Differential Logical Relations

Ugo Dal Lago  Francesco Gavazzo  Akira Yoshimizu

Abstract

We introduce a new form of logical relation which, in the spirit of metric relations, allows to assign each pair of programs a quantity measuring their distance, rather than a boolean value standing for their being equivalent. The novelty of differential logical relations consists in measuring the distance between terms not (necessarily) by a numerical value, but by a mathematical object which somehow reflects the interactive complexity, i.e. the type, of the compared terms. We exemplify this concept in the simply-typed lambda-calculus, and show a form of soundness theorem. We also see how ordinary logical relations and metric relations can be seen as instances of differential logical relations. Finally, we show that differential logical relations can be organised in a cartesian closed category, contrarily to metric relations, which are well-known not to have such a structure, but only that of a monoidal closed category.

1 Introduction

Modern software systems tend to be heterogeneous and complex, and this is reflected in the analysis methodologies we use to tame their complexity. Indeed, in many cases the only way to go is to make use of compositional kinds of analysis, in which parts of a large system can be analysed in isolation, without having to care about the rest of the system, the environment. As an example, one could consider a component $A$ and replace it with another, e.g. more efficient component $B$ without looking at the context $C$ in which $A$ and $B$ are supposed to operate, see Figure 1. Of course, for this program transformation to be safe, $A$ should be equivalent to $B$ or, at least, $B$ should be a refinement of $A$.

Program equivalences and refinements, indeed, are the cruxes of program semantics, and have been investigated in many different programming paradigms. When programs have an interactive behaviour, like in concurrent or higher-order languages, even defining a notion of program equivalence is not trivial, while coming out with handy methodologies for proving concrete programs to be equivalent can be quite challenging, and has been one of the major research topics in programming language theory, stimulating the development of techniques like logical relations [14, 12], applicative bisimilarity [1], and to some extent denotational semantics [16, 17] itself.

Coming back to our example, may we say anything about the case in which $A$ and $B$ are not equivalent, although behaving very similarly? Is there anything classic program semantics can say about this situation? Actually, the answer is negative: the program transformation turning such an $A$ into $B$ cannot be justified, simply because there is no guarantee about what the possible negative effects that turning $A$ into $B$ could have on the overall system formed by $C$ and $A$. There are, however, many cases in which program transformations like the one we just described are indeed of interest, and thus desirable. Many examples can be, for instance, drawn from the field of approximate computing [13], in which equivalence-breaking program transformations are considered as beneficial provided the overall behaviour of the program is not affected too much by the transformation, while its intensional behaviour, e.g. its performance, is significantly improved.
One partial solution to the problem above consists in considering program metrics rather than program equivalences. This way, any pair of programs are dubbed being at a certain numerical distance rather than being merely equivalent (or not). This, for example, can be useful in the context of differential privacy [15, 4, 22] and has also been studied in the realms of domain theory [7, 3, 8, 10, 2] (see also [18] for an introduction to the subject) and coinduction [20, 19, 9, 5]. The common denominator among all these approaches is that on one hand, the notion of a congruence, crucial for compositional reasoning, is replaced by the one of a Lipschitz-continuous map: any context should not amplify (too much) the distance between any pair of terms, when it is fed with either the former or the latter:

$$\delta(C[M], C[N]) \leq c \cdot \delta(M, N).$$

This enforces compositionality, and naturally leads to consider metric spaces and Lipschitz functions as the underlying category. As is well-known, this is not a cartesian closed category, and thus does not form a model of typed $\lambda$-calculi, unless one adopts linear type systems, or type systems in which the number of uses of each variable is kept track of, like FUUZ [15]. This somehow limits the compositionality of the metric approach [7, 11].

There are however program transformations which are intrinsically unjustifiable in the metric approach. Consider the following two programs of type REAL $\rightarrow$ REAL

- $M_{\text{SIN}} := \lambda x. \sin(x)$
- $M_{\text{ID}} := \lambda x. x$.

The two terms compute two very different functions on the real numbers, namely the sine trigonometric function and the identity on $\mathbb{R}$, respectively. The distance $|\sin x - x|$ is unbounded when $x$ ranges over $\mathbb{R}$. As a consequence, the numerical distance between $M_{\text{SIN}}$ and $M_{\text{ID}}$, however defined, is infinite, and the program transformation turning $M_{\text{SIN}}$ into $M_{\text{ID}}$ cannot be justified this way, for very good reasons. As highlighted by Westbrook and Chaudhuri [21], this is not the end of the story, at least if the environment in which $M_{\text{SIN}}$ and $M_{\text{ID}}$ operate feed either of them only with real numbers close to 0, then $M_{\text{SIN}}$ can be substituted with $M_{\text{ID}}$ without affecting too much the overall behaviour of the system.

The key insight by Westbrook and Chaudhuri is that justifying program transformations like the one above requires taking the difference $\delta(M_{\text{SIN}}, M_{\text{ID}})$ between $M_{\text{SIN}}$ and $M_{\text{ID}}$ not merely as a number, but as a more structured object. What they suggest is to take $\delta(M_{\text{SIN}}, M_{\text{ID}})$ as yet another program, which however describes the difference between $M_{\text{SIN}}$ and $M_{\text{ID}}$:

$$\delta(M_{\text{SIN}}, M_{\text{ID}}) := \lambda x. \lambda \varepsilon. |\sin x - x| + \varepsilon.$$

This reflects the fact that the distance between $M_{\text{SIN}}$ and $M_{\text{ID}}$, namely the discrepancy between their output, does not only depend on the discrepancy on the input, namely on $\varepsilon$, but also on the input itself, namely on $x$. It both $x$ and $\varepsilon$ are close to 0, $\delta(M_{\text{SIN}}, M_{\text{ID}})$ is itself close to 0.

In this talk, we develop Westbrook and Chaudhuri’s ideas, and turn them into a framework of differential logical relations. We will do all this in a simply-typed $\lambda$-calculus with real numbers as the only base type. Starting from such a minimal calculus has at least two advantages: on the one hand one can talk about meaningful examples like the one above, and on the other hand the induced metatheory is simple enough to highlight the key concepts.

References


