Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 203 (2008) 79–92

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

Interaction Nets With Nested Pattern Matching

Abubakar Hassan[a,1] and Shinya Sato[b,2]

[a] *Department of Computer Science*
King's College London
Strand, London WC2R 2LS, UK

[b] *Faculty of Econoinformatics*
Himeji Dokkyo University
7-2-1 Kamiohno, Himeji-shi, Hyogo 670-8524, JAPAN

Abstract

Reduction rules in Interaction Nets are constrained to pattern match exactly one argument at a time. Consequently, a programmer has to introduce auxiliary rules to perform more sophisticated matches. We propose an extension of Interaction Nets which facilitates nested pattern matching on interaction rules. We then define a practical compilation scheme from extended rules to pure interaction rules. We achieve a system that provides convenient ways to express Interaction Net programs without defining auxiliary rules.

Keywords: Interaction nets, pattern matching, programming language design.

1 Introduction

Interaction Nets [5] can be considered as a graphical–or visual–programming language. Programs are expressed as graphs, and computation is graph reduction. From another perspective, Interaction Nets are also a low level implementation language: we can define systems of Interaction Nets that are instructions for the target of compilation schemes of other programming languages. For instance, Interaction Nets have been used for the implementation of optimal reduction [4,6] and other efficient implementations of the λ-calculus [8]. In addition, there has been various implementations of Interaction Nets [7,9]. Despite that we can already program in Interaction Nets (they are Turing complete), they still remain far from being used as a programming language. Drawing an analogy with functional programming, we

[1] abubakar.hassan@kcl.ac.uk
[2] shinya@himeji-du.ac.jp

doi:10.1016/j.entcs.2008.03.035

only have the pure λ-calculus without syntactic sugar, constants, data-structures, etc.

In this paper we take a step towards developing a richer language based on Interaction Nets. Interaction Nets have a very primitive notion of pattern matching since only two agents can interact at a time. Consequently, many auxiliary agents and rules are needed to implement more sophisticated matches. These auxiliaries are implementation details and should be generated automatically other than by the programmer. To achieve this, we extend Interaction Nets to allow rules with nested patterns to be defined. We then give a compilation scheme from extended to ordinary interaction rules.

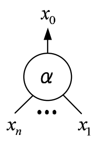
There has been several works that extend Interaction Nets in some way (see Section 6.2). Sinot and Mackie's Macros for Interaction Nets [10] are quite close to what we present in this paper. They allow pattern matching on more than one argument by relaxing the restriction of one principal port per agent. The main difference with our work is that their system does not allow nested pattern matching. Our system facilitates nested/deep pattern matching of agents.

The rest of this paper is organised as follows: In the next section we give a brief introduction to Interaction Nets. In Section 3 we motivate our work through an example. We give the proposed extensions in Section 4, followed by the compilation schemes in Section 5. In Section 6 we discuss some implementation issues. Finally, we conclude the paper in Section 7

2 Interaction Nets

We review the basic notions of Interaction Nets. See [5] for a more detailed presentation. Interaction Nets are specified by the following data:

- A set Σ of *symbols*. Elements of Σ serve as *agent* (node) labels. Each symbol has an associated arity *ar* that determines the number of its *auxiliary ports*. If $ar(\alpha \in \Sigma) = n$, then α has $n+1$ *ports:* n auxiliary ports and a distinguished one called the *principal port*.



  We use the textual notation $x_0 - \alpha(x_1, ..., x_n)$ to represent an agent α where x_0 is the principal port and $x_1, ..., x_n$ are its auxiliary ports.

- A *net* built on Σ is an undirected graph with agents at the vertices. The edges of the net connect agents together at the ports such that there is only one edge at every port. A port which is not connected is called a *free port*.

- Two agents $(\alpha, \beta) \in \Sigma \times \Sigma$ connected via their principal ports form an *active pair* (analogous to a redex). An interaction rule $((\alpha, \beta) \to N) \in \mathcal{R}$ replaces the pair (α, β) by the net N. All the free ports are preserved during reduction, and there is at most one rule for each pair of agents. The following diagram illustrates the