On the Degeneration of Program Generators by Program Composition*

Robert GLÜCK  
*Department of Computer Science,  
University of Copenhagen,  
Universitets parken 1  
DK-2100 Copenhagen, Denmark.

Andrei KLI MOV  
*Keldysh Institute of Applied Mathematics,  
Russian Academy of Sciences,  
Miusskaya Square 4  
RU-125047 Moscow, Russia.

Received 8 July 1996  
Revised manuscript received 27 February 1997

Abstract One of the main discoveries in the seventies was that the concept of a generating extension covers a very wide class of apparently different program generators. Program specialization, or partial evaluation, is powerful because it provides uniform techniques for the automatic implementation of generating extensions from ordinary programs. The Futamura projections stand as the cornerstone of the development of program specialization.

This paper takes the idea of the Futamura projections further. Three degeneration projections are formulated which tell us how to achieve the reverse goal by program composition, namely turning a generating extension into an ordinary program. The fact that program composition can invert the effect of program specialization shows that these projections are dual in a sense. The degeneration projections complete a missing link between programs and generating extensions and allow for novel applications of program transformation.

* This work was partly supported by the project “Design, Analysis and Reasoning about Tools” funded by the Danish Natural Sciences Research Council. The first author was also supported by an Erwin-Schrödinger-Fellowship of the Austrian Science Foundation (FWF) under grant J0780 & J0964; the second author partly by the Russian Basic Research Foundation grant 93-01-00628 & 96-01-01315, the US Office of Naval Research grant N00014-96-1-0800, and the US Civilian Research and Development Foundation grant RM1-254.
§1 Introduction

One of the main discoveries in the seventies was that the concept of a generating extension\(^1\) covers a very wide class of apparently different program generators. This has the big advantage that program generators for diverse applications such as parsing, translation, theorem proving, and pattern matching can be implemented with uniform techniques.\(^2\) Program specialization, or partial evaluation, is powerful because it provides transformation techniques for the automatic implementation of generating extensions.\(^3,4,5\) Program specialization can now be considered as one of the most advanced techniques for automatic program transformation. The Futamura projections\(^6\) stand as the cornerstone of this development.

This paper takes the idea of the Futamura projections further. Three degeneration projections are formulated which tell us how to reverse the effect of the Futamura projections by program composition, namely how to turn a generating extension into an ordinary program. The degeneration projections, similar in structure to the Futamura projections but their inverse, complete a missing link between programs and generating extensions. This is interesting in its own right and allows for several novel applications of program specialization and program composition. It is quite remarkable that, although program specialization has been used for some time to generate generating extensions, the reverse operation, degeneration, has not been studied before.

In this paper we shall mainly be concerned with what can be achieved by program composition, and not how it can be achieved. Although promising results have been obtained for program composition,\(^7,8,9,10\) this method is still at a research stage. The transformation problems presented here can be seen as test cases for existing methods, and as challenging goals for future research.

Suppose \(P\) is a program with two arguments and let \(<P X, Y>\) denote the application of program \(P\) to its input \(X, Y\). Computation of \(P\) producing result \(\text{Out}\) is described by

\[<P X, Y> \Rightarrow \text{Out}\]

A generating extension of \(P\) is a program \(\text{Gen}_P\) that takes one part of \(P\)'s input, say \(X\), and produces another program \(P_X\), the specialization of \(P\) to that input. The specialized program returns the same result when applied to the remaining input \(Y\) as the original program \(P\) when applied to input \(X, Y\). Computation in two stages is described by

\[<\text{Gen}_P X> \Rightarrow P_X\]
\[<P_X Y> \Rightarrow \text{Out}\]