Analyse d'Efficacité de Services en π-calcul (Proposition Stage de L3 ENS de Lyon 2019)

Romain Demangeon
LIP6, Sorbonne Université

Résumé

Ce stage a pour but l'étude d'une analyse d'efficacité pour des réseaux de services décrits dans le formalisme du π-calcul. L'analyse en question a été décrite dans une publication récente [2] et un premier prototype [6] a été produit. A l'aide d'un système de types et de contraintes syntaxiques, l'analyse décide si un processus du π-calcul modélise un réseau de service "efficace" (génération de messages polynomiale en la taille de la requête initiale).

L'objectif du stage est d'étudier cette analyse et son implémentation et de proposer des améliorations d'efficacité (optimiser les algorithmes qui parcourentt et manipulent des types et des termes du π-calcul) et d'expressivité (pouvoir reconnaître un plus grand nombre de processus du π-calcul efficaces) pour le prototype et l'analyse elle-même.

Cadre du Stage

La motivation de [2] est la détection d'applications distribuées *efficaces*, c'est-à-dire, d'applications qui, au cours de leur exécution, génèrent un nombre de messages maitrisé (par exemple, borné par une fonction polynomiale d'une quantité initiale, comme la taille d'une requête). Cette analyse applique des méthodes issues de la *complexité implicite* à la théorie des *algèbres de processus*.

La complexité implicite [3] est un domaine de recherche qui vise à définir des cadres de programmation (systèmes de types, syntaxes contraintes, ...) dans lesquels chaque programme vérifie, a priori, une borne de complexité. Les algèbres de processus (CSS, π-calcul [5], ...) sont des langages formels qui décrivent les programmes distribués sous formes de termes (appelés *processus*). Des propriétés désirables pour les applications distribuées peuvent être traduites dans les algèbres de processus et vérifiées à l'aide de systèmes de types.

Dans [2], une analyse classique de complexité implicite [1] est adaptée dans le formalisme du π-calcul [5], permettant la détection de processus au comportement "polynomial". Un prototype naïf d'implémentation de cette analyse a été réalisée en OCaml [6].

Objectifs du Stage

Une fois que le stagiaire sera familiarisé avec le cadre du stage, plusieurs directions sont envisageables:

1. **améliorer** (ou réécrire, totalement ou partiellement) le prototype en cherchant de meilleures structures de données et de meilleurs algorithmes pour manipuler des termes et des types du π-calcul, réaliser leur unification, et propager les mises à jour des types au sein de l'environnement,
2. **étendre** l'analyse (et, in fine, le prototype) afin d'améliorer son expressivité en étudiant la prise en charge de structures de données plus complexes ou en adaptant une autre technique issue de la complexité implicite (par exemple [4]),

Domaines du Stage

Le stage a de meilleure chance de succès si:

- le stagiaire aime lire du (et écrire en) OCaml,

- le stagiaire aime les calculs formels (λ-calcul, π-calcul, ...),
- le stagiaire aime les règles d'inférence (particulièrement, les règles de typage),
- le stagiaire aime à la fois la programmation et l'algorithmique.

Le stage a des chances de mobiliser des connaissances acquises en:

I3101 : structures de données et efficacité d'algorithme de parcours, complexité,

I3104 : programmation en OCaml, sémantique et typage.

Cadre logistique

Le stagiaire aura un **bureau** au LIP6 (https://www.lip6.fr/) à proximité de celui de l'encadrant pendant la durée du stage.
Une **gratification** est peu probable (mais possible, à discuter).

References

[1] Stephen Bellantoni and Stephen A. Cook. A new recursion-theoretic characterization of the polytime functions. In *STOC*, pages 283–293. ACM, 1992.

[2] Romain Demangeon and Nobuko Yoshida. Causal computational complexity of distributed processes. In Anuj Dawar and Erich Grädel, editors, *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, pages 344–353. ACM, 2018.

[3] Ugo Dal Lago. A short introduction to implicit computational complexity. In Nick Bezhanishvili and Valentin Goranko, editors, *Lectures on Logic and Computation - ESSLLI 2010 Copenhagen, Denmark, August 2010, ESSLLI 2011, Ljubljana, Slovenia, August 2011, Selected Lecture Notes*, volume 7388 of *Lecture Notes in Computer Science*, pages 89–109. Springer, 2011.

[4] Daniel Leivant and Jean-Yves Marion. Lambda calculus characterizations of poly-time. *Fundam. Inform.*, 19(1/2):167–184, 1993.

[5] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes. Technical report, Laboratory for Foundations of Computer Science, Department of Computer Science, University of Edinburgh, 1989.

[6] Prototype. `https://www-apr.lip6.fr/~demangeon/Recherche/protolics.ml`.