A model of costs and benefits of meta-level computation

Frank van Harmelen

SWI
University of Amsterdam
frankh@swi.psy.uva.nl

Abstract. It is well known that meta-computation can be used to guide other computations (at the object-level), and thereby reduce the costs of these computations. However, the question arises to what extent the cost of meta-computation offsets the gains made by object-level savings. In this paper we discuss a set of equations that model this trade-off between object-savings and meta-costs. The model shows that there are a number of important limitations on the usefulness of meta-computation, and we investigate the parameters that determine these limitations.

1 Introduction

One of the most often stated aims of meta-programming is search-control: a meta-program is used to guide the computation at the object-level. Often, this takes the form of a meta-program choosing among multiple applicable object-level computations. A large body of literature exists on this type of meta-programs, in areas like knowledge-representation, (logic-)programming and theorem proving.

Although many other types of meta-programs are both possible and useful, this paper will only consider meta-programs that are used to guide the computation at the object-level. This type of meta-program gives rise to a trade-off situation, in which costs should be compared with benefits. The benefit of meta-computation is that it leads to a better choice among object-computations, and therefore to savings at the object-level, since useless or expensive object-computations can be avoided (see e.g. [1] for results in the area of theorem proving). On the other hand, meta-computations themselves often have a considerable cost, and this cost might offset any savings that are obtained by that very same computation.

This trade-off (between savings made by meta-level choices and the costs of having to make these choices) has been recognised in the literature: [5], [2] and [4, chapter 7] report on experimental results on measuring the size of the meta-level overhead, and the large literature on partial evaluation tries to reduce the size of this overhead.

The goal of this paper is to investigate a theoretical model of the costs and benefits of meta-computation. After setting out the formal assumptions that underlie this work (Sect. 2), we present in Sect. 3 a quantitative model developed by [6]. In the context of this model, we postulate some reasonable properties of
meta-computations (Sect. 4), and illustrate the model with some examples (Sect. 5). In Sect. 6 we extend and generalise the basic model from Sect. 3.

2 Assumptions

We will assume that there are two independent methods for solving a particular object-level problem, and we will call these methods $x$ and $y$. We also assume that each of these methods has a certain expected cost, which we will denote by $c_x$ and $c_y$. Furthermore, we assume that $x$ and $y$ are heuristic methods, i.e. they are not guaranteed to solve the object-problem. Instead, we will assume that each method has a specific chance of solving the given object-problem, which we will write as $p_x$ and $p_y$.

The goal of the meta-computation is to choose among the two object-methods $x$ and $y$ in order to solve a given problem in such a way that the overall cost of the object-computation is minimised. Because in general $x$ and $y$ are not guaranteed to solve the problem ($p_x$ and $p_y$ might be smaller than 1), the meta-computation must not choose between either $x$ or $y$, but it must choose the ideal ordering of $x$ and $y$: first try $x$, and if it fails, then try $y$, or vice versa. We will write $c_{x;y}$ to denote the expected cost of first executing $x$ followed by $y$ if $x$ fails, and similarly for $c_{y;x}$.

The meta-computation that determines this choice will again have a certain cost, which we will write as $c_m$. Again, we will assume that this meta-computation is heuristic, i.e. it will make the correct choice of how to order $x$ and $y$ only with a certain chance, which we will write as $p_m$.

We assume that without meta-level computation, the system will try to use the two methods in a random order to solve the object-problem. The goal of the model will be to compute the savings (or losses, i.e. negative savings) that are obtained by using meta-computation to choose the ordering of $x$ and $y$ instead of making a random choice. These expected savings will be denoted by $s$.

All of this notation (plus some additional notation used in later sections) is summarised in Table 1.

3 The Savings Function

In this section, we will derive the expression for the expected savings obtained by making a correct meta-level choice concerning the optimal order in which to execute two object-level methods.

Given the assumptions about $x$, $y$, $c_x$, $c_y$, $p_x$ and $p_y$, the expected cost of executing $x$ before $y$, $c_{x;y}$ is:

$$c_{x;y} = c_x + (1 - p_x)c_y$$

1 In Sect. 6, we will show how the model can be extended to deal with an arbitrary number of methods.

2 In Sect. 6 we will show how the model can be adjusted to accommodate the more realistic assumption that the system will execute the methods in some fixed order if no meta-computation is done.