TP6

Ce TP est constitué de deux fichiers `Information.java` et `ConstInference.java` à compléter et à rendre. Le TP s'appuie sur le CM6.

Comme dans le TP5, vous devrez implémenter une analyse de propagation de constantes qui va calculer la valeur de chaque variable dans le treillis ⊤ - ℕ - ⊥. Contrairement au précédent TP, cette fois-ci l'analyse porte sur la forme SSA. Grâce à cette forme l'analyse est simplifiée : chaque variable n'est définie qu'en un unique point, et le calcul des ensembles `DefUse` ne nécessite qu'une lecture linéaire du programme.

Le TP consiste en la pré analyse du programme en forme SSA afin de calculer les ensembles `DefUse` (partie 1), puis en l'inférence des variables constantes (partie 2). La transformation du programme en elle-même, propageant les constantes, n'est pas à faire et est donnée. La transformation ne sera cependant correcte que dans le cas où les parties 1 et 2 sont bien réalisées.

N'hésitez pas à aller visiter la documentation de RTL !
https://people.irisa.fr/Timothee.Haudebourg/teaching/ast/doc/index.html

En particulier :
https://people.irisa.fr/Timothee.Haudebourg/teaching/ast/doc/rtl/OperandVisitor.html
https://people.irisa.fr/Timothee.Haudebourg/teaching/ast/doc/rtl/instr/InstrVisitor.html
https://people.irisa.fr/Timothee.Haudebourg/teaching/ast/doc/rtl/instr/package-summary.html

Partie 1 - Pré analyse du programme en forme SSA

Dans cette partie vous devrez implementer la classe située dans le fichier `Information.java` chargée du calcul des ensembles `DefUse`.

L'objectif est de calculer la valeur de la table associative `definitions` liant chaque variable à son unique définition, ainsi que la table `defUses` associant à chaque variable l'ensemble des variables utilisant cette première dans leur définition.

Par exemple, dans le programme suivant:

```
a = 4

b = a + 1

c = a * 2
```

on a
definitions.get(a) -> a = 4
defintions.get(b) -> b = a + 1
defintions.get(c) -> c = a * 2

defUses.get(a) = {b, c}
defUses.get(b) = {}
defUses.get(c) = {}

Une définition peut prendre différente forme. Lisez attentivement le type du champ `definitions` ainsi que la documentation de l'interface `Definition`.
En particulier, notez que la classe `Parameter` représente une variable donnée en paramètre de la fonction et définie à l'extérieur de celle-ci.

Une implémentation de référence vous est donnée par la classe `rtl.ssa.DefaultInformation`. N'hésitez pas à vous en servir dans le fichier `Test.java` pour comparer avec votre propre implémentation.
Bien entendu, vous n'avez pas accès au code source de l'implémentation de référence.

La documentation utile :
https://people.irisa.fr/Timothee.Haudebourg/teaching/ast/doc/rtl/Definition.html
https://people.irisa.fr/Timothee.Haudebourg/teaching/ast/doc/rtl/DefinitionVisitor.html
https://people.irisa.fr/Timothee.Haudebourg/teaching/ast/doc/rtl/Parameter.html
https://people.irisa.fr/Timothee.Haudebourg/teaching/ast/doc/rtl/ssa/Information.html

Partie 2 - Inférence des constantes

Dans cette partie, vous devrez implémenter la classe située dans le fichier `ConstInference.java` chargée de l'analyse d'inférence des constantes.

L'objectif est de calculer la valeur de la table `map` associant à chaque variable sa valeur dans le treillis $\top$ - $\mathbb{N}$ - $\bot$, représenté par le type `rtl.ssa.BotOrIntOrTop`. L'analyse est faite à partir des informations calculées dans la partie 1.

À la place d'équations `In` et `Out` et de calcul d'un point fixe vu dans les précédents TPs, nous profitons ici des propriétés de la forme SSA pour simplifier l'analyse.

Pour cela, vous utiliserez une "worklist" comme vu en cours. Cette "worklist" contient en permanence les identifiants des variables pour lesquelles la valeur associée dans la `ConstMap` doit être mise à jour. Vous devrez donc déterminer:
- Comment initialiser cette worklist ?
- Quelle est la condition d'ajout d'un identifiant dans la worklist ?
- Quelle est la condition d'arrêt de l'analyse ?

La mise à jour d'une valeur se fera à l'aide le fonction `evalDefinition` (que vous êtes chargés d'implémenter) calculant le valeur de treillis résultant de l'évaluation d'une définition.

N'hésitez pas à aller voir la documentation de la classe `rtl.ssa.ConstMap` ainsi que de la classe `rtl.ssa.BotOrIntOrTop` pour savoir quelles sont les méthodes et fonctions à votre disposition.

Une implémentation de référence vous est donnée par la classe `rtl.ssa.DefaultConstInference`. N'hésitez pas à vous en servir dans le fichier `Test.java` pour comparer avec votre propre implémentation.
Bien entendu, vous n'avez pas accès au code source de l'implémentation de référence.

La documentation utile :
https://people.irisa.fr/Timothee.Haudebourg/teaching/ast/doc/rtl/analysis/ConstMap.html
https://people.irisa.fr/Timothee.Haudebourg/teaching/ast/doc/rtl/analysis/BotOrIntOrTop.html
https://people.irisa.fr/Timothee.Haudebourg/teaching/ast/doc/rtl/ssa/ConstInference.html

Tests

Un fichier `Test.java` est mis à votre disposition dans le dossier `test`.
Vous pouvez l'exécuter comme une application Java pour tester votre TP.