```
Mini-projet 2 : Synthèse d'invariants en SMT-LIB
```

Cours de Logique, L3 Informatique, Université de Paris, 2022–2023

Objectif du mini-projet

Dans ce projet, vous allez automatiser la synthèse d'invariants de boucle pour un très petit langage de programmation, décrit ci-dessous. Comme échauffement, vous écrirez d'abord un fichier SMT-LIB à la main. Il est recommandé de lire en détail la section 16.4 des notes de cours avant de réaliser ce mini-projet.

Documentation

Une attention particulière sera portée sur la documentation de votre code : la pertinence et la clarté de cette dernière comptera pour un pourcentage important de votre note finale. Vous devrez donc soigner les explications qui accompagneront votre code et ce sur deux volets principaux :

1. Le fichier RENDU contient des questions précises sur votre implémentation. Vous êtes requis d'y répondre en complétant le même fichier, qui devra faire partie de votre rendu final. Vos réponses doivent être claires, précises et préférablement succinctes. À la lecture de vos réponses, une compréhension globale, sans ambiguïté, de votre implémentation devra facilement se dégager.

   Un mini-projet sans fichier RENDU rempli ne recevra pas de note.

2. Vous devez impérativement commenter votre code dans le but de complémenter les explications fournies lors du remplissage du fichier RENDU (volet 1 ci-dessus). Dans le même esprit, vos commentaires doivent être clairs et précis, ils doivent donner une compréhension plus fine des détails de votre implémentation.

   Un code non commenté entraînera automatiquement une note finale lourdement pénalisée.

Consignes

** Exercice 1 **

Écrire un fichier SMT-LIB `invariants.smt2` qui, lorsqu'il est donné à Z3, trouve un invariant de boucle pour le programme Java suivant :

```
int i = 0;
int v = 0;
while (i < 3) {
    v = v + 3;
    i = i + 1;
}
assert v == 9;
```

** Exercice 2 **

Nous définissons un langage de programmation « WA » (While-Assert), qui modélise des programmes Java de la forme générale suivante :

```
int x1 = a1;
// (...)
int xk = ak;
while (s) {
    x1 = b1;
    // (...)
    xk = bk;
}
assert (t);
```

Plus formellement, un *programme WA* est défini comme un uplet `(k, a1, ..., ak, b1, ..., bk, s, t)`, où `k` est un entier représentant le nombre de variables utilisées par le programme, `a1`, ..., `ak` et `b1`, ..., `bk` sont des *termes*, et `s` et `t` sont des *tests*. Les variables utilisées dans les termes et les tests seront `x1`, ..., `xk`, toujours de type entier. Ici, un *terme* de WA est construit à partir de variables et de constantes entières en appliquant les opérations arithmétiques `+` et `*`, et un *test* est défini comme une formule atomique, qui compare deux termes par l'égalité ou par l'ordre. Par exemple, le programme Java donné dans l'exercice 1 correspond, après l'application du renommage de `i` en `x1` et de `v` en `x2`, au programme WA suivant : `(2, 0, 0, x1 + 1, x2 + 3, x1 < 3, x2 = 9)`.

Écrire une fonction OCaml `smtlib_of_wa : program -> string` qui prend en argument un programme WA et qui renvoie un programme SMT-LIB, qui, lorsqu'il est donné à Z3, vérifiera l'existence d'un invariant de boucle pour le programme WA donné en argument.

Le fichier `invariants.ml` fourni avec ce mini-projet vous suggère une définition possible des types en OCaml, ainsi qu'une subdivision de l'exercice en cinq questions. Vous devez donc compléter le code du fichier `invariants.ml` :

- la fonction `str_of_term : term -> string`
- la fonction `str_of_test : test -> string`
- la fonction `str_condition : term list -> string`
- la fonction `str_assert : int -> string -> string`
- la fonction auxiliaire `loop_condition : program -> string`
- la fonction auxiliaire `assertion_condition : program -> string`

Les types et les commentaires dans `invariants.ml` sont indicatifs. D'autres choix peuvent être pertinents. Vous pouvez ajouter d'autres fonctions auxiliaires.

Tester son mini-projet

Le fichier `invariants.ml` contient le programme de l'exercice 1 dans `p1`. Proposez un autre programme WA `p2` pour tester votre solution pour l'exercice 2. Utilisez Z3 pour vérifier que votre code SMT-LIB est bien correct (en ligne : https://jfmc.github.io/z3-play/).

Rendre son mini-projet

- date limite : 19 décembre 2022 à 22:00
- en binôme, qui est sauf exception le même que pour MP1
- sous la forme d'une archive `XX-nom1-nom2.zip` où `XX` est le numéro de binôme déclaré à la page https://moodle.u-paris.fr/mod/choicegroup/view.php?id=82687 et `nom1` et `nom2` sont les noms de famille des deux membres du binôme, contenant l'arborescence suivante :

```
XX-nom1-nom2/invariants.smt2
XX-nom1-nom2/invariants.ml
XX-nom1-nom2/Makefile
XX-nom1-nom2/RENDU
```