Typed ψ-calculi

Hans Hüttel*

Department of Computer Science, Aalborg University, Selma Lagerlöfs Vej 300, Denmark

Abstract. A large variety of process calculi extend the π-calculus with more general notions of messages. Bengtson et al. have shown that many of these π-like calculi can be expressed as so-called ψ-calculi. In this paper, we describe a simple type system for ψ-calculi. The type system satisfies a subject reduction property and a general notion of channel safety. A number of existing systems are shown to be instances of our system, and other, new type systems can also be obtained. We first present a new type system for the calculus of explicit fusions by Wischik and Gardner, then one for the distributed π-calculus of Hennessy and Riely and finally show how existing type systems for secrecy and authenticity in the spi calculus can be represented and shown to be safe.

1 Introduction

Process calculi are formalisms that allow us to describe and reason about parallel and distributed computations. A prominent example is the π-calculus due to Milner et al. [17,23], and one of the techniques for reasoning about properties of processes is that of *type systems*. The first type system, due to Milner [17], dealt with the notion of correct usage of channels, ensuring that only names of the correct type could be communicated.

Since then, a plethora of type systems have been introduced. Pierce and Sangiorgi [19] described a type system that uses subtyping and capability tags to control the use of names as input or output channels. There are also several type systems for variants of the π-calculus. In the spi-calculus [3], type systems have been used to capture properties of cryptographic protocols such as secrecy [1,2] and authenticity [11,9]. The distributed π-calculus, Dπ, by Hennessy and Riely [20] can describe located computations in which subprocesses can migrate between locations, and in this setting type systems have been proposed [20,21] for controlling migration.

The type systems mentioned above are seemingly unrelated and arise in different settings but share certain characteristics. They only classify processes as either well-typed or not, so type judgments for processes P are of the form $E \vdash P$, where E is a type environment that records the types of the free names in P. On the other hand, terms M are given a type T, so that type judgments are of the form $E \vdash M : T$.

Bengtson et al. introduced ψ-calculi [4] as a generalization of the many variants of the π-calculus in which structured message terms appear. A main innovation is the use of a small set of process constructs that generalize those of the π-calculus, and the use of general notions of terms, assertions and conditions. Each of these syntactic categories is then assumed to form a *nominal set* in the sense of Gabbay and Mathijssen [10]. It

* hans@cs.aau.dk

J.-P. Katoen and B. König (Eds.): CONCUR 2011, LNCS 6901, pp. 265–279, 2011.

has been shown [4] that both existing calculi such as the π-calculus and the spi-calculus and new variants can be captured as ψ-calculi.

The goal of the present paper is to describe a general framework for type systems for variants of the π-calculus within the above tradition by means of a type system for ψ-calculi. Our type system generalizes the so-called simply typed π-calculus introduced by Sangiorgi and Walker [23]. Within it, we can capture several existing, seemingly quite different type systems, including those mentioned above, and formulate new ones. An important advantage is that we can formulate general soundness results that allow us to conclude the soundness of several such type systems.

There has been other work aimed at giving a general account of type systems for process calculi. Most work has focused on general type systems for the π-calculus, and in much of it processes have types that come equipped with a notion of behaviour. Early work includes that of Honda [12] who introduces so-called typed algebras which can be provided with a notion of reduction semantics. Later, [13], Igarashi and Kobayashi described a general type system for a subset of the polyadic π-calculus. This type system can be instantiated to describe e.g. deadlock freedom and race freedom. The type of a π-process is a term in a process calculus reminiscent of CCS.

Bigraphs [18] provide another general setting for process calculi. Here, Elsborg et al. have proposed a type system[8]; however, there are as yet few actual results that show how this type system can be instantiated.

In [16], Makholm and Wells describe a general process calculus Meta* and a general type system Poly*. Meta* can be instantiated by a concrete set of reduction rules, and the resulting type system will satisfy a subject reduction property. Here, there are also significant differences from our approach. Firstly, the main focus of [16] is that of understanding variants of the calculus of Mobile Ambients [6], not variants of the π-calculus. Secondly, in the type system Poly* the typable entities are processes, not names. Finally, the type system of [16] is not instantiated to any existing type system, and it is not clear how this is to be done, given the importance of typed names in many existing type systems for variants of the π-calculus. In the present paper we develop a framework for π-calculus variants in which this can be achieved.

The rest of our paper is structured as follows. Section 2 gives a short introduction to ψ-calculi. In Section 3, we describe our type system for ψ-calculi. In Section 4 we establish important properties of the type system. In particular, we prove a subject reduction property and introduce a general notion of safety. Finally, in Section 5 we show how a number of existing type systems can be seen as instances of our type system.

2 ψ-calculi

The intention of ψ-calculi [4] is to generalize the common characteristics of variants of the π-calculus that allow for transmission of message terms that may be structured, i.e. that need not be names.

2.1 Syntax

A ψ-calculus has a set of *processes*, ranged over by $P, Q, \ldots$. Processes contain occurrences of *terms*, ranged over by $M, N \ldots$ and both processes and terms can contain

names. We assume a countably infinite set of names $\mathcal{N}$. The set of names that appear in patterns (defined below) is called the set of *variable names* $\mathcal{N}_V$ and is ranged over by $x, y \ldots$. The set of other names, $\mathcal{N} \setminus \mathcal{N}_V$ is ranged over by $a, b, \ldots, m, n \ldots$. We let $u, v \ldots$ range over $\mathcal{N}$.

The formation rules for processes are the following.

$P ::=$	$\underline{M}(\lambda \boldsymbol{x})N.P$	input
	$\overline{M}N.P$	output
	$P_1 \mid P_2$	parallel composition
	$(\nu n : T)P$	restriction of name n
	$!P$	replication
	case $\varphi_1 : P_1, \ldots, \varphi_k : P_k$	conditional
	$(\!\vert \Psi \vert\!)$	assertion process

The process constructs are similar to those of the π-calculus; however, the object M of an input or output prefix can be an arbitrary term and in the input construct $\underline{M}(\lambda \boldsymbol{x})N.P$ the subject $(\lambda \boldsymbol{x})N$ is a *pattern* whose variable names $\boldsymbol{x}$ can occur free in N and P. Any term received on channel M must match this pattern; a term N_1 matches the pattern $(\lambda \boldsymbol{x})N$ if N_1 can be found by instantiating the variable names $\boldsymbol{x}$ in N with terms. Finally note that assertions (see below) can also be used as processes.

As we consider typed ψ-calculi, we assume that a name n in a restriction $(\nu n : T)P$ is annotated with a type T; types are introduced in Section 3.1.

To understand the properties of names, an important notion is that of a *nominal set*. Informally speaking, this is a set whose members can be affected by names being bound or swapped. If x is an element of a nominal set and $a \in \mathcal{N}$, we write $a\#x$, to denote that a is fresh for x; the notion extends to sets of names in the expected way.

Another difference from [4] is that the sets of types, assertions and terms are each assumed to be generated by a signature of constructors. A *nominal data type* [4] is a nominal set with internal structure. Let Σ be a signature. Then a nominal data type over Σ is a Σ-algebra, whose carrier set is a nominal set (for the precise definition, see [10]). In the nominal data types of ψ-calculi we use simultaneous term substitution $X[\boldsymbol{z} := \boldsymbol{Y}]$ – terms in $\boldsymbol{Y}$ replace the names in $\boldsymbol{z}$ in X.

Further, nominal data types are assumed to be *distributive*: for every function symbol f and term substitution σ acting on variable names, we have $f(M_1, \ldots, M_k)\sigma = f(M_1\sigma, \ldots, M_k\sigma)$. In other words, term substitution distributes over function symbols. This requirement of distributivity, which is also not required in [4], will ensure that a standard substitution lemma for type judgments will hold if the substitution is well-typed, i.e. if $\sigma(x)$ and x have the same type for any variable name $x \in \mathrm{dom}(\sigma)$.

The set of types $\mathcal{T}$ is a nominal datatype, since names can appear in types. We let T range over $\mathcal{T}$ and let $\mathrm{fn}(T)$ denote the set of free names in type T.

Terms, *assertions* and *conditions* belong to the following nominal data types.

T	terms	M, N
C	conditions	φ
A	assertions	Ψ

Given valid choices of $\mathbf{T}$, $\mathbf{C}$ and $\mathbf{A}$, the following operations on data types are always assumed:

$$\begin{aligned} \otimes &: \mathbf{A} \times \mathbf{A} \to \mathbf{A} && \text{composition of assertions} \\ \leftrightarrows &: \mathbf{T} \times \mathbf{T} \to \mathbf{C} && \text{channel equivalence} \\ \mathbf{1} &\in \mathbf{A} && \text{unit assertion} \\ \models &\subseteq \mathbf{A} \times \mathbf{C} && \text{entailment} \end{aligned}$$

The notion of channel equivalence tells us which terms represent the same communication channel; $\leftrightarrows$ thus appears in the rules describing communication and prefixes. The entailment relation $\models$ describes when conditions are true, given an assertion set, and is needed to describe the behaviour of conditionals and to decide channel equivalence.

For any process P, we let $\mathrm{n}(P)$ denote the support of P, i.e. the names of P, and let $\mathrm{fn}(P)$ denote the set of free names in P. This notion is also defined for terms.

2.2 Labelled Semantics

In ψ-calculi, the assertion information of a process P can be extracted as its *frame* $\mathcal{F}(P) = \langle E_P, \Psi_P \rangle$, where Ψ_P is the composition of assertions in P and E_P records the types of the names local to Ψ_P. We call these *qualified frames*, since (unlike [4]) names are now annotated with types. Composition of frames is defined by $\langle E_1, \Psi_1 \rangle \otimes \langle E_2, \Psi_2 \rangle = \langle E_1 E_2, \Psi_1 \otimes \Psi_2 \rangle$ whenever we have $\mathrm{dom}(E_1) \# \mathrm{dom}(E_2)$, $\mathrm{dom}(E_1) \# \Psi_2$, and $\mathrm{dom}(E_2) \# \Psi_1$. Moreover, we write $(\nu b : T)F$ to mean $\langle b : T, E_F, \Psi_F \rangle$.

Definition 1 (Frame of a process).

$$\mathcal{F}(P \mid Q) = \mathcal{F}(P) \otimes \mathcal{F}(Q) \qquad \mathcal{F}((\nu b : T)P) = (\nu b : T)\mathcal{F}(P)$$
$$\mathcal{F}(\lfloor\!\lfloor \Psi \rfloor\!\rfloor) = \langle \varepsilon, \Psi \rangle \qquad \mathcal{F}(P) = \mathbf{1} \quad \textit{otherwise}$$

Labelled transitions are of the form $\Psi \rhd P \xrightarrow{\alpha} P'$ where the label α is defined by the formation rules

$$\alpha ::= \tau \mid \underline{M}N \mid \overline{M}N \mid (\nu \boldsymbol{b} : \boldsymbol{T})\overline{M}N$$

We let $\mathrm{bn}((\nu \boldsymbol{b} : \boldsymbol{T})\overline{M}N) = \boldsymbol{b}$ and $\mathrm{bn}(\alpha) = \emptyset$ otherwise.

The transitions are given by the rules in Table 1. We omit the symmetric versions of (COM) and (PAR). In (COM) we assume that $\mathcal{F}(P) = \langle E_P, \Psi_P \rangle$ and $\mathcal{F}(Q) = \langle E_Q, \Psi_Q \rangle$, where $\mathrm{dom}(E_P)$ is fresh for all of Ψ, Ψ_Q, Q, M, and P – and that the symmetric freshness condition holds for E_Q. In (PAR) we assume that $\mathcal{F}(Q) = \langle E_Q, \Psi_Q \rangle$, where $\mathrm{dom}(E_Q)$ is fresh for Ψ, P and α. In (CASE), if more than one condition holds, the choice between them is nondeterministic.

Our semantics differs from that of [4], since types may contain names and appear in restrictions, so the order of restrictions now matters. In the rule (OPEN) we write $\nu \boldsymbol{a} \cup \{b : T\}$ to denote the typed *sequence* $\boldsymbol{a}$ extended with $b : T$ and extend the side condition of [4] to deal with the case where an extruded name appears in the type. To see why this is necessary, assume a type $\mathbf{Ch}(b)$ of channels that can carry the name b and consider the process $(\nu b : T_1)(\nu c : \mathbf{Ch}(b))\overline{a}c.$. Here, both b and c must be extruded to make use of c as a channel, even though b does not appear as a name in $\overline{a}c$.

Table 1. Labelled transition rules (for name freshness conditions, see Section 2.2)

$$\text{(In)}\quad \frac{\Psi \models M \leftrightarrow K}{\Psi \rhd \underline{M}(\lambda \boldsymbol{x})N.P \xrightarrow{\underline{K}N[\boldsymbol{x}:=\boldsymbol{L}]} P[\boldsymbol{x}:=\boldsymbol{L}]} \qquad \text{(Out)}\quad \frac{\Psi \models M \leftrightarrow K}{\Psi \rhd \overline{M}N.P \xrightarrow{\overline{K}N} P}$$

$$\text{(Com)}\quad \frac{\begin{array}{c}\Psi_Q \otimes \Psi \rhd P \xrightarrow{\overline{M}(\nu \boldsymbol{a}:\boldsymbol{T})N} P' \\ \Psi_P \otimes \Psi \rhd Q \xrightarrow{\underline{K}N} Q' \\ \Psi \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrow K\end{array}}{\Psi \rhd P \mid Q \xrightarrow{\tau} (\nu \boldsymbol{a}:\boldsymbol{T})(P' \mid Q')}\ \boldsymbol{a}\#Q \qquad \text{(Case)}\quad \frac{\Psi \rhd P_i \xrightarrow{\alpha} P' \quad \Psi \models \varphi_i}{\Psi \rhd \mathbf{case}\ \tilde{\varphi} : \tilde{P} \xrightarrow{\alpha} P'}$$

$$\text{(Par)}\quad \frac{\Psi_Q \otimes \Psi \rhd P \xrightarrow{\alpha} P'}{\Psi \rhd P \mid Q \xrightarrow{\alpha} P' \mid Q}\ \mathrm{bn}(\alpha)\#Q \qquad \text{(Scope)}\quad \frac{\Psi \rhd P \xrightarrow{\alpha} P' \quad b\#\alpha, \Psi}{\Psi \rhd (\nu b : T)P \xrightarrow{\alpha} (\nu b : T)P'}$$

$$\text{(Open)}\quad \frac{\Psi \rhd P \xrightarrow{\overline{M}(\nu \boldsymbol{a}:\boldsymbol{T})N} P'}{\Psi \rhd (\nu b : T_1)P \xrightarrow{\overline{M}(\nu \boldsymbol{a}:\boldsymbol{T} \cup \{b:T_1\})N} P'}\ b\#\boldsymbol{a}, \Psi, M \text{ and } b \in \mathrm{n}(N) \cup \mathrm{n}(\boldsymbol{T}) \cup \mathrm{n}(T_1) \qquad \text{(Rep)}\quad \frac{\Psi \rhd P \mid !P \xrightarrow{\alpha} P'}{\Psi \rhd !P \xrightarrow{\alpha} P'}$$

3 A Simple Type System for ψ

Our type system extends that of the simply typed π-calculus of [23] in two ways. Firstly, we assign types to messages, and secondly, typability may depend on assertions.

3.1 Types and Type Environments

In type systems for process calculi with names, a type environment records the types of free names. Since typability in our setting can also depend on assertions, a type environment E can also contain assertions. We define the set of type environments by

$$E ::= E, x : T \mid E, \Psi \mid \emptyset$$

A type environment E is *well-formed*, written $E \vdash \diamond$, if it is a finite partial function from names to types such that if $E = E_1, \Psi, E_2$ and x is a name in Ψ, then $x \in \mathrm{dom}(E_1)$. We refer to the type annotations of E as $\mathsf{E}(E)$ and to the composed assertion $\bigotimes_{\Psi \in E} \Psi$ as $\Psi(E)$.

Definition 2. *We write $E_1 <_T E_2$ if $E_1 \vdash \diamond$, $E_2 \vdash \diamond$, $E_1 = E_{10}, E_{11}, E_{12}, \ldots, E_{1(k+1)}$ and $E_2 = E_{10}, u_1 : T_1, E_{11}, \ldots, u_k : T_k, E_{1k}, E_{1(k+1)}$ for some $u_1 : T_1, \ldots, u_k : T_k$.*

Thus, $E <_T E'$ if E' extends E with additional type annotations.

Definition 3. *Let E and E' be type environments. We write $E <_A^0 E'$ if $E' = E, \Psi$ for some assertion Ψ. We let $<_A$ denote the transitive closure of $<_A^0$ and let $<$ denote the least preorder containing $<_T \cup <_A$.*

3.2 Type Judgements and Type Rules

A type system for a ψ-calculus must have type judgements for each of the three nominal datatypes: $E \vdash M : T$, $E \vdash \varphi$ and $E \vdash \Psi$. Let $\mathcal{J}$ range over judgements, i.e.

$$\mathcal{J} ::= M : T \mid \varphi \mid \Psi$$

We consider only *qualified* judgements $\mathcal{J}_E$ of the form $E \vdash \mathcal{J}$ where $\mathrm{fn}(\mathcal{J}) \subseteq dom(E)$.

Definition 4. *Let $\mathcal{J}_E = E \vdash \mathcal{J}$ and $\mathcal{J}'_{E'} = E' \vdash \mathcal{J}'$ be qualified judgements. We write $\mathcal{J}_E < \mathcal{J}'_{E'}$ if $\mathcal{J} = \mathcal{J}'$ and $E < E'$.*

General Assumptions. We make a series of general assumptions concerning our type rules that are sufficient for establishing usual properties – in particular, the weakening and strengthening properties – of any concrete instance of our type system.

A natural assumption is that bindings in the type environment can be used:

$$(\textsc{Var}) \quad E \vdash x : T \quad \text{if } E(x) = T$$

Type rules have zero or more *premises* and a *conclusion* that are qualified judgements and may also include a *side condition*; a side condition is a predicate that is not a qualified type judgement but can depend on judgements in the rule. A side condition is dependent on the qualified judgements $\boldsymbol{\mathcal{J}_E}$ of the rule it occurs in and is therefore denoted $\chi(\boldsymbol{\mathcal{J}_E})$. We require the following to hold for all rules for terms and assertions.

- Every side condition χ must be *monotone* wrt. environment extensions. Let $\boldsymbol{\mathcal{J}_E}$ be an arbitrary instance of a rule. Suppose that whenever $\chi(\boldsymbol{\mathcal{J}_E})$ holds and $\boldsymbol{\mathcal{J}_E} < \boldsymbol{\mathcal{J}'_{E'}}$ for another rule instance $\boldsymbol{\mathcal{J}'_{E'}}$ then also $\chi(\boldsymbol{\mathcal{J}'_{E'}})$. Then χ is monotone.
- Every side condition χ must be *topical*; removing unnecessary type annotations will not affect the validity of a side condition. Let $\boldsymbol{\mathcal{J}_E}$ be an arbitrary instance of a rule, and suppose that whenever $\boldsymbol{\mathcal{J}_E} <_T \boldsymbol{\mathcal{J}'_{E'}}$ for some other instance $\boldsymbol{\mathcal{J}'_{E'}}$ and $\chi(\boldsymbol{\mathcal{J}'_{E'}})$ holds, then also $\chi(\boldsymbol{\mathcal{J}_E})$. Then χ is topical.
- Assertion typability must *respect composition* of environment assertions. If $E \vdash \Psi$, then $\mathsf{E}(E), \Psi(E) \vdash_E \Psi$.

Finally, we require *compositionality*.

- For composite assertions, we require that there is a compositional rule

$$\frac{E \vdash \Psi_1 \quad E \vdash \Psi_2}{E \vdash \Psi_1 \otimes \Psi_2}$$

- If a type rule types a composite condition $g(\varphi_1, \ldots, \varphi_k)$, it must be of the form

$$\frac{E \vdash \varphi_i \quad 1 \leq i \leq k \quad \chi}{E \vdash g(\varphi_1, \ldots, \varphi_k)}$$

- If a type rule types a composite term $f(M_1, \ldots, M_k)$, it must be of the form

$$\frac{E \vdash M_i : T_i \quad 1 \leq i \leq k \quad \chi}{E \vdash f(M_1, \ldots, M_k) : T}$$

We do not impose any constraints on how the result type arises from the types of immediate constituents, but we require channels to have the same type in environment E, if they are equivalent for an assertion that is well-typed wrt. E.

$$\text{If } E \vdash M : T \text{ , } E \vdash \Psi \text{ and } \Psi \models M \dot{\leftrightarrow} N \text{ then } E \vdash N : T \tag{1}$$

To deal with pattern matching in inputs, we introduce the following type rule for message patterns:

$$\text{(PATTERN)} \quad \frac{E, \boldsymbol{x} : \boldsymbol{T} \vdash M : U}{E \vdash (\lambda \boldsymbol{x}) M : \boldsymbol{T} \to U}$$

If an input abstraction has type $\boldsymbol{T} \to U$, it will receive any term of type U if this term contains pattern variables of types corresponding to $\boldsymbol{T}$.

Type Rules. The type rules are found in Table 2. In the rule (PAR), for each component P we collect the relevant assertions Ψ_P and type bindings E_P associated with bound names that may occur in the types and include these when typing the other component. In the rules (IN) and (OUT), the type of the subject M and the type of the object (the term transmitted on channel M) must be compatible wrt. a *compatibility predicate* $\leftrightharpoons$.

$$\text{(IN)} \quad \frac{\begin{array}{c} E, \boldsymbol{x} : \boldsymbol{T} \vdash P \\ E \vdash (\lambda \boldsymbol{x}) N : \boldsymbol{T} \to U_o \\ E \vdash M : U_s \end{array}}{E \vdash \underline{M}(\lambda \boldsymbol{x}) N.P} \qquad U_s \leftrightharpoons U_o$$

$$\text{(OUT)} \quad \frac{E \vdash M : T_s \quad E \vdash N : T_o \quad E \vdash P}{E \vdash \overline{M} N.P} \qquad T_s \leftrightharpoons T_o$$

$$\text{(PAR)} \quad \frac{E, E_{P_2}, \Psi_{P_2} \vdash P_1 \quad E, E_{P_1}, \Psi_{P_1} \vdash P_2}{E \vdash P_1 \mid P_2}$$

$$\text{(RES)} \quad \frac{E, x : T \vdash P}{E \vdash (\nu x : T) P} \qquad \text{(REP)} \quad \frac{E \vdash P}{E \vdash !P}$$

$$\text{(ASS)} \quad \frac{E \vdash \Psi}{E \vdash (\!|\Psi|\!)} \qquad \text{(CASE)} \quad \frac{E \vdash \varphi_i \quad E \vdash P_i \quad 1 \leq i \leq k}{E \vdash \textbf{case } \varphi_1 : P_1, \ldots, \varphi_k : P_k}$$

Table 2. Type rules for processes

Example 1. To capture a type system with channel types such as the original sorting system by Milner [17] we can let $\leftrightharpoons$ be defined by $T_1 \leftrightharpoons T_2$ if $T_1 = \mathbf{Ch}(T_2)$.

4 Properties of the Simple Type System

Type systems normally guarantee two properties of well-typed programs. A *subject reduction* property ensures that if a program is well-typed, it stays well-typed under

subsequent reduction steps. A *safety property* ensures that if a program is well-typed, a certain safety predicate will hold. We now consider these two properties in our setting.

4.1 Standard Lemmas

The following standard properties follow from the assumptions of compositionality, monotonicity and topicality.

Lemma 1. *The following properties hold for all instances that satisfy the general assumptions of Section 3.2.*

Substitution. *Let σ be a term substitution. If $E \vdash \mathcal{J}$, dom$(E) =$ dom(σ) and $E \vdash \sigma(x) : E(x)$ for all $x \in$ dom(σ) then $E \vdash \mathcal{J}\sigma$*

Interchange. *If $E, E_1, x : T, E_2 \vdash \mathcal{J}$ and $x \notin$ dom(E_1) and $x \notin \mathrm{fn}(E_1)$ then $E, x : T, E_1, E_2 \vdash \mathcal{J}$.*

Weakening. *If $E \vdash \mathcal{J}$ and $E, E' \vdash \diamond$ then $E, E' \vdash \mathcal{J}$*

Strengthening. *If $E, x : T_1 \vdash \mathcal{J}$ and $x \notin \mathrm{fn}(\mathcal{J})$ then $E \vdash \mathcal{J}$*

4.2 The Subject Reduction Property

Our type system guarantees a subject reduction property, namely that typability is preserved under τ-actions.

Theorem 1. *Suppose $E \vdash \Psi$ and $E \vdash P$ and that $\Psi \rhd P \stackrel{\tau}{\longrightarrow} P'$. Then also $E \vdash P'$.*

Proof. (Sketch) Induction in the labelled transition rules. We first establish a lemma for transitions not labelled with τ: that if $E \vdash \Psi$, $E \vdash P$ and $\Psi \rhd P \stackrel{\alpha}{\longrightarrow} P'$ where $\alpha \neq \tau$ and E' is any type environment such that $E, E' \vdash \alpha$ (i.e. the terms in α can be typed), then $E, E' \vdash P'$. The proof of the theorem then consists in an induction in the rules defining τ-transitions.

4.3 Safety in the Simple Type System

The notion of safety depends on the particular instantiation of the type system. However, all instantiations guarantee a general notion of channel safety.

Given a predicate on process configurations, $\mathsf{now}(E, \Psi, P)$, which defines a notion of *now-safety* of process P relative to a type environment E and an assertion Ψ. We can then define the general notion of safety as invariant now-safety.

Definition 5. *Given assertion Ψ, process P, type environment E and predicate* now*, $\Psi \rhd P$ is* safe *for E if for any P' where $\Psi \rhd P \stackrel{\tau}{\longrightarrow}{}^{*} P'$, we have that* $\mathsf{now}(E, \Psi, P')$.

Following this approach, we can show that an instance of the simple type system has a safety property by defining the property in terms of a suitable notion of now-safety and then showing that typability implies now-safety.

The side conditions of the type rules (IN) and (OUT) guarantee a general form of *channel safety*. This property guarantees that channels are always used to transmit messages whose type is compatible with that of the channel. Note that in Definition 6 we may need to extend the type environment; this is the case, since an input action may involve the reception of a term containing extruded names.

Definition 6. *Define* $\mathsf{now}_{\mathrm{Ch}}(E, \Psi, P)$ *to hold if when* $\Psi \rhd P \xrightarrow{\alpha} P'$ *and* $E \vdash \Psi$, *then*

1. *If* $\alpha = \overline{M}(\nu \boldsymbol{x} : \boldsymbol{T})N$ *and* $E \vdash M : U_s$ *, then we have* $E, \boldsymbol{x} : \boldsymbol{T}, \Psi_1 \otimes \cdots \otimes \Psi_k \vdash N : U_o$ *for some* $\Psi_1, \ldots, \Psi_k$ *and* U_o *where* $U_s \leftrightharpoons U_o$
2. *If* $\alpha = \underline{M}N$, *if* $\mathrm{fn}(N) = \boldsymbol{x}$ *and it is the case that* $E \vdash M : U_s$, *then for some* U_o, $\boldsymbol{T}$ *and* $\Psi_1, \ldots, \Psi_k$ *we have* $E, \boldsymbol{x} : \boldsymbol{T}, \Psi_1 \otimes \cdots \otimes \Psi_k \vdash N : U_o$, *such that* $U_s \leftrightharpoons U_o$.

We say that $\Psi \rhd P$ *is* now-channel safe for E.

Theorem 2. *If* $E \vdash P$ *and* $E \vdash \Psi$, *then* $\mathsf{now}_{\mathrm{Ch}}(E, \Psi, P)$.

Safety now follows from the subject reduction property of Theorem 1.

Theorem 3. *If* $E \vdash \Psi$ *and* $E \vdash P$ *then* $\Psi \rhd P$ *is channel-safe for* E.

5 Instances of the Simple Type System

We now describe how a series of new and existing type systems can be expressed as instances of our simple type system and how their safety properties can be established using the general results that we now have.

5.1 The Calculus of Explicit Fusions

In [4], a ψ-calculus translation is given of the calculus of explicit fusions of Wischik and Gardner [24]. No type systems were formulated for this calculus; here is one.

Fusions express that names a and b are considered equivalent and can therefore be represented as assertions. For any binary relation R, we let $\mathrm{EQ}(R)$ denote its reflexive, transitive and symmetric closure. The ψ-calculus representation of the calculus explicit fusions is now as follows.

The set of terms $\mathbf{T}$ is taken to be $\mathcal{N}$. The set of conditions $\mathbf{C}$ is the set of name identities of the form $a = b$ with $a, b \in \mathbf{T}$. The assertion set $\mathbf{A}$ is the family of finite sets of identities of the form $\{a_1 = b_1, \ldots, a_n = b_n\}$ for $n \geq 0$. Composition $\otimes$ is set union, the identity assertion $\mathbf{1}$ is $\emptyset$ and the entailment relation is defined by

$$\Psi \models a = b \text{ if } a = b \in \mathrm{EQ}(\Psi)$$

In the calculus of explicit fusions, a relevant notion of safety is that only names of the same type will ever be fused.

Definition 7. *P is now-safe wrt. E if for any assertion $a = b$ in P, $E(a) = E(b)$.*

To capture this, we use a notion of channel type. Names have types of the form

$$T ::= \mathbf{Ch}(T) \mid X$$

where X ranges over the set of *type variables* **TVar**. Actual types are defined by means of a *sorting*, which is a finite function $\Delta : \mathbf{TVar} \to \mathcal{T}$. Again, the compatibility relation is defined by $\mathbf{Ch}(T) \leftrightharpoons T$. The type rule for assertions is then as follows.

$$E \vdash \{a_1 = b_1, \ldots, a_n = b_n\} \quad \text{if } E(a_i) = E(b_i) \text{ for } 0 \leq i \leq n$$

For conditions, the type rules are similar:

$$E \vdash a = b \quad \text{if } E(a) = E(b) \qquad\qquad E \vdash a \dot{\leftrightarrow} b \quad \text{if } E(a) = E(b)$$

The safety result is now the following.

Theorem 4. *If $E \vdash P$, then P is safe wrt. E.*

5.2 The Distributed π-calculus

The distributed π-calculus was first proposed by Hennessy and Riely [20]. In Dπ, processes are located at named locations ($l[P]$) and can migrate to a named location ($\mathbf{go}\ k.P$). In our version P ranges over the set of processes and N over networks.

$$P ::= \mathbf{0} \mid \overline{a}\langle x\rangle.P_1 \mid a(x).P_1 \mid P_1 \mid P_2 \mid (\nu n : T)P_1 \mid !P_1 \mid \mathbf{go}\ k.P_1$$
$$N ::= \mathbf{0} \mid N_1 \mid N_2 \mid (\nu n : T)N_1 \mid l[P]$$

The labelled transition semantics, which involves a notion of structural congruence $\equiv$, has transitions of the form $P \xrightarrow{l@\alpha} P'$ for processes and $N \xrightarrow{\alpha} N'$ for networks, where either $\alpha = \overline{a}n$, $\alpha = an$ or $\alpha = \tau$.

Dπ can be represented as a ψ-calculus by a translation, due to Carbone and Maffeis [5]. The central insight is to view a channel a at location l as a composite term $l \cdot a$, so the set of terms is $\mathbf{T} = \{l \cdot a \mid l, a \in \mathcal{N}\}$. This set is not closed wrt. arbitrary substitutions, but the set of well-typed terms is closed under well-typed substitutions, which suffices. The set of conditions $\mathbf{C}$ is $\mathbf{C} = \{l \cdot a \dot{\leftrightarrow} l \cdot a \mid l, a \in \mathcal{N}\}$.

For networks, the translation is

$$[\![l[P]]\!] = [\![P]\!]_l \qquad\qquad [\![\mathbf{0}]\!] = \mathbf{1}$$
$$[\![N_1 \mid N_2]\!] = [\![N_1]\!] \mid [\![N_2]\!] \qquad\qquad [\![(\nu n)N_1]\!] = (\nu n)[\![N_1]\!]$$

For processes, we have

$$[\![\mathbf{go}\ k.P]\!]_l = [\![P]\!]_k \qquad\qquad [\![a(x).P]\!]_l = (l \cdot a)(x).[\![P]\!]_l$$
$$[\![\overline{a}\langle x\rangle.P]\!]_l = \overline{(l \cdot a)}\langle x\rangle.[\![P]\!]_l \qquad\qquad [\![P_1 \mid P_2]\!]_l = [\![P_1]\!]_l \mid [\![P_2]\!]_l$$
$$[\![!P_1]\!]_l = !\,[\![P_1]\!]_l \qquad\qquad [\![(\nu n)P]\!] = (\nu n)[\![P]\!]$$

The only assertion is the trivial assertion $\mathbf{1}$; we always have $E \vdash \mathbf{1}$.

Most type systems for Dπ are concerned about notions of channel safety. Here, we describe a type system that assigns types to both location and channel names. We therefore consider types of the form

$$T ::= \mathbf{Ch}(T) \mid \mathbf{Loc}\{a_i : \mathbf{Ch}(T_i)\}_{i \in I} \mid B$$

where I is a finite index set and B ranges over a set of base types. A location type $\mathbf{Loc}\{a_i : \mathbf{Ch}(T_i)\}_I$ describes the available interface of a location: only the specified

channel names can be used for communication at locations of this particular type. The type rule for composite names is then

$$\frac{E \vdash l : \mathbf{Loc}\{a_i : \mathbf{Ch}(T_i)\}_{i\in I} \quad E \vdash a_i : \mathbf{Ch}(T_i) \quad \text{for some } i \in I}{E \vdash l \cdot a_i : \mathbf{Ch}(T_i)}$$

where I is a finite set. The compatibility relation is given by $\mathbf{Ch}(T) \leftrightarrows T$

Definition 8. *A* Dπ *network* N *is now-safe wrt.* E *if whenever* $N \equiv (\nu \boldsymbol{k})l[P] \mid N'$ *and* $P \xrightarrow{l@\alpha}$ *where either* $\alpha = \overline{a}n$ *or* $\alpha = an$*, then* $E(l) = \boldsymbol{Loc}\{a_i : \boldsymbol{Ch}(T_i)\}_I$ *where* $a = a_i$ *and* $E(n) = T_i$ *for some* $i \in I$*.*

The following result follows from Theorem 3.

Theorem 5. *Let* $P = [\![N]\!]$ *for a* Dπ *network* N*. If* $E \vdash P$*, then* N *is safe wrt.* E*.*

5.3 A Type System for Secrecy

Next, we show how to represent a type system for secrecy in the spi calculus due to Abadi and Blanchet [2]. The version of the spi calculus used in [2] has primitives for asymmetric cryptography and its formation rules are

$$\begin{aligned} M &::= x \mid a \mid \{\boldsymbol{M}\}_M \\ P &::= \overline{M}\langle \boldsymbol{M}\rangle \mid a(x_1,\ldots,x_n).P \mid \mathbf{0} \mid P|Q \mid !P \mid (\nu a)P \\ &\quad \mid \mathbf{case}\ M\ \mathbf{of}\ \{x_1,\ldots,x_n\}_a : P\ \mathbf{else}\ Q \mid \mathbf{if}\ M = N\ \mathbf{then}\ P\ \mathbf{else}\ Q \end{aligned}$$

The term $\{\boldsymbol{M}\}_M$ denotes that the tuple $\boldsymbol{M} = (M_1,\ldots,M_k)$ is encrypted with key M. The process constructs resemble those of ψ-calculi, but output is asynchronous, and the subject of an input must be a *name* a (as opposed to a variable x). Similarly, in a decryption $\mathbf{case}\ M\ \mathbf{of}\ \{x_1,\ldots,x_n\}_k : P\ \mathbf{else}\ Q$, the key k must be a name. Thus, input and decryption capabilities cannot be transmitted – the latter lets us distinguish between private keys (only used for decryption) and public keys (only used for encryption).

The representation of this as a ψ-calculus is straightforward and based on [4]. We introduce conditions $M = \mathsf{enc}(*,k)$ and $M \neq \mathsf{enc}(*,k)$ whose intended meaning is that M is, resp. is not, encrypted using key k. Similarly, we introduce match conditions $M = N$ and mismatch $M \neq N$.

The decryption construct now becomes

$$(\nu \boldsymbol{x})(\mathbf{case}\ M = \mathsf{enc}(*,k) : (\!|\boldsymbol{x} = \mathsf{dec}(M,k)|\!) \mid P)\ ;\ M \neq \mathsf{enc}(*,k) : Q)$$

and the biconditional construct is $\mathbf{case}\ M = N : P\ ; M \neq N : Q$. Channel equality $\leftrightarrows$ is defined to be the identity relation on terms.

Abadi and Blanchet consider a notion of *secrecy under all opponents* [2]. If RW is a finite set of names and W a finite set of closed terms, then an (RW, W)-opponent is a spi-calculus process Q such that $Q = Q'[\boldsymbol{x} := \boldsymbol{N}]$ where $\mathrm{fn}(Q') \subseteq RW$, $W = \boldsymbol{N}$ and such that the set of free variables of Q' contains at most $\boldsymbol{x}$.

Definition 9. *Let RW be a finite set of names and W a finite set of closed terms.. Process P preserves the secrecy of M wrt. RW if whenever $P \xrightarrow{\tau}{}^{*} P'$ then it is not the case that $P' \xrightarrow{\overline{c}M} P''$ for any P'' or $c \in RW$. P preserves the secrecy of M from (RW, W) if $P|Q$ preserves the secrecy of M wrt. RW for any (RW, W)-opponent Q.*

The types given in [2] have the syntax

$$T ::= \mathsf{Pub} \mid \mathsf{Sec} \mid C^{\mathsf{Pub}}[\boldsymbol{T}] \mid C^{\mathsf{Sec}}[\boldsymbol{T}] \mid K^{\mathsf{Sec}}[\boldsymbol{T}] \mid K^{\mathsf{Pub}}[\boldsymbol{T}]$$

where $\boldsymbol{T}$ is any tuple of types. $C^{\mathsf{Pub}}[\boldsymbol{T}]$ and $C^{\mathsf{Sec}}[\boldsymbol{T}]$ are channel types for sending public, resp. secret, data and $K^{\mathsf{Sec}}[\boldsymbol{T}]$ and $K^{\mathsf{Pub}}[\boldsymbol{T}]$ are key types for encrypting these.

Abadi and Blanchet introduce a subtype relation which states that public types are subtypes of Pub (so e.g. $C^{\mathsf{Pub}}[\boldsymbol{T}] \leq \mathsf{Pub}$) and secret types are subtypes of Sec. Moreover, a special judgment $E \vdash M : \mathcal{S}$ describes the set of types $\mathcal{S}$ that M can have.

In our representation of the type system we must distinguish between names and variables. Our syntax of types is therefore

$$T ::= \mathsf{Pub} \mid \mathsf{Sec} \mid C^{A}_{B}[\boldsymbol{T}] \mid K^{A}_{B}[\boldsymbol{T}] \mid \boldsymbol{T}$$

where $A \in \{\mathsf{Pub}, \mathsf{Sec}\}$ and $B \in \{\mathsf{Name}, \mathsf{Var}\}$. We capture the subtype relation and the type set judgment by rules including the following.

$$\text{(PubVar)} \quad \frac{E(x) = \mathsf{Pub}}{E \vdash x : C^{\mathsf{Pub}}_{\mathsf{Var}}[\boldsymbol{T}]} \qquad \text{(KeyP)} \quad \frac{E \vdash M : C^{\mathsf{Pub}}_{\mathsf{Var}}[\boldsymbol{T}]}{E \vdash M : \mathsf{Pub}}$$

The original type rules for the decryption construct are now captured via type rules for assertions, two of which are shown below.

$$\text{(Dec-PK1)} \quad \frac{\begin{array}{c} E \vdash M : \mathsf{Pub} \quad E(k) = K^{\mathsf{Pub}}_{\mathsf{Name}}[\boldsymbol{T}] \\ E(x_i) = T_i \\ \text{for } 1 \leq i \leq |\boldsymbol{x}| \quad |\boldsymbol{T}| = |\boldsymbol{x}| \end{array}}{E \vdash \boldsymbol{x} = \mathsf{dec}(M, k)} \qquad \text{(Equal)} \quad \frac{E \vdash M : T \quad E \vdash N : T}{E \vdash M = N}$$

For the assertion $M = N$, the type system of [2] allows $\mathbf{if}\ M = N\ \mathbf{then}\ P\ \mathbf{else}\ Q$ to be well-typed if $E \vdash M : S_1$ and $E \vdash N : S_2$ but $S_1 \cap S_2 = \emptyset$. This is not allowed in our type system; our rules mirror the stronger requirement (originally made in [1]) that $S_1 \cap S_2 \neq \emptyset$. The remaining assertions are defined to be always well-typed.

For the input and output rules the original type system allows channels of type $C^{\mathsf{Pub}}_{B}[\boldsymbol{T}]$ to transmit either messages of type Pub or a tuple of type $\boldsymbol{T}$. We can capture this by defining the compatibility predicate as follows (where **Pub** stands for a tuple type of arbitrary length, all of whose components are Pub).

$$C^{\mathsf{Pub}}_{B}[\boldsymbol{T}] \leftharpoonup \mathbf{Pub} \qquad C^{\mathsf{Pub}}_{B}[\boldsymbol{T}] \leftharpoonup \boldsymbol{T} \qquad C^{\mathsf{Sec}}_{B}[\boldsymbol{T}] \leftharpoonup \boldsymbol{T}$$

The secrecy result for the type system is the following.

Theorem 6 ([2]). *Let P be a closed process. Suppose that $E \vdash P$ and $E \vdash s : \mathsf{Sec}$. Let*

$$RW = \{a \mid E(a) = \mathsf{Pub}\} \quad W = \{a' \mid E(a') = C^{\mathsf{Pub}}[\cdots] \text{ or } E(a') = K^{\mathsf{Pub}}[\cdots]\}$$

Then P preserves the secrecy of s from (RW, W).

We obtain this safety result from Theorem 3 by first noticing that channel safety implies that if a process R satisfies that $E \vdash R$, then a secret name s cannot be leaked on any channel in RW as defined. To conclude the proof, now apply the result (established in [2]) that $E \vdash P \mid Q$ for any (RW, W)-opponent Q.

5.4 Correspondence Types for Authenticity

Next, we represent a type and effect system for a spi-calculus capturing non-injective authenticity using correspondence assertions[9]. This case shows how the typability of processes and terms can depend on assertions.

We now consider symmetric encryption, and the decryption operation is written as $\mathbf{case}\ M\ \mathbf{of}\ \{x_1, \ldots, x_n\}_M : P\ \mathbf{else}\ Q$, where the key M can be an arbitrary term.

We introduce *correspondence assertions* **begin** $\ell(M)$ and **end** $\ell(M)$; these are labelled message terms where ℓ ranges over a set of labels disjoint from the set of message terms. In a process, **begin** assertions are placed where authentication is to be initiated, and **end** assertions are placed where authentication should be completed.

The set of message terms is defined by the formation rules

$$M ::= x \mid (M_1, M_2) \mid \{M_1\}_{M_2} \mid \mathbf{fst}\ M \mid \mathbf{snd}\ M \mid \mathbf{ok}$$

where $\mathbf{fst}\ M$ and $\mathbf{snd}\ M$ extract the first, respectively second coordinate of a pair M. The $\mathbf{ok}$ term is an *explicit effect term* used to transfer the capability to match **begin** assertions. $\mathbf{ok}$ terms and correspondence assertions do not influence process behaviour; their only role is in the type system.

Definition 10. *A process P is* safe *for type environment E if whenever $P \xrightarrow{\tau}{}^{*} (\nu \boldsymbol{n})(\textbf{\textit{end}}\ \ell(M) \mid P')$, then either we have $P' \equiv (\!|\textbf{\textit{begin}}\ \ell(M)|\!) \mid P^{(2)}$ or $\textbf{\textit{begin}}\ \ell(M) \in E$. An* opponent *is a spi calculus process Q that has no* $\mathbf{end}$ *assertions and where every term in P can be typed with opponent type* $\mathbf{Un}$*. Process P is* robustly safe *if $P \mid Q$ is safe for any opponent Q.*

Here, $\equiv$ is the structural congruence relation of the spi calculus [3]. Note that the safety property employs the following notion of now-safety: P is now-safe, if every end-assertion can be matched by a begin-assertion with the same label.

The important property is similar to that for the secrecy type system: if P can be well-typed when all its free variables have opponent type $\mathbf{Un}$, then P is robustly safe.

Theorem 7. *[9] If $x_1 : \mathbf{Un}, \ldots, x_k : \mathbf{Un} \vdash P$ where $\{x_1, \ldots, x_k\} = \mathrm{fv}(P)$, then P is robustly safe.*

The representation of the spi calculus is as in Section 5.3, and we can keep the original types of [9].

$$T ::= \mathbf{Ch}(T) \mid \mathbf{Pair}(x : T_1, T_2) \mid \mathbf{Ok}(S) \mid \mathbf{Un}$$

Here $\mathbf{Pair}(x : T_1, T_2)$ is a dependent pair type, $\mathbf{Ok}(S)$ is an ok-type, where S is a finite set of assertions called an *effect*, and $\mathbf{Un}$ is an *opponent type*. We incorporate opponent types by defining the compatibility relation by $\mathbf{Un} \leftrightharpoons \mathbf{Un}$.

Correspondence assertions **begin** $\ell(M)$ and **end** $\ell(M)$ are added to the assertions $\mathbf{A}$ of Section 5.3. Their type rules are defined below. All other assertions are assumed to be always well-typed. Note that there are two rules for end-assertions, since effects can either occur directly in the type environment or be hidden within an ok-type. Also note that these rules show how typability can depend on assertions in the type environment.

$$\text{(BEGIN)}\ E \vdash \textbf{begin}\ \ell(M) \qquad \text{(END-1)}\ \frac{E = E_1, \ell(M), E_2 \quad \mathrm{fn}(M) \subseteq \mathrm{dom}(E_1)}{E \vdash \textbf{end}\ \ell(M)}$$

$$\text{(END-2)}\ \frac{E = E_1, x : \mathbf{Ok}(S), E_2 \quad \ell(M) \in S}{E \vdash \textbf{end}\ \ell(M)} \qquad \mathrm{fn}(M) \subseteq \mathrm{dom}(E_1)$$

Some of the type rules for terms are shown below.

$$\text{(ENC)}\ \frac{E \vdash M : T \quad E \vdash N : \mathbf{Key}(T)}{E \vdash \{M\}_N : \mathbf{Un}} \qquad \text{(OK)}\ \frac{E \vdash \diamond \quad E \vdash \psi \quad \forall \psi \in S}{E \vdash \mathbf{ok} : \mathbf{Ok}(S)}$$

$$\text{(ENC UN)}\ \frac{E \vdash M : \mathbf{Un} \quad E \vdash N : \mathbf{Un}}{E \vdash \{M\}_N : \mathbf{Un}} \qquad \text{(OK UN)}\ \frac{E \vdash \diamond}{E \vdash \mathbf{ok} : \mathbf{Un}}$$

The authenticity result of Theorem 7 can now be established by Theorem 1 combined with a lemma that every well-typed process is now-safe wrt. correspondences. This is easily proved by induction in the type derivation of a well-typed P.

6 Conclusions and Further Work

In this paper we have presented a simple type system for ψ-calculi where term types belong to a nominal data type and judgements for processes are of the form $E \vdash P$ and given by a fixed set of rules. Terms, assertions and conditions are assumed to form nominal datatypes, and only a few requirements on type rules are imposed, such as compositionality and substitutivity. The type system lets us represent existing type systems for secrecy and authenticity in the spi calculus and location safety in the distributed π-calculus and also gives rise to a first type system for the calculus of explicit fusions.

The type system represents forms of channel subtyping by a compatibility relation and deals with opponent typability, but a more general account of subtyping is another important line of future research.

Other type systems in the literature deal with *resource-aware* properties such as linearity or receptiveness of names [14,22], or notions of termination [7,15]. Common to these systems is that the rules for parallel composition and prefixes are modified and the

use of replication is limited; most often by only allowing replicated inputs. An extension of our work to such type systems is a topic of ongoing work.

References

1. Abadi, M.: Secrecy by typing in security protocols. J. ACM 46(5), 749–786 (1999)
2. Abadi, M., Blanchet, B.: Secrecy types for asymmetric communication. Theor. Comput. Sci. 298(3), 387–415 (2003)
3. Abadi, M., Gordon, A.D.: A calculus for cryptographic protocols: the spi calculus. In: Proc. CCS 1997, pp. 36–47. ACM, New York (1997)
4. Bengtson, J., Johansson, M., Parrow, J., Victor, B.: Psi-calculi: Mobile Processes, Nominal Data, and Logic. In: Proc. of LICS 2009, pp. 39–48. IEEE, Los Alamitos (2009)
5. Carbone, M., Maffeis, S.: On the expressive power of polyadic synchronisation in π-calculus. Nordic Journal of Computing 10(2), 70–98 (2003)
6. Cardelli, L., Gordon, A.D.: Mobile ambients. In: Nivat, M. (ed.) FOSSACS 1998. LNCS, vol. 1378, p. 140. Springer, Heidelberg (1998)
7. Deng, Y., Sangiorgi, D.: Ensuring termination by typability. Inf. Comput. 204(7), 1045–1082 (2006)
8. Elsborg, E., Hildebrandt, T., Sangiorgi, D.: Type systems for bigraphs. In: Kaklamanis, C., Nielson, F. (eds.) TGC 2008. LNCS, vol. 5474, pp. 126–140. Springer, Heidelberg (2009)
9. Fournet, C., Gordon, A.D., Maffeis, S.: A type discipline for authorization policies. ACM Trans. Program. Lang. Syst. 29(5) (2007)
10. Gabbay, M.J., Mathijssen, A.: Nominal (universal) algebra: Equational logic with names and binding. J. Log. Comput. 19(6), 1455–1508 (2009)
11. Gordon, A.D., Jeffrey, A.: Authenticity by typing for security protocols. J. Comput. Secur. 11(4), 451–519 (2003)
12. Honda, K.: Composing processes. In: Proc. of POPL 1996, pp. 344–357. ACM, New York (1996)
13. Igarashi, A., Kobayashi, N.: A generic type system for the Pi-calculus. Theor. Comput. Sci. (11), 121–163 (2004)
14. Kobayashi, N., Pierce, B.C., Turner, D.N.: Linearity and the pi-calculus. ACM Trans. Program. Lang. Syst. 21(5), 914–947 (1999)
15. Kobayashi, N., Sangiorgi, D.: A hybrid type system for lock-freedom of mobile processes. ACM Trans. Program. Lang. Syst. 32(5) (2010)
16. Makholm, H., Wells, J.B.: Instant polymorphic type systems for mobile process calculi: Just add reduction rules and close. In: Sagiv, M. (ed.) ESOP 2005. LNCS, vol. 3444, pp. 389–407. Springer, Heidelberg (2005)
17. Milner, R.: The polyadic pi-calculus: a tutorial, pp. 203–246. Springer, Heidelberg (1993)
18. Milner, R.: The Space and Motion of Communicating Agents. Cambridge University Press, Cambridge (2009)
19. Pierce, B.C., Sangiorgi, D.: Typing and subtyping for mobile processes. Mathematical Structures in Computer Science 6(5), 409–453 (1996)
20. Riely, J., Hennessy, M.: A typed language for distributed mobile processes. In: Proceedings of POPL 1998, pp. 378–390. ACM, New York (1998)
21. Riely, J., Hennessy, M.: Trust and partial typing in open systems of mobile agents. J. Autom. Reasoning 31(3-4), 335–370 (2003)
22. Sangiorgi, D.: The name discipline of uniform receptiveness. Theor. Comput. Sci. 221(1-2), 457–493 (1999)
23. Sangiorgi, D., Walker, D.: The π-Calculus - A Theory of Mobile Processes. Cambridge University Press, Cambridge (2001)
24. Wischik, L., Gardner, P.: Explicit fusions. Theor. Comput. Sci. 340(3), 606–630 (2005)