Sujet de stage: programmation vérifiée de fonctions de flot
Encadrants: Jean-Pierre Talpin <talpin@irisa.fr>, Yuanro Zhang
Equipe-projet INRIA TEA, Rennes en collaboration avec UC San Diego (équipe associé Composite)
mots-clé: programmation vérifiée, modèles flot-de-données, logique temporelle, vérification, preuve

Des avancées majeures en théorie de la preuve, théorie des types et vérification formelle (notamment SAT-SMT) ont conduit à revisiter le paradigme "preuve = programme" en introduisant la notion de typage par raffinement. Le raffinement <v : t | p> définit le domaine t de la valeur v mais aussi les propriétés p de cette valeur dans ce domaine de définition. Par exemple, le type <n: int | n mod 2 = 0> dénote les entiers pairs.

Ce paradigme est mis en œuvre au moyen de langages de programmation algorithmique comme Liquid Haskell et F*, mais aussi impératifs comme Frama-C, et sont encore présents depuis longtemps dans la méthode B. Leur utilisation permet de certifier la correction d'un programme vis à vis d'exigences exprimées au moyen de raffinement en utilisant des outils de vérification (Z3) ou de preuve (Coq).

Dans ce stage, nous nous intéressons au langage de programmation vérifiée F* (Inria-Microsoft) pour y définir un modèle de concurrence supportant la génération de code réactif (exécution préhemptive, ordonnancement dynamique) ou temp-réel (non-préhemptive, ordonancement statique) en utilisant le générateur de code Kremlin de F* et les outils de méta-programmation Meta-F*.

L'objet du stage sera d'élaborer les briques de base du modèle visé, dans l'optique de leur poursuite en thèse, en définissant la logique, les types, et les opérateurs de base d'un langage "réactif", de manière à permettre la vérification statique de leur bon typage ainsi que de leur propriétés de logique temporelle.

REFERENCES

F*: A Higher-Order Effectful Language Designed for Program Verification. https://www.fstar-lang.org
LTL Linear Temporal Logic Symbolic Model Checking. Kristin Rozier. Computer Science Review, Volume 5, Issue 2, May 2011.
A collection of tutorials, guidelines, examples, patterns and half-baked ideas on functional reactive programming (FRP). Heinrich Apfelmus. https://github.com/HeinrichApfelmus/frp-guides