CALODS - Concise Algorithm Language for Observing Distributed Systems

Chaboche - Marais

1er juin 2020

Introduction

Notre objectif

Projet de recherche avec M. Sangnier et M. Laroussinie, *CALODS* :

- définir un langage pour décrire les algorithmes distribués
- représenter graphiquement toutes les exécutions d'un programme
- créer un système de vérification de propriétés
- permettre d'ordonner les processus afin de contraindre l'exécution des programmes

Introduction

Notre projet

Que fait *CALODS* ?

- production d'un fichier Graphviz représentant le diagramme d'états
- production d'un fichier *Prism* correspondant à la traduction d'un programme *CALODS*
- production d'un automate de Büchi non-déterministe pour ordonner l'exécution des programmes

Architecture

Composition du projet :

- compiler :
  - But : générer une structure abstraite représentant le programme
  - Dépendance : menhir
- prism :
  - But : traduire la structure abstraite en un programme *Prism*
  - Dépendance : Prism Model Checker
- graph :
  - But : représenter le programme sous forme de diagramme d'états
  - Dépendance : Graphviz

Conception

Éléments de conception du projet :

- **Modularité** : Compiler est une interface qui permet d'abstraire le code notre compilateur
- **Inter-dépendance** : Prism et Graph dépendent de Compiler pour la structure de données $\Rightarrow$ pas de dépendance entre Prism et Graph
- **Avantage** : La structure en modules permet de changer le contenu ou d'ajouter de nouveaux éléments facilement
- **Difficulté** : comprendre *Prism* et générer les graphes en utilisant les bonnes structures de données

Gestion de projet

Répartition des rôles et organisation :

- **Rôles** :
  - Compiler : ensemble
  - Prism : Valentin, rejoint par Étienne (module plus conséquent)
  - Graph : Étienne
- **Organisation** : réunions bi-mensuelles avec M. Sangnier et M. Laroussinie et échanges par mails
- **Git** : utilisation de branches pour éviter les conflits et d'issues pour les bugs
- **Tests** : présentation aux enseignants et exécution sur des exemples avec des résultats connus

Module de compilation

```
(* Implement the compiler *)
module Calods  : Compiler with type ast = Ast.program = struct
  type ast = Ast.t

  let parse_filename file =
    let input = open_in file in
    let lexbuf = Lexing.from_channel input in
    let (header, processes) =
      try
        Parser.program Lexer.token lexbuf
      with e -> Error.readable e lexbuf
    in
    let main =
      try
        Parser.main Lexer.main lexbuf
      with e -> Error.readable e lexbuf
    in
    Ast.Program (header, processes, main)

  let print_ast ast =
    CalodsPrettyPrinter.print_program
      Format.std_formatter ast

  let check ast =
    Checker.type_check ast
end
```

COVID-19

Les impacts du Covid-19 sur notre travail :

- absence de réunion physique $\Rightarrow$ communication par mails, plus complexe pour expliquer les avancements
- résultat semblable mais avec plus de possibilité d'éprouver le projet $\Rightarrow$ production plus robuste

Conclusion

Ce que l'on a tiré du projet :

- **Apprentissage** :
  - compréhension des algorithmes distribués et du matériel de preuves (model checkers, schedulers, etc)
  - maîtrise de *OCaml* et mise en pratique du cours de PFA
  - manipulation des arbres de syntaxe abstraite pour la transformation de programmes.
  - utilisation poussée de git
- **Continuité** : vérification de propriétés sur les graphes pour les comparer avec celles de *Prism*
- **Rétrospective** :
  - utilisation plus poussée des modules et interfaces de *OCaml*.
  - amélioration de la communication au fil des mois