Projet n° 01 : Construction de Glushkov

Victor MIKHAILOVICH GLUSHKOV, 1923-1982.

L'objectif de ce petit projet est d'implémenter un petit moteur d'expressions régulières en utilisant la construction de Glushkov (que l'on appelle aussi algorithme de Berry-Sethi) pour construire un automate à partir d'une expression régulière.

Consignes

Ce petit projet est à réaliser par groupes de *deux* ou *trois* pour le vendredi 22 décembre, au sein d'un même groupe de TP. Le projet est un projet de groupe et donc tous les élèves du groupe doivent bien en comprendre toutes les parties. Cependant, les élèves les plus à l'aise peuvent proposer des prolongements et des modules plus délicats, mais à la condition de bien en expliquer le principe à tous les autres membres du groupe. Il est possible de se répartir certaines parties entre membres du groupe, par exemple chaque membre peut être responsable de l'implémentation de diverses fonctions, *qui seront ensuite relues et vérifiées par les autres membres*, ce qui nécessite alors de bien spécifier le comportement attendu de ces fonctions. Mais le fonctionnement général complet du programme doit être impérativement maîtrisé par tous.

Le choix du langage OCAML est imposé.

Le code source du projet devra être clairement structuré, raisonnablement commenté et respecter les bonnes pratiques de programmation.

Vous pouvez utiliser des outils de travail collaboratif de votre choix, des logiciels de gestion de version (`git`, `mercurial`, `fossil`) et des logiciels de partage, mais ceci n'est nullement obligatoire, surtout si vous ne maîtrisez pas déjà ces outils. Vous pouvez simplement désigner un membre qui sera responsable de mettre en commun les fonctions écrites dans un même programme principal et qui s'assurera de la cohérence globale. Attention tout de même, responsable ne veut pas dire seul à la manœuvre : tous les membres du groupes doivent parfaitement comprendre l'ensemble du programme.

Vous devez rendre un unique fichier `projet01_<nom1>_<nom2>_<nom3>.zip` où évidemment les `<nom?>` sont à remplacer par les noms de famille des membres du groupes. Une fois

décompressé cette archive doit contenir un ou plusieurs fichiers sources, ainsi qu'éventuellement des fichiers d'en-tête `.h` ou d'interface `.mli` (ce n'est pas obligatoire, en particulier si vous ne savez pas ce que c'est !). Un petit fichier `README.txt` doit expliquer très clairement comment compiler votre petit projet. Vous pouvez également y ajouter des commentaires qui vous sembleraient utiles.

Cahier des charges

Une fois compilé, par exemple en un exécutable `mygrep`, votre programme doit se comporter comme une version rudimentaire (mais utilisable) de `grep`, exactement comme cela a été réalisé au TP n° 13. Il est nécessaire de bien relire la première partie de ce TP et il est conseillé de bien le retravailler avant de réaliser ce projet. On pourra faire les mêmes hypothèses sur les expressions régulières limitées acceptées.

La seule différence avec le TP n° 13 est qu'il vous est demandé d'utiliser la construction de Glushkov plutôt que la construction de Thompson et de réaliser ce projet en OCaml. Un des avantages de cette constructions c'est qu'il n'y a pas d'ϵ-transitions dans l'automate non déterministe obtenu.

On pourra déterminiser l'automate puis utiliser une fonction simple pour déterminer si un mot est reconnu ou bien exécuter directement l'automate non déterministe en utilisant la fonction $\widehat{\delta}$ qui travail sur un ensemble d'états possibles.

Squelettes

Un squelette `squelette_minimal.ml` permettant de réaliser les fonctions d'entrées-sorties vous est proposé. Il propose le même comportement que le programme réalisé dans le TP n° 13. Vous êtes libres de l'utiliser.

Dans ce projet il sera donc nécessaire de définir ses propres types et ses propres structures de données. Cependant, si vous lancer sans aide dans ce projet vous semble compliqué ou vous prendrait trop de temps, un exemple de squelette avec les fonctions utilisées dans le corrigé vous est proposé. Les types et les signatures de ces fonctions sont disponibles dans le fichier `squelette_proposition.mli` et certaines de ces fonctions ou types sont implémentés dans le fichier `squelette_proposition.ml`. Vous êtes entièrement libres d'utiliser ou de vous inspirer de ce squelette et ceci ne sera nullement pénalisé. Pour les élèves moins autonomes, ceci est vivement conseillé.

Extensions

Si vous avez encore du temps ou si vous avez déjà réalisé ce projet l'année dernière et que vous ne souhaitez pas le refaire, voici deux extensions proposées :

- Implémenter l'opération réciproque, c'est-à-dire le passage d'un automate à une expression régulière en utilisant l'algorithme de McNaughton et Yamada ;
- Implémenter l'opération réciproque, c'est-à-dire le passage d'un automate à une expression régulière en utilisant l'algorithme de l'élimination des états, dit aussi algorithme de McCluskey et Brzozowski.

Dans les deux cas, on pourra implémenter une fonction de simplification, plus ou moins élaborée, des expressions régulières pour limiter leur croissance.