Probabilistic pi-calculus and Event Structures

Daniele Varacca, Nobuko Yoshida

Probabilistic π-Calculus and Event Structures [1]

Daniele Varacca[a] Nobuko Yoshida[b]

[a] *PPS - Université Paris 7, France*

[b] *Imperial College London, UK*

Abstract

This paper proposes two semantics of a probabilistic variant of the π-calculus: an *interleaving semantics* in terms of Segala automata and a *true concurrent semantics*, in terms of probabilistic event structures. The key technical point is a use of *types* to identify a good class of non-deterministic probabilistic behaviours which can preserve a compositionality of the parallel operator in the event structures and the calculus. We show an operational correspondence between the two semantics. This allows us to prove a "probabilistic confluence" result, which generalises the confluence of the linearly typed π-calculus.

Keywords: Event structures, probabilistic processes, π-calculus, linear types

1 Introduction and motivations

Probabilistic models for concurrency have an extensive literature: most of the studies concern interleaving models [19,25,9], but recently, true concurrent ones have also been studied [18,1,28,31]. This paper presents an interleaving and a true concurrent semantics to a probabilistic variant of the π-calculus. The variant we consider is similar to the ones presented in [14,6], yet contains important differences. The main difference, which motivates all the others, is the presence of *types*.

The various typing systems for mobile processes have been developed in order to provide disciplines to control non-deterministic behaviours statically and compositionally. In probabilistic concurrency, a restriction of non-determinism becomes more essential, for example, for preservation of the associativity of parallel composition or to guarantee freedom from any specific scheduling policies [28]. This paper performs an initial step towards a "good" typing discipline for probabilistic name passing, which can preserve expressiveness and can harmonise with existing probabilistic concurrent semantics and programming languages [22,10,7].

We present a typing system for the probabilistic π-calculus, inspired from a linear typing systems for the π-calculus [3,35]. The linearly typed π-calculus can embed a family of λ-calculi *fully abstractly*. Linearly typed processes enjoy several interesting properties.

[1] Work partially supported by EPSRC grant GR/T04724/01, and ANR project ParSec ANR-06-SETI-010-02.

This paper is electronically published in
Electronic Notes in Theoretical Computer Science
URL: `www.elsevier.nl/locate/entcs`

In particular they are guaranteed to be *confluent*, that is the computation they perform is deterministic. In the true concurrent setting, confluence can be viewed as absence of conflicts, or *conflict freeness*. In a conflict free system, one can have different partial runs, for instance because one chooses to execute different subsystems. However, under some basic fairness assumptions, and if we abstract away from the order in which concurrent events happen, the system will always produce the same run.

In [30], we extend the linear π-calculus by adding a nondeterministic choice. The typing system no longer guarantees conflict freeness, but the more general behavioural property of *confusion freeness*. This property has been studied in the form of free choice Petri Nets [23,8]. Confusion free event structures are also known as concrete data structures [4], and their domain-theoretic counterpart are the concrete domains [17]. In a confusion free system, all nondeterministic choices are *localised* and are independent from any other event in the system. In the probabilistic setting [28], the intuition is that local choices can be resolved by a local coin, or die. The result in [28] show that probabilistic confusion free systems are *probabilistically confluent*. We have argued that confluence entails the property of having only one maximal computation, up to the order of concurrent events. It is then reasonable to define probabilistic confluence as the property of having only one maximal probabilistic computation, where a probabilistic computation is defined as a probability measure over the set of computations.

We provide an interleaving and a true concurrent semantics to this probabilistic π-calculus. The interleaving semantics is given as Segala automata [25], which are an operational model that combine probability and nondeterminism. The nondeterminism is necessary to account for the different possible schedulings of the independent parts of a system. The true concurrent semantics is given as probabilistic event structures [28]. In this model, we do not have to account for the different schedulings, and that leads to the probabilistic confluence result (Theorem 6.2), one of the main original contributions of this work.

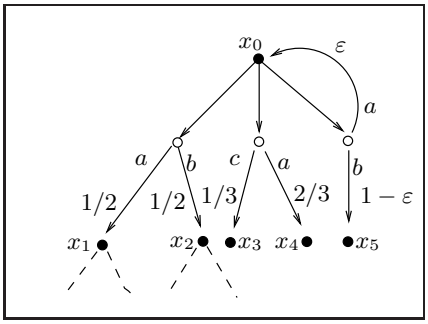
In order to relate the two semantics, we show how a probabilistic event structure generates a Segala automaton. This allows us to show an operational correspondence between the two semantics.

Types play an important role for a compositional semantics, which is given as a clean generalisation of Winskel's original event structure semantics of CCS [32] to the π-calculus. In this sense, this work offers a concrete syntactic representation of the probabilistic event structures as name passing processes, closing an open issue in [28,30]. The work opens a door for event structure semantics for probabilistic λ-calculi and programming languages, using the probabilistic linear π-calculus as an intermediate formalism.

Due to the space limitation, the proofs are omitted and some non-probabilistic materials are left to [30,29].

2 Segala automata

To give an operational semantics to the probabilistic π-calculus we use *Segala automata*, a model that combines probability and nondeterminism. Segala automata can be seen as an extension both of Markov chains and of labelled transition systems. They were introduced by Segala and Lynch [26,25]. A recent presentation of Segala automata can be found in [27]. The name "Segala automata" appears first in [2].



In the initial state x_0 there are three possible transition groups, corresponding to its three hollow children. The left-most transition group is $x_0\{\xrightarrow[p_i]{a_i} x_i\}_{i\in I}$ where $I = \{1,2\}$, $a_1 = a, a_2 = b$ and $p_1 = p_2 = 1/2$. The right-most transition group is $x_0\{\xrightarrow[p_j]{a_j} x_j\}_{j\in J}$ where $J = \{0,5\}$, $a_0 = a, a_5 = b$ and $p_0 = \varepsilon, p_5 = 1-\varepsilon$.

Fig. 1. A Segala automaton

2.1 Notation

A *probability distribution* over a finite or countable set X is a function $\xi : X \to [0,1]$ such that $\sum_{x\in X}\xi(x) = 1$. The set of probability distributions over X is denoted by $V(X)$. By $\mathscr{P}(X)$, we denote the powerset of X. A *Segala automaton* over a set of labels A is given by a finite or countable set of states X together with a transition function $t : X \to \mathscr{P}(V(A\times X))$. This model represents a process that, when it is in a state x, nondeterministically chooses a probability distribution ξ in $t(x)$ and then performs action a and enters in state y with probability $\xi(a,y)$.

The notation we use comes from [14]. Consider a transition function t. Whenever a probability distribution ξ belongs to $t(x)$ for a state $x \in X$ we will write

$$x\{\xrightarrow[p_i]{a_i} x_i\}_{i\in I} \tag{1}$$

where $x_i \in X$, $i \neq j \Longrightarrow (a_i,x_i) \neq (a_j,x_j)$, and $\xi(a_i,x_i) = p_i$. Probability distributions in $t(x)$ are also called *transition groups* of x.

A good way of visualising probabilistic automata is by using alternating graphs [13]. In Figure 1, black nodes represent states, hollow nodes represent transition groups.

2.2 Runs and schedulers

An *initialised* Segala automaton, is a Segala automaton together with an initial state x_0. A *finite path* of an initialised Segala automaton is an element in $(X\times V(X\times A)\times A)^*X$, written as $x_0\xi_1a_1x_1\ldots\xi_na_nx_n$, such that $\xi_{i+1} \in t(x_i)$. An *infinite* path is defined in a similar way as an element of $(X\times V(X\times A)\times A)^\omega$.

The probability of a finite path $\tau := x_0\xi_1a_1x_1\ldots\xi_na_nx_n$ is defined as

$$\Pi(\tau) = \prod_{1\le i\le n}\xi_i(a_i,x_i)\,.$$

The last state of a finite path τ is denoted by $l(\tau)$. A path τ is *maximal* if it is infinite or if $t(l(\tau)) = \emptyset$.

A *scheduler* for a Segala automaton with transition function t is a partial function $\mathscr{S} : (X\times V(X\times A)\times A)^*X \to V(X\times A)$ such that, if $t(l(\tau)) \neq \emptyset$ then $\mathscr{S}(\tau)$ is defined and $\mathscr{S}(\tau) \in t(l(\tau))$. A scheduler chooses the next probability distribution, knowing the history of the process. Using the representation with alternating graphs, we can say that, for every path ending in a black node, a scheduler chooses one of his hollow sons.

Given an (initial) state $x_0 \in X$ and a scheduler $\mathscr{S}$ for t, we consider the set $\mathscr{B}(t,x_0,\mathscr{S})$

of maximal paths, obtained from t by the action of $\mathscr{S}$. Those are the paths $x_0\xi_1a_1x_1\ldots\xi_na_nx_n$ such that $\xi_{i+1} = \mathscr{S}(x_0\xi_1a_1x_1\ldots\xi_ia_ix_i)$. The set of maximal paths is endowed with the σ-algebra $\mathscr{F}$ generated by the finite paths. A scheduler induces a probability measure on $\mathscr{F}$ as follows: for every finite path τ, let $K(\tau)$ be the set of maximal paths extending τ. Define $\zeta_{\mathscr{S}}(K(\tau)) := \Pi(\tau)$, if $\tau \in \mathscr{B}(t, x_0, \mathscr{S})$, and 0 otherwise. It can be proved [25] that $\zeta_{\mathscr{S}}$ extends to a unique probability measure on $\mathscr{F}$.

Given a set of labels $B \subseteq A$ we define $\zeta_{\mathscr{S}}(B)$ to be $\zeta_{\mathscr{S}}(Z)$, where Z is the set of all maximal paths containing some label from B.

3 A probabilistic π-calculus

3.1 Syntax and Operational Semantics

We assume the reader is familiar with the basic definitions of the π-calculus [21]. We consider a restricted version of the π-calculus, where only bound names are passed in interaction. This variant is known as πI-calculus [24]. In the typed setting has the same expressiveness as the full calculus [34]. The labelled transition semantics of the πI-calculus is simpler than that of the full calculus and its labels more naturally corresponds to those of event structures. Syntactically we restrict an output to the form $(\boldsymbol{\nu}\,\tilde{y})\overline{x}\langle\tilde{y}\rangle.P$ (where names in $\tilde{y}$ are pairwise distinct), which we write $\overline{x}(\tilde{y}).P$.

We extend this framework to a probabilistic version of the calculus, where the output is generative, while the input is reactive. Reactive input is similar to the "case" construct and selection is "injection" in the typed λ-calculi. The formal grammar of the calculus is defined below with $p_i \in [0,1]$.

$$P ::= x\,\&_{i\in I}\,\mathtt{in}_i(\tilde{y_i}).P_i \;\mid\; \overline{x}\bigoplus_{i\in I} p_i\,\mathtt{in}_i(\tilde{y_i}).P_i \;\mid\; P \mid Q \;\mid\; (\boldsymbol{\nu}\,x)P \;\mid\; \mathbf{0} \;\mid\; !x(\tilde{y}).P$$

$x\,\&_{i\in I}\,\mathtt{in}_i(\tilde{y_i}).P_i$ is a reactive input, and no probability is attached to its events, $\overline{x}\bigoplus_{i\in I} p_i\mathtt{in}_i(\tilde{y_i}).P_i$ is a generative output, and the events are given probability denoted by the p_i, $P \mid Q$ is a parallel composition, $(\boldsymbol{\nu}\,x)P$ is a restriction and $!x(\tilde{y}).P$ is a replicated input. When the input or output indexing set is a singleton we use the notation $x(\tilde{y}).P$ or $\overline{x}(\tilde{y}).P$; when the indexing set finite, we can write $x(\mathtt{in}_1(\tilde{y_1}).P_1\;\&\;\ldots\;\&\;\mathtt{in}_n(\tilde{y_n}).P_n)$ or $\overline{x}(p_1\mathtt{in}_1(\tilde{y_1}).P_1 \oplus p_n\mathtt{in}_n(\tilde{y_n}).P_n)$. We omit the empty vector and $\mathbf{0}$: for example, $\overline{a}$ stands for $\overline{a}().\mathbf{0}$. The bound/free names are defined as usual. We assume that names in a vector $\tilde{y}$ are pairwise distinct. We use $\equiv_\alpha$ and $\equiv$ for the standard α and structural equivalences [21,15].

The operational semantics is given in terms of Segala automata, using the notation defined in (1) in Section 2. The labels we use are of the following form:

$$\alpha,\beta ::= \quad x\mathtt{in}_i\langle\tilde{y}\rangle \quad\mid\quad \overline{x}\mathtt{in}_i\langle\tilde{y}\rangle \quad\mid x\mathtt{pr}_i\langle\tilde{y}\rangle \mid\; \overline{x}\mathtt{pr}_i\langle\tilde{y}\rangle \quad\mid \qquad \boldsymbol{\tau}$$

(branching) (selection) (offer) (request) (synchronisation)

With the notation above, we say that x is the *subject* of the label β, denoted as $subj(\beta)$, while $\tilde{y} = y_1,\ldots,y_n$ are the *object* names, denoted as $obj(\beta)$. For branching/selection labels, the index i is the *branch* of the label. The notation "$\mathtt{in}_i$" comes from the injection of the typed λ-calculus.

The rules for deriving the transitions are presented in Figure 2. The partial operation $\bullet$ on labels is defined as follows: $x\mathtt{in}_i\langle\tilde{y_i}\rangle \bullet \overline{x}\mathtt{in}_i\langle\tilde{y_i}\rangle = x\mathtt{pr}_i\langle\tilde{y}\rangle \bullet \overline{x}\mathtt{pr}_i\langle\tilde{y}\rangle = \boldsymbol{\tau}$, and

$$\overline{x}\bigoplus_{i\in I} p_i \mathtt{in}_i(\tilde{y}_i).P_i\{\xrightarrow[p_i]{\overline{x}\mathtt{in}_i\langle\tilde{y}_j\rangle} P_i\}_{i\in I} \qquad x\,\&_{i\in I}\,\mathtt{in}_i(\tilde{y}_i).P_i\{\xrightarrow[1]{x\mathtt{in}_j\langle\tilde{y}_j\rangle} P_j\}$$

$$!x(\tilde{y}).P\{\xrightarrow[1]{x\mathtt{pr}_i\langle\tilde{y}\rangle} P \mid !x(\tilde{y}).P\} \qquad \overline{x}(\tilde{y}).P\{\xrightarrow[1]{\overline{x}\mathtt{pr}_i\langle\tilde{y}\rangle} P\}$$

$$\frac{P\{\xrightarrow[p_i]{\beta_i} P_i\}_{i\in I} \quad subj(\beta_i)\neq x}{(\boldsymbol{\nu}\, x)P\{\xrightarrow[p_i]{\beta_i}(\boldsymbol{\nu}\, x)P_i\}_{i\in I}} \qquad \frac{P\{\xrightarrow[p_i]{\beta_i} P_i\}_{i\in I}}{P\mid Q\{\xrightarrow[p_i]{\beta_i} P_i\mid Q\}_{i\in I}}$$

$$\frac{P\{\xrightarrow[p_i]{\alpha_i} P_i\}_{i\in I} \quad Q\{\xrightarrow[1]{\beta_i} Q_i\} \;\; obj(\alpha_i)=\tilde{y}}{P\mid Q\{\xrightarrow[p_i]{\alpha_i\bullet\beta_i}(\boldsymbol{\nu}\,\tilde{y})(P_i\mid Q_i)\}_{i\in I}} \qquad \frac{P\equiv_\alpha P' \quad P\{\xrightarrow[p_i]{\beta_i} Q_i\}_{i\in I}}{P'\{\xrightarrow[p_i]{\beta_i} Q_i\}_{i\in I}}$$

Fig. 2. Segala automaton for the probabilistic πI-Calculus

undefined otherwise. In particular, the generative output synchronises with the reactive input, and a synchronisation step takes place, with the probability chosen by the output process.

3.2 Linear types for the probabilistic π-calculus

This subsection outlines a basic idea of the linear types for a probabilistic π-calculus.

The linear type discipline [3,35] controls a composition of processes in two ways: first, for each linear name there are a unique branching input and a unique selecting output; and secondly, for each replicated name there is a unique stateless replicated input (offer, or server) with zero or more dual outputs (request, or client).

Let us consider the following example where branching and selection provide probabilistic behaviour, preserving linearity:

$$Q_1 \stackrel{\text{def}}{=} \overline{a}.(p\mathtt{in}_1.b \,\oplus\, (1-p)\mathtt{in}_2.c) \mid a.(\mathtt{in}_1.\overline{d}\,\&\,\mathtt{in}_2.\overline{e})$$

Q_1 is typable, and we have either $Q\xrightarrow[p]{\tau}(b\mid\overline{d})$ or $Q\xrightarrow[1-p]{\tau}(c\mid\overline{e})$. The following process is also typable:

$$Q_2 \stackrel{\text{def}}{=} \overline{a}.(p\mathtt{in}_1.b \,\oplus\, (1-p)\mathtt{in}_2.b) \mid a.(\mathtt{in}_1.\overline{d}\,\&\,\mathtt{in}_2.\overline{e})$$

since whichever branch is selected, b is used once. However $\overline{a}.b \mid \overline{a}.c \mid a$ is untypable as linear output $\overline{a}$ appears twice. As an example of the offer-request constraint, let us consider the following process:

$$Q_3 \stackrel{\text{def}}{=} !\,a(x).\overline{x}.(p\mathtt{in}_1 \,\oplus\, (1-p)\mathtt{in}_2) \mid \overline{a}(x)x.(\mathtt{in}_1.\overline{d}\,\&\,\mathtt{in}_2.\overline{e}) \mid \overline{a}(x)x.(\mathtt{in}_1.\overline{f}\,\&\,\mathtt{in}_2.\overline{g})$$

Q_3 is typable since, while output at a appears twice, a replicated input at a appears only once. Note that x under the replication preserves the linearity after each invocation at a. On the other hand, $!\,b.\overline{a} \mid !\,b.\overline{c}$ is untypable because b is associated with two replicators. In the context of deterministic processes, the typing system guarantees confluence, in the presence of nondeterminism it guarantees confusion freeness [30].

Channel types are inductively made up from type variables and action modes: the *input modes* $\downarrow, \mathbf{!}$, and the dual *output modes* $\uparrow, \mathbf{?}$. Then the syntax of types is given as follows:

$$\tau ::= \&_{i\in I}\,(\tilde{\tau}_i)^{\downarrow} \mid \oplus_{i\in I}\,(\tilde{\tau}_i)^{\uparrow} \mid (\tilde{\tau})^{!} \mid (\tilde{\tau})^{?} \mid \updownarrow$$

(branching) (selection) (offer) (request) (closed)

where $\tilde{\tau}$ is a tuple of types.

Branching types represent the notion of "environmental choice": several choices are available for the environment to choose. Selection types represent the notion of "process choice": some choice is made by the process, possibly probabilistically. In both cases the choice is alternative: one excludes all the others. Offer types represent the notion of "available resource": I offer to the environment something that is available regardless of whatever else happens. Request types represent the notion of "concurrent client": I want to use an available resource. The closed type is used to represent a channel that cannot be composed further.

We write $MD(\tau)$ for the outermost mode of τ. The *dual of* τ, written $\overline{\tau}$, is the result of recursively dualising all action modes, with $\updownarrow$ being self-dual. A type environment Γ is a finite mapping from channels to channel types. Sometimes we will write $x \in \Gamma$ to mean $x \in Dom(\Gamma)$.

Types restrict the composability of processes: if P is typed under environment Γ_1, Q is typed under Γ_2 and if Γ_1, Γ_2 are "compatible", then a new environment $\Gamma_1 \odot \Gamma_2$ is defined, such that $P \mid Q$ is typed under $\Gamma_1 \odot \Gamma_2$. If the environments are not compatible, $\Gamma_1 \odot \Gamma_2$ is not defined and the parallel composition cannot be typed. Formally, we introduce a partial commutative operation $\odot$ on types, defined as follows [2] :

(i) $\&_{i\in I}\,(\tau_i)^{\downarrow} \odot \oplus_{i\in I}\,(\overline{\tau_i})^{\uparrow} = \updownarrow$
(ii) $(\tau)^{?} \odot (\overline{\tau})^{!} = (\overline{\tau})^{!}$
(iii) $(\tau)^{?} \odot (\tau)^{?} = (\tau)^{?}$

Then, the environment $\Gamma_1 \odot \Gamma_2$ is defined homomorphically. Intuitively, the rule (i) says that once we compose input-output linear channels, the channel becomes uncomposable. The rule (ii) says that a server should be unique while rule (iii) says that an arbitrary number of clients can request interactions. Other compositions are undefined.

The rules defining typing judgments $P \triangleright \Gamma$ (where Γ is an environment which maps a channel to a type) are identical to the affine π-calculus [3] except a straightforward modification to deal with the generative output, which is defined by the same rule for confusion free processes in [30], without any additional complexity due to the probability. The rules are presented in Figure 3. In (Par), $\Gamma_1 \odot \Gamma_2$ guarantees the consistent channel usage like linear inputs being only composed with linear outputs, etc. In (Res), we do not allow $\uparrow$, $?$ or $\downarrow$-channels to be restricted since they carry actions which expect their dual actions to exist in the environment. (WeakOut) and (WeakCl) weaken with $?$-names or $\updownarrow$-names, respectively, since these modes do not require *further* interaction. (LIn) and (LOut) ensure that x occurs precisely once. (RIn) is the same as (LIn) except that no free linear channels are suppressed. This is because a linear channel under replication could be used more than once. (ROut) is similar with (LOut). Note we need to apply (WeakOut) before the first application of (ROut).

We then obtain a typed version of the operational semantics by restricting the actions that are not allowed by the type environment. Informally an action is allowed by an envi-

[2] to simplify the notation we omit the ˜ that denotes polyadicity

$$\frac{P \triangleright \Gamma, a:\tau \quad a \notin \Gamma \quad MD(\tau) = \mathbf{!}, \updownarrow}{(\boldsymbol{\nu}\, a)P \triangleright \Gamma}\ \text{Res} \qquad \frac{}{\mathbf{0} \triangleright \emptyset}\ \text{Zero} \qquad \frac{P \triangleright \Gamma \quad x \notin \Gamma}{P \triangleright \Gamma, x:\updownarrow}\ \text{WeakCl}$$

$$\frac{P_i \triangleright \Gamma, \tilde{y}_i:\tilde{\tau}_i \quad a \notin \Gamma}{\overline{a} \bigoplus_{i\in I} p_i \mathtt{in}_i(\tilde{y}_i).P_i \triangleright \Gamma, a:\bigoplus_{i\in I}(\tilde{\tau}_i)^{\uparrow}}\ \text{LOut} \qquad \frac{P \triangleright \Gamma \quad x \notin \Gamma}{P \triangleright \Gamma, x:(\tilde{\tau})^{?}}\ \text{WeakOut}$$

$$\frac{P_i \triangleright \Gamma, \tilde{y}_i:\tilde{\tau}_i \quad a \notin \Gamma}{a \,\&_{i\in I}\, \mathtt{in}_i(\tilde{y}_i).P_i \triangleright \Gamma, a:\&_{i\in I}(\tilde{\tau}_i)^{\downarrow}}\ \text{LIn} \qquad \frac{P_i \triangleright \Gamma_i \quad (i=1,2)}{P_1 \mid P_2 \triangleright \Gamma_1 \odot \Gamma_2}\ \text{Par}$$

$$\frac{P \triangleright \Gamma, \tilde{y}:\tilde{\tau} \quad a \notin \Gamma \quad \forall (x:\tau) \in \Gamma.\ MD(\tau) =?}{!a(\tilde{y}).P \triangleright \Gamma, a:(\tilde{\tau})^{!}}\ \text{RIn} \qquad \frac{P \triangleright \Gamma, a:(\tilde{\tau})^{?}, \tilde{y}:\tilde{\tau}}{\overline{a}(\tilde{y}).P \triangleright \Gamma, a:(\tilde{\tau})^{?}}\ \text{ROut}$$

Fig. 3. Linear Typing Rules

ronment if the subject of the action has a branching, selection or server type. The formal definition can be found in [29]. For example, the output transition at a in $\overline{a} \mid a.\mathbf{0}$ is not allowed since a is linear so that $\overline{a}$ is assumed to interact with only $a.\mathbf{0}$, not with the external observer. The typed automaton, $P \triangleright \Gamma\{\xrightarrow[p_i]{\beta_i} P_i \triangleright \Gamma_i\}_{i\in I}$, is defined by adding the constraint:
if $P\{\xrightarrow[p_i]{\beta_i} P_i\}_{i\in I}$ and Γ allows β_i for all $i \in I$ then $P \triangleright \Gamma\{\xrightarrow[p_i]{\beta_i} P_i \triangleright \Gamma_i\}_{i\in I}$ The nature of the typing system is such that for every transition group, either all actions are allowed, or all are not, and therefore the above semantics is well defined.

3.3 Example of a probabilistic process

We consider the model of traffic lights from [22]. Let a be a driver, and let $\mathtt{in}_{\mathtt{red}}, \mathtt{in}_{\mathtt{yell}}, \mathtt{in}_{\mathtt{green}}$ represent colours of the traffic light. The process $\overline{a}\mathtt{in}_{\mathtt{red}}(y)$ represents the traffic light signalling to the driver it is red, at the same time communicating the name y of the crossing. The behaviour of the driver at the crossing is either braking, staying still, or driving ($\mathtt{in}_{\mathtt{brake}}, \mathtt{in}_{\mathtt{still}}, \mathtt{in}_{\mathtt{drive}}$).

A cautious driver is represented by the process:

$$D_c^a = a \,\&_{i\in\{\mathtt{red},\mathtt{yell},\mathtt{green}\}}\, \mathtt{in}_i(y).P_i \quad \text{with} \quad \begin{array}{l} P_{\mathtt{red}} = \overline{y}(0.2\mathtt{in}_{\mathtt{brake}} \oplus 0.8\mathtt{in}_{\mathtt{still}}) \\ P_{\mathtt{yell}} = \overline{y}(0.9\mathtt{in}_{\mathtt{brake}} \oplus 0.1\mathtt{in}_{\mathtt{drive}}) \\ P_{\mathtt{green}} = \overline{y}(\mathtt{in}_{\mathtt{drive}}) \end{array}$$

A cautious driver watches what colour the light is and behaves accordingly. If it is red, she stays still, or finishes braking. If it is yellow, most likely she brakes. If it is green, she drives on.

A driver in a hurry is represented by the process

$$D_h^a = a\,\&_{i\in\{\mathtt{red},\mathtt{yell},\mathtt{green}\}}\,\mathtt{in}_i(y).Q_i \quad \text{with} \quad \begin{array}{l} Q_{\mathtt{red}} = \overline{y}(0.3\mathtt{in}_{\mathtt{brake}} \oplus 0.6\mathtt{in}_{\mathtt{still}} \oplus 0.1\mathtt{in}_{\mathtt{drive}}) \\ Q_{\mathtt{yell}} = \overline{y}(0.1\mathtt{in}_{\mathtt{brake}} \oplus 0.9\mathtt{in}_{\mathtt{drive}}) \\ Q_{\mathtt{green}} = \overline{y}(\mathtt{in}_{\mathtt{drive}}) \end{array}$$

This is similar to the cautious driver, but he is more likely to drive on at red and yellow. In fact, both have the same *type*, they check the light, and they choose a behaviour:

$$D_c^a, D_h^a \triangleright a : \&_{i\in\{\mathtt{red},\mathtt{yell},\mathtt{green}\}}\,(\textstyle\bigoplus_{j\in\{\mathtt{brake},\mathtt{still},\mathtt{drive}\}}()^{\uparrow})^{\downarrow}$$

where $\&_{i\in I}\,(\tau_i)^{\downarrow}$ is a branching type which inputs a value of type τ_i and $\bigoplus_{i\in I}\,(\tau_i)^{\uparrow}$ is a selection type which selects a branch i with a value of type τ_i. Note that the type actually states that the driver chooses the behaviour *after* seeing the light. We can represent two independent drivers:

$$D2 = (\nu a, a')(\overline{a}\mathtt{in}_{\mathtt{red}}(y).R \mid D_c^a \mid \overline{a'}\mathtt{in}_{\mathtt{green}}(y).R \mid D_h^{a'})$$

where $R = y\,\&_{i\in\{\mathtt{brake},\mathtt{still},\mathtt{drive}\}}\,\mathtt{in}_i()$ represents the traffic light accepting the behaviour of the driver. We have that $D2$ has two transition groups, corresponding to the two drivers. Note that linearity and confusion-freedom of y guarantees that each driver can perform only one of three actions, i.e. either `brake`, `still` or `drive` at any one time.

4 Probabilistic event structures

We now present the model of probabilistic event structures, that we use to give an alternative semantics to the probabilistic π-calculus. Probabilistic event structures were first introduced by Katoen [18], as an extension of the so called *bundle* event structure. A probabilistic version of *prime* event structures was introduced in [28]. Below we start from basic definitions without probability.

4.1 Basic definitions

An *event structure* is a triple $\mathscr{E} = \langle E, \leq, \smile\rangle$ such that

- E is a countable set of *events*;
- $\langle E, \leq\rangle$ is a partial order, called the *causal order*;
- for every $e \in E$, the set $[e) := \{e' \mid e' < e\}$, called the *enabling set* of e, is finite;
- $\smile$ is an irreflexive and symmetric relation, called the *conflict relation*, satisfying the following: for every $e_1, e_2, e_3 \in E$ if $e_1 \leq e_2$ and $e_1 \smile e_3$ then $e_2 \smile e_3$.

The reflexive closure of conflict is denoted by $\asymp$. We say that the conflict $e_2 \smile e_3$ is *inherited* from the conflict $e_1 \smile e_3$, when $e_1 < e_2$. If a conflict $e_1 \smile e_2$ is not inherited from any other conflict we say that it is *immediate*, denoted by $e_1 \smile_\mu e_2$. The reflexive closure of immediate conflict is denoted by $\asymp_\mu$. Causal dependence and conflict are mutually exclusive. If two events are not causally dependent nor in conflict they are said to be *concurrent*. A *labelled event structure* is an event structure $\mathscr{E}$ together with a labelling function $\lambda : E \to L$, where L is a set of labels.

We introduce an interesting class of event structures where every choice is *localised*. To specify what "local" means that we need the notion of *cell*, a set of events that are pairwise in immediate conflict and have the same enabling sets.

Definition 4.1 A *partial cell* is a set c of events such that $e, e' \in c$ implies $e \asymp_\mu e'$ and $[e) = [e')$. A maximal partial cell is called a *cell*. An event structure is *confusion free* if its cells are closed under immediate conflict.

Equivalently, in a confusion free event structure, the reflexive closure of immediate conflict is an equivalence with cells being its equivalence classes.

4.2 Probabilistic event structures

Once an event structure is confusion-free, we can associate a probability distribution with some cells. Intuitively it is as if, for every such cell, we have a die local to it, determining the probability with which the events at that cell occur.

We can think of the cells with a probability distribution as *generative*, while the other cells will be called *reactive*. Reactive cells are awaiting a synchronisation with a generative cell in order to be assigned a probability.

Definition 4.2 Let $\mathscr{E} = \langle E, \leq, \smile \rangle$ be a confusion free event structure, let G be a set of cells of $\mathscr{E}$ and let G' be the set of events of the cells in G. The cells in G are called *generative*. The cells not in G are called *reactive*. A *cell valuation* on $(\mathscr{E}, G)$ is a function $p : G' \to [0, 1]$ such that for every $c \in G$, we have $\sum_{e \in c} p(e) = 1$. A *partial probabilistic event structure* is a confusion free event structure together with a cell valuation. It is called simply *probabilistic event structure* if $G' = E$.

This definition generalises the definition given in [28], where it is assumed that $G' = E$. Note also that a confusion free event structure can be seen as a probabilistic event structure where the set G is empty.

4.3 Operators on event structures

Several operations can be defined on event structures.

- prefixing $a.\mathscr{E}$. This is obtained by adding a new minimum event, labelled by a. Conflict, order, and labels remain the same on the old events.
- prefixed sum $\sum_{i \in I} a_i.\mathscr{E}_i$. This is obtained by disjoint union of copies of the event structures $a_i.\mathscr{E}_i$, where the order relation is the disjoint union of the orders, the labelling function is the disjoint union of the labelling functions, and the conflict is the disjoint union of the conflicts extended by putting in conflict every two events in two different copies. It is a generalisation of prefixing, where we add an initial *reactive cell*, instead of an initial event.
- probabilistic prefixed sum $\sum_{i \in I} p_i a_i.\mathscr{E}_i$, where $\mathscr{E}_i$ are partial probabilistic event structures. This is obtained as above, but with the condition that the initial cell is generative, and that the probability of the new initial events are p_i.
- restriction $\mathscr{E} \setminus X$ where $X \subseteq A$ is a set of labels. This is obtained by removing from E all events with label in X and all events that are above one of those. On the remaining events, order, conflict and labelling are unchanged.

- relabelling $\mathscr{E}[f]$. This is just composing the labelling function λ with a function $f : L \to L$. The new event structure has thus labelling function $f \circ \lambda$.
- parallel composition The parallel composition of event structures is not so simple to define, due to the possibility of synchronisation among events. For lack of space we skip the details, that can be found in [33,29,30].

Intuitively, events in the parallel composition are the events of the two event structures, plus some new event representing synchronisation. For a labelled event structures with labels in L, the labels of the synchronisation events are obtained via a *synchronisation algebra* S, a partial binary operation $\bullet_S$ defined on L. If the labels of the two synchronising event are l_1, l_2, the syncronisation event will have label $l_1 \bullet_S l_2$, if defined, orelse it will be restricted away. The simplest synchronsation algebra is always undefined and represents the absence of synchronisation. In this case the parallel composition can be represented as the disjoint union of the sets of events, of the causal orders, and of the conflict. This can be also generalised to an arbitrary family of event structures $(\mathscr{E}_i)_{i\in I}$. In such a case we denote the parallel composition as $\prod_{i\in I} \mathscr{E}_i$.

All constructors above, except the parallel composition, preserve the class of partial probabilistic event structures. In the next section we present a typing system, which is designed to allow parallel composition to preserve that class.

4.4 Typed event structures

In this section we recall the notion of type for an event structure, which was defined in [29]. Types and type environments for event structures are inspired by those of the π-calculus, but they recursively keep track of the names communicated along the channels. They are generated by the following grammar:

$$\Gamma, \Delta ::= \quad y_1 : \sigma_1, \ldots, y_n : \sigma_n$$

$$\tau, \sigma ::= \quad \&_{i\in I}\, \Gamma_i \quad | \quad \bigoplus_{i\in I} \Gamma_i \quad | \bigotimes_{i\in I} \Gamma_i \; | \; \biguplus_{i\in I} \Gamma_i \quad | \qquad \updownarrow$$

(branching) (selection) (offer) (request) (closed type)

A type environment Γ is *well formed* if any name appears at most once. Only well formed environments are considered for typing event structures. The intuition behind the types is similar to the π-calculus. The main difference is that offer is not restricted to a replicated server, but represents different concurrent resources.

Given a labelled confusion free event structure $\mathscr{E}$ on π-calculus labels (defined in Section 3), we can define when $\mathscr{E}$ is typed in the environment Γ, written as $\mathscr{E} \triangleright \Gamma$. Informally, a confusion free event structure $\mathscr{E}$ has type Γ if cells are partitioned in branching, selection, request, offer and synchronisation cells, all the non-synchronisation events of $\mathscr{E}$ are represented in Γ and causality in $\mathscr{E}$ refines the name causality implicit in Γ. This means that if name y appears inside the type of a name x, any event whose subject is y must be causally related with en event whose subject is x.

The types are designed so that the parallel composition of typed event structures will also be typed. To define the parallel composition, we use the following same synchronisation algebra used in Section 3: $x\mathtt{in}_i\langle\tilde{y}_i\rangle \bullet \overline{x}\mathtt{in}_i\langle\tilde{y}_i\rangle = x\mathtt{pr}_i\langle\tilde{y}\rangle \bullet \overline{x}\mathtt{pr}_i\langle\tilde{y}\rangle = \boldsymbol{\tau}$, and undefined otherwise. Moreover, the parallel composition of two typed event structures $E_1 \triangleright \Gamma_1$ and

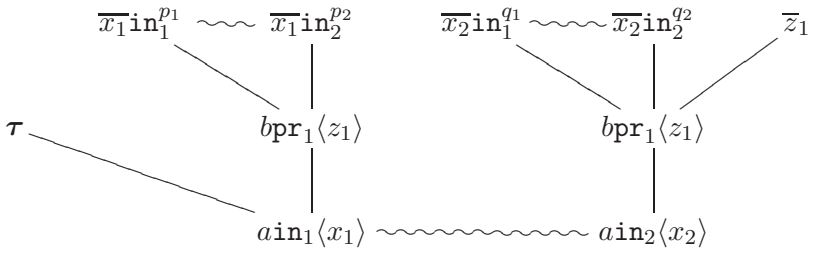


Fig. 4. A typed event structure

$E_2 \triangleright \Gamma_2$ is defined only when the environment $\Gamma_1 \odot \Gamma_2$ is defined, and in such a case the parallel composition has type $\Gamma_1 \odot \Gamma_2$ The formal definition of $\odot$ is similar to the corresponding notion for the π-calculus, but it is recursively applied to the object names. It is designed to preserve the well formedness (linearity) of the environment. The details can be found in [29,30].

To type a partial probabilistic event structure, we type it as a non probabilistic event structure. We also make sure that only the branching cells are reactive, as they are waiting to synchronise with a dual selection cell.

Definition 4.3 Let $\mathscr{E} = \langle E, \leq\!\!\smile\ \lambda, G, p\rangle$ be a partial probabilistic event structure. We say that $\mathscr{E} \triangleright \Gamma$, if the following conditions are satisfied:

- $\mathscr{E} \triangleright \Gamma$ as for the non-probabilistic case;
- G includes all cells, except the branching ones.

From the fact that the parallel composition of typed event structures is typed, one can easily derive that the parallel composition of typed probabilistic event structures [29] is still a probabilistic event structure, and that it is typed.

4.5 Example of typed event structure

Figure 4 represents a typed (partial) probabilistic event structure $\mathscr{E} \triangleright \Gamma$, where

$$\Gamma = a : \&_{i\in\{1,2\}}(x_i : \oplus_{k\in\{1,2\}}()),\ b : \otimes_{i\in\{1\}}(z_i : \uplus_{k\in\{1\}}())$$

Immediate conflict is represented by curly lines, while causal order proceeds upwards along the straight lines. The selection cells $\overline{x_1}\mathtt{in}_1, \overline{x_1}\mathtt{in}_2$ and $\overline{x_2}\mathtt{in}_1, \overline{x_2}\mathtt{in}_2$ are generative. The branching cell $a\mathtt{in}_1\langle x_1\rangle, a\mathtt{in}_2\langle x_2\rangle$ is reactive. Every other cell is generative, and contains only one event, that has probability 1. We can see that the causality in $\mathscr{E}$ refines the name causality in Γ: for instance, Γ forces the labels with subject x_i to be above the label $a\mathtt{in}_i\langle x_i\rangle$, but does not force the causal link between the events labelled by $a\mathtt{in}_i\langle x_i\rangle$ and $b\langle z_1\rangle$. Note also that the synchronisation event is not represented in the type.

5 Event structure semantics of the probabilistic π-calculus

This section presents the event structure semantics of the π-calculus and its properties As in [30], the semantics is given by a family of partial functions $[\![-]\!]^\rho$, parametrised by a

"choice function" ρ, that take a judgment of the π-calculus and return an event structure. The "choice function" ρ assigns to every bound name a set (possibly a singleton) of fresh distinct names. The parametrisation is necessary because π-calculus terms are identified up to α-conversion, and so the identity of bound names is irrelevant, while in typed event structures, the identity of the object names is important. Also, since servers are interpreted as infinite parallel compositions, every bound name of a server must correspond to infinitely many names in the interpretation.

The semantics is defined as in the non probabilistic case [29]. As an example, we list the semantics of the selection:

$$[\![\overline{a}\bigoplus_{i\in I} p_i\mathtt{in}_i(\tilde{y}_i).P_i \triangleright \Gamma, a : \bigoplus_{i\in I}(\tilde{\tau}_i)]\!]^\rho = \sum_{i\in I} p_i\overline{a}\mathtt{in}_i\langle\tilde{z}_i\rangle.[\![P_i[\tilde{z}_i/\tilde{y}_i] \triangleright \Gamma, \tilde{z}_i : \tilde{\tau}_i]\!]^{\rho_i}$$

where $\tilde{z}_i = \rho(\tilde{y}_i)$ and ρ_i is ρ restricted to to the bound names of P_i. We can see that syntactic prefix is modelled using the prefixing operator of event structures, while the parameter ρ chooses an instantiation of the bound names.

The main property of the typed semantics is that all denoted event structures are typed.

Theorem 5.1 *Let P be a process and Γ an environment such that $P \triangleright \Gamma$. Suppose that $[\![P \triangleright \Gamma]\!]^\rho$ is defined. Then there is a environment Δ such that $[\![P \triangleright \Gamma]\!]^\rho \triangleright \Delta$.*

This theorem means that all denoted event structures are indeed partial probabilistic event structures. Note that the set of generative cells includes all synchronisation cells. Therefore a closed process denotes a probabilistic event structure.

Corollary 5.2 *The event structure $[\![P \triangleright \emptyset]\!]^\rho$ is a probabilistic event structure.*

This implies that there exists a unique probability measure over the set of maximal runs [28]. In other words, for closed processes, the scheduler only influences the order of independent events, in accordance with the intuition that probabilistic choice are local and not influenced by the environment.

6 Event structures and Segala automata

In this section we show a formal correspondence between Segala automata and probabilistic event structures.

6.1 From event structures to Segala automata

Definition 6.1 Let $\mathscr{E} = \langle E, \leq, \smile, \lambda\rangle$ be a labelled event structure and let e be one of its minimal events. The event structure $\mathscr{E}\lfloor e = \langle E', \leq', \smile', \lambda'\rangle$ is defined by: $E' = \{e' \in E \mid e' \not\asymp e\}$, $\leq' = \leq_{|E'}$, $\smile' = \smile_{|E'}$, and $\lambda' = \lambda_{E'}$.

Roughly speaking, $\mathscr{E}\lfloor e$ is $\mathscr{E}$ minus the event e, and minus all events that are in conflict with e. We can then generate a Segala automata on event structures as follows:

$$\mathscr{E}\{\xrightarrow[p_i]{a_i}\mathscr{E}\lfloor e_i\}_{i\in I}$$

if there exists a minimal generative cell $c = \{e_i \mid i \in I\}$, such that $p(e_i) = p_i$ and $\lambda(e_i) = a_i$. We also put

$$\mathscr{E}\{\xrightarrow[1]{a}\mathscr{E}\lfloor e\}$$

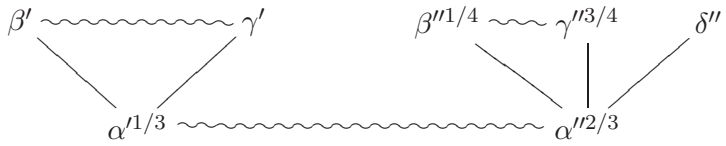


Fig. 5. A probabilistic event structure

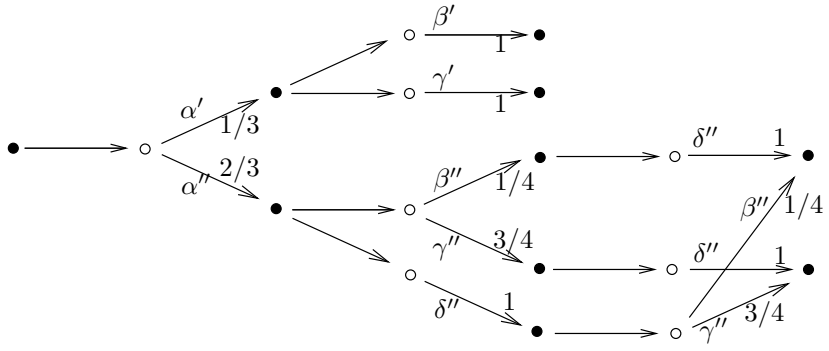


Fig. 6. The corresponding Segala automaton

if there exists an event e belonging to a minimal reactive cell, such that $\lambda(e) = a$. The initialised Segala automaton generated by an event structure $\mathscr{E}$ is the above automation intialised at $\mathscr{E}$.

A probabilistic event structure (where every cell is generative) generates a somewhat "deterministic" Segala automaton. The general formalisation of this property requires several technicalities (see [28], for instance). Here we state a simplified result. Consider a scheduler $\mathscr{S}$ for a Segala automaton (t, x_0). We say that $\mathscr{S}$ is *fair* if for every path $\tau \in \mathscr{B}(t, x_0, \mathscr{S})$, there does not exist a transition group ξ and an index j, such that $\xi \in t(x_i)$ for all $i > j$, and ξ is never chosen by $\mathscr{S}$.

Theorem 6.2 *Let $\mathscr{E}$ be a probabilistic event structure, and consider the Segala automaton generated as above. For every set of labels B, and for every fair schedulers $\mathscr{S}, \mathscr{T}$, we have $\zeta_{\mathscr{S}}(B) = \zeta_{\mathscr{T}}(B)$.*

In a non-probabilistic confluent system, all (fair) resolutions of the nondeterministic choices give rise to the same set of events, possibly in different order. In this sense we can see Theorem 6.2 as expressing probabilistic confluence.

Figure 5 shows an example of a (partial) probabilistic event structure. The generative cells are $\{\alpha', \alpha''\}, \{\beta'', \gamma''\}$ and the probability is indicated as superscript of the label. Figure 6 shows the Segala automaton corresponding to the event structure of Figure 5.

6.2 The adequacy theorem

There is a correspondence between the two semantics and of the π-calculus. It is formalised by the following theorem, which shows the correspondence between the Segala automata semantics defined in Section 3, and the Segala automaton derived from the event structure semantics, as described above.

Theorem 6.3 *Let $\cong$ denote isomorphism of probabilistic event structures.*

Suppose $P \triangleright \Gamma\{\xrightarrow[p_i]{\beta_i} P_i \triangleright \Gamma_i\}_{i\in I}$ in the π-calculus. Then there exists ρ, ρ_i such that $[\![P \triangleright \Gamma]\!]^\rho$ is defined and $[\![P \triangleright \Gamma]\!]^\rho\{\xrightarrow[p_i]{\beta_i} \cong [\![P_i \triangleright \Gamma_i]\!]^{\rho_i}\}_{i\in I}$.

Conversely, suppose $[\![P \triangleright \Gamma]\!]^\rho\{\xrightarrow[p_i]{\beta_i} \mathscr{E}_i\}_{i\in I}$, for some ρ. Then there exist P_i, ρ_i such that $P \triangleright \Gamma\{\xrightarrow[p_i]{\beta_i} P_i \triangleright \Gamma \setminus \beta_i\}_{i\in I}$ and $[\![P_i \triangleright \Gamma \setminus \beta_i]\!]^{\rho_i} \cong \mathscr{E}_i$, for all $i \in I$.

The proof is analogous to the one for the non-probabilistic case [29] by induction on the operational rules, the difficult case being the parallel composition.

6.3 Example of probabilistic confluence

Theorem 6.3 and Theorem 6.2 together show that the linearly typed probabilistic π-calculus is "probabilistically confluent".

To exemplify this, consider a process P such that $P \triangleright a : \bigoplus_{i\in I}$. This is a process that emits only one visible action, whose subject is a. For every $j \in I$ we can define the probability P emits $a\mathtt{in}_j$ as $p_{\mathscr{S}}(a\mathtt{in}_j)$ for some fair scheduler $\mathscr{S}$. By Theorem 6.2, we have that this probability is independent from the scheduler, so we can define it as $p(a\mathtt{in}_j)$. This independence from the scheduling policy is what we call probabilistic confluence.

Note also that it can be shown that $\sum_{i\in I} p(a\mathtt{in}_i) \leq 1$. When the inequation is strict, the missing probability is the probability that the process does not terminate. This reasoning rely on the typing in that there exist untyped processes that are not probabilistically confluent. For instance consider

$$(\nu b)(\overline{b} \mid b.\overline{a} \bigoplus_{i\in\{1,2\}} p_i\mathtt{in}_i \mid b.\overline{a} \bigoplus_{i\in\{1,2\}} q_i\mathtt{in}_i)$$

The above process also emits only one visible action, whose subject is a. The probability of $\overline{a}\mathtt{in}_1$ is p_1, or q_1, depending on which synchronisation takes place, i.e. depending on the scheduler. Note, however, that this process is not typable.

7 Related and future work

7.1 Related work

This paper has provided an event structures semantics for a probabilistic version of the π-calculus. It is the first true concurrent semantics of a probabilistic π-calculus. Related work with true concurrency models for the π-calculus and (confusion-free) event structures are already discussed in [30,29]. There, the importance of confusion freeness and the use of types in event structures is also discussed in depth. Another recent event structure semantics of the π-calculus was presented in [5].

The natural comparison is with the probabilistic π-calculus by Herescu and Palamidessi [14]. Their and our calculi both have a semantics in terms of Segala automata, while we also provide an event structure semantics. The key of our construction is the typing system, which allows us to stay within the class of probabilistic event structures.

Our typing system is designed to provide a "probabilistically confluent" calculus, and therefore their calculus is more expressive, as it allows non-confluent computations. At the core of their calculus, there is a renormalisation of probabilities, which is absent in our

setting, i.e. in our calculus, all probabilistic choices are local, and are not influenced by the environment.

A simpler calculus, without renormalisation, is presented in [6]. This version is very similar to ours, in that all choices are local; in fact, the protocol example presented in [6] (via an encoding into our calculus) is linearly typable. We believe we could apply a typing system similar to ours to the calculus in [6], prove the same results in this paper and identify a good class of probabilistic name-passing behaviours.

7.2 Future work

We have shown a correspondence between event structures and Segala automata. We would have liked to extend this correspondence to a categorical adjunction between two suitable categories, ideally extending the setting presented in [33]. It is possible to do so, by a simple definition of morphisms for Segala automata, and by extending the notion of probabilistic event structures. Unfortunately neither category has products, which are used in [33] to define parallel composition. The reason for this is nontrivial and it is has to do with the notion of stochastic correlation, a phenomenon already discussed in [28] in the context of true concurrent models. This issue needs to be investigated further.

The linearly typed π-calculus is the target of a sound and complete encodings of functional language [3,34]. Our traffic light example in Section 5 suggests that our calculus captures the core part of the expressiveness represented by the Stochastic Lambda Calculus [22]. We plan to perform similar encodings in the probabilistic version, notably the probabilistic functional language [22], probabilistic λ-calculus [10] and Probabilistic PCF [7]. Since the linear type structures are originated from game semantics [16], this line of study would lead to a precise expressive analysis between the probabilistic event structures, Segala automata, probabilistic programming languages and probabilistic game semantics [7], bridged by their representations of or encodings into probabilistic π-calculi. Finally, there are connections between event structures, concurrent game [20], and ludics [11,12] that should be investigated also in the presence of probabilities.

References

[1] Samy Abbes and Albert Benveniste. Probabilistic models for true-concurrency: branching cells and distributed probabilities for event structures. *Information and Computation*, 204(2):231–274, 2006.

[2] Falk Bartels, Ana Sokolova, and Erik de Vink. A hierarchy of probabilistic system types. In *Electronic Notes in Theoretical Computer Science*, volume 82. Elsevier, 2003.

[3] Martin Berger, Kohei Honda, and Nobuko Yoshida. Sequentiality and the π-calculus. In *Procceedingss of TLCA'01*, volume 2044 of *LNCS*, pages 29–45, 2001.

[4] Gérard Berry and Pierre-Louis Curien. Sequential algorithms on concrete data structures. *Theoretical Computer Science*, 20(265–321), 1982.

[5] Roberto Bruni, Hernán Melgratti, and Ugo Montanari. Event structure semantics for nominal calculi. In *Proceedings of 17th CONCUR*, volume 4137 of *LNCS*, pages 295–309. Springer, 2006.

[6] Kostas Chatzikokolakis and Catuscia Palamidessi. A framework to analyze probabilistic protocols and its application to the partial secrets exchange. In *Proceedings of Symposium on Trustworthy Global Computing, 2005*, volume 3705 of *LNCS*. Springer, 2005.

[7] Vincent Danos and Russell S. Harmer. Probabilistic game semantics. *ACM Transactions on Computational Logic*, 3(3):359–382, 2002.

[8] Jörg Desel and Javier Esparza. *Free Choice Petri Nets*. Cambridge University Press, 1995.

[9] Josée Desharnais, Abbas Edalat, and Prakash Panangaden. Bisimulation for labelled markov processes. *Information and Computation*, 179(2):163–193, 2002.

[10] Alessandra Di Pierro, Chris Hankin, and Herbert Wiklicky. Probabilistic lambda calculus and quantitative program analysis. *Journal of Logic and Computation*, 2005. To appear.

[11] Claudia Faggian and François Maurel. Ludics nets, a game model of concurrent interaction. In *Proceedings of 20th LICS*, pages 376–385, 2005.

[12] Claudia Faggian and Mauro Piccolo. A graph abstract machine describing event structure composition. In *Proceedings of the GT-VC workshop*, 2006. Short paper.

[13] Hans Hansson. *Time and Probability in Formal Design of Distributed systems*. PhD thesis, Uppsala University, 1991.

[14] Mihaela Herescu and Catuscia Palamidessi. Probabilistic asynchronous π-calculus. In *Proceedings of 3rd FoSSaCS*, volume 1784 of *LNCS*, pages 146–160. Springer, 2000.

[15] Kohei Honda and Nobuko Yoshida. On reduction-based process semantics. *Theoretical Computer Science*, 151(2):385–435, 1995.

[16] J. Martin E. Hyland and C.-H. Luke Ong. On full abstraction for PCF: I, II, and III. *Information and Computation*, 163(2):285–408, 2000.

[17] Gilles Kahn and Gordon D. Plotkin. Concrete domains. *Theoretical Computer Science*, 121(1-2):187–277, 1993.

[18] Joost-Pieter Katoen. *Quantitative and Qualitative Extensions of Event Structures*. PhD thesis, University of Twente, 1996.

[19] Kim G. Larsen and Arne Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94(1):1–28, 1991.

[20] Paul-André Melliès. Asynchronous games 2: The true concurrency of innocence. In *Proceedings of 15th CONCUR*, pages 448–465, 2004.

[21] Robin Milner. *Communicating and Mobile Systems: The Pi Calculus*. Cambridge University Press, 1999.

[22] Norman Ramsey and Avi Pfeffer. Stochastic lambda calculus and monads of probability distributions. In *Proceedings of 29th POPL*, pages 154–165, 2002.

[23] Grzegorz Rozenberg and P.S. Thiagarajan. Petri nets: Basic notions, structure, behaviour. In *Current Trends in Concurrency*, volume 224 of *LNCS*, pages 585–668. Springer, 1986.

[24] Davide Sangiorgi. π-calculus, internal mobility and agent passing calculi. *Theoretical Computer Science*, 167(2):235–271, 1996.

[25] Roberto Segala. *Modeling and Verification of Randomized Distributed Real-Time Systems*. PhD thesis, M.I.T., 1995.

[26] Roberto Segala and Nancy Lynch. Probabilistic simulations for probabilistic processes. *Nordic Journal of Computing*, 2(2):250–273, 1995. An extended abstract appears in *Proceedings of 5th CONCUR*, Uppsala, Sweden, LNCS 836, pages 481–496, August 1994.

[27] Mariëlle Stoelinga. An introduction to probabilistic automata. *Bulletin of the European Association for Theoretical Computer Science*, 78:176–198, 2002.

[28] Daniele Varacca, Hagen Völzer, and Glynn Winskel. Probabilistic event structures and domains. *TCS*, 358(2-3):173–199, 2006. Full version of the homonymous paper in CONCUR 2004.

[29] Daniele Varacca and Nobuko Yoshida. Event structures, types and the π-calculus. Technical Report 2005/06, Imperial College London, 2005. Available at `www.doc.ic.ac.uk/~varacca`.

[30] Daniele Varacca and Nobuko Yoshida. Typed event structures and the π-calculus. In *Proceedings of XXII MFPS*, ENTCS, 2006.

[31] Hagen Völzer. Randomized non-sequential processes. In *Proceedings of 12th CONCUR*, volume 2154 of *LNCS*, pages 184–201, 2001. Extended version as Technical Report 02-28 - SVRC - University of Queensland.

[32] Glynn Winskel. Event structure semantics for CCS and related languages. In *Proceedings of 9th ICALP*, volume 140 of *LNCS*, pages 561–576. Springer, 1982.

[33] Glynn Winskel and Mogens Nielsen. Models for concurrency. In *Handbook of logic in Computer Science*, volume 4. Clarendon Press, 1995.

[34] Nobuko Yoshida, Martin Berger, and Kohei Honda. Strong Normalisation in the π-Calculus. In *Proceedings of LICS'01*, pages 311–322. IEEE, 2001. The full version in *Journal of Inf. & Comp.*., 191 (2004) 145–202, Elsevier.

[35] Nobuko Yoshida, Kohei Honda, and Martin Berger. Linearity and bisimulation. In *FoSSaCs02*, volume 2303 of *LNCS*, pages 417–433. Springer, 2002.