Convergence of Program Transformers in the Metric Space of Trees

Morten Heine B. Sørensen
Department of Computer Science, University of Copenhagen (DIKU)
Universitetsparken 1, DK-2100 Copenhagen Ø, Denmark
E-mail: rambo@diku.dk

Abstract. In recent years increasing consensus has emerged that program transformers, e.g., partial evaluation and unfold/fold transformations, should terminate; a compiler should stop even if it performs fancy optimizations! A number of techniques to ensure termination of program transformers have been invented, but their correctness proofs are sometimes long and involved.

We present a framework for proving termination of program transformers, cast in the metric space of trees. We first introduce the notion of an abstract program transformer; a number of well-known program transformers can be viewed as instances of this notion. We then formalize what it means that an abstract program transformer terminates and give a general sufficient condition for an abstract program transformer to terminate. We also consider some specific techniques for satisfying the condition. As applications we show that termination of some well-known program transformers either follows directly from the specific techniques or is easy to establish using the general condition.

Our framework facilitates simple termination proofs for program transformers. Also, since our framework is independent of the language being transformed, a single correctness proof can be given in our framework for program transformers using essentially the same technique in the context of different languages. Moreover, it is easy to extend termination proofs for program transformers to accommodate changes to these transformers. Finally, the framework may prove useful for designing new termination techniques for program transformers.

1 Introduction

Numerous program transformation techniques have been studied in the areas of functional and logic languages, e.g., partial evaluation and unfold/fold transformations. Pettorossi and Proietti [30] show that many of these techniques can be viewed as consisting of three conceptual phases which may be interleaved: symbolic computation, search for regularities, and program extraction.

Given a program, the first phase constructs a possibly infinite tree in which each node is labeled with an expression; children are added to the tree by unfolding steps. The second phase employs generalization steps to ensure that one constructs a finite tree. The third phase constructs from this finite tree a new program.

The most difficult problem for most program transformers is to formulate the second phase in such a way that the transformer both performs interesting optimizations and always terminates. Solutions to this problem now exist for most transformers.

The proofs that these transformers indeed terminate—including some proofs by the author—are sometimes long, involved, and read by very few people. One reason for this
is that such a proof needs to formalize what it means that the transformer terminates, and significant parts of the proof involve abstract properties about the formalization.

In this paper we present a framework for proving termination of program transformers. We first introduce the notion of an abstract program transformer, which is a map from trees to trees expressing one step of transformation. A number of well-known program transformers can be viewed as instances of this notion. Indeed, using the notion of an abstract program transformer and associated general operations on trees, it is easy to specify and compare various transformers, as we shall see.

We then formalize what it means that an abstract program transformer terminates and give a sufficient condition for an abstract program transformer to terminate. A number of well-known transformers satisfy the condition. Indeed, using the notion of an abstract program transformer and associated general operations on trees, it is easy to specify and compare various transformers, as we shall see.

Developing the condition once and for all factors out this common part; a termination proof within our framework for a program transformer only needs to prove properties that are specific to the transformer. This yields shorter, less error-prone, and more transparent proofs, and means that proofs can easily be extended to accommodate changes in the transformer. Also, our framework isolates exactly those parts of a program transformer relevant for ensuring termination, and this makes our framework useful for designing new termination techniques for existing program transformers.

The insight that various transformers are very similar has led to the exchange of many ideas between researchers working on different transformers, especially techniques to ensure termination. Variations of one technique, used to ensure termination of positive supercompilation [35], have been adopted in partial deduction [23], conjunctive partial deduction [16], Turchin's supercompiler [41], and partial evaluation of functional-logic programs [1]. While the technique is fairly easily transported between different settings, a separate correctness proof has been given in each setting.

It would be better if one could give a single proof of correctness for this technique in a setting which abstracts away irrelevant details of the transformers. Therefore, we consider specific techniques, based on well-known transformers, for satisfying the condition in our framework. The description of these techniques is specific enough to imply termination of well-known transformers, and general enough to establish termination of different program transformers using essentially the same technique in the context of different languages. As applications we demonstrate that this is true for positive supercompilation and partial deduction (in the latter case by a brief sketch).

The set of trees forms a metric space, and our framework can be elegantly presented using such notions as convergence and continuity in this metric space. We also use a few well-known results about the metric space of trees, e.g., completeness. However, we do not mean to suggest that the merits of our approach stem from the supposed depth of any of these results; rather, the metric space of trees offers concepts and terminology useful for analyzing termination of abstract program transformers.

Section 2 introduces program transformers as maps from trees to trees. This is then formalized in the notion of an abstract program transformer in Section 3. Section 4 presents positive supercompilation as an abstract program transformer. Section 5 presents the metric space of trees, and Section 6 uses this to present our sufficient condition for termination, as well as the specific techniques to satisfy the condition. Section 7 shows that positive supercompilation terminates. It also sketches Martens and Gallagher's [26] generic algorithm for partial deduction as an abstract program transformer and sketches a proofs that it terminates.