On Bisimulations for the Asynchronous $\pi$-Calculus *

Roberto M. Amadio  Ilaria Castellani  Davide Sangiorgi
CNRS, Sophia-Antipolis  INRIA, Sophia-Antipolis  INRIA, Sophia-Antipolis

Abstract

The asynchronous $\pi$-calculus is a variant of the $\pi$-calculus where message emission is non-blocking. Honda and Tokoro have studied a semantics for this calculus based on bisimulation. Their bisimulation relies on a modified transition system where, at any moment, a process can perform any input action.

In this paper we propose a new notion of bisimulation for the asynchronous $\pi$-calculus, defined on top of the standard labelled transition system. We give several characterizations of this equivalence including one in terms of Honda and Tokoro's bisimulation, and one in terms of barbed equivalence. We show that this bisimulation is preserved by name substitutions, hence by input prefix. Finally, we give a complete axiomatization of the (strong) bisimulation for finite terms.

1 Introduction

Process interaction in a distributed system without global clock is usually modelled by message passing. In this context, one often distinguishes between synchronous and asynchronous message passing. In the former, the send and receive events can be regarded as happening at the same time. In the latter, one can imagine that messages are sent and travel in the ether till they reach their destination, while the sending process accomplishes other tasks.

In the distributed algorithms community the distinction synchronous vs. asynchronous communication is not considered a very important issue. For instance [Tel95], pp 44 says:

Messages in distributed systems can be passed either synchronously or asynchronously. (...) For many purposes synchronous message passing can be regarded as a special case of asynchronous message passing (...)

Indeed one can simulate a synchronous communication with two asynchronous ones. On the other hand in the language design community the distinction seems

*The authors were partially supported by France Télécom, CTI-CNET 95-1B-182 Modélisation de Systèmes Mobiles.
to be quite relevant. Basically, asynchronous communication is easier to implement than the synchronous one as it is closer to the communication primitives offered by available distributed systems. In particular, asynchronous communication has become a popular choice in the design of languages for the programming of distributed applications. An early proposal is Agha's actors model [Agh86], while more recent contributions based on the theory of the \( \pi \)-calculus include Pict [PT96] and the join calculus [FG96].

A second community where the distinction synchronous vs. asynchronous is gaining momentum is that concerned with the semantics of programs. In this community one is often interested in comparing calculi. Certain translations turn out to be fully abstract in an asynchronous setting, where the observer has less power. Examples include the encoding of input-guarded choice [NP96] into the asynchronous \( \pi \)-calculus and the encoding of the asynchronous \( \pi \)-calculus into the join calculus [FG96].

A way to restrict a process calculus to asynchronous communications is to remove output prefixing. In other terms, an asynchronous output \( \alpha \) followed by a process \( P \) is the same as the parallel composition \( \alpha | P \). If the calculus has a non-deterministic sum, then we also disallow output guards. We can justify this decision as follows: (i) An output on a choice point forces synchronizations at the implementation level, this seems to contradict the very essence of asynchronous communication (we are not aware of any programming language which allows this). (ii) At the semantic level a calculus with output guards is more discriminating, in particular certain desirable equations such as (2) in section 4 fail to hold.

The resulting calculus is still quite expressive when working in a framework where channel names are transmissible values, e.g. the \( \pi \)-calculus [MPW92]. Indeed it is quite easy to simulate the synchronous \( \pi \)-calculus in the asynchronous one: the sending process waits for an acknowledgment from the receiving process on a private channel. Basic results on the expressiveness of the asynchronous \( \pi \)-calculus can be found in the works by Honda and Tokoro, and Boudol [HT91, Bou91], where the asynchronous \( \pi \)-calculus was first proposed.

When communications are asynchronous, the sender of an output message does not know when the message is actually consumed. In other words, an asynchronous observer, as opposed to a synchronous one, cannot directly detect the input actions of the observed process. Consequently, the asynchronous calculus requires the development of an appropriate semantic framework.

In this paper we develop a theory of bisimulation for the asynchronous \( \pi \)-calculus both in the strong and in the weak case. Our starting point is an original notion of asynchronous bisimulation over the standard labelled transition system. As a first contribution, we provide several characterizations of this bisimulation, and in particular we study under which conditions it coincides with barbed equivalence. We also show that our asynchronous bisimulation coincides with that proposed by Honda and Tokoro, which is based on a modified transition system for the \( \pi \)-calculus, on the sublanguage that they consider. As a second result, we observe that asynchronous bisimulation is preserved by the input prefix of the \( \pi \)-calculus (a similar property is proved in [HT92]) and coincides with ground bisimulation (a bisimulation where only one fresh name is considered in the input clause). Finally, we give a complete axiomatization of asynchronous bisimulation in the strong case for finite terms.