Reflections on Reflections

Gilles Barthe\textsuperscript{1}, John Hatcliff\textsuperscript{2}, and Morten Heine Sørensen\textsuperscript{3}

\textsuperscript{1} Centrum voor Wiskunde en Informatica (CWI)\textsuperscript{\dagger}
\textsuperscript{2} Computer Science Department, Oklahoma State University\textsuperscript{\ddagger}
\textsuperscript{3} Department of Computer Science, University of Copenhagen (DIKU)\textsuperscript{§}

Abstract. In the functional programming literature, compiling is often expressed as a translation between source and target program calculi. In recent work, Sabry and Wadler proposed the notion of a \textit{reflection} as a basis for relating the source and target calculi. A reflection elegantly describes the situation where there is a kernel of the source language that is isomorphic to the target language. However, we believe that the reflection criteria is so strong that it often excludes the usual situation in compiling where one is compiling from a higher-level to a lower-level language.

We give a detailed analysis of several translations commonly used in compiling that fail to be reflections. We conclude that, in addition to the notion of reflection, there are several relations weaker a reflection that are useful for characterizing translations. We show that several familiar translations (that are not naturally reflections) form what we call a \textit{reduction correspondence}. We introduce the more general notion of a \((R_1, R_2, R_3, R_4)\)-correspondence as a framework for describing relations between source and target calculi.

1 Introduction

In the functional programming literature, compiling is often expressed as a translation between source and target program calculi. The target calculus is typically based on continuation-passing style (CPS) terms \cite{1,27}, monadic-style terms \cite{3,12}, \textit{A}-normal forms \cite{10}, or some other sort of intermediate language that makes explicit things such as intermediate values and closure operations. In recent work \cite{23,24}, Sabry and Wadler question: what is the appropriate relationship between source and target calculi?

Program calculi are often presented as equational theories. So one might answer the above question by stating that compiling should preserve the equality relation of the source calculi. That is,

\begin{equation}
M =_S N \implies M^* =_T N^*
\end{equation}

where \(=_S\) and \(=_T\) denote the convertibility relation in the source and target calculi (respectively), and where \(M^*\) denotes the compilation of \(M\). One might

\textsuperscript{\dagger} PO Box 94079, 1090 GB Amsterdam, The Netherlands, gilles@cwi.nl
\textsuperscript{\ddagger} 219 Math Sciences, Stillwater, OK, USA, 74078, hatcliff@cs.okstate.edu
\textsuperscript{§} Universitetsparken 1, DK-2100 Copenhagen, Denmark, rambo@diku.dk
even require that the converse of the above implication holds (i.e., that compiling preserves and reflects $\equiv_s$).

However, it is usually more fruitful to focus on the reduction theories which typically underlie the equational theories. For example, one might require that

$$M \rightarrow_S N \text{ implies } M^* \rightarrow_T N^*. \quad (2)$$

If reductions are taken to represent computational steps or optimizations, this states that a series of computational steps in the source language can be expressed in the target language. Again, the converse may also be of interest: one would like that every optimization expressible in the target calculus be reflected in the source calculus as well. This is especially advantageous if the conceptual complexity of the target language (e.g., CPS) is significantly greater than that of the source language. Such a result would allow one to reason about the optimizations in the target language while working only with the source language.

Sabry and Felleisen [22] popularized the use of a *decompiling* translation for establishing the converse of the implications. For example, for the latter implication at line (2), one might define a decompiling translation $\dagger$ which maps target language terms back to source terms and require that

$$P \rightarrow_T Q \text{ implies } P^\dagger \rightarrow_S Q^\dagger.$$  

This, along with some other simple properties describing the interaction between $*$ and $\dagger$ is sufficient to establish the converse of line (2).

Now that there are two calculi (source and target) and two translations (compiling and decompiling) under consideration, there are are variety of ways in which these four components can be related. Sabry and Wadler sought to emphasize reduction properties over equational properties, and they adopted the topological notion of *Galois connection* as their foundation for judging the possible relationships. After noting that some Galois connections describe undesirable computational relationships, they proposed a special case of a Galois connection called a *reflection* as a basis for relating source and target calculi. They then showed that several compiling-related translations could be viewed as reflections between various source and target calculi.

However, a close examination of the definitions of Galois connection and reflection reveals what we feel are some overly strong computational properties. A reflection elegantly describes the situation where there is a kernel of the source language that is isomorphic to the target language. Yet we believe that the reflection criteria is so strong that it often excludes the usual situation in compiling where one is compiling from a higher-level to a lower-level language. In many such cases, it is simply impossible to obtain a reflection naturally.

The present work has two goals.

1. We reflect upon Sabry and Wadler’s proposal of Galois connection and reflection, and give what we feel are the strengths and weaknesses of these properties as criteria for translations used in compiling. We conclude that, in addition to the notion of reflection, there are several relations weaker than