Forward Slicing of Multi-paradigm Declarative Programs Based on Partial Evaluation

Germán Vidal

DSIC, Technical University of Valencia
Camino de Vera s/n, E-46022 Valencia, Spain
gvidal@dsic.upv.es

Abstract. Program slicing has been mainly studied in the context of imperative languages, where it has been applied to many software engineering tasks, like program understanding, maintenance, debugging, testing, code reuse, etc. This paper introduces the first forward slicing technique for multi-paradigm declarative programs. In particular, we show how program slicing can be defined in terms of online partial evaluation. Our approach clarifies the relation between both methodologies and provides a simple way to develop program slicing tools from existing partial evaluators.

1 Introduction

Essentially, program slicing [34] is a method for decomposing programs by analyzing their data and control flow. It has many applications in the field of software engineering (e.g., program understanding, maintenance, debugging, merging, testing, code reuse, etc). This concept was originally introduced by Weiser [33] in the context of imperative programs. Surprisingly, there are very few approaches to program slicing in the context of declarative programming (some notable exceptions are, e.g., [13,21,24,25,29]). Roughly speaking, a program slice consists of those program statements which are (potentially) related with the values computed at some program point and/or variable, referred to as a slicing criterion. Program slices are usually computed from a program dependence graph [10,19] that makes explicit both the data and control dependences for each operation in a program. Program dependences can be traversed backwards or forwards (from the slicing criterion), which is known as backward or forward slicing, respectively. Additionally, slices can be dynamic or static, depending on whether a concrete program’s input is provided or not. A complete survey on program slicing can be found, e.g., in [30].

The main purpose of partial evaluation techniques is to specialize a given program w.r.t. part of its input data—hence it is also known as program specialization. The partially evaluated program will be (hopefully) executed more

* This work has been partially supported by CICYT TIC 2001-2705-C03-01, by the Generalitat Valenciana under grant CTIDIA/2002/205, and by the MCYT under grants HA2001-0059, HU2001-0019 and HI2000-0161.
efficiently since those computations that depend only on the known data are performed—at partial evaluation time—once and for all. Many online partial evaluation schemes follow a common pattern: given a program and a partial call, the partial evaluator builds a finite representation—generally a graph—of the possible executions of the partial call and, then, systematically extracts a residual program (the partially evaluated program) from this graph. This view of partial evaluation clearly shows the similarities with program slicing: both techniques should construct a finite representation of some program execution, usually with part (or none) of the input data.

In this paper, we present a forward slicing method based on (online) partial evaluation. While the construction of a graph representing some program execution is quite similar in both techniques, the extraction of the final program is rather different. Partial evaluation usually achieves its effects by compressing paths in the graph and by renaming expressions in order to remove unnecessary function symbols. Hence, partial evaluation constructs a new, residual program. In contrast, program slicing should preserve the structure of the original program: statements can be (totally or partially) deleted but new statements cannot be introduced. Following [12], partial evaluators can be classified into the following categories:

- **monovariant**: each function of the original program gives rise to (at most) one residual function,
- **polyvariant**: each function of the original program may give rise to one or more residual functions,
- **monogenetic**: each residual function stems from one function of the original program, and
- **polygenetic**: each residual function may stem from one or more functions of the original program.

According to this classification, forward slicing can be seen as a particular form of monovariant and monogenetic partial evaluation (in order to preserve a one-to-one relation between the functions of the original and residual programs).

Unfortunately, monovariant/monogenetic partial evaluation could be rather imprecise, thus resulting in an unnecessarily large residual program (i.e., slice). To overcome this problem, we introduce an extended operational semantics to perform partial evaluations, which helps us to preserve as much information as possible while maintaining the monovariant/monogenetic nature of the process. In order to center the discussion, we present our developments in the context of a multi-paradigm declarative language which integrates features from functional and logic programming (like, e.g., Curry [17] or Toy [22]).

The main contributions of this work are the following: (1) We define the first forward slicing technique for functional logic programs. Moreover, the application of our developments to pure (lazy) functional programs would be straightforward, since either the syntax and the underlying (online) partial evaluators (e.g., positive supercompilation [28]) share many similarities. (2) Our method is defined in terms of an existing partial evaluation scheme; therefore, it is easy to implement by adapting current partial evaluators. Furthermore, we do not need