. Conf. on Applications and Theory of Petri Nets*, volume 616 of *Lecture Notes in Computer Science*, pages 113–133. Springer-Verlag, 1992.

[10] S. Donatelli and G. Franceschinis. PSR Methodology: integrating hardware and software models. In *Proceedings Appl. and Theory of Petri Nets*, number 1091 in Lecture Notes in Computer Science. Springer-Verlag, June 1996.

[11] E.Brinksma, H. Hermanns, and J-P Katoen, editors. *Lectures on Formal Methods and Performance Analysis*, number 2090 in Lecture Notes in Computer Science. Springer-Verlag, 2001.

[12] H. Hermanns and M. Rettelbach. Syntax, semantics, equivalences and axioms for MTIPP. Technical report, Univ. Erlangen-Nurnberg, 1995.

[13] J. Hillston. *A Compositional Approach to Performance Modelling*. PhD thesis, University of Edinburgh, 1994.

[14] R. Hopkins. A process algebra subsuming composable stochastic nets. In *Proc SPN 2003*, 2003.

[15] R. P. Hopkins and P. King. A visual formalism for the composition of stochastic Petri nets. In *Tools 2002*, number 2324 in Lecture Notes in Computer Science, pages 239–258. Springer-Verlag, 2002.

[16] K. Jensen. *Coloured Petri Nets.* EATCS Monographs on Theoretical Computer Science. Springer-Verlag, 1992.

[17] J-P. Katoen and P. D'Argenio. General distributions in process algebra. In E.Brinksma et al. [11], pages 375–430.

[18] M. Marsan, G. Balbo, A. Bobbio, G. Chiola, G. Conte, and A. Cumani. The effect of execution policies on the semantics and analysis of stochastic Petri nets. *IEEE transactions on software engineering*, 15(7), July 1989.

[19] R. Milner. *Communication and Concurrency.* Prentice Hall, 1989.

[20] A. Roscoe. Denotational semantics for OCCAM. In *Proc. 1984 Seminar on Concurrency*, number 197 in Lecture Notes in Computer Science, 1984.