Types for combining polynomial time programs

proposé par Patrick Baillot (LIP, ENS Lyon)

Coordonnées :
patrick.baillot@ens-lyon.fr
http://perso.ens-lyon.fr/patrick.baillot/

Laboratoire d’accueil : LIP, ENS Lyon, Équipe Plume (Programs and proofs),

City : Lyon

MPRI courses related to the proposal : 2.1, 2.2, 2.4.

Context :
Feasible computation is in first approximation classically associated to polynomial time (Ptime) complexity. But how can one combine Ptime programs as building blocks in such a way to stay in the Ptime class? We know for instance that the simple composition of two Ptime programs is Ptime... But this is not true in general for more elaborate constructions such as iteration or recursion.

Implicit computational complexity (ICC) aims at designing languages whose programs stay within a given class, for instance Ptime, e.g. [BC92, Hof03, BBR18]. However this leads to distinct languages, which rely on different principles and turn out to be in general incomparable. Although some automatic complexity analysis have been inspired by some of these works, see e.g. [PH20], ICC has not provided general methods to reason about arbitrary Ptime programs.

Objective :
In this internship we propose to take a fresh look at the question of Ptime program construction by adopting a semantics-inspired approach. We aim at designing program construction rules which preserve the property of being Ptime and which take as assumptions only informations about output-size and time bounds of programs, and not about their code.

The method we suggest for that is to consider a generalist language and to define some classes of programs, called types, which contain all Ptime programs satisfying a certain property about output-size (not only those coming from ICC restricted programming disciplines). Then we will define some rules for constructing new Ptime programs from programs of certain types.

For this goal we propose to start with a first-order functional language and to take inspiration from Implicit computational complexity languages, e.g. to start with, Bellantoni-Cook’s safe recursion [BC92] and Hofmann’s non-size-increasing programs [Hof03]. We also expect that some ideas from linear logic [Gir98, BBR18] will be useful on this path.

Expected ability of the student : Some simple notions of computational complexity. Some basic knowledge of a functional language, λ-calculus or term rewriting, and of type systems. Knowledge of linear logic would be a plus but is not mandatory.

Références


