Rapport de projet - Model-checker pour Lustre
Systèmes synchrones - M2 MPRI

Clément Devatine

January 9, 2022

1 Architecture du projet

Le projet utilise `dune` pour être compilé. Il est donc nécessaire que celui-ci soit installé, via `opam` par exemple (`opam install dune`). Cela a nécessité quelques légères modifications dans le code source de la bibliothèque `Aez` et dans le code accompagnant le lexeur et parseur.
Le dossier est structuré comme suit :

- `bin` contient le fichier pour l'exécutable et deux fichiers exemples. L'exécutable reprend le code qui était donné initialement.
- `lib` contient tout le code auxiliaire. J'ai pour habitude de développer tout comme une bibliothèque, car cela facilite la maintenance et la lecture.
  - `aez` contient le code de la bibliothèque `Aez`
  - `backend_aez` permet d'interfacer la représentation intermédiaire avec celle de `Aez`, et de faire tourner le solveur sur l'algorithme de k-induction avec cette bibliothèque.
  - `common` contient des bibliothèques utilitaires, notamment un petit dictionnaire sur les chaînes de caractères et une extension des entiers 64 bits (utile pour la conversion des flottants en nombres relatifs).
  - `frontend` contient le code fournit initialement pour le projet, c'est-à-dire le lexeur, le parseur et le vérificateur du typage.
  - `ir` contient une description de la représentation intermédiaire et tout ce qui est nécessaire pour transformer la représentation du *frontend* vers la représentation intermédiaire.
  - `lmoch` regroupe toutes les sous-bibliothèques en une seule, dénommée `lmoch`.
- `test` est juste le dossier de test par défaut requis par `dune`. Il ne contient rien puisque tout les essais ont été faits à la main (et n'ont pu être automatisé car le projet est un échec).

Pour compiler le projet, dans le dossier principal, il faut taper la commande `dune build`. Pour exécuter le programme principal, il faut taper `dune exec lmoch <paramètres du programme>`.

2 Méthode employée

Après la partie qui décode et parse le fichier Lustre, le programme fonctionne en deux temps.

Il transforme dans un premier temps la représentation en sortie du parseur en une représentation intermédiaire plus proche de celle utilisée par `Aez`. En particulier, de nouvelles équations et variables sont introduites pour permettre de raisonner sur des booléens dans des termes, car `Aez` ne le permet pas. En particulier, dès qu'un terme t booléen intervient, celui-ci est remplacée par une variable auxiliaire booléene v, et l'équation $v \iff t$ est ajoutée.

Le deuxième temps fait passer la représentation intermédiaire à celle de `Aez`. Cela nécessite en particulier de s'occuper des expressions contenant des tuples, car ceux-ci ne sembles pas acceptés par `Aez` (il existe la possibilité d'avoir des types de données énumérés avec des constructeurs, mais je n'ai pas compris avec la documentation comment faire, et je n'ai pas trouvé d'exemples à ce sujet). Pour délier les tuples, toutes les expressions sont représentées par des arbres selon leur type, de sorte que les nœuds représentent les tuples et les feuilles les types de base (appelés `Atom` dans le code). Chaque variable en Lustre peut ainsi correspondre à tout un arbre de nouvelles variables atomiques. Lors de l'instantiation de *nodes* Lustre, le programme rebranche ensuite toutes les variables de l'arbre entre elle en créant les bonnes égalités. Des transformations sont aussi nécessaires pour faire correspondre les tuples de part et d'autre d'une équation : chaque côté d'une formule est donc calculé, puis les feuilles sont mises en relation avec les autres. Une nouvelle formule est donc ajoutée pour chaque couple de feuilles correspondantes. Il en va de même avec d'autres termes, notamment avec l'opérateur conditionnel.

3 Difficulté rencontrées

Malheureusement, je n'ai pas pu tester ce que j'ai réalisé, car je me suis heurté à des levées d'assertions dans la bibliothèque `Aez`, et je n'ai pu trouver leur origine. J'ai donc essayé de développer un backend pour Z3, ce qui présente aussi l'avantage de simplifier l'implémentation pour les tuples. Cependant, la documentation est tellement imprécise et diffuse que je n'ai pu y parvenir. Puisque le code de ce backend est désordonné et non-fonctionnel, je ne l'ai pas inclus dans le code.