Modeling Imperative Constructs in the Pi-Calculus[⋆]

Daniel Hirschkoff[1], Enguerrand Prebet[1], and Davide Sangiorgi[2]

[1] École Normale Supérieure de Lyon, France
[2] Università di Bologna, Italy, and INRIA

Abstract. We describe an extension of the pi-calculus with primitive operations to manipulate an imperative store. We study how this extended calculus can be encoded into the pi-calculus. This leads to the definition of a pi-calculus with location names, for which we present a behavioural equivalence. We show the full abstraction for the encoding and we analyse several variants of the encoding and of the bisimulation.

We study the representation of imperative constructs in the π-calculus. The intent of this study is to provide a building block for a framework to reason about rich programming languages, like, e.g., ML, where higher-order functions are combined with imperative aspects. In order to handle such languages, we plan to combine the results of this work with existing approaches to reason about higher-order functions in the π-calculus [6, 9, 2]. Because of its expressiveness, the π-calculus can be used as a target language in which rich programming languages can be translated. We can then benefit from the powerful techniques available to reason about π-calculus processes in order to establish equivalences between programs in the source language.

Equivalences between higher-order functional languages with state are known to be hard to establish. Several kinds of approaches, like Kripke logical relations or trace semantics [4, 5] have been investigated to prove such equivalences. There also exists a rich literature on game semantics to represent and analyse program execution [7, 1]. The operational approach, as put forward using the π-calculus, can serve as a complement to the "operational/denotational" point of view promoted by game semantics to reason about high-level languages.

In the present work, we analyse a simple representation of references in the Asynchronous π-calculus [3]. Processes are equipped with a store, which can be accessed and modified using standard operations of allocation, read and write. This defines our source calculus, called $\pi^{\mathtt{ref}}$. For example, the $\pi^{\mathtt{ref}}$ process $x \leftarrow \ell.\overline{a}\langle x\rangle$ reads the value which is in location ℓ of the store, and then emits it along channel a ($x \leftarrow \ell$ is the construct for reading the store at ℓ, and $\overline{a}\langle x\rangle$ represents the emission of x on channel a).

[⋆] Copyright © 2019 for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0)

The encoding is somehow standard, but has not been studied per se until now, as far as we know — it is different, in particular, from the encoding studied in [8]. A location (or reference) in the store contains a name, which can be either a π-calculus channel, or a location. In the encoding, a message $\overline{l}\langle m\rangle$ represents a location ℓ containing name m. Reading and updating the location is achieved using a simple protocol: for instance, $\ell(x).(\overline{\ell}\langle x\rangle \mid P)$ is the process that reads the content of ℓ and then proceeds as P, where x is bound in P. Accordingly, the process $x \leftarrow \ell.\overline{a}\langle x\rangle$ seen above is encoded as $\ell(x).(\overline{\ell}\langle x\rangle \mid \overline{a}\langle x\rangle)$.

The target of the encoding is therefore the Asynchronous π-calculus with a distinct notion of location names, a calculus we call π_ℓ. In order to analyse this language, we introduce a type system to capture the usage of location names. We also present a notion of bisimilarity, whose definition is rather sophisticated. We show that this equivalence is fully abstract with respect to barbed congruence in $\pi^{\mathtt{ref}}$, the source language. Thus, reasoning via the encoding into π_ℓ can be seen as a proof technique to establish equivalences between $\pi^{\mathtt{ref}}$ programs.

We now provide some intuitions about why the bisimulation in π_ℓ is not standard. If we consider processes

$$P \;=\; \ell(x).(\overline{\ell}\langle x\rangle \mid \overline{c}\langle d\rangle) \qquad \text{and} \qquad Q \;=\; \overline{c}\langle d\rangle \;,$$

which are the π-calculus translations of, respectively,

$$x \leftarrow \ell.\overline{c}\langle d\rangle \qquad \text{and} \qquad \overline{c}\langle d\rangle \;,$$

we may ask ourselves whether P and Q should be equivalent. They are not in the asynchronous π-calculus, even for asynchronous bisimilarity, because the output at c is guarded by the input at ℓ in P.

If we want to take into account the fact that ℓ stands for a location (or an address), we have two possibilities: either deem P and Q inequivalent, because P tries to access an unallocated address, and thus crashes before being able to output at c; or impose that behavioural equivalence should consider that a store is always available, in which case P and Q are behaviourally equal, because the reading performed by P has no influence on its behaviour. In the present work, we choose the latter option.

The starting point of our study is barbed congruence for $\pi^{\mathtt{ref}}$ *configurations*, which consist in a process and a store (with some private names shared between the process and the store). We provide a coinductive characterisation of this relation, which we call *closing equivalence*. Based on our encoding from $\pi^{\mathtt{ref}}$ into π_ℓ, we introduce a notion of typing, and define a notion of closing bisimilarity in π_ℓ, which is fully abstract w.r.t. its counterpart in $\pi^{\mathtt{ref}}$. We also study to what extent the standard notion of asynchronous bisimilarity can be used to reason about $\pi^{\mathtt{ref}}$ configurations.

References

1. Castellan, S., Clairambault, P., Hayman, J., Winskel, G.: Non-angelic concurrent game semantics. In: Baier, C., Lago, U.D. (eds.) Foundations of Software Science and

Computation Structures - 21st International Conference, FOSSACS 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings. Lecture Notes in Computer Science, vol. 10803, pp. 3–19. Springer (2018). https://doi.org/10.1007/978-3-319-89366-2_1, https://doi.org/10.1007/978-3-319-89366-2_1

2. Durier, A., Hirschkoff, D., Sangiorgi, D.: Eager functions as processes. In: Dawar, A., Grädel, E. (eds.) Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018. pp. 364–373. ACM (2018). https://doi.org/10.1145/3209108.3209152, https://doi.org/10.1145/3209108.3209152
3. Honda, K., Tokoro, M.: An object calculus for asynchronous communication. In: America, P. (ed.) ECOOP'91 European Conference on Object-Oriented Programming, Geneva, Switzerland, July 15-19, 1991, Proceedings. Lecture Notes in Computer Science, vol. 512, pp. 133–147. Springer (1991). https://doi.org/10.1007/BFb0057019, https://doi.org/10.1007/BFb0057019
4. Hur, C., Dreyer, D., Neis, G., Vafeiadis, V.: The marriage of bisimulations and kripke logical relations. In: Field, J., Hicks, M. (eds.) Proceedings of the 39th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2012, Philadelphia, Pennsylvania, USA, January 22-28, 2012. pp. 59–72. ACM (2012). https://doi.org/10.1145/2103656.2103666, https://doi.org/10.1145/2103656.2103666
5. Jaber, G., Tzevelekos, N.: Trace semantics for polymorphic references. In: Grohe, M., Koskinen, E., Shankar, N. (eds.) Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016. pp. 585–594. ACM (2016). https://doi.org/10.1145/2933575.2934509, https://doi.org/10.1145/2933575.2934509
6. Milner, R.: Functions as processes. In: Paterson, M. (ed.) Automata, Languages and Programming, 17th International Colloquium, ICALP90, Warwick University, England, July 16-20, 1990, Proceedings. Lecture Notes in Computer Science, vol. 443, pp. 167–180. Springer (1990). https://doi.org/10.1007/BFb0032030, https://doi.org/10.1007/BFb0032030
7. Murawski, A.S., Tzevelekos, N.: Full abstraction for reduced ML. Ann. Pure Appl. Logic **164**(11), 1118–1143 (2013). https://doi.org/10.1016/j.apal.2013.05.007, https://doi.org/10.1016/j.apal.2013.05.007
8. Röckl, C., Sangiorgi, D.: A pi-calculus process semantics of concurrent idealised ALGOL. In: Thomas, W. (ed.) Foundations of Software Science and Computation Structure, Second International Conference, FoSSaCS'99, Held as Part of the European Joint Conferences on the Theory and Practice of Software, ETAPS'99, Amsterdam, The Netherlands, March 22-28, 1999, Proceedings. Lecture Notes in Computer Science, vol. 1578, pp. 306–321. Springer (1999). https://doi.org/10.1007/3-540-49019-1_21, https://doi.org/10.1007/3-540-49019-1_21
9. Sangiorgi, D.: The lazy lambda calculus in a concurrency scenario. Inf. Comput. **111**(1), 120–153 (1994). https://doi.org/10.1006/inco.1994.1042, https://doi.org/10.1006/inco.1994.1042