Investigating the Logical Aspects of the Pi-calculus

Noël Bernard

INVESTIGATING THE LOGICAL ASPECTS OF THE PI-CALCULUS

NOËL BERNARD

1. The π-calculus: a nominal process calculus

The π-calculus has been introduced by R. Milner [MPW92] as a model for communicating processes. This calculus has been stressed by A. D. Gordon [Gor01] to be an example of nominal calculus, as defined by R. M. Needham [Nee89]: "*A nominal calculus is a computational formalism that includes a set of pure names and allows the dynamic generation of fresh, unguessable names*".

In this formalism, messages are passed along channels each of which has a name; moreover, the names constitute also the possible content of the messages. The dynamic generation of names has founded the concept of mobility, which has spread as a domain of research in itself covering, in addition to the π-calculus, HOπ [San93], the ambients [CG98], the join-calculus [Fou98], ... Another recent development in the domain of secure communications makes an essential use of the unguessable character of names, for example in the Spi-calculus [AG97].

Up to miscellaneous variants, the processes of the π-calculus are given in the following syntax, in which $\bar{a}\,\langle\vec{b}\rangle.P$ represents sending a list $\vec{b}$ of messages on a channel a, then performing continuation P; $a(\vec{x}).P$ means receiving on channel a some values which are then substituted for $\vec{x}$ in continuation P; operators $\mid$ and $+$ respectively express parallel composition, and indeterministic choice between two processes; $(\nu\,\vec{x})P$ expresses the generation of a list of fresh names $\vec{x}$ whose knowledge is restricted to P; and $[x = y]P$ allows execution of P only if names x and y match. Finally, infinite processes are provided by construction $D(\vec{v})$ representing a call to a named process having a (possibly recursive) definition $D \stackrel{def}{=} (\lambda\,\vec{x})P$:

$$P \;::=\; 0 \;\mid\; \bar{a}\,\langle\vec{b}\rangle.P \;\mid\; a(\vec{x}).P \;\mid\; P \,+\, P \;\mid\; P \mid P \;\mid\; (\nu\,\vec{x})P \;\mid\; [x = y]P \;\mid\; D(\vec{v})$$

Though closely related in spirit to the λ-calculus, the π-calculus is a calculus of processes. R. Milner [Mil92] has investigated the relations between these theories, and pointed essential differences between the two. In contrast with the deep investigations of the correspondence between the λ-calculus and proof theory, the main stress as regards the π-calculus is put on methods for proving equivalence between processes. A quite complete account of the present state of the art can be found in [SW01].

Date: 30 10 2002.

2. Model-checking the π-calculus

Model-checking deals with Kripke-like relations $P \models \varphi$ between processes and formulas. It was first introduced for finite state automata and temporal logic [EC80]. It was later used for CCS, using a modal logic HML (for Hennessy-Milner Logic) later improved as the μ-logic described below.

Let us give as an example the syntax of LTL (Linear Temporal Logic). Its formulas are defined as follows:

$$\varphi ::= tt \mid ff \mid a = b \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \Diamond\varphi \mid \Box\varphi \mid \varphi \mathcal{U} \varphi \mid \forall x\, \varphi \mid \exists x\, \varphi$$

It presents three temporal modalities: $\Diamond\varphi$ means that φ will happen to be true in the future, $\Box\varphi$ that it will always be true, and $\varphi\mathcal{U}\psi$ that φ will remain true until ψ gets true (For sake of simplicity, we do not mention other possible modalities).

A nice tool for model-checking with LTL is the SPIN checker [Hol97] built on the PROMELA language. The distribution of this software involves some samples among which a translation of a celebrated example of a π-calculus process, the cell-phone handoff strategy [OP92]. Nevertheless using LTL model-checkers to verify infinite processes remains out of reach at this time.

The μ-logic has different modalities, and involves fixpoint constructors adapted to infinite processes. Its use for the π-calculus has been studied by M. Dam [Dam93]. The syntax is as follows:

$$\varphi ::= tt \mid ff \mid a = b \mid \neg\varphi \mid \varphi\wedge\varphi \mid \varphi\vee\varphi \mid \langle A\rangle\varphi \mid [A]\varphi \mid \forall x\varphi \mid \exists x\varphi \mid \Sigma x\varphi \mid \mu X\varphi \mid \nu X\varphi$$

It depends on a set of variables which will be identified with the names in the π-calculus. These names appear in atomic expressions $a = b$ and also in the modalities $\langle A\rangle$ and $[A]$ in which A is a set of actions of one of the three forms a, $\bar{a}$ (with a any name) and τ. Then we can give sense to an expression $P \models \varphi$ to be understood as "P satisfies φ", and we concentrate below on the particular features of this syntax. Satisfaction is understood in a Kripke model-like sense, where the Kripke structure would be the transition tree starting from a process P.

Modalities and quantifiers, in combination, allow to deal with outputs or inputs: $P \models [A]\,\varphi$ expresses that for any action $P \xrightarrow{\alpha} P'$ with $\alpha \in A$ then $P' \models \varphi$. $P \models \langle A\rangle\,\varphi$ means there is at least one action $P \xrightarrow{\alpha} P'$ such that $\alpha \in A$ and $P' \models \varphi$. We write $a(x).P \models [a]\,\forall x\,\varphi$ if for any name n received by P along channel a we have $P\{n/x\} \models \varphi\{n/x\}$; existential quantifier is treated in a similar way, while $P \models [\bar{a}]\,\sum x\,\varphi$ expresses that after any emission $P \xrightarrow{\bar{a}\langle n\rangle} P'$ we have $P' \models \varphi\{n/x\}$.

Least fixpoint μ and a greater fixpoint ν allow expressing infinite features of the processes. For example, we can translate the modalities in LTL by writing $\Box\varphi \equiv (\nu\, X)(\varphi \wedge [-]X)$ and $\Diamond\varphi \equiv (\mu\, X)(\varphi \vee (\langle -\rangle\, tt \wedge [-]X))$ (notations $[-]$ and $\langle -\rangle$ are conventional for the case A is the set of all names).

The results of M. Dam led to an implementation incorporated in B. Victor's Mobility Workbench (MWB) [VM94], one of the scarce model-checkers for the π-calculus.

In collaboration with Y. Dumond, we have investigated the use of the π-calculus associated to the μ-logic for dealing with a case-study proposed at the workshop AFADL'98 [LIS98]: the access controller. It consists in modeling the control of a set of doors for a set of buildings in a protected area. Each user attempting to pass a door must be checked to be authorized by entering a personal card into a reader. Various answers using methods such as B, LOTOS, UNITY, LPG, UML, Coq, were given in the AFADL meetings. Our contribution [BD00] led us to verify that the π-calculus enjoys interesting possibilities of modular treatment of complex systems, while the mobility inherent in this calculus allows to take into account intricate changes of configuration.

The method used in our treatment is to associate a process for each entity, such as a door, a card reader, a user, and dynamically add a new process at each attempt of a user to pass a door. The fact that all events are signalled via messages transmitted through a BUS also used for giving opening or closing orders makes it easy to translate the description into the π-calculus. The dynamic creation of a process is made possible by the (ν)-construct in this calculus.

The specifications of the case-study described above were given using a list of LTL formulas. For example, the following formula expresses that if unauthorized user u attempts to enter door a then a is locked:

$$\Box\big(u.attempts(a) \wedge \neg u.authorized(a) \rightarrow a.locked\big)$$

The translation of LTL into the μ-logic provides automatically a specification in this last language.

We tried MWB on the μ-logic specification of the controller, and as expected, this tool is unable to manage a problem of that size. The question remains whether it is possible to devise an efficient model-checker for the π-calculus.It would be interesting to further investigate how infinite processes can be addressed by tools related to LTL such as SPIN, well adapted to deal with concrete verifications of specifications.

3. Curry-Howard and the π-calculus

The use, in the λ-calculus, of a typing system in which types are formulas in second-or higher-order logic was a the origin of some of the most outstanding results of this calculus. The fact, for a λ-term, to have a type ensures that this term terminates, which is a central property in this theory. Moreover, the Curry-Howard correspondence between proofs in intuitionistic logic and the λ-calculus ensures that, if a specification is expressed by some formula, any proof of this formula provides, by just using the typing rules, a program fulfilling the specification. This allows at a theoretical level to devise means of automatic generation of correct programs, and has led to practical applications such as the typed language ML or the Coq prover.

The π-calculus has been equipped by R. Milner with a typing system called sorts. Made necessary in the polyadic version of the π-calculus [Mil91], where during a single communication a list of messages can be transmitted, typing is especially crucial for HOπ, a Higher-Order π-calculus, in which messages can be not only

names of channels, but also processes. The sorts associated to channels are their arity, that is, not only the number of messages transmitted together in one communication on this channel, but also in a recursive way, the sort of each message.

Sorts do not supply the π-calculus with a correctness verification tool. The sorting discipline only specifies the properties of channels, while for processes the only possible type is OK -meaning channels are used with the correct arity- which gives no indication about termination or fulfillment of specification. It may be noticed anyway that problems of great importance in the sequential context of the λ-calculus are not the main concern in process calculi: termination is not always a goal and conflicts with liveness properties often looked for.

Several attempts have been made to give the π-calculus more elaborate typing systems. The blue calculus [Bou98] makes a synthesis of the λ- and π-calculi: In addition to a kernel of π we find a λ-abstraction while communication on a channel a is replaced here by a "resource fetching" rule $(az_1 \dots z_k \mid \langle a \Leftarrow P \rangle) \to Pz_1 \dots z_k$. This rich language can be typed with a system combining the simple types of the λ-calculus and the sorts of the π-calculus. This type system does not ensure termination, but enjoys the subject reduction property. An inspection of how the typing rules apply to the π-calculus component of the blue calculus makes it appear that the typing reduces here to a sorting discipline.

Another attempt in [YBH01],[BHY01] extends the sorting discipline using ideas from the linear logic. It uses a slightly modified on the π-calculus, as here output $\bar{a}(\vec{x}).P$ is bound, leading to a communication rule of the form $\bar{a}(\vec{x}).P \mid a(\vec{x}).P \to (\nu\vec{x})(P \mid Q)$. Typing uses dual I/O modes $\downarrow$ (linear input), $\uparrow$ (linear output), ! (replicated input) and ? (output to !). This system allows a fully abstract encoding of PCF into the π-calculus, related to the sequential character of the processes which can be typed; a further refinement ensures the strong normalization property as well as confluence.

The two examples cited above have in common that a typing system stronger than the sorts is associated to a sequential calculus, introduced either in addition to, or as a fragment of the process calculus. This raises the question of whether ensuring properties of processes with type systems implies to restrict one's attention to sequential processes. We present in the following a (work in progress) sketch of a possible Curry-Howard correspondence defined directly for the π-calculus.

A characteristic of the π-calculus is to make a balance between local actions and global actions. Thus, in [Mil91] R. Milner defines abstractions $(\lambda\,\vec{x})\,P$ and a metaconstruct $\bullet$ called "pseudo-application"; we would like to sketch here a system giving a real status to local actions. A logic fitting with such a system should take advantage from having local cuts like in natural deduction and global cuts as one finds in the sequent calculus. A mechanism of explicit substitutions, known to be related to the cut of sequent calculus, can been defined in the π-calculus: we denote it $x \lhd A$. This construct, not without similarities with the resource fetching of the blue calculus, is accompanied with abstractions and application (now fully integrated in the calculus). Explicit substitutions are processes and can interact

with other processes, but for sake of keeping a local character to the calculus, they only modify the head variables of a process.

For giving the π-calculus a Curry-Howard property, due its modularity, we look for some kind of distributed logic. As processes are spread on different locations, each proof associated to a process is also situated in a particular location. Using a property proven at some place supposes therefore a call to that property under a name, in a way suggested by the usual system of references to lemmas or theorems:

$$\textbf{Lemma A} : \ldots \qquad \leftrightsquigarrow \qquad \text{Using lemma A, } \ldots$$

We distinguish between theorems, usually supposed to be used many times, and lemmas, considered here as a resource limited to one use.

The typing rules for processes and abstractions are defined in a mutually recursive way. Abstractions and application correspond to the local notion of cut, as in the following example:

$$\text{if} \quad (\lambda\, x, y).P \ : \ p, q \to r \quad \text{and} \quad M \ : \ p \ , \ N \ : \ q \quad \text{then} \quad (\lambda\, x, y).P \bullet (M, N) \ : \ r$$

To the global cut, whose usual definition uses limited resource, corresponds the communication in the π-calculus. We give below the typing and its intuitive signification in terms of lemma:

$a(x, y).P : p, q \to r$	:	Lemma a: under hyp. p, q then r.
$\bar{a}\langle M, N\rangle : r$ (where $M : p, N : q$)	:	Use lemma a knowing that p and q then r.

Theorems, though closely related to lemmas, are supposed to behave as infinite resources. They are intuitively associated to explicit substitutions. It should be noticed that this choice of representation actually strongly separates theorems from lemmas and relates them in some extent to abstractions.:

$a \lhd N : p$	:	Theorem a: p.
$a \bullet M : q$ (where $M : p$)	:	Apply Theorem a using p then q.

These propositions intend to provide a correspondence between processes and what we call a distributed natural deduction. While the sorts provide a static verification tool, we hope the achievement of this work to bring a better grasp on dynamic properties of the calculus.

4. Conclusion

We have presented the logical properties of the π-calculus under two aspects: model-checking, and typing. Many motivations exist for adopting a logical point of view: equivalence, security, time, space, locality, mobility. To various approaches correspond the use of various logics: this is even increased by the numerous variants of the π-calculus. Two comments can be given in conclusion: First, few efficient practical realizations have been yet made in this direction; second, it is not so obvious what best choice of a logical system should be made. An interesting idea would be to allow the fixpoints of the μ-logic or modalities of LTL into the syntax of the logic. This is the direction in which the present work is engaged.

References

[AG97] Martín Abadi and Andrew D. Gordon. A calculus for cryptographic protocols: The Spi calculus. In *Fourth ACM Conference on Computer and Communications Security*, pages 36–47. ACM Press, 1997.

[BD00] Noël Bernard and Yves Dumond. Spécification en pi-calcul de l'étude de cas relative au contrôle d'accès. In *AFADL'2000*, LSR/IMAG – BP 72 38402 Saint-Martin d'Heres Cedex – Grenoble – France, January 2000. LSR/IMAG, LSR/IMAG.

[BHY01] Martin Berger, Kohei Honda, and Nobuko Yoshida. Sequentiality and the pi-calculus. In *International Conference on Typed Lambda Calculi and Applications (TCLA 2001, Krakow, Poland)*, volume 2044 of *LNCS*, pages 29–45. Springer, 2001.

[Bou98] Gerard Boudol. The π-calculus in direct style. *Higher-Order and Symbolic Computation*, 11(2):177–185, 1998.

[CG98] Luca Cardelli and Andrew D. Gordon. Mobile ambients. In Maurice Nivat, editor, *Foundations of Software Science and Computational Structures*, volume 1378 of *LNCS*, pages 140–155. Springer Verlag, 1998.

[Dam93] M. Dam. Model checking mobile processes. *Lecture Notes in Computer Science*, 715:22–36, 1993.

[EC80] E. Allen Emerson and Edmund M. Clarke. Characterizing correctness properties of parallel programs using fixpoints. In J. W. de Bakker and Jan van Leeuwen, editors, *Automata, Languages and Programming, 7th Colloquium*, volume 85 of *Lecture Notes in Computer Science*, pages 169–181, Noordweijkerhout, The Netherland, 14–18 July 1980. Springer-Verlag.

[Fou98] Cédric Fournet. *The Join-Calculus: a Calculus for Distributed Mobile Programming*. Ph.D. thesis, Ecole Polytechnique, 1998.

[Gor01] Andrew D. Gordon. Notes on nominal calculi for security and mobility. *Lecture Notes in Computer Science*, 2171:262–330, 2001.

[Hol97] G.J. Holzmann. The model checker spin. *IEEE Trans. on Software Engineering*, 23(5):279–295, May 1997. Special issue on Formal Methods in Software Practice.

[LIS98] LISI/ENSMA. *Approches Formelles dans l'Assistance au Développement de Logiciels (Proceedings of AFADL'98)*, Téléport2 - Avenue 1 - BP109 - 86960 FUTUROSCOPE Cedex, October 1998. LISI/ENSMA.

[Mil91] Robin Milner. The polyadic π-calculus: a tutorial. LFCS Report Series ECS-LFCS-91–180, Laboratory for Foundations of Computer Science, Department of Computer Science, University of Edinburgh, UK, Heidelberg, October 1991.

[Mil92] R. Milner. Functions as processes. *Mathematical Structures in Computer Science*, 2:119–141, 1992.

[MPW92] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes (Parts I and II). *Information and Computation*, 100:1–77, 1992.

[Nee89] R. M. Needham. Names. In Sape Mullender, editor, *Distributed Systems*, chapter 5., pages 89–102. acpr, 1989.

[OP92] Fredrik Orava and Joachim Parrow. An algebraic verification of a mobile network. *Formal Aspects of Computing*, 4(6):497–543, 1992.

[San93] Davide Sangiorgi. *Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms*. Ph.D. thesis, Computer Science Dept., University of Edinburgh, May 1993.

[SW01] Davide Sangiorgi and David Walker. *The π-calculus: a Theory of Mobile Processes*. Cambridge University Press, 2001.

[VM94] Björn Victor and Faron Moller. The Mobility Workbench — A tool for the π-calculus. In David Dill, editor, *Computer Aided Verification (Proc. of CAV'94)*, volume 818 of *LNCS*, pages 428–440. Springer, 1994.

[YBH01] N. Yoshida, M. Berger, and K. Honda. Strong normalisation in the π-calculus. In *16th Annual IEEE Symposium on Logic in Computer Science (LICS '01)*, pages 311–322, Washington - Brussels - Tokyo, June 2001. IEEE.

LAMA-Université de Savoie, Campus Scientifique, F-73376 Le Bourget-du-Lac
E-mail address: Noel.Bernard@univ-savoie.fr