Previous Academic Work: MSc Thesis

Title: *Termination Analysis of λ-calculus and a subset of core ML.*

Abstract:

Termination analysis is a very important component of software verification: it is futile trying to prove a property on a program result if the program does not terminate and therefore never returns the result. Turing showed that termination is an undecidable property. However in [2], Lee, Jones and Ben-Amram introduced "size-change termination", a decidable property strictly stronger than termination. They proposed a method called the "size-change principle" to analyze it.
Size-change analysis relies on a finite approximation of the program computational behavior. A call semantics is defined such that the presence of infinite call sequences characterizes non-termination. Since the approximated computational space is finite, infinite call sequences must contain loops. Deciding the size-change property then amounts to analyze loops of the program through the use of "size-change graphs" describing program calls.

We first explain the size-change principle in the first-order case ([2]) and its adaptation to the untyped λ-calculus ([5]). My implementation provides some improvements over the original method: it avoids variable renaming and generates a more accurate approximation.
Finally we extend the size-change principle to a subset of ML featuring ground type values, higher-order type values and recursively defined functions. Compared to other works, this is the first time that the size-change principle is applied to a higher-order functional language.
In a first attempt, the ML program is converted into a λ-calculus expression, by means of Church numerals and the Y combinator, and analyzed using the algorithm of [5]. Implementing numbers with church numerals has two important drawbacks: the size of the converted program increases proportionally to the integer values used in its definition. Secondly, since the decrease in integer values is not properly reflected by church numerals, most recursively defined functions operating on numbers are not recognized as terminating!
In the second approach, being inspired by [5] we redefine from scratch an algorithm for the core ML case which handles natively `if-then-else` and `let rec` structures with no conversion. This algorithm produces the same result as [5] for higher-order values but can also analyze the size of ground type values. This enhances the scope of the termination analyzer to some recursively defined function operating on numbers.

Brief description of the proposed topic of research

Title: *Verification of program properties using a fully abstracted game semantics for a Caml like language*

Description

I propose to work in the field of computer-assisted formal verification and program analysis. I would like to contribute to the work already achieved in the Game Semantics research group in Oxford.

Prior research by the Game Semantics group in Oxford shows that Game Semantics is a powerful tool for giving semantics to computer programs ([1]).

In [6] and [3] full abstractions of PCF programming language has been established using Game Semantics concepts.

The case of functional programming is of particular interest for me. Being inspired by the work done by Luke Ong and Samson Abramsky in the Oxford Computer Laboratory, I would like to explore the possibilities of extension of their work to the case of a Caml like programming language.

My work will consist in investigating applications of Algorithmic Game Semantics to the case of a ML programming language like Caml. The aim will be to find a full abstraction of a ML programming language such as Objective Caml ([4]) where functionnal and imperative programming paradigms are present. This language is a call-by-value language implementing recursive procedure, conditionals, sequential composition, variable assignment, while/for loops, and object oriented programming paradigms such as classes, objects and methods.

The full abstraction will then be used as a basis for different verification tools. These tools should be able to read Objective Caml code as an input and to do verification on that program. The following properties should be supported by the verfication tools:

- proof of termination (when it is possible)
- equivalence of two ocaml programs
- other properties to be expressed in a temporal logic which need to be defined.

The full abstraction should permit to generate structures which approximate the semantics of the program. These structures should be adapted to the properties which need to be proved.

One possible approach to this project can be to find an interesting extension of the size-change principle introduced by Neil Jones in [2]. The size-change principle has been used by Neil Jones in [5] to develop an algortihm for the termination analysis of the untyped lambda-calculus (subject of my MSc project last years).

References

[1] S. Abramsky. Algorithmic game semantics: a tutorial introduction. 2001.

[2] N.D. Jones C.S. Lee and A. M. Ben-Amram. The size-change principle for program termination. 2001.

[3] J. M. E. Hyland and C.-H. L. Ong. On full abstraction for pcf. 2000.

[4] INRIA. Objective caml programming language, 2003. `http://caml.inria.fr/`.

[5] N.D. Jones and N. Bohr. Termination of the untyped lambda calculus. 2004.

[6] R. Jagadeesan S. Abramsky and P. Malacaria. Full abstraction for pcf. 2000.