The Challenge of Typed Expressiveness in Concurrency

Jorge A. Pérez

The Challenge of Typed Expressiveness in Concurrency

Jorge A. Pérez

University of Groningen, The Netherlands

Abstract. By classifying behaviors (rather than data values), *behavioral types* abstract structured protocols and enforce disciplined message-passing programs. Many different behavioral type theories have been proposed: they offer a rich landscape of models in which types delineate concurrency and communication. Unfortunately, studies on formal relations between these theories are incipient. This paper argues that clarifying the *relative expressiveness* of these type systems is a pressing challenge for formal techniques in distributed systems. We overview works that address this issue and discuss promising research avenues.

1 Introduction

Communication and *types* are increasingly relevant in (concurrent) programming. To bear witness of this trend, several languages promoted by industry offer advanced type systems (or type-based analysis tools) and/or support (message-passing) communication. For instance, Facebook's Flow [1] is a type checker for JavaScript based on gradual typing; Mozilla's Rust [4] exploits affine, ownership types to balance safety and control; Google's Go [3] supports process concurrency and channel-based communication. Other languages (e.g., Erlang [2]) also offer forms of (message-passing) communication.

If communication and types are here to stay, on what foundations languages integrating both features should rest? Much research within formal techniques in distributed systems has been devoted to models for concurrency and communication. In particular, *process calculi* have been widely promoted as a basis for type systems for concurrent programs. Indeed, building upon the π-calculus, a variety of *behavioral type systems* have been put forward [24,35]: by classifying behaviors (rather than values), these type structures abstract structured protocols and enforce disciplined message-passing programs. Existing work suggests that rather than a shortage of foundations for types and communication, we have the opposite problem: there are many formal foundations and it is unclear how to build upon them to transfer analysis techniques into practice.

The current situation calls for rigorous comparisons between well-established (but distinct) behavioral typed frameworks. Besides revealing bridges between different models of typed processes, such comparisons should clarify the complementarities and shortcomings of analysis techniques based on types. Developing a theory of *typed expressiveness* is thus a challenge for the specification and

analysis of distributed systems. The consolidation of *communication-centered* software systems (collections of interacting, heterogeneous services) and the renewed interest of software practitioners in communication and types endow this research challenge with practical significance.

We argue that the much needed formal comparisons may draw inspiration from results and frameworks for *relative expressiveness*, as studied in concurrency theory. This area has elucidated important formal relations between *untyped* process languages (see [49] for a survey); it thus appears as an adequate basis for analogous formal results in a typed setting.

This short paper is organized as follows. Next we overview main achievements in untyped expressiveness (§ 2). Then, we briefly review expressiveness results that consider behavioral types and/or behaviorally typed processes (§ 3). We conclude by discussing promising research directions (§ 4).

2 Expressiveness in Concurrency: The Untyped Case

Important studies of expressiveness in concurrency concern *process calculi*, a well-established approach to the analysis of concurrent systems. These formalisms, in particular the π-calculus, also provide a basis for typed systems for communicating programs. Process calculi promote the view of *concurrency as communication*: they abstract the interactions of a system by means of atomic communication actions. This view is simple and powerful, and has deep ramifications to *computing* at large.

The process calculi approach to concurrency has proved prolific. Starting from a few "basic" process calculi (including CSP, CCS, the π-calculus, amongst others), many extensions have been put forward, following modeling requirements (e.g., quantitative information) but also the need of representing forms of interaction typical of application domains such as, e.g., systems biology, security, and service-oriented computing. Given this *jungle* of process models [45], the need for establishing formal relationships between them was soon evident. To rise to this challenge, since the early 1990s a sustained body of work has assessed the *expressiveness* of process languages. These studies may pertain, for instance, to the nature of a process construct (e.g., whether it can be implemented using more basic ones) or to the transfer of reasoning techniques (e.g., process equivalences) across languages.

The main device used in expressiveness studies is the notion of *encoding*: a translation of a *source language* $\mathcal{L}_S$ into a *target language* $\mathcal{L}_T$ that satisfies certain properties. Such properties are usually defined as *syntactic and semantic criteria*; they are abstract indicators of an encoding's quality. Two common criteria are *homomorphism with respect to parallel composition*, which promotes compositional encodings, and *operational correspondence*, which relates the (visible) behavior of source and target processes. Abstract formulations of encodings and their criteria have been put forward (cf. [30]). An encoding of $\mathcal{L}_S$ into $\mathcal{L}_T$ (a *positive* encodability result) indicates that $\mathcal{L}_T$ is at least as expressive as $\mathcal{L}_S$: all behaviors expressible in $\mathcal{L}_S$ can also be expressed in $\mathcal{L}_T$. The non existence

of an encoding (a *negative* encodability result) may indicate that $\mathcal{L}_S$ is more expressive than $\mathcal{L}_T$: there are behaviors expressible in $\mathcal{L}_S$ but not in $\mathcal{L}_T$.

Encodability results have clarified our understanding of notions of interaction and communication as abstracted by process calculi. They have been crucial to:

a) *Formally relate different computational models.* Two notable examples are (i) the encodability of the λ-calculus in the π-calculus [44], and (ii) decidability results for termination/convergence for CCS, which distinguish the expressive power of processes with replication and recursion [6].
b) *Assess the expressive power of process languages.* As an example, several influential works have clarified the interplay of choice operators and (a)synchronous communication, and its influence on the expressive power of the π-calculus (see, for instance, [48]).
c) *Transfer proof techniques between different calculi.* For instance, the fully abstract encodability of process mobility (process passing) into name mobility has important consequences on the behavioral theory of higher-order process calculi [50].

Expressiveness results can be useful in the design of concurrent languages. For instance, the formalization of compilers usually requires encodings and behavior-preserving translations. Similarly, the development of type systems for process calculi, in particular *behavioral types*, has further reconciled process models and actual concurrent languages.

3 Towards Relative Expressiveness for Behavioral Types

The development of process languages with verification techniques based on *type systems* has received much attention. From Milner's *sortings* [43] until recently discovered logic foundations of concurrency [9,52]—passing through, e.g., *graph types* [53], *linear types* [39], *session types* [31], and *generic types* [36]—type systems have revealed a rich landscape of concurrent models with disciplined communication.

Within these different type systems, behavioral types stand out by their ability to abstract notions such as causality, alternatives (choices), and repetition [35]. A behavioral type defines the resource-usage policy of a communication channel, but also describes a series of actions realized through that channel along time. Behavioral types are often defined on top of process calculi; this enables the definition of general verification techniques that may be then adapted to different languages. Given this jungle of *typed* process models, the need for establishing formal relations between them rises again. Mirroring the achievements of untyped expressiveness (cf. (a)–(c) above), we believe that research on formal methods for distributed systems would benefit from efforts aimed at:

i) Formally relating different typed models.
ii) Assessing the expressiveness of languages governed by behavioral types.
iii) Transferring analysis techniques between different typed languages and formalisms.

These are challenging issues: as different type systems rely on unrelated concepts, discrepancies on typability arise easily. Hence, objective comparisons are hard to establish. Next we briefly review some works that address (i)–(iii) from various angles. Most of them concern session types, one of the most representative classes of behavioral types. Session types abstract structured communications (protocols) by allowing sequential actions (input and output), labeled choices (internal and external), and recursion. *Binary* session types [31] abstract protocols between two partners; *multiparty* session types [32] overcome this limitation: a *global type* (*choreography*) offers a high-level perspective for the *local types* realized by each partner. Different formulations for binary/multiparty session types have been developed, with varying motivations [35].

Linear Types and Session Types. Kobayashi appears to be the first to have related distinct type disciplines for the π-calculus by encoding the (finite) session π-calculus into a π-calculus with linear types with usages and variant types [38, §10]. His encoding represents a session name by multiple linear channels, using a continuation-passing style. Since Kobayashi does not explore the correctness properties of his encoding, Dardha et al. [17] revisit it by establishing type and operational correspondences and by giving extensions with subtyping, polymorphism, and higher-order communication. An extension of [17] with recursion in processes and types is reported in [16].

In a similar vein as [17], Demangeon and Honda [19] encode a session π-calculus into a linear/affine π-calculus with subtyping based on choice and selection types, which generalize input and output types, resp. Their encoding preserves subtyping. In contrast to [17], the encoding in [19] is fully abstract up to may/must barbed congruences. Their linear/affine π-calculus can also encode a call-by-value λ-calculus.

Gay et al. [28] encode a monadic π-calculus with (finite) binary session types (as in [27]) into a polyadic π-calculus with generic process types (as in [36], instantiated with linear, race-free communications). They aim at retaining original constructs in both models and to ensure operational and type correspondences. Their encoding enjoys these correspondences, but does not preserve subtyping. The authors notice that encoding labeled choice into guarded, unlabeled summation is challenging; also, that the target languages considered in [17,19] admit simple encodings of labeled choice.

Concerning transfer of techniques, Carbone et al. [11] use the encoding in [17] to relate *progress* in the session π-calculus to *lock-freedom* [37] in the linear π-calculus. This path to progress in binary sessions is an improvement with respect to previous works [25,12], as more session processes with progress are accepted as well-typed.

Session Types and Automata-like Models. Just as (untyped) process calculi have been related to sequential models of computation to characterize (un)decidability properties (therefore identifying expressiveness gaps—see, e.g., [6]), session types have been related to automata-like models. Initial results are due to Villard [51] who, in the context of dual channel contracts, shows that a subclass

of communicating finite state machines (CFSMs) characterizes binary session types. Deniélou and Yoshida [22] extend this characterization to the multiparty setting by giving a class of generalized multiparty session types and their interpretation into CFSMs. This class inherits key safety and liveness properties from CFSMs. The work [23] complements [22] by offering a sound and complete characterization of multiparty session types as CFSMs, identifying the key notion of multiparty compatibility. Lange et al. [41] extend this work by developing techniques to synthesize global graphs from CFSMs, covering more global protocols than [22,23]. Fossati et al. [26] overcome modeling limitations of textual multiparty session types through an integration with Petri nets. Using Petri nets token semantics they define a conformance relation between syntactic local types and *session nets*.

Types in the ψ-calculi. The ψ-calculi [5] are a family of name-passing process calculi that extends the π-calculus with terms, conditions, and assertions. Generality arises by treating terms as subjects in communication prefixes and by allowing input patterns. Hüttel [33] proposes a type system for ψ-calculi that generalizes simple types for the π-calculus; it enjoys generic formulations of type soundness and type safety. Instantiations are shown for other process languages, including the fusion calculus, the distributed π (Dπ), and the spi-calculus. The work [34] extends [33] to the case of resource-aware type systems: it proposes a type system for ψ-calculi that subsumes the system for linear channel usage of [39], the action types of [54], and a type system similar to that in [21].

Other Approaches. López et al. [42] encode binary session processes into a declarative process calculus, enabling the transfer of LTL reasoning to scenarios of session communications. Cano et al. [10] encode a session π-calculus into declarative process calculus with explicit linearity and establish strong operational correspondences. Padovani [47] relates labeled choice operators in session types with intersection and union types. Orchard and Yoshida [46] encode PCF with an effect type system into a session π-calculus; a reverse encoding that embeds the session π-calculus into PCF with concurrency primitives is used to offer a new implementation of sessions into Haskell.

4 Concluding Remarks and Future Directions

The works discussed above define insightful relations between different frameworks of behavioral types. Although some works define the transfer of techniques between typed models and/or enable practical applications (cf. [11,46,42]), most existing works define isolated studies: since each work uses different techniques to relate typed models, further research is needed to integrate their results. There are promising research directions towards this ambitious goal. For space reasons, below we discuss only three of them.

4.1 Expressiveness Without Processes

Using process calculi to study behavioral types allows to establish their properties by exploiting well-established process techniques. In practical multiparty communications [13,23,41], however, one is mostly interested in types (rather than in processes), as they offer contracts/interfaces to derive safe implementations. These works use process calculi techniques to define, e.g., operational semantics and equivalences for global types. If one focuses on types then expressiveness results should relate different behavioral types and their semantics. Demangeon and Yoshida [20] develop this approach: they contrasted the expressivity of different models of multiparty session types by comparing their trace-based denotations. As global types become richer in structure, comparisons will need to use techniques from relative expressiveness. The recent work of Cruz-Filipe and Montesi [15] goes in this direction.

4.2 A Logic Yardstick for Typed Expressiveness

The variety of behavioral types rises a natural question: what should be a "fair yardstick" to compare them? Recently developed Curry-Howard correspondences for concurrency [9,52,14] define a fundamental model of behavioral types: by relating session types and linear logic, they identify type preserving and deadlock-free process models.

Some works connect [9,52] and other models. Wadler [52] gives a deadlock-free fragment of the session-typed functional language in [29] by encodings into concurrent processes. The work [18] contrasts two classes of deadlock-free, session processes: the first class results from [9,52]; the other results by combining the encoding in [17] and usage types [37]. The former class is shown to be strictly included in the latter. The work [8,7] defines an analysis of multiparty session types (lock-free choreographies, as in [23]) using techniques for binary session types (as in [9]). Further work should relate [9,52,14] and other behavioral types.

4.3 Typed Encodability Criteria

Related to § 4.1, another direction is clarifying how existing frameworks for relative expressiveness (such as [30]) can account for behavioral types. This is needed to understand how types induce encodability results, but also to relate different behavioral types in source/target languages.

In their recent study of the relative expressivity of higher-order session communication, Kouzapas et al. [40] define a notion of typed encoding that extends [30] with session types. This work considers translations of processes and types and two new encodability criteria: type soundness and type preservation. The former says that translations preserve typability (well-typed source processes are encoded into well-typed target processes), the latter says that encodings of types preserve the structure of the session protocol, as represented by session type operators. While demanding and focused to the case of binary session types, these criteria are shown to be appropriate in classifying session-based

π-calculi. Further work is needed to generalize/relax these typed encodability criteria, so to consider source and target languages with different type systems.

Acknowledgments. I would like to thank Ilaria Castellani, Ornela Dardha, Mariangiola Dezani-Ciancaglini, Dimitrios Kouzapas, Hugo A. López, and Camilo Rueda for their useful feedback on previous drafts of this paper.

References

1. Flow: A Static Type Checker for JavaScript, http://flowtype.org
2. The Erlang Programming Language, http://www.erlang.org
3. The Go Programming Language, https://golang.org
4. The Rust Programming Language, https://www.rust-lang.org
5. 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 Computer Society (2009), http://doi.ieeecomputersociety.org/10.1109/LICS.2009.20
6. Busi, N., Gabbrielli, M., Zavattaro, G.: On the expressive power of recursion, replication and iteration in process calculi. Math. Struc. in Comp. Sci. 19(6), 1191–1222 (2009), http://dx.doi.org/10.1017/S096012950999017X
7. Caires, L., Pérez, J.A.: A typeful characterization of multiparty structured conversations based on binary sessions. CoRR abs/1407.4242 (2014), http://arxiv.org/abs/1407.4242
8. Caires, L., Pérez, J.A.: Multiparty session types within a canonical binary theory, and beyond. In: Proc. of FORTE 2016. LNCS, Springer (2016), to appear
9. Caires, L., Pfenning, F.: Session types as intuitionistic linear propositions. In: CONCUR'2010. LNCS, vol. 6269, pp. 222–236. Springer (2010)
10. Cano, M., Rueda, C., López, H.A., Pérez, J.A.: Declarative interpretations of session-based concurrency. In: Proc. of PPDP 2015. pp. 67–78. ACM (2015), http://doi.acm.org/10.1145/2790449.2790513
11. Carbone, M., Dardha, O., Montesi, F.: Progress as compositional lock-freedom. In: Proc. of COORDINATION 2014. LNCS, vol. 8459, pp. 49–64 (2014), http://dx.doi.org/10.1007/978-3-662-43376-8_4
12. Carbone, M., Debois, S.: A graphical approach to progress for structured communication in web services. In: Proc. of ICE 2010. EPTCS, vol. 38, pp. 13–27 (2010), http://dx.doi.org/10.4204/EPTCS.38.4
13. Carbone, M., Montesi, F.: Deadlock-freedom-by-design: multiparty asynchronous global programming. In: Proc. of POPL '13. pp. 263–274. ACM (2013), http://doi.acm.org/10.1145/2429069.2429101
14. Carbone, M., Montesi, F., Schürmann, C., Yoshida, N.: Multiparty session types as coherence proofs. In: Proc. of CONCUR 2015. LIPIcs, vol. 42, pp. 412–426 (2015), http://dx.doi.org/10.4230/LIPIcs.CONCUR.2015.412
15. Cruz-Filipe, L., Montesi, F.: Choreographies, computationally. CoRR abs/1510.03271 (2015), http://arxiv.org/abs/1510.03271
16. Dardha, O.: Recursive session types revisited. In: Proc. of BEAT 2014. EPTCS, vol. 162, pp. 27–34 (2014), http://dx.doi.org/10.4204/EPTCS.162.4
17. Dardha, O., Giachino, E., Sangiorgi, D.: Session types revisited. In: Proc. of PPDP 2012. pp. 139–150. ACM (2012), http://doi.acm.org/10.1145/2370776.2370794
18. Dardha, O., Pérez, J.A.: Comparing deadlock-free session typed processes. In: Proc. of EXPRESS/SOS. EPTCS, vol. 190, pp. 1–15 (2015), http://dx.doi.org/10.4204/EPTCS.190.1

19. Demangeon, R., Honda, K.: Full abstraction in a subtyped pi-calculus with linear types. In: Proc. of CONCUR 2011. LNCS, vol. 6901, pp. 280–296. Springer (2011), http://dx.doi.org/10.1007/978-3-642-23217-6_19
20. Demangeon, R., Yoshida, N.: On the expressiveness of multiparty sessions. In: Proc. of FSTTCS 2015. LIPIcs, vol. 45, pp. 560–574. Schloss Dagstuhl (2015), http://dx.doi.org/10.4230/LIPIcs.FSTTCS.2015.560
21. Deng, Y., Sangiorgi, D.: Ensuring termination by typability. Inf. Comput. 204(7), 1045–1082 (2006)
22. Deniélou, P., Yoshida, N.: Multiparty session types meet communicating automata. In: Proc. of ESOP 2012. LNCS, vol. 7211, pp. 194–213. Springer (2012), http://dx.doi.org/10.1007/978-3-642-28869-2_10
23. Deniélou, P., Yoshida, N.: Multiparty compatibility in communicating automata: Characterisation and synthesis of global session types. In: Proc. of ICALP 2013. LNCS, vol. 7966, pp. 174–186. Springer (2013), http://dx.doi.org/10.1007/978-3-642-39212-2_18
24. Dezani-Ciancaglini, M., de'Liguoro, U.: Sessions and session types: An overview. In: WS-FM 2009. LNCS, vol. 6194, pp. 1–28. Springer (2010)
25. Dezani-Ciancaglini, M., de'Liguoro, U., Yoshida, N.: On progress for structured communications. In: Proc. of TGC 2007. LNCS, vol. 4912, pp. 257–275. Springer (2008), http://dx.doi.org/10.1007/978-3-540-78663-4_18
26. Fossati, L., Hu, R., Yoshida, N.: Multiparty session nets. In: Proc. of TGC 2014. LNCS, vol. 8902, pp. 112–127. Springer (2014), http://dx.doi.org/10.1007/978-3-662-45917-1_8
27. Gay, S., Hole, M.: Subtyping for session types in the pi calculus. Acta Inf. 42, 191–225 (2005), http://portal.acm.org/citation.cfm?id=1104643.1104646
28. Gay, S.J., Gesbert, N., Ravara, A.: Session types as generic process types. In: Proc. of EXPRESS 2014 and SOS 2014. EPTCS, vol. 160, pp. 94–110 (2014), http://dx.doi.org/10.4204/EPTCS.160.9
29. Gay, S.J., Vasconcelos, V.T.: Linear type theory for asynchronous session types. J. Funct. Program. 20(1), 19–50 (2010), http://dx.doi.org/10.1017/S0956796809990268
30. Gorla, D.: Towards a unified approach to encodability and separation results for process calculi. Inf. Comput. 208(9), 1031–1053 (2010), http://dx.doi.org/10.1016/j.ic.2010.05.002
31. Honda, K., Vasconcelos, V.T., Kubo, M.: Language primitives and type discipline for structured communication-based programming. In: ESOP'98. LNCS, vol. 1381, pp. 122–138. Springer (1998)
32. Honda, K., Yoshida, N., Carbone, M.: Multiparty asynchronous session types. In: POPL '08. pp. 273–284. ACM (2008)
33. Hüttel, H.: Typed ψ-calculi. In: Proc. of CONCUR 2011. LNCS, vol. 6901, pp. 265–279. Springer (2011), http://dx.doi.org/10.1007/978-3-642-23217-6_18
34. Hüttel, H.: Types for resources in ψ-calculi. In: Proc. of TGC 2013. LNCS, vol. 8358, pp. 83–102. Springer (2014), http://dx.doi.org/10.1007/978-3-319-05119-2_6
35. Huttel, H., Lanese, I., Vasconcelos, V., Caires, L., Carbone, M., Deniélou, P.M., Mostrous, D., Padovani, L., Ravara, A., Tuosto, E., Vieira, H.T., Zavattaro, G.: Foundations of session types and behavioural contracts. ACM Computing Surveys (2016), to appear
36. Igarashi, A., Kobayashi, N.: A generic type system for the pi-calculus. Theor. Comput. Sci. 311(1-3), 121–163 (2004), http://dx.doi.org/10.1016/S0304-3975(03)00325-6

37. Kobayashi, N.: A type system for lock-free processes. Inf. Comput. 177(2), 122–159 (2002), http://dx.doi.org/10.1006/inco.2002.3171
38. Kobayashi, N.: Type systems for concurrent programs. In: Formal Methods at the Crossroads. LNCS, vol. 2757, pp. 439–453. Springer (2003), http://dx.doi.org/10.1007/978-3-540-40007-3_26
39. Kobayashi, N., Pierce, B.C., Turner, D.N.: Linearity and the pi-calculus. In: POPL. pp. 358–371 (1996)
40. Kouzapas, D., Pérez, J.A., Yoshida, N.: On the relative expressiveness of higher-order session processes. In: ESOP 2016. LNCS, vol. 9632, pp. 446–475. Springer (2016)
41. Lange, J., Tuosto, E., Yoshida, N.: From communicating machines to graphical choreographies. In: Proc. of POPL 2015. pp. 221–232. ACM (2015), http://doi.acm.org/10.1145/2676726.2676964
42. López, H.A., Olarte, C., Pérez, J.A.: Towards a unified framework for declarative structured communications. In: Proc. of PLACES 2009. EPTCS, vol. 17, pp. 1–15 (2009), http://dx.doi.org/10.4204/EPTCS.17.1
43. Milner, R.: The Polyadic pi-Calculus: A Tutorial. Tech. rep., ECS-LFCS-91-180 (1991)
44. Milner, R.: Functions as processes. Math. Struc. in Comp. Sci. 2(2), 119–141 (1992)
45. Nestmann, U.: Welcome to the jungle: A subjective guide to mobile process calculi. In: Proc. of CONCUR 2006. LNCS, vol. 4137, pp. 52–63. Springer (2006), http://dx.doi.org/10.1007/11817949_4
46. Orchard, D.A., Yoshida, N.: Effects as sessions, sessions as effects. In: Proc. of POPL 2016. pp. 568–581. ACM (2016), http://doi.acm.org/10.1145/2837614.2837634
47. Padovani, L.: Session types = intersection types + union types. In: Proc. of ITRS 2010. EPTCS, vol. 45, pp. 71–89 (2010), http://dx.doi.org/10.4204/EPTCS.45.6
48. Palamidessi, C.: Comparing the expressive power of the synchronous and asynchronous pi-calculi. Mathematical Structures in Computer Science 13(5), 685–719 (2003), http://dx.doi.org/10.1017/S0960129503004043
49. Parrow, J.: Expressiveness of process algebras. ENTCS 209, 173–186 (2008), http://dx.doi.org/10.1016/j.entcs.2008.04.011
50. Sangiorgi, D.: Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms. PhD thesis CST–99–93, University of Edinburgh (1992)
51. Villard, J.: Heaps and Hops. Ph.D. thesis, École Normale Supérieure de Cachan (Feb 2011)
52. Wadler, P.: Propositions as sessions. J. Funct. Program. 24(2-3), 384–418 (2014), http://dx.doi.org/10.1017/S095679681400001X
53. Yoshida, N.: Graph types for monadic mobile processes. In: Proc. of FSTTCS'96. LNCS, vol. 1180, pp. 371–386. Springer (1996), http://dx.doi.org/10.1007/3-540-62034-6_64
54. Yoshida, N., Berger, M., Honda, K.: Strong normalisation in the pi -calculus. Inf. Comput. 191(2), 145–202 (2004)