The Early π-Calculus in Ticked Cubical Type Theory

Niccolò Veltri[1] and Andrea Vezzosi[2]

[1] Dept. of Software Science, Tallinn University of Technology, Estonia
niccolo@cs.ioc.ee
[2] IT University of Copenhagen, Denmark
avez@itu.dk

The Nakano modality $\triangleright$ [10] is an operator that, when added to a logic or a type system, encodes time at the level of formulae or types. The formula $\triangleright A$ stands for "A holds one time step in the future", similarly the inhabitants of type $\triangleright A$ are the inhabitants of A in the next time step. The Nakano modality comes with a guarded fixpoint combinator $\mathsf{fix}_A : (\triangleright A \to A) \to A$ ensuring the existence of a solution for all guarded recursive equations in any type. Logically, this corresponds to a version of Löb's axiom for the $\triangleright$ modality.

Guarded recursion has been added to Martin-Löf dependent Type Theory in two different ways: using delayed substitutions as in Guarded Dependent Type Theory (gDTT) [2] or using ticks as in Clocked Type Theory (CloTT) [1]. In these settings, the Nakano modality is employed for constructing guarded recursive types, i.e. recursive types in which the recursive variables are guarded by $\triangleright$. These are computed using the fixpoint combinator at type U, which is the universe of small types. For example, the guarded recursive type of infinite streams of natural numbers is obtained as $\mathsf{Str} = \mathsf{fix}_{\mathsf{U}} X.\, \mathbb{N} \times \triangleright X$ and it satisfies the type equivalence $\mathsf{Str} \simeq \mathbb{N} \times \triangleright \mathsf{Str}$. Recursively defined terms of guarded recursive types are causal and productive by construction.

Dependent type theories with guarded recursion have proved themselves suitable for the development of denotational semantics of programming languages, as demonstrated by Paviotti et al's formalization of PCF [11] and Møgelberg and Paviotti's formalization of FPC in gDTT [8]. Here we continue on this line of work by constructing a denotational model of Milner's early π-calculus in a suitable extension of CloTT. Traditionally, the denotational semantics of π-calculus is developed in specific categories of (presheaves over) profunctors [3] or domains [12, 6]. Fundamentally, the semantic domains have to be sufficiently expressive to handle the non-deterministic nature of π-calculus processes. In domain theoretic semantics, for example, this is achieved by employing powerdomains. Synthetic analogues of these constructions are not available in guarded type theories such as gDTT or CloTT, but it can be constructed if we set our development in extensions of these type systems with Higher Inductive Types (HITs), a characterizing feature of Homotopy Type Theory (HoTT).

We work in Ticked Cubical Type Theory (TCTT) [9], an extension of Cubical Type Theory (CTT) [5] with guarded recursion and the ticks from CloTT. CTT is an implementation of a variant of HoTT, giving computational interpretation to its characteristic features: the univalence axiom and HITs. Roughly speaking, the univalence axiom provides an extensionality principle for types, allowing to consider equivalent types as equal. A HIT A can be thought of as an inductive type in which the introduction rules not only specify the generators of A, but can also specify the generators of the higher equality types of A. The latter are commonly referred to as *path constructors*, due to the interpretation of equality proofs as paths in HoTT. For example, here is a presentation of the countable powerset datatype as a HIT [4], that will serve as our synthetic powerdomain:

$$\frac{}{\varnothing : \mathsf{P}_\infty\, A} \qquad \frac{a : A}{\{a\} : \mathsf{P}_\infty\, A} \qquad \frac{s : \mathbb{N} \to \mathsf{P}_\infty\, A}{\bigcup s : \mathsf{P}_\infty\, A}$$

$$\frac{x, y : \mathsf{P}_\infty\, A}{\_ : x \cup y = y \cup x} \qquad \frac{x, y, z : \mathsf{P}_\infty\, A}{\_ : (x \cup y) \cup z = x \cup (y \cup z)} \qquad \frac{x : \mathsf{P}_\infty\, A}{\_ : x \cup \varnothing = x} \qquad \frac{x : \mathsf{P}_\infty\, A}{\_ : x \cup x = x}$$

$$\frac{s : \mathbb{N} \to \mathsf{P}_\infty\, A \quad n : \mathbb{N}}{\_ : s\, n \cup \bigcup s = \bigcup s} \qquad \frac{s : \mathbb{N} \to \mathsf{P}_\infty\, A \quad x : \mathsf{P}_\infty\, A}{\_ : \bigcup s \cup x = \bigcup(\lambda n. s\, n \cup x)} \qquad \text{the 0-truncation constructor}$$

The type $\mathsf{P}_\infty\, A$ has three generators: the empty subset, the singleton constructor and countable union. Binary union $x \cup y$ is defined as the countable union of the sequence $x, y, y, y, \ldots$ The path constructors are the equations of the theory of countable-join semilattices, while the 0-truncation constructor forces $\mathsf{P}_\infty\, A$ to be a "set" in the sense of HoTT, that is a type with trivial higher paths. A membership relation $\in : A \to \mathsf{P}_\infty\, A \to \mathsf{U}$ is definable by induction on the second argument.

TCTT also has ticks. The Nakano modality is now indexed over the sort of ticks, $\triangleright(\alpha : \mathsf{tick}).A$, and its inhabitants are to be thought of as dependent functions taking in input a tick β and returning an inhabitant of $A[\beta/\alpha]$. So ticks correspond to resources witnessing the passing of time that can be used to access values available only at future times. We write $\triangleright A$ for $\triangleright(\alpha : \mathsf{tick}).A$ when α does not occur free in A. The $\triangleright$ modality is an applicative functor, its unit is called next. Ticks allow to extend the applicative structure to dependent types.

For the specification of the π-calculus syntax, we assume the existence of a countable set of names. Practically, for every natural number n, we assume given a type $\mathsf{Name}\, n$, the set containing the first n names. Each process can perform an output, an input or a silent action. The type of actions is indexed by two natural numbers, representing the number of free names and the sum of free and bound names, respectively. The input action binds the input name.

$$\frac{ch, v : \mathsf{Name}\, n}{\mathsf{out}\, ch\, v : \mathsf{Act}\, n\, n} \qquad \frac{ch : \mathsf{Name}\, n}{\mathsf{inp}\, ch : \mathsf{Act}\, n\, (n+1)} \qquad \frac{}{\tau : \mathsf{Act}\, n\, n}$$

The π-calculus syntax includes the nil process, prefixing, binary sums, parallel composition, restriction, a matching operator and replication.

$$\frac{}{\mathsf{end} : \mathsf{Pi}\, n} \qquad \frac{a : \mathsf{Act}\, n\, m \quad P : \mathsf{Pi}\, m}{a \cdot P : \mathsf{Pi}\, n} \qquad \frac{P : \mathsf{Pi}\, n \quad Q : \mathsf{Pi}\, n}{P \oplus Q : \mathsf{Pi}\, n} \qquad \frac{P : \mathsf{Pi}\, n \quad Q : \mathsf{Pi}\, n}{P \| Q : \mathsf{Pi}\, n}$$

$$\frac{P : \mathsf{Pi}\, (n+1)}{\nu P : \mathsf{Pi}\, n} \qquad \frac{x, y : \mathsf{Name}\, n \quad P : \mathsf{Pi}\, n}{\mathsf{guard}\, x\, y\, P : \mathsf{Pi}\, n} \qquad \frac{P : \mathsf{Pi}\, n}{!P : \mathsf{Pi}\, n}$$

The processes in $\mathsf{Pi}\, n$ are quotiented by a structural congruence relation $\approx$, which, among other things, characterizes the replication operator in terms of parallel composition: given a process $P : \mathsf{Pi}\, n$, we have $!P \approx P \| !P$. The early operational semantics is inductively defined as a type family $- [-] \mapsto - : \mathsf{Pi}\, n \to \mathsf{Label}\, n\, m \to \mathsf{Pi}\, m \to \mathsf{U}$. Following [7], the type $\mathsf{Label}\, n\, m$ of transition labels include a silent action, free and bound outputs, and free and bound inputs.

For the denotational semantic domain, we consider the guarded recursive type

$$\mathsf{Proc} := \mathsf{fix}_{\mathbb{N} \to \mathsf{U}} X.\, \lambda n.\, \mathsf{P}_\infty(\mathsf{Step}\, (\lambda m. \triangleright \alpha. X\, \alpha\, m)\, n)$$

where $\mathsf{Step}\, Y\, n := \Sigma(m : \mathbb{N}).\, \mathsf{Label}\, n\, m \times Y\, m$. In other words, $\mathsf{Proc}\, n$ is the type satisfying the type equivalence $\mathsf{Proc}\, n \simeq \mathsf{P}_\infty(\Sigma m : \mathbb{N}.\, \mathsf{Label}\, n\, m \times \triangleright \mathsf{Proc}\, m)$. Let Unfold be the right-to-left morphism underlying the latter equivalence. To each syntactic process $P : \mathsf{Pi}\, n$ we associate a semantic process $[\![P]\!] : \mathsf{Proc}\, n$. The interpretation respects the structural congruence relation, that is $P \approx Q$ implies $[\![P]\!] = [\![Q]\!]$. The early operational semantics transitions are modelled using the membership operation: given $P\,[a]{\mapsto}Q$ with $a : \mathsf{Label}\,n\,m$, we have $(m, a, \mathsf{next}\,[\![Q]\!]) \in \mathsf{Unfold}\,[\![P]\!]$. Nevertheless, Proc is not closed under name substitutions. Therefore, to obtain a sound interpretation of π-calculus, we need to move to the following domain:

$$\begin{aligned}\mathsf{PiMod}\,n := \Sigma(P : \Pi(m : \mathbb{N}).(\mathsf{Name}\,n \to \mathsf{Name}\,m) \to \mathsf{Proc}\,m).\\ \Pi(m, m' : \mathbb{N})(f : \mathsf{Name}\,m \to_{\mathsf{inj}} \mathsf{Name}\,m')(\rho : \mathsf{Name}\,n \to \mathsf{Name}\,m).\\ \mathsf{mapProc}\,f\,(P\,m\,\rho) = P\,m'\,(f \circ \rho)\end{aligned}$$

where $A \to_{\mathsf{inj}} B$ is the type of injective maps between A and B, while $\mathsf{mapProc}$ corresponds to the action of the functor Proc on injective renamings.

TCTT provides an extensionality principle for guarded recursive types: strong bisimilarity is equivalent to path equality [9]. For $\mathsf{Proc}\,n$, this says that semantic early bisimilarity is equivalent to path equality. In our work, we also define a syntactic notion of early bisimilarity and early congruence and we prove the denotational semantics fully abstract wrt. it.

We formalized the whole development in our own version of the Agda proof assistant based on TCTT, called Guarded Cubical Agda, an extension of Vezzosi et al's Cubical Agda [13].

Acknowledgments Niccolò Veltri was supported by the ESF funded Estonian IT Academy research measure (project 2014-2020.4.05.19-0001).

References

[1] P. Bahr, H. B. Grathwohl, and R. E. Møgelberg. The clocks are ticking: No more delays! In *Proc. of 32nd Ann. ACM/IEEE Symp. on Logic in Computer Science, LICS 2017*, pages 1–12. IEEE, 2017.

[2] A. Bizjak, H. B. Grathwohl, R. Clouston, R. E. Møgelberg, and L. Birkedal. Guarded dependent type theory with coinductive types. In B. Jacobs and C. Löding, editors, *Proc. of 19th Int. Conf. on Foundations of Software Science and Computation Structures, FOSSACS 2016*, volume 9634 of *Lecture Notes in Computer Science*, pages 20–35. Springer, 2016.

[3] G. L. Cattani, I. Stark, and G. Winskel. Presheaf models for the pi-calculus. In E. Moggi and G. Rosolini, editors *Proc. of 7th Int. Conf. on Category Theory and Computer Science, CTCS 1997*, volume 1920 of *Lecture Notes in Computer Science*, pages 106–126. Springer, 1997.

[4] J. Chapman, T. Uustalu, and N. Veltri. Quotienting the delay monad by weak bisimilarity. *Math. Struct. Comput. Sci.*, 29(1):67–92, 2019.

[5] C. Cohen, T. Coquand, S. Huber, and A. Mörtberg. Cubical type theory: A constructive interpretation of the univalence axiom. *IFCoLog J. Log. Appl.*, 4(10):3127–3170, 2017.

[6] M. P. Fiore, E. Moggi, and D. Sangiorgi. A fully-abstract model for the pi-calculus (extended abstract). In *Proc. of 11th Ann. IEEE Symp. on Logic in Computer Science, LICS 1996*, pages 43–54. IEEE, 1996.

[7] R. Milner, J. Parrow, and D. Walker. Modal logics for mobile processes. *Theor. Comput. Sci.*, 114(1):149–171, 1993.

[8] R. E. Møgelberg and M. Paviotti. Denotational semantics of recursive types in synthetic guarded domain theory. *Math. Struct. Comput. Sci.*, 29(3):465–510, 2019.

[9] R. E. Møgelberg and N. Veltri. Bisimulation as path type for guarded recursive types. *Proc. ACM Program. Lang.*, 3(POPL):4:1–4:29, 2019.

[10] H. Nakano. A modality for recursion. In *Proc. of 15th Ann. IEEE Symp. on Logic in Computer Science, LICS 2000*, pages 255–266. IEEE, 2000.

[11] M. Paviotti, R. E. Møgelberg, and L. Birkedal. A model of PCF in guarded type theory. *Electron. Notes Theor. Comput. Sci.*, 319:333–349, 2015.

[12] I. Stark. A fully abstract domain model for the pi-calculus. In *Proc. of 11th Ann. IEEE Symp. on Logic in Computer Science, LICS 1996*, pages 36–42. IEEE, 1996.

[13] A. Vezzosi, A. Mörtberg, and A. Abel. Cubical Agda: A dependently typed programming language with univalence and higher inductive types. *Proc. ACM Program. Lang.*, 3(ICFP):87:1–87:29, 2019.