Presheaf Models for the $\pi$-Calculus

Gian Luca Cattani, Ian Stark and Glynn Winskel

BRICS*, Department of Computer Science, University of Aarhus, Denmark

Abstract. Recent work has shown that presheaf categories provide a general model of concurrency, with an inbuilt notion of bisimulation based on open maps. Here it is shown how this approach can also handle systems where the language of actions may change dynamically as a process evolves. The example is the $\pi$-calculus, a calculus for 'mobile processes' whose communication topology varies as channels are created and discarded. A denotational semantics is described for the $\pi$-calculus within an indexed category of profunctors; the model is fully abstract for bisimilarity, in the sense that bisimulation in the model, obtained from open maps, coincides with the usual bisimulation obtained from the operational semantics of the $\pi$-calculus. While attention is concentrated on the 'late' semantics of the $\pi$-calculus, it is indicated how the 'early' and other variants can also be captured.

1 Introduction

The gap between domain theory and the theory of concurrency is narrowing. In particular, the $\pi$-calculus, which for a long time resisted all but operational semantics, has yielded to a fully abstract denotational semantics; a key idea was to move to domains indexed by a category of name sets (Stark 1996; Fiore, Moggi, and Sangiorgi 1996; Hennessy 1996). Why add yet another model, based this time not on familiar, complete partial orders but instead on presheaf categories?

Several reasons can be given, even at this preliminary stage.

Models for concurrency are best presented as categories where one can take advantage of the universality of constructions (see Winskel and Nielsen (1995)). If domain theory is to meet models for concurrency, it seems that the points of information in a complete partial order need to be replaced by more detailed objects in a category. A problem with the categories of traditional models is that they do not support higher-order constructions. Presheaf categories, on the other hand, not only include important traditional models like synchronisation trees and event structures, but also support function spaces. We see presheaf models as taking us towards a new domain theory in which presheaf models are analogous to nondeterministic domains (Hennessy and Plotkin 1979).

Another motivation comes in getting a more systematic and algebraic understanding of bisimulation. A traditional way to proceed in giving a theory to

* Basic Research in Computer Science, a centre of the Danish National Research Foundation.

1 In the sense that these traditional models embed fully, faithfully and densely in particular presheaf models (Joyal, Nielsen, and Winskel 1996).
a process language has been first to endow it with an operational semantics, provide a definition of bisimulation, follow this by the task of verifying that the bisimulation is a congruence (maybe by modifying it a bit), and then establish proof rules. Often the pattern is standard, but sometimes even getting a passable definition of bisimulation can be tricky, as, for example, when higher-order features are involved. An advantage of presenting models for concurrency as categories has been that they then support a general definition of bisimulation based on open maps (Joyal et al. 1996). We can then exploit the universality of various constructions in showing that they preserve bisimulation (Cattani and Winskel 1997). Presheaf categories come along with a concept of open map and bisimulation. They themselves form a category (strictly a bicategory of profunctors), in which the objects are presheaf categories, yielding ways to combine presheaf categories and their bisimulations.

The problem with general definitions is that it’s not always so easy to see what their instances amount to. Indeed, a major task of this paper is showing that the bisimulation on processes of the π-calculus obtained from open maps coincides with a traditional definition (following up on earlier work for value-passing processes (Winskel 1996)). However, the presheaf model for the π-calculus contributes more than this. Along the way, the presheaf model casts light on bisimulation for the π-calculus and why operations preserve it; the normal form for processes in the π-calculus (Fiore et al. 1996), which can be read off from the definition of the model (Theorem 5); and suggests smooth translations between variants of the π-calculus (Section 5). We also claim that, compared to the domain model, the “domain equation” of our model is rather simpler, because we seek a category of paths, not processes; the presheaf construction then fills in the necessary nondeterminism. As a result the system is particularly flexible: here we present the full ‘late’ π-calculus in detail, but we also sketch how the same category holds models of the ‘early’ π-calculus and other popular variants.

The recent domain models for the π-calculus lie within a functor category \( \mathcal{Cpo} \) (Fiore et al. 1996; Stark 1996). Here \( \mathcal{I} \) is the category of finite sets of names and injections between them, representing the fact that over time new names may be created and old names relabelled to avoid clashes. Hennessy (1996) has followed the same approach in his model for testing equivalences. The key to capturing the π-calculus is that the categorical requirements of functoriality and naturality give uniformity over varying name sets. Most notably \( \mathcal{Cpo} \) is cartesian closed, and the function space correctly handles the fact that old processes must be prepared to receive new names. This paper makes the same step up in the presheaf approach, to give a model of the π-calculus not in \( \mathcal{Cpo} \) but in \( \mathcal{Cpo}^\mathcal{I} \).

1.1 The π-calculus

The version of the π-calculus we use is entirely standard. We summarise it only very briefly here: for discussion and further detail see the original papers (Milner, Parrow, and Walker 1992a,b; Milner 1991). Processes have the following syntax

\[
P :: = \bar{x}y.P \mid x(y).P \mid \nu x P \mid [x=y]P \mid 0 \mid P + P \mid P | P \mid !P
\]