Evolution of Partial Evaluators: Removing Inherited Limits

Torben E. Mogensen
DIKU, University of Copenhagen, Denmark

Abstract. We show the evolution of partial evaluators over the past ten years from a particular perspective: the attempt to remove limits on the structure of residual programs that are inherited from structural bounds in the original programs. It will often be the case that a language allows an unbounded number or size of a particular features, but each program (being finite) will only have a finite number or size of these features. If the residual programs cannot overcome the bounds given in the original program, that can be seen as a weakness in the partial evaluator, as it potentially limits the effectiveness of residual programs. The inherited limits are best observed through specializing a self-interpreter and examining the object programs produced by specialisation of this. We show how historical developments in partial evaluators gradually remove inherited limits, and suggest how this principle can be used as a guideline for further development.

1 Introduction

Much has happened in the field of partial evaluation in the past ten years. The evolution has taken many different paths, focusing on different problems. We will show below that many seemingly unrelated developments serve similar purposes: removal of inherited limits. More often than not, the developments have been introduced for solving immediate practical problems observed through experiments, without explicit awareness of the general underlying principle.

In this paper we first introduce the concept of inherited limits, and discuss why we think it may be a problem. We then, as a case study, define a small first-order functional language and show how successive refinements of an initially very simple specialisation method each remove an inherited limit, until our final version can be seen to have none. We then round up by discussing how to use the principle of inherited limits to suggest further studies in partial evaluation methods, and noting some important developments in partial evaluation that can not be related to removal of inherited limits.

2 Inherited Limits

It has been stated (I forget by whom) that in computer science (as well as many other areas) there are only three reasonable limits: zero, one and infinity.
This principle has often been used in language design, such that there typi-
cally is no arbitrary bounds on nesting depth, size of program, number of vari-
ables etc, unless that bound is uniformly set to one, as e.g. the nesting depth of
function definitions in C.

But while the language as a whole imposes no such bounds, each program
(being finite) will normally be limited to a finite depth, size or number of these
features. For example, all Pascal programs will have a finite nesting depth of
functions, a finite number of variables and a finite dimension of arrays, even
though the language definition imposes no bound on these features.

The fact that any single program only uses a bounded number or size of a
feature does not usually cause any problem, and there really is no natural way
to avoid it, even if it did.

It is when new programs are derived by specialisation or other transforma-
tions from a single original programs that we may encounter a problem: the
derived programs may not be able to exceed the limits imposed by the original
program. Why is this a problem? Philosophically speaking, we can see that the
derived programs can not use the full potential of the programming language:
a better result may be obtained by using one more level of nesting, one more
variable or one more dimension in an array. It is less clear to what extent this
is a practical problem, though. I will argue that if partial evaluators are to be
used for compiling by specialising interpreters, then it is indeed a problem.

When compiling by partial evaluation, each object program is a specialised
version of the interpreter. The interpreter, being a single program, will have finite
bounds on features that the language itself has no limitation for. If the residual
programs can not exceed these bounds, only a subset of the target language is
exploited. If the source language has uniform limits for these features, it is no
problem. But if not, in particular if the source language is an extended version
of the target language, this is a problem.

In the “challenging problems” collection from the ’87 PEMC workshop, (Neil
Jones 1987) suggested that a partial evaluator is “strong enough” (later the
term optimal is used) if it is possible to completely remove the interpretation
overhead of a self-interpreter by specialisation. The case of self-interpretation is
interesting, as it is easy to judge to what extent the interpretation overhead has
been removed by comparing the source and target programs, as these are in the
same language. If the target program is identical to the source program (modulo
renaming and reordering) it is safe to conclude that all interpretation overhead
has been removed.

Note that for the test above to be a test on the quality of the partial evaluator,
not the self-interpreter, we must not insist on a particular self-interpreter. We
must argue that no matter how a self-interpreter is written, we can not obtain
optimality. This is where considering inherited limits becomes a useful yardstick:
if the partial evaluator inherits a bound from the original program relating to
a feature that the language has no limit for, then we can argue that optimality
can not be achieved.

Inherited limits are only interesting for features where it costs something to