An operational approach to blindness for ground references in the π-calculus

Iwan Quémerais

In the continuation of works that link operational semantics (OGS) with process calculi, we build on top of a type system that enforces visibility in a sequential π-calculus to also enforce blindness and obtain a full-abstraction result for a sequential π-calculus with ground references. Blindness is a game semantics notion modeling ground references in languages with exchanges of references; it was introduced by Murawski and Tzevelekos in order to prove full-abstraction for Reduced ML [3].

In π-calculus computation steps are communications between processes by means of message-passing synchronizations. This makes the π-calculus adequate as a model to interpret and give semantics to higher-order languages. In particular, techniques from OGS can be adapted to form type systems for π-calculi to enforce corresponding notions. For example, Prebet et al. ported the notions of alternation and well-bracketing from games semantics, in their operational semantics presentation, to define sequential and well-bracketed π-calculi [1].

More recently, with Hirschkoff and Sangiorgi, we ported the notion of visibility to a sequential π-calculus in order to define a type system enforcing that only first-order store can be used [2]. First-order store means that references should not store functions; in π-calculus it means that no higher-order name should be stored. However, in π-calculus, the notion of store is not necessarily explicit; references can be encoded in the π-calculus. In π-calculus, we represent references using specific channel names, ranging over $h, k, \ldots$, that follow a certain discipline [4]. Each reference should have a pending output $\overline{h}\langle n\rangle$, so that the ground value n (here an integer) can be read at any time using an input $h(x)$. The channel names that are not reference names will be called higher-order names, ranging over $a, b, c, d, \ldots$. Moreover, behaviours corresponding to the use of store can occur implicitly. For example, the following process makes use of implicit higher-order store:

$$P := \overline{a}(b).b(c).\overline{c}(\_).b(d).\overline{c}(\_)$$

where $\overline{a}(b)$ is an output sending channel name b through channel a and $b(c)$ is an input receiving channel name c through channel b, we can think of inputs as functions taking some parameters and outputs as calls to functions. In the process P, after the second input on b, an output at c is made; however the only appearance of c is as the parameter of the first call to b therefore this breaks the scope of names that b (as a function) should be able to call. In game semantics terms, the player view after $b(d)$ contains exactly the names a and d. In order to disallow these behaviours corresponding to implicit higher-order store, we need to impose visibility as a type system for processes. To each name that can be used as input we associate a view of names that can be used as output after such input. When the process is active (i.e. running) it is also given a current view for the outputs currently available. One difference with the treatment of visibility in OGS is that names are not introduced to take place of functions in an LTS but are directly present in the processes; this means that, in the processes, both Opponent and Player names coexist and therefore the view needs to take both into account.

When considering first-order store, two distinct forms of first-order store can be considered. Fully ground references: this is when references can store values of ground types (e.g. integers, booleans,...) and other references; and ground references: this is when references can store exclusively values of ground types. This difference is particularly important in languages where references can be exchanged. In particular, when considering ground references in ML-like languages, visibility is no longer sufficient to obtain a full-abstraction result; unlike for functions, a context can test the equality between two references; therefore, we need to ensure that this test can only be made between references in the current views. Murawski and Tzevelekos introduce blindness [3] as a Game semantics notion to capture this phenomenon and obtain a full-abstraction result for Reduced ML, a fragment of standard ML with ground references. As far as we know, no formulations of blindness have been proposed for OGS; we propose an operational formulation for the π-calculus and we hope that the ideas can be used to obtain a formulation for OGS.

We extend the language used for visibility in π-calculus so that reference names can be exchanged: $a(h, b)$, $\overline{a}\langle h\rangle(b)$. In order to restrict the processes to ground references we also apply the restrictions of visibility to reference names in the type system; previously it was only applied to higher-order names. Previously, the visibility restriction ensured that no higher-order names were implicitly stored, now neither can reference names. We remark that like in Reduced ML, enforcing visibility (similarly to what is done in the type system) in a

behavioural equivalence is not sufficient to obtain full-abstraction. If h and k are two references that store integer n and none of them appear in the view of a higher-order name a, then sending any one of them to a should have the same result; however, $\overline{a}\langle h\rangle$ and $\overline{a}\langle k\rangle$ are different actions and will be distinguished by the bisimilarity. The solution to this issue is to accurately mask reference names in the transitions made by processes. In the LTS, instead of directly sending the reference name, we send an avatar of this name. When sending a reference name to a higher-order name that already has some avatar of that reference name in the view we need to make sure that the same avatar is used for this action; however when the higher-order name does not have the reference name in the view we can send any fresh avatar instead of the reference name. Using this technique we show a full-abstraction result between a typed bisimilarity and barbed congruence.

We conjecture that a similar technique can be used in OGS using move-with-store where the store is masked in order to only show avatars of references that are in the view of the environment.

References

[1] Daniel Hirschkoff, Enguerrand Prebet, and Davide Sangiorgi. "On sequentiality and well-bracketing in the π-calculus". In: *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 - July 2, 2021*. IEEE, 2021, pp. 1–13. ISBN: 978-1-6654-4895-6. DOI: 10.1109/LICS52264.2021.9470559. URL: https://doi.org/10.1109/LICS52264.2021.9470559.

[2] Daniel Hirschkoff, Iwan Quémerais, and Davide Sangiorgi. "First-Order Store and Visibility in Name-Passing Calculi". In: *36th International Conference on Concurrency Theory, CONCUR 2025, Aarhus, Denmark, August 26-29, 2025*. Ed. by Patricia Bouyer and Jaco van de Pol. Vol. 348. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2025, 23:1–23:21. ISBN: 978-3-95977-389-8. DOI: 10.4230/LIPICS.CONCUR.2025.23. URL: https://doi.org/10.4230/LIPIcs.CONCUR.2025.23.

[3] Andrzej S. Murawski and Nikos Tzevelekos. "Full abstraction for Reduced ML". In: *Ann. Pure Appl. Log.* 164.11 (2013), pp. 1118–1143. DOI: 10.1016/J.APAL.2013.05.007. URL: https://doi.org/10.1016/j.apal.2013.05.007.

[4] Enguerrand Prebet. "Functions and References in the Pi-Calculus: Full Abstraction and Proof Techniques". In: *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022, Paris, France, July 4-8, 2022*. Ed. by Mikolaj Bojanczyk, Emanuela Merelli, and David P. Woodruff. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, 130:1–130:19. DOI: 10.4230/LIPICS.ICALP.2022.130. URL: https://doi.org/10.4230/LIPIcs.ICALP.2022.130.