TP67 - Certification d'un analyseur statique de programme

Ce TP est à rendre au plus tard le *Lundi 13 novembre 2017 à 20h00*. Envoyez par mail à votre encadrant de TP une archive dont le nom sera de la forme **Nom1_Nom2_TP67_ACF** et contenant :

1. la théorie `tp67.thy`
2. le contenu de votre projet Eclipse `TP67` en utilisant : File>Export>General>Archive file

---

L'objectif de ce TP est de programmer un analyseur acceptant **le plus possible** de programmes **sûrs** !

1. Programmer en Isabelle un analyseur de programmes
2. Ecrire et de vérifier les propriétés attendues sur l'analyseur (**Bonus** : prouvez-les)
3. Intégrer le code Scala de l'analyseur à l'évaluateur de programmes pour empêcher l'évaluation d'un programme pouvant mener à une attaque.

1 Le langage de programmation `bilbo`

Grammaire	Exemple de programme

`Expression::= BinExpression | Constant | Variable`
`BinExpression::= SumExp | SubExp`
`SumExp::= Expression "+" Expression`
`SubExp::= Expression "-" Expression`
`Condition::= Expression "=" Expression`
`Constant::= i:Integer`
`Variable::= s:String`
 	
`Statement::= Aff | Read | Print | Exec | Seq | If | Skip`
`Aff::= Variable ":=" Expression`
`Read::= "read(" Variable ")"`
`Print::= "print(" Expression ")"`
`Exec::= "exec(" Expression ")"`
`If::= "if(" Condition ") then" Statement "else" Statement`
`Seq::= "{" Statement* "}"`
	`{`
	`  print(1)`
	`  print(2)`
	`  read(z)`
	`  print(z+1)`
	`  x := 0-3`
	`  print(x)`
	`  print(y)`
	`  if (x+5=3) then`
	`  {print(10)`
	`  exec(0)}`
	`  else exec(x)`
	`}`

Dans ce langage, on suppose que l'instruction `exec` fait des appels systèmes. On suppose que l'appel système associé à l'entier `0` est dangereux (une opération exécutée avec les privilèges de `root`, par exemple).

2 Préambule

1. Copiez et chargez dans Isabelle le fichier `/share/m1info/ACF/TP67/tp67.thy`. Ce fichier contient le type abstrait des programmes `bilbo` ainsi qu'un évaluateur `evalS` pour ces programmes. On utilisera cet évaluateur comme la base de confiance de notre théorie.
2. Dans Eclipse Scala, importer le projet se trouvant dans l'archive `/share/m1info/ACF/TP67/TP67ACF.zip` :

   File>Import>General>Existing Projects into Workspace>Select archive file

   Le projet contient deux paquets : `tp67` et `utilities`. Dans ces paquets vous trouverez, en particulier :
   - Un objet `Main` et sa méthode `main` qui vous permet soit (choix `1`) d'exécuter des programmes fournis dans l'archive, soit (choix `2`) de lancer des tests automatiques de votre analyseur.
   - Un objet `Evaluator` qui contient une méthode `eval` permettant d'interpréter un programme écrit dans le langage de programmation `bilbo`.
   - Un objet `Analyzer` dont la méthode `safe` appellera votre analyseur statique de programme. Pour l'instant cet analyseur refuse tous les programmes `bilbo` : il rend `false` par défaut.
   - Des exemples de programmes `bilbo` : `bad1.txt`, `bad2.txt`, ...qui mènent à des attaques et des programmes inoffensifs `ok1.txt`, `ok2.txt`, ...

3 Prise en main de l'interprète de programmes `bilbo`

Dans la théorie Isabelle, vous trouverez des exemples de programmes `p1`, `p2`, `p4` et `p5` ainsi que l'interprète :

`evalS:: statement ⇒ (symTable * inchan * outchan) ⇒ (symTable * inchan * outchan)`

1. Dites pour chaque programme `p1`, `p2`, `p4` et `p5` s'il est inoffensif ou non ;
2. Observez le résultat de l'évaluation de ces programmes par `evalS` avec des entrées particulières (`inchan`) ;
3. Confrontez ces résultats avec vos réponses à la question 1. ;
4. Définissez une fonction `BAD:: (symTable * inchan * outchan) ⇒ bool` qui dit si le résultat de l'évaluation d'un programme a provoqué un `exec(0)`.

4 Définition de l'analyseur statique

Vous devez définir une fonction `san:: statement ⇒ bool` qui sera votre analyseur statique de programmes. Tout programme accepté par l'analyseur est inoffensif. Un programme `p` est inoffensif si, *quelle que soient* les entrées utilisateur, l'exécution de `p` sur ces entrées ne peut provoquer l'exécution de l'instruction `exec(e)` telle que `e` s'évalue à 0. L'analyseur répond `False` dans tous les autres cas : soit il existe une attaque soit il ne sait pas répondre. Définissez l'analyseur statique en Isabelle dans `tp67.thy`. Il est conseillé de procéder par itérations et de conserver le code de chaque version !

1. Un analyseur qui accepte un programme s'il ne comporte pas d'instructions `exec`
2. Un analyseur qui accepte un programme s'il ne comporte que des `exec` sur des constantes différentes de `0`
3. Un analyseur qui accepte un programme dès lors que ses `exec` portent sur des expressions qui ne s'évaluent jamais à `0`.

5 Après chaque itération : vérification de l'analyseur statique

Les propriétés attendues d'un analyseur statique sont les suivantes :

Correction : Tout programme accepté par l'analyseur est inoffensif.

Complétude : Tout programme inoffensif est accepté par l'analyseur.

Pour les langages de programmation classiques, la complétude de l'analyseur est impossible à assurer car le problème est indécidable. Dans le cas particulier du langage `bilbo`, garantir la complétude de votre analyseur serait possible, car le problème est décidable, mais sort du cadre de ce TP. On se servira essentiellement du théorème de complétude pour détecter des programmes inoffensifs mais non acceptés par l'analyseur. Dans `tp67.thy`, définissez les propriétés de correction et de complétude de votre analyseur.

Remarque : sur vos propriétés, cherchez des contre-exemples en poussant les temps de calcul des outils. **Ne passez à la preuve que s'il vous reste du temps en fin de TP** (en commençant par les analyseurs les plus simples).

6 Intégration de l'analyseur statique

Avant de générer le code Scala, complétez la section `imports` de votre théorie `tp67.thy` avec les théories suivantes :

```
imports Main  "~~/src/HOL/Library/Code_Target_Int" "~~/src/HOL/Library/Code_Char"
```

La première permet d'obtenir un code généré qui utilise des `BigInt` Scala pour représenter les `int` Isabelle au lieu d'entiers codés avec `0`, `Succ` et `Pred`. La deuxième permet d'obtenir du code Scala dans lequel les `string` Isabelle sont représentés par des `List[Char]` Scala.

Remarque : ces théories ne doivent être utilisées que pour la génération. En particulier, elles font échouer l'utilisation de `quickcheck`.

Exportez le code Scala de votre analyseur et complétez le code Scala de la méthode `safe` de l'objet `Analyzer`. Vous êtes encouragés à compléter la base de programmes inoffensifs/malveillants avec vos propres jeux d'essais.