Actor Network Procedures as Psi-calculi for Security Ceremonies *

Cristian Prisacariu †
Dept. of Informatics, University of Oslo, – P.O. Box 1080 Blindern, N-0316 Oslo, Norway.
cristi@ifi.uio.no

The *actor network procedures* of Pavlovic and Meadows are a recent graphical formalism developed for describing security ceremonies and for reasoning about their security properties. The present work studies the relations of the actor network procedures (ANP) to the recent psi-calculi framework. Psi-calculi is a parametric formalism where calculi like spi- or applied-pi are found as instances. Psi-calculi are operational and largely non-graphical, but have strong foundation based on the theory of nominal sets and process algebras. One purpose of the present work is to give a semantics to ANP through psi-calculi. Another aim was to give a graphical language for a psi-calculus instance for security ceremonies. At the same time, this work provides more insight into the details of the ANPs formalization and the graphical representation.

1 Introduction

Actor Network Procedures (ANP) is a formalism introduced in [33] for modeling security ceremonies [12, 38]. Reasoning about security properties of ceremonies is done using the Procedure Derivation Logic, which comes from a line of research on logics for composition of protocols that started with the Protocol Composition Logic [11, 8]. This formalism that we concentrate on in this paper should not be confused with the work with similar purposes from [35]. Both these approaches [35, 33] aim to be used for describing security ceremonies by drawing inspiration from the actor network theory in sociology, where the book [25] gives a good overview.

Security ceremonies are well motivated in [12, 38] with convincing examples. Technically, security ceremonies are meant to extend security protocols by including the human in the formalization and making explicit the environment (and the attacker). A ceremony may also combine protocols. In consequence, a formalism for security ceremonies is expected to be expressive enough to include existing formalisms for protocols as special cases. Such a formalism should offer the possibility to model human behavior related to the ceremony. Since ceremonies would tend to be large, because of all the assumptions that are included explicitly, we expect compositionality to be a main aspect of the formalism, both for design and for reasoning.

For usability purposes a desired formalism for security ceremonies would offer a graphical language for developing the ceremonies, as well as for reasoning. Yet the graphics should be formally grounded, so to have guarantees for the security results obtained. A good example of such formally grounded graphical languages can be the statecharts [17, 19] or the live sequence charts [18], which were intended for describing concurrent and reactive systems.

*This work was partially supported by the project OffPAD with number E!8324 part of the Eurostars program funded by the EUREKA and European Community.

†**Acknowledgements:** I would like to thank Audun Jøsang for introducing me to security ceremonies and for explaining their practical usefulness and the need for an adequate formalism (i.e., graphical, expressive, and with reasoning capabilities).

Pre-proceedings of
The International Workshop on Graphical Models for Security 2014

The aim of the actor network procedures [33] is to be graphical, expressive, compositional, and with formal logical reasoning capabilities. In this paper we look more carefully at this formalism and relate it to the psi-calculi framework, which offers solid semantical analysis and possibility for correlations with existing security formalisms coming from the process algebra approach.

Psi-calculi [3] are a semantics framework where various existing calculi can be found as instances. In particular, the spi- and applied-pi calculi [2, 1] are two instances of interest for security protocols. Psi-calculi are though a non-graphical algebraic formalism; but with strong mathematical background. Results and tools of psi-calculi, e.g., involving theorem provers or true concurrency semantics, could be thus used in the setting of actor network procedures. Nevertheless these could easily be hidden from the security ceremony developer behind the graphical formalism ANPs provide.

With the danger of seaming somewhat pedantic to some readers, a little bit more motivation for use of formal techniques for security protocols is in order here. Arguably, for security systems perfection and assurance of perfection are highly important, since bugs cannot be considered “features”, as is the case in other areas. For a security system one often wants to be provided with guarantees that some expected security properties are met. This can be even more difficult to achieve for security ceremonies, which are more complex, composing protocols, including hidden assumptions and human models. In the case of security protocols one hardly can rely on testing to provide assurance; and experience has shown that protocols that are thoroughly tested in practice for years turn out to contain serious flaws, where a famous example is [26].

This motivates why considerable amounts of research have been put into providing mathematical models and theories for studying security protocols. But more practical are the formal tools that have been built on top of these theories so to have a (semi-)automated way of ensuring security properties. Examples of tools include model checkers like Murphi [29, 40] or FDR [15] which are push-button tools with yes/no answers; or theorem provers like ProVerif [4] for the symbolic (process calculi) approach and CryptoVerif [5] for the computational approach, or Isabell/HOL [32], which often need interaction with expect users but which achieve stronger results than model checkers do.

The psi-calculi is a framework that goes well with the ProVerif and FDR tools which are also based on process calculi. But more than this, psi-calculi have been built (i.e., all the related meta-results have been proven) using the proof assistant Isabell/HOL [30]. Therefore one could say that psi-calculi could be seen as lying at the intersection of the two kinds of tools, making use of the strengths of both.

A downside of such strong formally grounded frameworks is that they are difficult to use by common developers of security protocols and ceremonies. This is where graphical formalisms usually can provide considerable simplifications and hide formal notation, concentrating on the concepts and methods instead. A quite appreciated example of graphical languages grounded in theory can be found in the area of developing reactive or concurrent systems. Here the groundbreaking was achieved through *statecharts* [17] which has become a standard and which have been extended into the more recent *live sequence charts* [18]. A long term goal of the present author is to have a similar graphical language and tool-set for security ceremonies; and this paper tries to identify a path in this direction.

2 Background on psi-calculi

Psi-calculus [3] has been developed as a framework for defining nominal process calculi, like the many variants of the pi-calculus [28]. The psi-calculi framework is based on nominal datatypes, and [3, Sec.2.1] gives a sufficient introduction to nominal sets used in psi-calculi. We will not refer much to nominal datatype i this paper, but refer the reader to the book [36] which contains a thorough treatment

of both the theory behind nominal sets as well as various applications (e.g., see [36, Ch.8] for nominal algebraic datatypes). We expect, though, some familiarity with notions of algebraic datatypes and term algebras.

The psi-calculi framework is parametric; instantiating the parameters accordingly, one obtains an *instance of psi-calculi*, like the pi-calculus, or the cryptographic spi-calculus. These parameters are:

$\mathbf{T}$	terms (data/channels)
$\mathbf{C}$	conditions
$\mathbf{A}$	assertions

which are nominal datatypes not necessarily disjoint; together with the following operators:

$\leftrightarrow : \mathbf{T} \times \mathbf{T} \to \mathbf{C}$	channel equality
$\otimes : \mathbf{A} \times \mathbf{A} \to \mathbf{A}$	composition of assertions
$\mathbf{1} \in \mathbf{A}$	minimal assertion
$\vdash \subseteq \mathbf{A} \times \mathbf{C}$	entailment relation

Intuitively, terms can be seen as generated from a signature, as in term algebras; the conditions and assertions can be those from first-order logic; the minimal assertion being top/true, entailment the one from first-order logic, and composition taken as conjunction. We will shortly exemplify how pi-calculus is instantiated in this framework. The operators are usually written infix, i.e.: $M \leftrightarrow N$, $\Psi \otimes \Psi'$, $\Psi \vdash \varphi$.

The above operators need to obey some natural requirements, when instantiated. Channel equality must be symmetric and transitive. The composition of assertions must be associative, commutative, and have $\mathbf{1}$ as unit; moreover, composition must preserve equality of assertions, where two assertions are considered equal iff they entail the same conditions (i.e., for $\Psi, \Psi' \in \mathbf{A}$ we define the equality $\Psi \simeq \Psi'$ iff $\forall \varphi \in \mathbf{C} : \Psi \vdash \varphi \Leftrightarrow \Psi' \vdash \varphi$).

The intuition is that assertions will be used to assert about the environment of the processes. Conditions will be used as guards for guarded (non-deterministic) choices, and are to be tested against the assertion of the environment for entailment. Terms are used to represent complex data communicated through channels, but will also be used to define the channels themselves, which can thus be more than just mere names, as in pi-calculus. The composition of assertions should capture the notion of combining assumptions from several components of the environment.

The syntax for building psi-process is the following (psi-processes are denoted by the $P, Q, \ldots$; terms from $\mathbf{T}$ by $M, N, \ldots$;):

$\mathbf{0}$	Empty/trivial process
$\overline{M}\langle N \rangle.P$	Output
$\underline{M}\langle(\lambda \tilde{x})N\rangle.P$	Input
case $\varphi_1 : P_1, \ldots, \varphi_n : P_n$	Conditional (non-deterministic) choice
$(\nu a)P$	Restriction of names a inside processes P
$P \mid Q$	Parallel composition
$!P$	Replication
$(\!\mid \Psi \mid\!)$	Assertions

The empty process has the same behavior as, and thus can be modeled by, the trivial assignment $(\!\mid \mathbf{1} \mid\!)$.

The input and output processes are as in pi-calculus only that the channel objects M can be arbitrary terms. In the input process the object $(\lambda \tilde{x})N$ is a pattern with the variables $\tilde{x}$ bound in N as well as in the continuation process P. Intuitively, any term message received on M must match the pattern N for some substitution of the variables $\tilde{x}$. The same substitution is used to substitute these variables in P after a successful match. The traditional pi-calculus input $a(x).P$ would be modeled in psi-calculi as $\underline{a}\langle(\lambda x)x\rangle.P$, where the simple names a are the only terms allowed.

The case process behaves like one of the P_i for which the condition φ_i is entailed by the current environment assumption, as defined by the notion of *frame* which we preset later. This notion of frame is familiar from the applied pi-calculus, where it was introduced with the purpose of capturing static information about the environment (or seen in reverse, the frame is the static information that the current process exposes to the environment). A particular use of case is as **case** $\varphi : P$ which can be read as **if** φ **then** P. Another special usage of case is as **case** $\top : P_1, \top : P_2$, where $\Psi \vdash \top$ is a special condition that is entailed by any assertion, like $a \leftrightarrow a$; this use is mimicking the pi-calculus nondeterministic choice $P_1 + P_2$. Restriction, parallel, and replication are the standard constructs of pi-calculus.

Assertions $(\!|\Psi|\!)$ can float freely in a process (i.e., be put in parallel) describing assumptions about the environment. Otherwise, assertions can appear at the end of a sequence of input/output actions, i.e., these are the guarantees that a process provides after it finishes (on the same lines as in assume/guarantee reasoning about programs). Assertions are somehow similar to the active substitutions of the applied pi-calculus, only that assertions do not have computational behavior, but only restrict the behavior of the other constructs by providing their assumptions about the environment.

Example 2.1 (pi-calculus as an instance) *To obtain pi-calculus [28] as an instance of psi-calculus use the following, built over a single set of names $\mathcal{N}$:*

$$\begin{array}{lcl} \mathbf{T} & \triangleq & \mathcal{N} \\ \mathbf{C} & \triangleq & \{a = a \mid a, b \in \mathbf{T}\} \\ \mathbf{A} & \triangleq & \{\mathbf{1}\} \\ \leftrightarrow & \triangleq & = \\ \vdash & \triangleq & \{(\mathbf{1}, a = a) \mid a \in \mathbf{T}\} \end{array}$$

with the trivial definition for the composition operation. The only terms are the channel names $a \in \mathcal{N}$, and there is no other assertion than the unit. The conditions are equality tests for channel names, where the only successful tests are those where the names are equal. Hence, channel comparison is defined as just name equality.

Psi-calculus is given an operational semantics in [3] using labeled transition systems, where the nodes are the process terms and the transitions represent one reduction step, labeled with the action that the process executes. The actions, generally denoted by α, β, represent respectively the input and output constructions, as well as τ the internal synchronization/communication action:

$$\overline{M}\langle(\nu\tilde{a})N\rangle \mid \underline{M}\langle N\rangle \mid \tau$$

Transitions are done in a context, which is represented as an assertion Ψ, capturing assumptions about the environment:

$$\Psi \rhd P \xrightarrow{\alpha} P'$$

Intuitively, the above transition could be read as: The process P can perform an action α in an environment respecting the assumptions in Ψ, after which it would behave like the process P'.

The context assertion is obtained using the notion of *frame* which essentially collects (using the composition operation) the outer-most assertions of a process. The frame $\mathcal{F}(P)$ is defined inductively on the structure of the process as:

$$\begin{array}{l} \mathcal{F}((\!|\Psi|\!)) = \Psi \\ \mathcal{F}(P \mid Q) = \mathcal{F}(P) \otimes \mathcal{F}(Q) \\ \mathcal{F}((\nu a)P) = (\nu a)\mathcal{F}(P) \\ \mathcal{F}(!P) = \mathcal{F}(\textbf{case } \tilde{\varphi} : \tilde{P}) = \mathcal{F}(\overline{M}\langle N\rangle.P) = \mathcal{F}(\underline{M}\langle(\lambda\tilde{x})N\rangle.P) = \mathbf{1} \end{array}$$

Any assertion that occurs under an action prefix or a condition is not visible in the frame.

We give only an exemplification of the transition rules for psi-calculus, and refer to [3, Table 1] for the full definition. The (CASE) rule shows how the conditions are tested against the context assertions. The communication rule (COM) shows how the environment processes executing in parallel contribute their top-most assertions to make the new context assertion for the input-output action of the other parallel processes. In the (com)-rule the assertions Ψ_P and Ψ_Q come from the frames of $\mathscr{F}(P) = (\nu \tilde{b_P})\Psi_P$ respectively $\mathscr{F}(Q) = (\nu \tilde{b_Q})\Psi_Q$.

$$\frac{\Psi \rhd P_i \xrightarrow{\alpha} P' \qquad \Psi \vdash \varphi_i}{\Psi \rhd \mathbf{case}\ \tilde{\varphi} : \tilde{P} \xrightarrow{\alpha} P'}\ (\text{CASE})$$

$$\frac{\Psi_Q \otimes \Psi \rhd P \xrightarrow{\overline{M}\langle(\nu \tilde{a})N\rangle} P' \qquad \Psi_P \otimes \Psi \rhd Q \xrightarrow{\underline{K}\langle N\rangle} Q' \qquad \Psi_Q \otimes \Psi_P \otimes \Psi \vdash M \leftrightarrow K}{\Psi \rhd P \,|\, Q \xrightarrow{\tau} (\nu \tilde{a})(P' \,|\, Q')}\ (\text{COM})$$

There is no transition rule for the assertion process; this is only used in constructing frames. Once an assertion process is reached, the computation stops, and this assertion remains floating among the other parallel processes and will be composed part of the frames, when necessary, like in the case of the communication rule.

3 A psi-calculus instance for actor network procedures

We do not introduce the notation and definitions used in the ANPs of [33] because our aim here is to develop teh ANP ideas in the formal language of psi-calculi. In consequence, we lie to see this section as a formal description of ANPs. The ideas and development from [33] of the ANPs require quite expressive theories which cannot be easily captured with traditional formalisms for security protocols, but which are available in the psi-calculi framework.

There are a few aspects of psi-calculi that offer us the possibility to give semantics to the actor network procedures in this section; and in fact to complex systems like ubiquitous systems where humans are part of the system.

One aspect is the expressiveness of the terms that are allowed to be used for representing messages. This is very liberal in psi-calculi, and thus can easily capture complex structures of messages. Moreover, nominals are allowed in the terms for capturing the important notion of names (like in pi-calculi, or object-oriented languages). Names appear in actor network procedures in three places, as we see further, as names for channels, system configurations, and names of participants in the ceremony.

Another aspect of psi-calculi is the way communication can be done through arbitrarily complex communication terms. This means we are not restricted to just one channel name, but more structure for channels is allowed. We exploit this when modeling the structure of the system configurations and their attached channels and participants. This more complex structure is responsible for the communications in the actor network procedures.

An important aspect of psi-calculi is also the logic that is available through the assertions and the conditions language, and the entailment relation between the two. The liberty that the psi-calculi framework offers for defining the logical part of the calculus allows one to choose the right language for the application purpose. In consequence, depending on the problem one can choose a more expressive logical entailment or one with better computation properties (i.e., feasible for automation). One way of using the assertions and conditions is as in the Hoare-style of reasoning. We may have pre-conditions (using the case construct) and post-conditions (using the trailing assertions) for individual actions as well

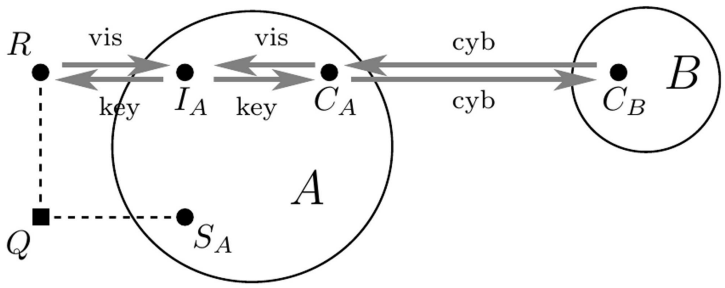

Figure 1: The structure of the configurations, part of the CAP procedure.

as for complex processes. Essentially, psi-calculi allow us to have an assume/guarantee reasoning style using the logical language of our choice and the granularity of the reasoning (i.e., process/action level). Processes are annotated with logical assertions so that an external logical reasoning system can be used on top of the process. But as well, the conditions are logically tested by the process while running, constituting a runtime level reasoning system.

The logical part of the psi-calculi will be exploited to capture the reasoning aspects that actor networks procedures and their PDL logic offers.

3.1 Example of an actor network procedure

We take the *two-factor authentication* example from [33] of the Chip Authentication Program (CAP) procedure. The configuration structure is described in [33, 2.4.2] whereas the runs of the ceremony are described in [33, 3.5.1]. The graphical formalization of this ceremony is given in [33, Fig.3] for the structure and in [33, Fig.7] for the run; we will use the same original figures so to stick with the graphical choices of the authors of [33].

The Figure 1 (taken from [33, Fig.3]) graphically represents the structure of the configurations and the attached communication channels for the actor network procedure that formalizes the CAP two-factor authentication [10]. The structure contains two identity names A (for Alice) and B (for the Bank) which are attached (as subscript) to some of the configurations. The circled areas are not strictly necessary, and become impossible to represent for larger examples; but are good visual aid for examples like this one where they encircle all those configurations corresponding to the respective identity.

The single configuration C_B represents the Bank's computer. Three configurations are under A's control: the computer C_A, the card S_A, and the human representation of Alice I_A. A card reader R is also available, which when coupled with Alice's card form the configuration Q. The human I_A can output through a keyboard channel information to Alice's computer and through another keyboard channel to the card reader. Both the card reader and the computer have visual displays that give information to the human. There are two cyber channels between the two computers; cyber channels are assumed to be untrusted and the information on them should be transmitted encrypted.

The arrows represent channels, and have attached a label denoting the type of the channel. The dark circles and squares represent configurations, where the squares are complex ones containing sub-configurations, whereas the circles are minimal configurations; which are called nodes in [33]. The dashed lines display the containment relation between the configurations.

Based on this structure, runs are drawn (usually one run, the desired/secure run). The Figure 2 (taken from [33, Fig.7]) graphically represents the desired run for the CAP authentication. For a better visual association, the structure of the configurations is displayed at the top, in a more simplistic manner.

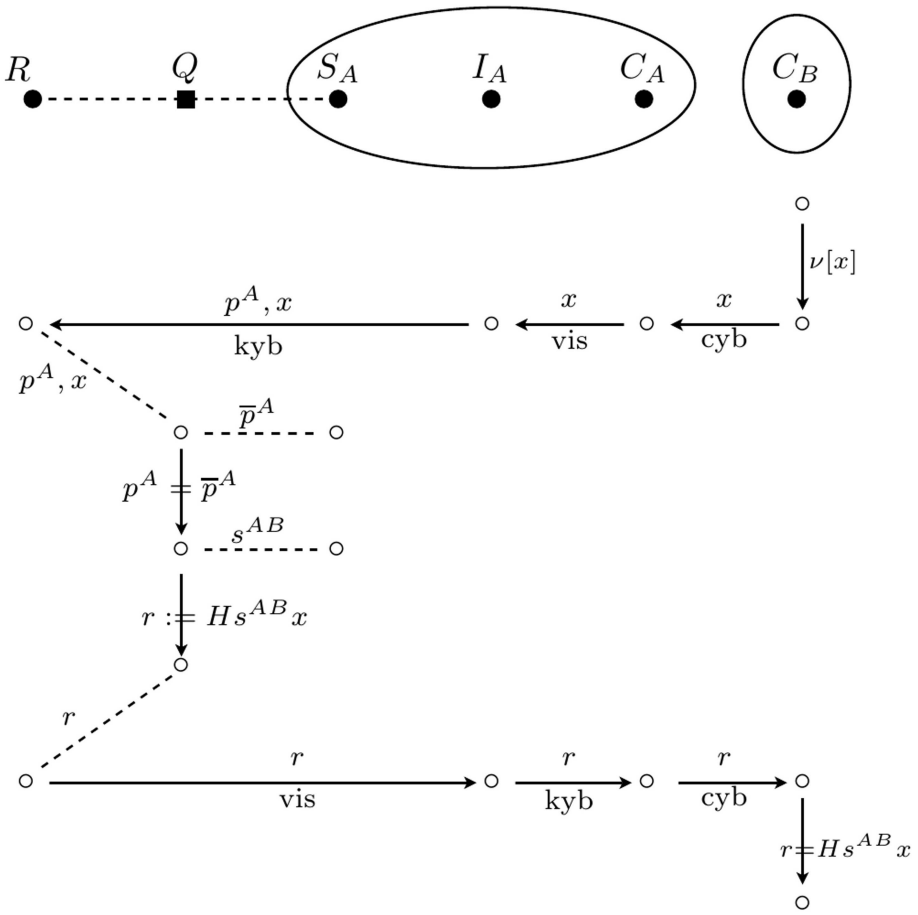

Figure 2: The desired run of the CAP procedure.

The run in Figure 2 shows several aspects of ANPs. Internal actions are drawn vertically, whereas communication between configurations are drawn horizontally. The actions are displayed on the arrows, as well as the channel type that is used for communication. There are actions of generation of fresh content, of transmission of information, of polyadic communication (tuples of messages are sent), tests, and assignments. The dashed lines again are related to configurations made from combination of other configurations, but in a run represent sharing of information.

In this run the Bank generates a fresh value x and sends it to the computer of Alice which communicates it to the human through a visual display channel. The human forwards this value and a password to the card reader through a keyboard input channel. The card reader and the card form a configuration inside which the information sent by the human is shared. This configuration compares the password send by the human to the one stored on the card S_A. If matched then the card gives away a long secret s^{AB}, which is difficult for a human to remember, opposed to a password. The configuration Q hashes this secret and the fresh value into a response which the card reader displays to the human to copy and forward it through the computer of Alice to the Bank's computer. The long secret s^{AB} is shared a priori between the card and the Bank. Therefore the Bank can perform the same calculation to generate a hashed value from this secret and the fresh value and to test it against the received response.

The fresh value is used to ensure that phishing cannot be done through recording just one session. The secret stored on the card is assumed to be a strong secret. The password is used only to make sure

that the right human has made the configuration Q by inserting her card into the card reader. Sending this weak password is done through a physical channel, which is assumed to be harder to attack.

3.2 Encoding

Actor network procedures essentially consist of a structure of configurations together with events/actions causally ordered in the concurrent runs of the procedure/protocol. Configurations are partially ordered by a containment relation, which specifies which sub-configurations form a larger configuration. Configurations have attached channel ends (input and output ends). There is information flow inside configurations; more details will come later in the text.

We therefore, identify one nominal datatype built over a set of *configuration names* $\mathcal{N}_C$. This datatype captures the partial order on the configurations. The terms that we will use are lists of configuration names describing reachability paths based on the parent-child relation between configurations; i.e., each configuration comes with the list of its ancestors.

$$[l_1, \cdots, l_n] \in \mathbf{T} \text{ for } l_i \in \mathcal{N}_C.$$

The list terms may also contain variables, and the names in the list may also be hidden behind a name binding restriction operation $(\nu \cdot)$.

Channels and channel types do not bare much distinction in [33]. In consequence we will treat them in this paper as one and the same *channel name*. If proper channel types are needed (like stating what kind of messages are allowed to be communicated) we could turn to the work on typed psi-calculi of [23] which extends the classic works on typed pi-calculus [34, 39] or on the distributed pi-calculus [22, 21] where types capture resources. More complex types of channels, like the ones that [33] talks about, could be captured with complex processes that are processing the messages received, before forwarding them to the intended recipient. This is how channels like a *random noise binary channel* would be described, or a *lossy channel*. Moreover, we do not restrict the formalism and do not assume that only one channel of one type exists between two configurations, as is done in [33].

In consequence, the nominal datatype from before is enriched with a set of *channel names* $\mathcal{N}_A$ which are paired with the list terms. This describes how a channel name is attached to the configuration described by the list term. In consequence a channel is represented by a term:

$$[l_1, \cdots, l_n]c \in \mathbf{T} \text{ for } c \in \mathcal{N}_A,$$

where the channel name is c and the list $[l_1, \cdots, l_n]$ denotes the configuration (and it's ancestor configurations) to which c is attached.

Communication in psi-calculus on such a channel is defined with the psi-calculus process syntax from the previous section, e.g.:

$$\overline{[l_1, \cdots, l_n]c}\langle N \rangle \quad \text{in ANP notation would be} \quad \overline{c}\langle [N]_{[l_1, \cdots, l_n]} \rangle$$

where $N \in \mathbf{T}$ is usually a message term. Intuitively, this says that there is an output (sending) of the message N on the channel c in the configuration $[l_1, \cdots, l_n]$.

We allow message terms to be constructed from some arbitrary signature, as it is allowed in the actor network procedures, and supported by the psi-calculi framework. For the purposes of this paper, the signature for building messages N is left open, and is unimportant for the developments that we do further. For a specific application to a security ceremony or protocol, the designer decides on the message terms needed in the procedure. In this way we allow one to specify the minimal signature for

message terms as needed for the specific protocol. Therefore the computation needed to analyze the specific protocol is dependent on the messages being sent part of the protocol.

One more nominal set $\mathcal{N}_I$ keeps track of the *identities* involved in the ceremony. Identities are associated to configurations through a partial function, describing which identity controls which configuration; some configurations may be uncontrolled (thus the partiality of the function). We use a pairing construct to form configurations which are controlled by some principal:

$$i[l_1,\cdots,l_n] \in \mathbf{T} \text{ for } i \in \mathcal{N}_I.$$

Channels can be attached to both uncontrolled configurations, like we did before, as well as to controlled configurations:

$$i[l_1,\cdots,l_n]c \in \mathbf{T} \text{ for } i \in \mathcal{N}_I, l_i \in \mathcal{N}_A, c \in \mathcal{N}_C.$$

This example of term involves all three nominal sets, and is arguably the most complex form of terms we will use for communications in an ANP. More complication would come only from the specific term algebra of the messages that a designer chooses for a specific protocol.

In the actor network procedures, the communication operations which are not controlled, i.e.,

$$\overline{[l_1,\cdots,l_n]c}\langle N\rangle \quad \text{as Output, or} \qquad \underline{[l_1,\cdots,l_n]c}\langle(\nu\tilde{a})N\rangle \quad \text{as Input}$$

are called *events*, whereas the controlled ones, like

$$\overline{i[l_1,\cdots,l_n]c}\langle N\rangle \quad \text{or} \qquad \underline{i[l_1,\cdots,l_n]c}\langle(\nu\tilde{a})N\rangle,$$

are called *actions*. But there is no essential difference, and thus the graphical notation does not make a distinction between the two.

The structure of the actor network procedure is intended to capture how information is shared within the sub-configurations. In particular, information could flow from a main configuration to its sub-configurations when received on a channel attached to the main configuration. Opposite, information could flow from a sub-configuration to its parents (and ancestors) configurations, and be sent out through channels attached to an ancestor, and not to the originator configuration. This form of information sharing is rather open and liberal in ANPs, which would leave room for a lot of hidden flow of information.

This means that there is not much control on the sharing of information in ANPs. It may be that the parent configuration wants to share some information only with part of its sub-configurations. In consequence, here we allow for more control on the information flow so a communication action can specify explicitly to which sub-configuration it wants to communicate. Moreover, we allow for hiding of the internal structure of configurations. When hiding is used then sharing can be implemented only internally, by first communicating with the main (public) configuration, which in turn decides to which sub-configurations to forward the message (possibly changed).

Until now we have encoded the communication structure of an actor network procedure through the terms $\mathbf{T}$ of psi-calculus. The approach that we took above is inspired by the nested distributed pi-calculus of [24], where locations can be nested (i.e., a location can have sub-locations). Depending on the list terms that one defines, different relations between the configurations can be obtained. For the ANPs in particular, we are aiming for a partial order relation.

But the formalism for ANPs should subsume existing formalisms for security protocols and communicating processes. Indeed, if the lists defined above are always empty, and only channel names are used, then we obtain the communication mechanism of pi-calculus. Assume that no identities are present in the terms. The distributed pi-calculus is then obtained when working only with singleton lists, i.e., a

flat structure of the locations (or configurations). Ambient calculi [7] or bigraphs [27] are obtained by making a tree-like structure between the configurations.

Internal sharing can be captured using local communication, hidden from outsiders. The same approach we use to model local actions, like assignment or generation of fresh names as in the CAP example. Sending a fresh value on the cyber channel in the CAP example would be encoded as:

$$(\nu fr)\,\overline{B[l_C]cyb}\langle fr\rangle.\mathbf{0}$$

For ease of notation, henceforth we will not add the empty process at the end. To encode a local/internal action we use a private channel on which the configuration communicates with itself, as:

$$(\nu loc)(\overline{B[l_C]loc}\langle fr\rangle \mid \underline{B[l_C]loc}\langle(\lambda x)x\rangle).$$

Example 3.1 *The local generation of the fresh value fr by Bank's computer $B[l_C]$, which is then communicated on the cyber channel is thus modeled as:*

$$(\nu loc)((\nu fr)\overline{B[l_C]loc}\langle fr\rangle \mid \underline{B[l_C]loc}\langle(\lambda x)x\rangle.\overline{B[l_C]cyb}\langle x\rangle).$$

The restriction operator ν applied to the name loc makes this communication channel visible only inside the scope of ν, whereas the restriction of the name fr models the uniqueness, thus freshness.

Up to now we have made use only of the **T** nominal datatype of the psi-calculi, not needing the logical part offered by the assertions **A** and conditions **C**. The terms we described until now, part of **T**, are used to capture the complex communication structure of ANP. This was used in conjunction with the process syntax for input/output, parallel composition and name restriction.

The **case** process of psi-calculus is powerful, offering both non-determinism as well as conditionals. Actor network procedures do not involve non-determinism, the same as the spi- and applied pi-calculus. These require only conditional constructs. This is natural when thinking that these are formalisms for describing security/communication protocols, i.e., deterministic runs of such protocols. But having the non-deterministic choice possibility in psi-calculi opens up for more modeling opportunities, like when wanting to refine the model of the human into a probabilistic one, involving probabilistic choices.

We have seen the necessity for the conditional in the CAP example, where terms were tested for equality. In consequence, we will include as conditions in **C**, tests for equality for any two nominal terms from **T**:

$$M = N \in \mathbf{C} \;\text{ for any }\; M, N \in \mathbf{T}.$$

The entailment relation defines when two terms are equal wrt. some assertion:

$$\Psi \vdash M = N \;\text{ iff }\; \vdash_\Sigma M = N$$

where Σ is the signature over which the message terms are built, and $\vdash_\Sigma$ is the equational logic entailment relation wrt. the signature Σ. In other words, $M = N$ is decided only looking at the terms, using syntactic unification in the term algebra described by the signature Σ. In many cases equations are defined for such a term algebra, which need to be considered when deciding the equality of two terms. This would then involve working modulo these equations; we use the same notation $\vdash_\Sigma$ for the case when equations are part of the definition of the algebra of terms too.

Testing for equality of message terms does not depend on the assertions. Nevertheless, we will use assertions to model the partially ordered runs that actor network procedures use. This way of capturing true concurrency models (like the pomsets [13, 37] used by ANPs) in psi-calculi is inspired by the recent [31]. Actor network procedures is among the few formalisms that acknowledges the need for true

concurrency [42] when modeling protocols, instead of interleaving concurrency or linear runs as most process algebras approaches do. Actor network procedures describe a run to be a *pomset*, i.e., a partially ordered multiset. Knowing that a linear run (word, string) is a totally ordered multiset (because symbols may appear multiple times in the string), then a pomset is a partially ordered run, i.e., a run where some of the actions are concurrent, not all being linearly ordered. It is natural to model a run of a protocol or ceremony as a partially ordered run because there are several parties involved, often distributed, thus executing actions/communication in parallel, concurrently. A single participant may be deterministic and sequential, thus exhibiting a linear run. But put together several participants exhibit a partially ordered run of the whole ceremony. More order constraints can come from the communications, where e.g., input actions depend on (i.e., must come after) the respective output actions.

Actor network procedures are declarative when defining their runs, in the same style as true concurrency models like event structures [41] or pomsets, or as languages are, as opposed to automata or process algebras which describe their runs in an operational manner. Psi-calculi allow also a declarative style of defining dependencies between actions by using the logic captured in the entailment relation and the assertions. In this way we have the full descriptive power of true concurrency models so to capture conjunctive (or disjunctive) dependencies as is the case with pomsets. This cannot be captured only through the sequence operation of the psi-processes.

We define assertions $\mathbf{A}$ to contain sets of communication terms, e.g.:

$$\{\underline{i[l_1,\ldots,l_k]c}\langle N\rangle, \overline{i[l_1,\ldots,l_k]c}\langle N\rangle,\ldots\} \in \mathbf{A} \text{ with } i[l_1,\ldots,l_k]c \in \mathbf{T} \text{ and } N \in \mathbf{T}.$$

In consequence, conditions are also enriched with such actions by combining them with conjunction. The entailment is defined to treat conjunction appropriately, as in classical logics. For the new kind of conditions the entailment is just set-containment:

$$\Psi \vdash \underline{i[l_1,\ldots,l_k]c}\langle N\rangle \text{ iff } \underline{i[l_1,\ldots,l_k]c}\langle N\rangle \in \Psi.$$

With the conditions and assertions in place we can capture orders on the communication actions in the form of pomsets as follows. Each action is conditioned by a set of other actions on which it depends. In this way the action cannot be performed until the condition is met, i.e., all the actions on which it depends have been done. Thus,

$$\textbf{case } \varphi : \underline{i[l_1,l_2]c}\langle N\rangle.P \text{ with } \varphi = \{\overline{j[l_1]d}\langle N'\rangle, \underline{i[l_1,l_2]b}\langle N''\rangle\}$$

describes the fact that action $\underline{i[l_1,l_2]c}\langle N\rangle$ must come after the two actions from the condition have been executed. The knowledge that an action has been executed is gathered in the context assertion through assertion processes which are left behind after each execution of an action. This is easily done by changing the continuation of an action:

$$\underline{i[l_1,l_2]c}\langle N\rangle.P \text{ becomes } \underline{i[l_1,l_2]c}\langle N\rangle.(P \mid (\!|\underline{i[l_1,l_2]c}\langle N\rangle|\!)).$$

After the action is performed, the trailing assertion will become available to both the continuation and the other parallel processes, as part of the context assertion collected by the frame of the parallel process.

The actor network procedure formalism uses the PDL (Procedure Description Logic) to reason about runs. PDL[1] uses two kinds of basic formulas: one states that an action has been executed; and another

[1]PDL for actor network procedures should not be confused with Propositional Dynamic Logic [20] (usually abbreviated PDL, or DL for the higher order case) used for reasoning about programs. On the other hand, propositional dynamic logic is a modal logic that reasons about actions in general, and could also be used for reasoning about ANPs once the special basic formulas and actions are set, as done for the ANPs. In fact, the reasoning style of PDL for ANPs resembles much the past temporal logic style of reasoning, and temporal reasoning can be done with propositional dynamic logic too.

states that some action has been executed before another action. Above, the assertions capture only the first kind of PDL basic formula. We now add another assertion that stands for the second kind of PDL formula. This second kind of assertions capture a whole pomset in the assertions only. This pomset is available to the process for inspection, during the execution, and it captures the partial run so far. It is like a history in Hoare-style reasoning, only that in our case it is a partially ordered history.

We thus add to the assertion terms, dependency pairs of actions, giving the possibility to describe partial orders of actions as assertions:

$$\underline{i[l_1,\ldots,l_k]c}\langle N\rangle \leq \overline{j[l_1,\ldots,l_k]d}\langle N'\rangle \in \mathbf{A}$$

signifying that the right-hand action depends on the left-hand action. If, moreover, both these actions are part of the assertions set then we conclude that the left action happened before the right action.

A question to ask is how do such dependency pairs get into the assertion set. Trailing assertion processes would be used, the same as we did earlier, only that more care needs to be taken when defining the assertion composition operation. We are not just using set union, but for building dependency pairs we must also achieve the transitivity property of the partial order relation we want to maintain.

Example 3.2 *For the CAP run in Figure 2 the execution of the process would reach a point (e.g., upper left-most corner) where the environment assertion* Ψ *would contain a dependency pair*

$$\overline{A[l_C]cyb}\langle fr\rangle \leq \underline{A[l_I]kyb}\langle(p^A, fr)\rangle,$$

saying that the action of Alice of typing at the keyboard of the password and the fresh value is dependent, thus should come after, the computer of Alice having received the fresh value.

We have thus covered all aspects of the actor network procedures graphical formalism through the psi-calculus operational formalism. The structure of the configurations has been captured through the nominal datatypes, and the message terms have been treated the same as in ANPs. The definition of the pomset runs of a ANP was done by making use of the encoding of the dependencies between the actions using the assertions and conditions. Communication is done through channels attached to configurations, and internal actions are modeled also as communications but on private channels.

We have thus seen use of all psi-constructs for building processes: input/output used for communication and simulation of other actions like assignment; the **case** for modeling conditionals (and Hoare-style pre-conditions); restriction of names for modeling private communications and fresh values; parallel composition for putting several identities in the protocol to run together; and assertion processes for capturing the Hoare-style guarantees. The *replication* construct has not been used; but it is essentially useful when needing to model ceremonies that can run through several sessions.

4 Conclusions and outlook

There are many possibilities that psi-calculi open up for modeling ubiquitous systems and security ceremonies, where humans are part of the system and are intended to be modeled in a more faithful way. The work we undertook in this paper shows that the psi-calculi framework is expressive enough to capture faithfully complex formalisms like the actor network procedures, and even with some generalizations thereof. We could easily capture concurrent computations in psi-calculi, besides sequential ones.

The logic that psi-calculi offers is opened to be tailored to the application needs. In the case of ANPs we could use it to capture the Hoare-style reasoning that ANPs use. Such a reasoning is essential for security ceremonies where the assumptions should be made explicit, opposed to what usually is done with formalisms for security protocols where many assumptions are left underspecified.

The term construction mechanisms are also rather liberal in psi-calculi. This offers the possibility to define with any degree of detail needed, the message terms exchanged in the ceremony. At the same time we are not constrained when defining communication terms. This allows for modeling a great wealth of communication mechanisms. We have exploited this second freedom in the construction of the nominal terms representing the complex communication structure of ANPs.

For ceremonies in particular, we are interested in more faithful modeling of the human nodes where we would like to either leave room for errors, i.e., using non-determinism, or we would like to integrate a statistical model of the human, using probabilistic choices. These go beyond what current actor network procedures allow. But psi-calculi can accommodate probabilistic models, e.g., by going through CC-pi [6, 9] which has already been treated as a psi-calculus. But the work on probabilities and psi-calculus is not yet mature and still needs more investigation.

An interesting future direction for a graphical formalism line the ANPs for modeling ceremonies is the notion of action refinement [14]. This is a technique for building (and working with) models in an incremental way, starting from an abstraction and refining actions into more concrete implementations. Action refinement is well behaved for true concurrency models and their equivalences like history preserving bisimulation. But it is not studied how to do action refined for graphical languages (except for the statecharts [17]), and not for ANPs either. Refinement for ANPs would allow a ceremony designer to refine abstract models, both the configuration structures and the runs. By refining, one can expand single actions into more complex runs, or can expand one configuration with sub-configurations.

Action refinement is a technique for building models compositionally, in a top-down manner, whereas process algebras have a compositional approach where they build a model from components plugged together using operators like choice, sequential, or parallel composition. Action refinement can work well in combination with the compositional approach of process algebras; and we encourage this combination.

The structure of the configurations in an ANP is static. It is not the case that during the execution of the ceremony some configurations are broken, disappear, or loose some of their sub-configurations. But in the psi-calculi one can easily model a dynamic structure, similar to how the ambients are dynamic in ambient calculus [7], or how communication changes locations in distributed pi-calculus [21]. The recent bigraphs formalism [27, 16] is especially focusing on how the structure of the system changes; the execution is defined in terms of change of structure (and interaction links). We see great possibility for investigating change of structure in ANPs, starting from the encoding we have given in this paper, and from the investigations of the above formalisms wrt. the psi-calculi framework.

References

[1] Martín Abadi & Cédric Fournet (2001): *Mobile values, new names, and secure communication*. In Chris Hankin & Dave Schmidt, editors: *POPL*, ACM, pp. 104–115. Available at `http://doi.acm.org/10.1145/360204.360213`.

[2] Martín Abadi & Andrew D. Gordon (1999): *A Calculus for Cryptographic Protocols: The spi Calculus*. *Inf. Comput.* 148(1), pp. 1–70. Available at `http://dx.doi.org/10.1006/inco.1998.2740`.

[3] Jesper Bengtson, Magnus Johansson, Joachim Parrow & Björn Victor (2011): *Psi-calculi: a framework for mobile processes with nominal data and logic*. *Logical Methods in Computer Science* 7(1). Available at `http://dx.doi.org/10.2168/LMCS-7(1:11)2011`.

[4] Bruno Blanchet (2004): *Automatic Proof of Strong Secrecy for Security Protocols*. In: *IEEE Symposium on Security and Privacy*, IEEE Computer Society, pp. 86–102. Available at `http://doi.ieeecomputersociety.org/10.1109/SECPRI.2004.1301317`.

[5] Bruno Blanchet (2008): *A Computationally Sound Mechanized Prover for Security Protocols*. *IEEE Trans. Dependable Sec. Comput.* 5(4), pp. 193–207. Available at http://dx.doi.org/10.1109/TDSC.2007.1005.

[6] Maria Grazia Buscemi & Ugo Montanari (2007): *CC-Pi: A Constraint-Based Language for Specifying Service Level Agreements*. In Rocco De Nicola, editor: *16th European Symposium on Programming Programming Languages and Systems (ESOP'07)*, *LNCS* 4421, Springer, pp. 18–32. Available at http://dx.doi.org/10.1007/978-3-540-71316-6_3.

[7] Luca Cardelli & Andrew D. Gordon (1998): *Mobile Ambients*. In M. Nivat, editor: *Foundations of Software Science and Computation Structure, FoSSaCS'98*, *Lecture Notes in Computer Science* 1378, Springer, pp. 140–155. Available at http://dx.doi.org/10.1007/BFb0053547.

[8] Anupam Datta, John C. Mitchell, Arnab Roy & Stephan Hyeonjun Stiller (2011): *Protocol Composition Logic*. In V. Cortier & S. Kremer, editors: *Formal Models and Techniques for Analyzing Security Protocols*, IOS Press.

[9] Rocco De Nicola, Gian Luigi Ferrari, Ugo Montanari, Rosario Pugliese & Emilio Tuosto (2005): *A Process Calculus for QoS-Aware Applications*. In Jean-Marie Jacquet & Gian Pietro Picco, editors: *7th International Conference on Coordination Models and Languages*, *LNCS* 3454, Springer, pp. 33–48. Available at http://dx.doi.org/10.1007/11417019_3.

[10] Saar Drimer, Steven J. Murdoch & Ross J. Anderson (2009): *Optimised to Fail: Card Readers for Online Banking*. In Roger Dingledine & Philippe Golle, editors: *Financial Cryptography and Data Security*, *LNCS* 5628, Springer, pp. 184–200. Available at http://dx.doi.org/10.1007/978-3-642-03549-4_11.

[11] Nancy A. Durgin, John C. Mitchell & Dusko Pavlovic (2003): *A Compositional Logic for Proving Security Properties of Protocols*. *Journal of Computer Security* 11(4), pp. 677–722. Available at http://iospress.metapress.com/content/lgf63yyhl3ajnddt/.

[12] Carl Ellison (2007): *Ceremony Design and Analysis*. Cryptology ePrint Archive, Report 2007/399. Available at http://eprint.iacr.org/2007/399.

[13] Jay L. Gischer (1984): *Partial Orders and the Axiomatic Theory of Shuffle*. Ph.D. thesis, CS, Stanford University.

[14] Rob J. van Glabbeek & Ursula Goltz (2001): *Refinement of actions and equivalence notions for concurrent systems*. *Acta Informatica* 37(4/5), pp. 229–327. Available at http://link.springer.de/link/service/journals/00236/bibs/1037004/10370229.htm.

[15] Michael Goldsmith, Gavin Lowe, Bill Roscoe, Peter Ryan & Steve Schneider (2000): *Modelling and analysis of security protocols*. Pearson Education.

[16] Davide Grohmann (2008): *Security, Cryptography and Directed Bigraphs*. In: *4th International Conference on Graph Transformations (ICGT'08)*, *LNCS* 5214, Springer-Verlag, pp. 487–489. Available at http://dx.doi.org/10.1007/978-3-540-87405-8_41.

[17] David Harel (1987): *Statecharts: A Visual Formulation for Complex Systems*. *Science of Computer Programmings* 8(3), pp. 231–274. Available at http://dx.doi.org/10.1016/0167-6423(87)90035-9.

[18] David Harel & Rami Marelly (2003): *Come, Let's Play – Scenario-Based Programming Using LSCs and the Play-Engine*. Springer. Available at http://www.springer.com/computer/programming/book/978-3-540-00787-6.

[19] David Harel & Amnon Naamad (1996): *The STATEMATE Semantics of Statecharts*. *ACM Trans. Softw. Eng. Methodol.* 5(4), pp. 293–333. Available at http://doi.acm.org/10.1145/235321.235322.

[20] David Harel, Jerzy Tiuryn & Dexter Kozen (2000): *Dynamic Logic*. MIT Press, Cambridge, MA, USA.

[21] Matthew Hennessy (2007): *A Distributed Pi-Calculus*. Cambridge Univ. Press.

[22] Matthew Hennessy & James Riely (2002): *Resource Access Control in Systems of Mobile Agents*. *Inf. Comput.* 173(1), pp. 82–120. Available at http://dx.doi.org/10.1006/inco.2001.3089.

[23] Hans Hüttel (2011): *Typed psi-calculi*. In Joost-Pieter Katoen & Barbara König, editors: *CONCUR, Lecture Notes in Computer Science* 6901, Springer, pp. 265–279. Available at http://dx.doi.org/10.1007/978-3-642-23217-6_18.

[24] Hans Hüttel (2013): *On Representing Located Process Calculi in the psi-calculus*. personal communication.

[25] Bruno Latour (2005): *Reassembling the Social – An Introduction to Actor-Network-Theory*. Oxford Univ. Press.

[26] Gavin Lowe (1996): *Breaking and Fixing the Needham-Schroeder Public-Key Protocol Using FDR*. *Software - Concepts and Tools* 17(3), pp. 93–102.

[27] Robin Milner (2009): *The Space and Motion of Communicating Agents*. Cambridge Univ.Press.

[28] Robin Milner, Joachim Parrow & David Walker (1992): *A Calculus of Mobile Processes, I-II*. *Information and Computation* 100(1), pp. 1–77, doi:dx.doi.org/10.1016/0890-5401(92)90008-4.

[29] John C. Mitchell, Mark Mitchell & Ulrich Stern (1997): *Automated analysis of cryptographic protocols using Murphi*. In: *IEEE Symposium on Security and Privacy*, IEEE Computer Society, pp. 141–151. Available at http://doi.ieeecomputersociety.org/10.1109/SECPRI.1997.601329.

[30] Tobias Nipkow, Lawrence C. Paulson & Markus Wenzel (2002): *Isabelle/HOL - A Proof Assistant for Higher-Order Logic*. *Lecture Notes in Computer Science* 2283, Springer. Available at http://dx.doi.org/10.1007/3-540-45949-9.

[31] Håkon Norman (2013): *Event Structures as Psi-calculi*. In Tarmo Uustalu & Juri Vain, editors: *25th Nordic Workshop on Programming Theory (NWPT13)*.

[32] Lawrence C. Paulson (1998): *The Inductive Approach to Verifying Cryptographic Protocols*. *Journal of Computer Security* 6(1-2), pp. 85–128. Available at http://iospress.metapress.com/content/5wlu8p2am1du051d/.

[33] Dusko Pavlovic & Catherine Meadows (2011): *Actor-network procedures: Modeling multi-factor authentication, device pairing, social interactions*. *arXiv.org*. Available at http://arxiv.org/abs/1106.0706.

[34] Benjamin C. Pierce & Davide Sangiorgi (1996): *Typing and Subtyping for Mobile Processes*. *Mathematical Structures in Computer Science* 6(5), pp. 409–453.

[35] Wolter Pieters (2011): *Representing humans in system security models: An actor-network approach*. *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications* 2(1), pp. 75–92.

[36] Andrew M. Pitts (2013): *Nominal Sets: Names and Symmetry in Computer Science*. *Cambridge Tracts in Theoretical Computer Science* 57, Cambridge Univ. Press.

[37] Vaughan R. Pratt (1986): *Modeling Concurrency with Partial Orders*. *J. Parallel Programming* 15(1), pp. 33–71. Available at http://dx.doi.org/10.1007/BF01379149.

[38] Kenneth Radke, Colin Boyd, Juan Manuel González Nieto & Margot Brereton (2011): *Ceremony Analysis: Strengths and Weaknesses*. In: *26th IFIP-TC-11 International Information Security Conference (SEC), IFIP Advances in Information and Communication Technology* 354, Springer, pp. 104–115. Available at http://dx.doi.org/10.1007/978-3-642-21424-0_9.

[39] Davide Sangiorgi & David Walker (2001): *The π-Calculus: a Theory of Mobile Processes*. Cambridge Univ. Press.

[40] Ulrich Stern & David L. Dill (1997): *Parallelizing the Murphi Verifier*. In Orna Grumberg, editor: *9th International Conference on Computer Aided Verification (CAV), LNCS* 1254, Springer, pp. 256–278. Available at http://dx.doi.org/10.1007/3-540-63166-6_26.

[41] Glynn Winskel (1986): *Event Structures*. In: *Advances in Petri Nets, LNCS* 255, Springer, pp. 325–392. Available at http://dx.doi.org/10.1007/3-540-17906-2_31.

[42] Glynn Winskel & Mogens Nielsen (1995): *Models for Concurrency*. In Samson Abramski, Dov M. Gabbay & Tom S.E. Maibaum, editors: *Handbook of Logic in Computer Science – vol 4 – Semantic Modelling*, Oxford University Press, pp. 1–148.