Types for parallel complexity analysis

Encadrants : Patrick Baillot (patrick.baillot@ens-lyon.fr) et Alexis Ghyselen (alexis.ghyselen@ens-lyon.fr)

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

City : Lyon

Context :
Some type systems have been devised to analyse the time complexity of functional programs. With such a system, given a type derivation for a program $M$, one can extract from it an upper bound on the (sequential) execution time of $M$ on any input.

A more recent challenge is that of complexity analysis for parallel programming languages. These languages are equipped with features to spawn several computation threads, in order to take advantage of multiprocessor machines. In this setting we are interested in the parallel complexity of the program, that is to say its execution time when independent threads are executed simultaneously, as much as possible. An example of type system to analyse the parallel complexity of parallel programs is given in [HS15].

Objective :
The $\pi$-calculus [SW03] is a formal calculus to study parallel and concurrent computation, just as $\lambda$-calculus allows to study functional computation. It represents processes communicating by means of channels. A type system was recently proposed in [BG21, BG20] to analyse the parallel complexity of $\pi$-calculus programs. The type system uses some size annotations to reason about integer values, and some time annotations to reason about communications.

In this internship we propose to investigate how the type system of [BG21] can be used to analyse the complexity of parallel functional programs. The work can proceed as follows :

— Consider a simple parallel functional language, such as the one studied in [HS15], and define a simulation of this language in $\pi$-calculus. Such a simulation can take inspiration from the simulations of $\lambda$-calculus in $\pi$-calculus [SW03].

— Define an operational semantics and a suitable notion of parallel complexity for this parallel language, by using the simulation in $\pi$-calculus and the parallel operational semantics in [BG21].

— Finally define a type system for the parallel language allowing to extract a bound on the parallel complexity of the program, again by taking advantage of the simulation in $\pi$-calculus and the type system for $\pi$-calculus.

— Then one can study some examples of algorithms written in the parallel language, examine whether they are typable and in this case how good is the complexity bound obtained by the type system.

Expected ability of the student : It is expected to have some knowledge about $\lambda$-calculus or functional programming languages as well as about type systems. Some basic notions of $\pi$-calculus or process calculi would also be welcome but are not compulsory. Some background in algorithmics can also be useful but is not necessary either.

Références

