Efficient $\lambda$-Evaluation with Interaction Nets

Ian Mackie
Department of Computer Science
King’s College London
Strand, London WC2R 2LS, UK
ian@dcs.kcl.ac.uk

Abstract. This paper presents an efficient implementation of the $\lambda$-calculus using the graph rewriting formalism of interaction nets. Building upon a series of previous works, we obtain one of the most efficient implementations of this kind to date: outperforming existing interaction net implementations, as well as other approaches. We conclude the paper with extensive testing to demonstrate the capabilities of this evaluator.

1 Introduction

One of the first algorithms to implement Lévy’s [9] notion of optimal reduction for the $\lambda$-calculus was presented by Lamping [7]. With the help of linear logic [4], this algorithm was tidied up and lead to the well-known algorithm of Gonthier, Abadi and Lévy [5]. Empirical and theoretical studies of this algorithm have revealed several causes of inefficiency (accumulation of certain nodes in the graph rewriting formalism). Asperti et al. [1] devised BOHM (Bologna Optimal Higher-Order Machine) to overcome some of these issues, which has stood until now as not only the most efficient (in terms of rewriting steps) implementation of optimal reduction, but also the most efficient implementation of the $\lambda$-calculus.

Interaction nets [6] (a particular form of graph rewriting) has played a role in the above algorithms. However, the focus has been on optimality, rather than on using the interaction net framework. A parallel thread of work takes interaction nets as a focus point rather than optimality. An important reason for this choice is that, in addition to offering insight into issues such as sharing computation, they provide an operational model which captures all the computation: in other words, counting the rewrite steps is sufficient to measure the cost of a computation. The key observation here is that in the previously mentioned implementations of the $\lambda$-calculus, $\beta$-reduction (not including substitution) is just another graph rewrite. Our aim therefore is to use a more pragmatic approach to optimal reduction where we aim to find the minimum number of rewrite steps ($\beta$ included). Historically, this notion of “practical” optimality began in [11], based on an interaction net encoding of linear logic due to Abramsky. Although this first $\lambda$-evaluator based on interaction nets performed fewer interactions (rewrite steps) for specific $\lambda$-terms than Lamping’s algorithm, it was never a match for BOHM. A further attempt, YALE [12], provided a substantial improvement.
which can systematically perform better than Lamping’s algorithm, and approximate BOHM on specific classes of terms. However, when the need for optimality kicks in, YALE is a very poor second best.

A question therefore remained: is there an efficient interaction net implementation of the \(\lambda\)-calculus which does less work than BOHM? The purpose of the present paper is to answer this question in the positive. Specifically, we give a new \(\lambda\)-evaluator: KCLE (King’s College Lambda Evaluator) which has the following features:

- It is efficient: although KCLE performs more \(\beta\)-reduction steps than optimal reducers, the overall number of graph rewrite steps is smaller.
- It evaluates \(\lambda\)-terms to full normal form, even for open terms (as a side effect, this offers a relatively simple notion of read-back, as normal forms are images of the translation function).
- It is an interaction net, so we can take advantage of many results and implementations, specifically parallel, where almost linear speedup has been achieved. We discuss other advantages of interaction nets later.

Relation to Previous Work. The present paper is a continuation of a programme of research by the author to use interaction nets as an efficient mechanism for the encoding of the \(\lambda\)-calculus. Specifically, it builds upon two pieces of work: [11] and [12]. It is also related to the work on interaction nets for Linear Logic [13].

Overview. The rest of this paper is structured as follows. In the next section we recall interaction nets, and motivate why we use them. In Section 3 we give the translation of the \(\lambda\)-calculus into interaction nets. Section 4 studies the reduction system. In Section 5 we examine properties of the encoding. Section 6 gives experimental evidence of our results, where we compare with other systems. Finally, we conclude the paper in Section 7.

2 Interaction Nets

An interaction net system [6] is specified by giving a set \(\Sigma\) of symbols, and a set \(\mathcal{R}\) of interaction rules. Each symbol \(\alpha \in \Sigma\) has an associated (fixed) arity. An occurrence of a symbol \(\alpha \in \Sigma\) will be called an agent. If the arity of \(\alpha\) is \(n\), then the agent has \(n + 1\) ports: a distinguished one called the principal port depicted by an arrow, and \(n\) auxiliary ports labelled \(x_1, \ldots, x_n\) corresponding to the arity of the symbol. Such an agent will be drawn in the following way:

\[
\begin{array}{c}
\alpha \\
\downarrow \\
\ldots \\
x_1 \\
\end{array}
\]

A net \(N\) built on \(\Sigma\) is a graph (not necessarily connected) with agents at the vertices. The edges of the graph connect agents together at the ports such that there is only one edge at every port. The ports of an agent that are not connected