Kohei and session types

Vasco Thudichum Vasconcelos
University of Lisbon
ETAPS, March 2013

1989


KEIO UNIVERSITY
1858
CALAMVS GLADIO FORTIOR

Mario Tokoro

Alzi' Yon

LFCS

Laboratory for Foundations of Computer Science
Department of Computer Science - University of Edinburgh

A Calculus of Mobile Processes, Part 1

A Calculus of Mobile Processes, Part I

by

Robin Milner
Joachim Parrow
David Walker

LFCS Report Series

ECS-LFCS-89-85
(also published as CSR-302-89)

LFCS
Department of Computer Science
University of Edinburgh
The King's Buildings

June 1989

INRIA

UNITÉ DE RECHERCHE
INRIA-SOPHIA ANTIPOLIS

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tél.:(1) 39 63 55 11

Rapports de Recherche

N° 1154

Programme 1
Programmation, Calcul Symbolique
et Intelligence Artificielle

FUNCTIONS AS PROCESSES

Robin MILNER

Février 1990

RR-1154

Corrections in red
- RM15

3-3
3-9
3-10
4-2
4-5
4-6
4-7

Sorts and Types in the π-calculus

Robin Milner

December 1990

1. Introduction
2. Abstractions, objects and sorts
3. Examples of sortings
4. Types, with elementary examples
5. Types of λ-agents
6. Higher-order?

APPENDIX: Structural congruence, actions, and transition rules

1991

An Object Calculus for Asynchronous Communication*

Kohei Honda and Mario Tokoro†

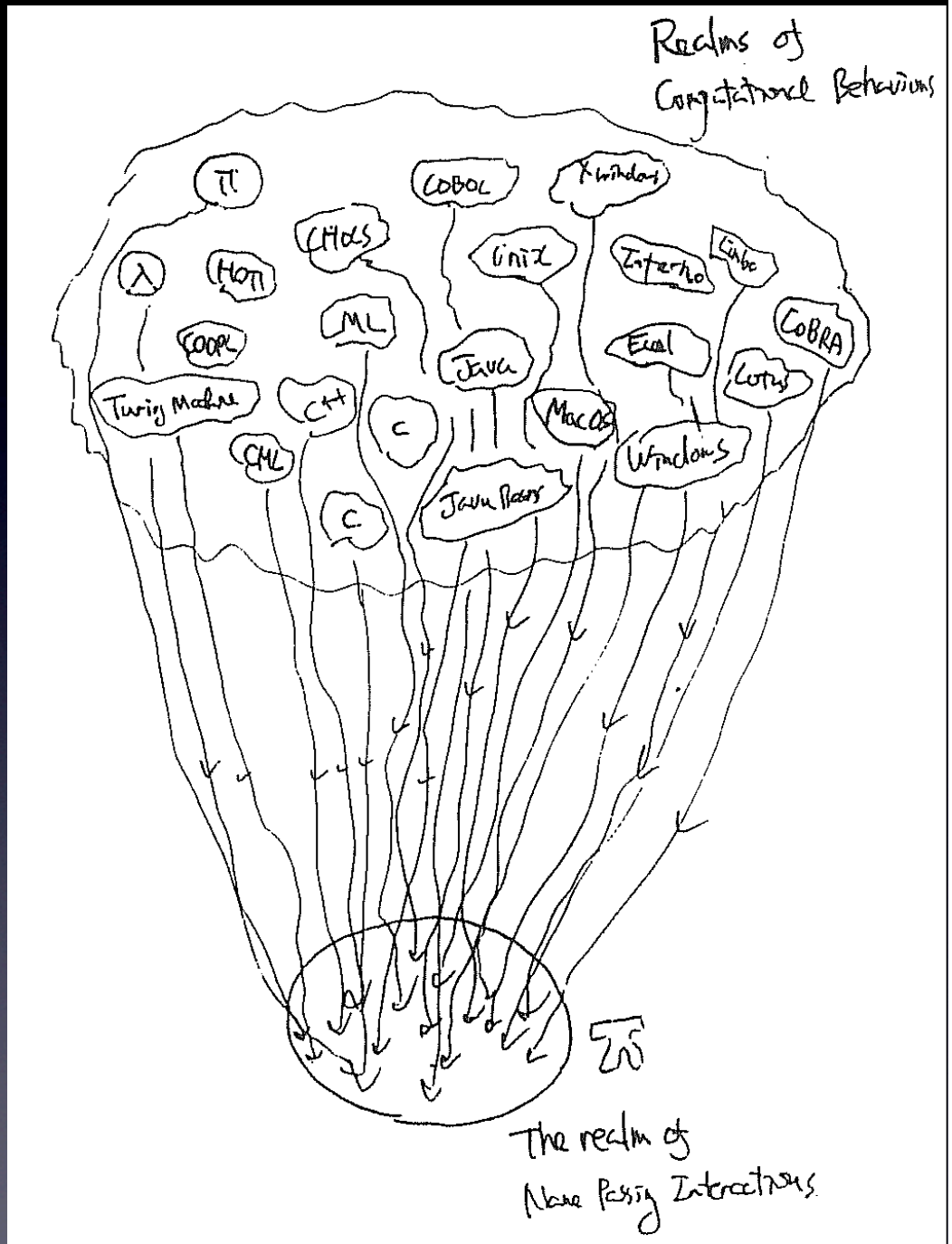
Department of Computer Science,
Keio University,
3-14-1 Hiyoshi, Kohoku-ku, Yokohama, 223,
Japan

Abstract

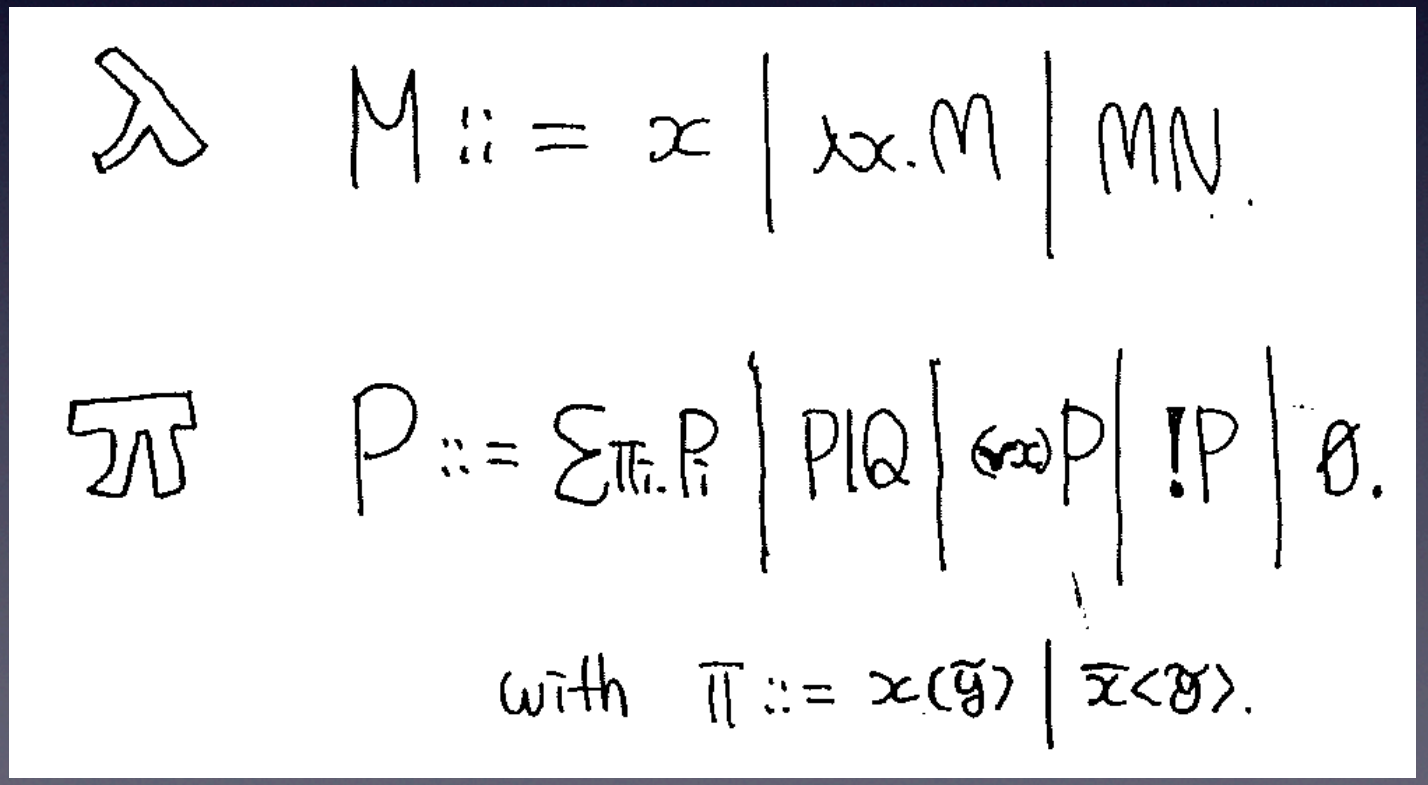
This paper presents a formal system based on the notion of objects and asynchronous communication. Built on Milner's work on π-calculus, the communication primitive of the formal system is purely asynchronous, which makes it unique among various concurrency formalisms. Computationally this results in a consistent reduction of Milner's calculus, while retaining the same expressive power. Seen semantically asynchronous communication induces a surprisingly different framework where bisimulation is strictly more general than its synchronous counterpart. This paper shows basic construction of the formal system along with several illustrative examples.

ECOOP 1991: Geneva, Switzerland

Every computational behaviour can be reduced to name passing



lambda to pi: "important suggestions at many stages of formal development"



1993

On Reduction-Based Process Semantics*

Kohei Honda Nobuko Yoshida

Department of Computer Science, Keio University
3-14-1 Hiyoshi, Kohoku-ku, Yokohama, 223, Japan

Abstract. A formulation of semantic theories for processes which is based on reduction relation and equational reasoning is studied. The new construction can induce meaningful theories for processes, both in *strong* and *weak* settings. The resulting theories in many cases coincide with, and sometimes generalise, observation-based formulation of behavioural equivalence. The basic construction of reduction-based theories is studied, taking a simple name passing calculus called ν-calculus as an example. Results on other calculi are also briefly discussed.

FSTTCS 1993: Bombay, India

Principal typing schemes in a polyadic π-calculus*

Vasco T. Vasconcelos *vasco@mt.cs.keio.ac.jp*
Kohei Honda *kohei@mt.cs.keio.ac.jp*
Department of Computer Science
Keio University
3-14-1 Hiyoshi Kohoku-ku Yokohama 223
Japan

Abstract

The present paper introduces a typing system for a version of Milner's polyadic π-calculus, and a typing inference algorithm linear on the size of the input. The central concept underlying the typing system is the notion of type assignment, where each free name in a term is assigned a type, the term itself being given multiple name-type pairs. This observation leads to a clean typing system for Milner's sorting, and induces an efficient algorithm to infer the typing of a term. The typing system enjoys a subject-reduction property and possesses a notion of principal typing scheme. The algorithm to reconstruct the principal typing scheme of a process, or to detect its inexistence, is proved correct with respect to the typing system.

CONCUR 1993: Hildesheim, Germany

Combinatory Representation of Mobile Processes *

Kohei Honda †
kohei@mt.cs.keio.ac.jp

Nobuko Yoshida
yoshida@mt.cs.keio.ac.jp

Department of Computer Science, Keio University
3-14-1, Hiyoshi, Kohoku-ku, Yokohama 223, Japan

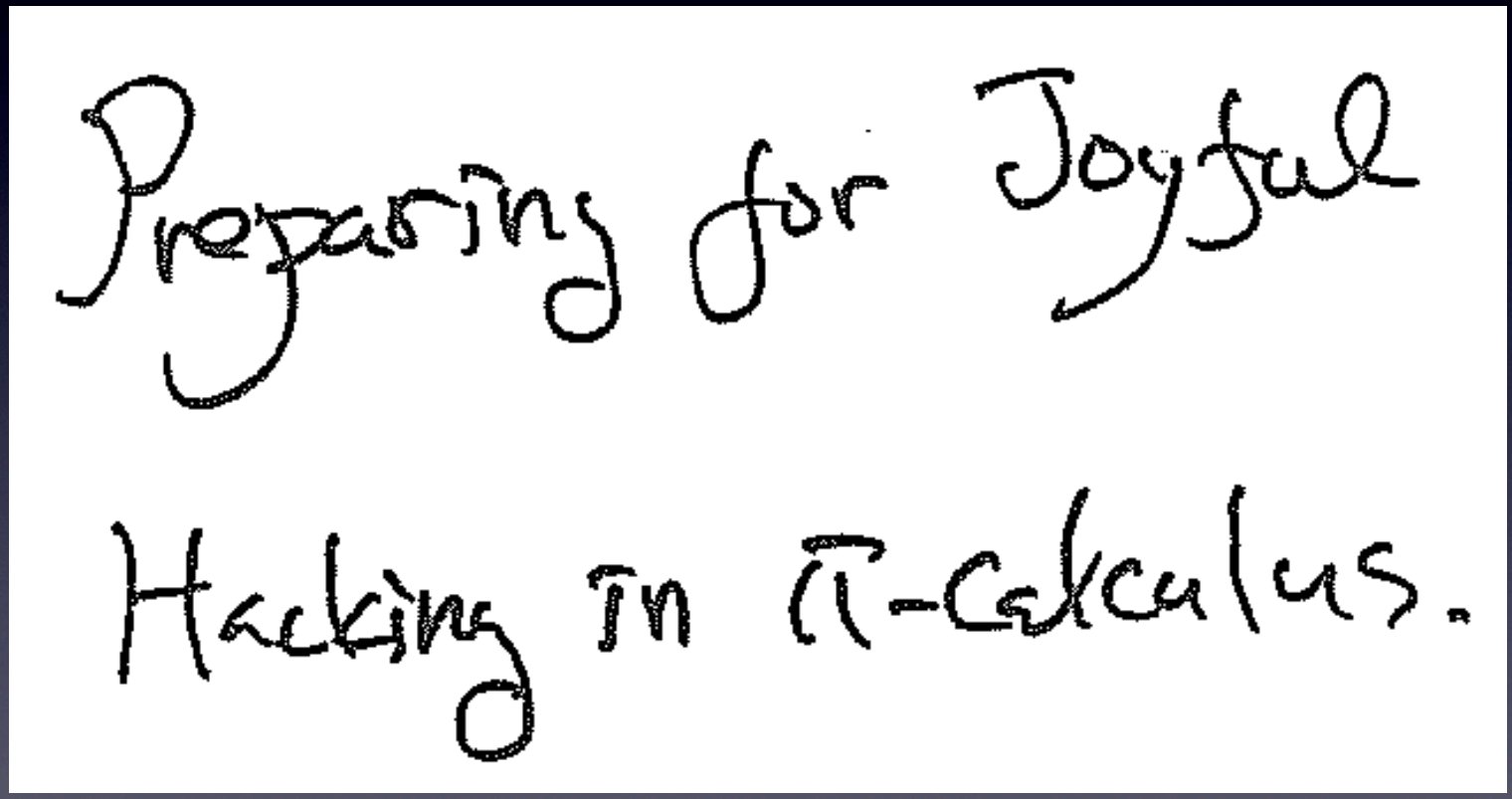
Abstract

A certain analogue of theory of combinators in the setting of concurrent processes is formulated. The new combinators are derived from the analysis of the operation called *asynchronous name passing*, just as the analysis of *logical substitution* gave rise to the sequential combinators. A system with seven atoms and fixed interaction rules, but with no notion of prefixing, is introduced, and is shown to be capable of representing input and output prefixes over arbitrary terms in a behaviourally correct way, just as SK-combinators are closed under functional abstraction without having it as a proper syntactic construct. The basic equational correspondence between concurrent combinators and a system of asynchronous mobile processes, as well as the embedding of the finite part of π-calculus in concurrent combinators, is proved. These results will hopefully serve as a cornerstone for further investigation of the theoretical as well as pragmatic possibilities of the arbitrarily complex applicative behaviour. This ultimately led us to the notion of *combinatory algebra* as a semantic foundation of typed and untyped λ-calculi. At the same time, the decomposition of the application-substitution process into finite dynamics in combinators has had a profound impact on the execution schemes of modern functional programming languages. Truly, parallel developments in the study of λ-calculi and that of combinators are essential to our current practice of sequential programming, both theoretical and pragmatic.

Such parallel developments, however, have not been known in the world of concurrent processes. Nor, at least until recently, has one agreed upon the existence of such an essential operation as *β-reduction* in the concurrency setting. Yet nowadays we find, especially among researchers on concurrency, growing interest in one simple yet powerful primitive, which, when coupled with basic operators like concurrent composition and name hiding, can represent quite versatile structures of concurrent computation. The operation is *name passing*,

POPL 1994: Portland, Oregon, USA

Types for interaction



“structuring constructs for communication-based programming which are born through the examination of pi-calculus encoding of various computational structures”

Sequentialisation (1)

- As a first step, we wish to realise sequencing in communication:

$$a!\langle x_1 \ldots x_n \rangle.P \mid \bar{a}:[V_1 \ldots V_n] \rightarrow P_{[V_1 \ldots V_n / x_1 \ldots x_n]} \quad (*)$$

cf. currying in λ-calculus.

$$\lambda(x_1 \ldots x_n).M \equiv \lambda x_1 \lambda x_2 \ldots \lambda x_n . M.$$

Branching (1).

- Next problem: Can we realise:

$$a:[\mathit{left}:P]\,\&\,[\mathit{right}:Q] \mid \bar{a}.\mathit{left} \rightarrow P,$$

$$a:[\mathit{left}:P]\,\&\,[\mathit{right}:Q] \mid \bar{a}.\mathit{right} \rightarrow Q.$$

I.e. branching/selection or method invocation?

Types for Dyadic Interaction*

Kohei Honda

kohei@mt.cs.keio.ac.jp

Department of Computer Science, Keio University

3-14-1 Hiyoshi, Kohoku-ku, Yokohama, 223, Japan

Abstract

We formulate a typed formalism for concurrency where types denote freely composable structure of dyadic interaction in the symmetric scheme. The resulting calculus is a typed reconstruction of name passing process calculi. Systems with both the explicit and implicit typing disciplines, where types form a simple hierarchy of types, are presented, which are proved to be in accordance with each other. A typed variant of bisimilarity is formulated and it is shown that typed β-equality has a clean embedding in the bisimilarity. Name reference structure induced by the simple hierarchy of types is studied, which fully characterises the typable terms in the set of untyped terms. It turns out that the name reference structure results in the deadlock-free property for a subset of terms with a certain regular structure, showing behavioural significance of the simple type discipline.

CONCUR 1993: Hildesheim, Germany

An Interaction-based Language and its Typing System*

Kaku Takeuchi
kaku@mt.cs.keio.ac.jp

Kohei Honda
kohei@mt.cs.keio.ac.jp

Makoto Kubo
kubo@mt.cs.keio.ac.jp

Department of Computer Science, Keio University
3-14-1 Hiyoshi, Kohoku-ku, Yokohama, 223, Japan

Abstract

We present a small language $\mathcal{L}$ and its typing system based on the idea of interaction, one of the important notions in parallel and distributed computing. $\mathcal{L}$ is based on, apart from such constructs as parallel composition and process creation, three pairs of communication primitives which use the notion of a *session*, a semantically atomic chain of communication actions which can interleave with other such chains freely, for high-level abstraction of interaction-based computing. Three primitives enable programmers to elegantly describe complex interactions among processes with a rigorous type discipline similar to ML [4]. The language is given formal operational semantics and a type inference system, regarding which we prove that if a program is well-typed in the typing system, it never causes run-time error due to type inconsistent communication patterns, offering a new foundation for type discipline in parallel programming languages.

PARLE 1994: Athens, Greece

"*session*, a semantically atomic chain of communication actions which can interleave with other such chains freely, for high-level abstraction of interaction-based computing."

LANGUAGE PRIMITIVES AND TYPE DISCIPLINE FOR STRUCTURED COMMUNICATION-BASED PROGRAMMING

KOHEI HONDA*, VASCO T. VASCONCELOS†, AND MAKOTO KUBO‡

ABSTRACT. We introduce basic language constructs and a type discipline as a foundation of structured communication-based concurrent programming. The constructs, which are easily translatable into the summation-less asynchronous π-calculus, allow programmers to organise programs as a combination of multiple flows of (possibly unbounded) reciprocal interactions in a simple and elegant way, subsuming the preceding communication primitives such as method invocation and rendez-vous. The resulting syntactic structure is exploited by a type discipline à la ML, which offers a high-level type abstraction of interactive behaviours of programs as well as guaranteeing the compatibility of interaction patterns between processes in a well-typed program. After presenting the formal semantics, the use of language constructs is illustrated through examples, and the basic syntactic results of the type discipline are established. Implementation concerns are also addressed.

ESOP 1998: Lisbon, Portugal

Thank you