The Essence of Eta-Expansion in Partial Evaluation

OLIVIER DANVY, KAROLINE MALMKJÆR, AND JENS PALSBERG

{danvy,karoline,palsberg}@daimi.aau.dk

Computer Science Department, Aarhus University, Ny Munkegade, DK-8000 Aarhus C, Denmark

Abstract. Selective eta-expansion is a powerful “binding-time improvement”, i.e., a source-program modification that makes a partial evaluator yield better results. But like most binding-time improvements, the exact problem it solves and the reason why have not been formalized and are only understood by few.

In this paper, we describe the problem and the effect of eta-redexes in terms of monovariant binding-time propagation: eta-redexes preserve the static data flow of a source program by interfacing static higher-order values in dynamic contexts and dynamic higher-order values in static contexts. They contribute to two distinct binding-time improvements.

We present two extensions of Gomard's monovariant binding-time analysis for the pure A-calculus. Our extensions annotate and eta-expand A-terms. The first one eta-expands static higher-order values in dynamic contexts. The second also eta-expands dynamic higher-order values in static contexts.

As a significant application, we show that our first binding-time analysis suffices to reformulate the traditional formulation of a CPS transformation into a modern one-pass CPS transformer. This binding-time improvement is known, but it is still left unexplained in contemporary literature, e.g., about “cps-based” partial evaluation.

We also outline the counterpart of eta-expansion for partially static data structures.

Keywords: Two-level λ-calculus, binding-time analysis, coercions.

1. Introduction

Partial evaluation is a program-transformation technique for specializing programs [10], [15]. In the last decade it has been described using the notion of binding times [18]. Essentially the computations in a source program are divided into “static” or specialization-time computations (performed by the partial evaluator) and “dynamic” or run-time computations (to be performed in the specialized program). Partial evaluation amounts to performing the static computations and constructing the specialized program so that running it performs the dynamic computations. Thus a partial evaluator evaluates static expressions (i.e., expressions that only depend on partial-evaluation time data) and reconstructs dynamic expressions (i.e., expressions that depend on run-time data). For this to work, the binding-time division must be congruent (also called consistent) [14], [21], [22], i.e., no static computation may depend on the result of a dynamic computation.

In this setting, two sorts of expressible values coexist: static values and dynamic values (i.e., residual expressions); correspondingly two sorts of contexts coexist:
static contexts and dynamic contexts. Recall that a context is an expression with one hole [I]. A higher-order (resp. partially static) context is an expression with a higher-order (resp. partially static) hole. A static (resp. dynamic) context is an expression with a static (resp. dynamic) hole. A hole is static (resp. higher-order, partially static, and dynamic) whenever the expression fitting this hole is static (resp. higher-order, partially static, and dynamic).

To obtain consistency, Mix-style partial evaluators [15] coerce static values and contexts to be respectively dynamic values and dynamic contexts, when they encounter a clash. This is acceptable if source programs are first-order and values are either fully static or fully dynamic. However these coercions are excessive for higher-order programs with partially static values and contexts.

Lacking better interface between higher-order and dynamic, source programs must often be modified “to improve their binding times” and thus “to make them specialize better”. In Section 12.4 of their textbook [15], Jones, Gomard, and Sestoft list eta-expansion as an effective binding-time improvement but give only a brief idea of why it works.

In the following section, we use the term dynamize, with the meaning “make dynamic”, to characterize the effect of eta-expansion. We explain how eta-redexes prevent a binding-time analysis from

- dynamizing static values in dynamic contexts, and
- dynamizing static contexts around dynamic values

when the values are higher-order. Preventing static values and contexts from being dynamized improves the annotation in case the static values are used elsewhere or in case other static values may also occur in the same context. In Section 3 we present two binding-time analyses that insert eta-redexes automatically, and we illustrate them with two continuation-based program transformations. Section 4 outlines the counterpart of eta-expansion for partially static data structures. After a comparison with related work, we conclude.

2. The essence of eta-expansion

We show three examples, where

- a number occurs both in a static and in a dynamic context,
- a higher-order value occurs both in a static and in a dynamic context, and
- a function is applied to both a static and a dynamic higher-order argument.

After the examples, we summarize why eta-expansion improves binding times, given a monovariant binding-time analysis. We use “@” (pronounced “apply”) to denote applications, and we abbreviate \((e_0@e_1)@e_2\) by \(e_0@e_1@e_2\) and \(e_0@(\lambda x.e)\) by \(e_0@\lambda x.e\).