TP3

Utilisez le projet fourni dans tp3.tar.gz, comme pour les TPs précédents (importez-le sous Eclipse, configurez l'exécution de MyTest.java pour qu'il reçoive un fichier en entrée).

Ce TP comporte 4 parties et s'appuie sur le CM3. Chaque partie correspond à un fichier à compléter. Il y a donc 4 fichiers à rendre au final. Le fichier TP3ConstSimplification.java n'est pas à rendre.

Partie 1: analyse des définitions possibles (fichier TP3ReachableDef.java)

Complétez le fichier pour construire une analyse des définitions possibles. Lancer MyTest pour voir les différentes étapes du calcul de point fixe de l'analyse obtenue.
Attention, contrairement à ce qui a été présenté en CM4, nous n'utiliserons pas ici de label spécifique "?" pour représenter les variables non-initialisées. Le domaine de l'analyse devient donc simplement SET(VAR * LAB). Il sera représenté par le type Set<RDPair> dans ce TP.

Partie 2: optimisation par propagation de constantes (fichier TP3ConstPropReachDef.java)

En utilisant le résultat de l'analyse des définitions possibles, remplacez les utilisations des variables qui sont toujours des constantes par la dite constante. La classe hérite de Transform. Pour modifier une instruction, la méthode transform doit retourner la nouvelle instruction.

```
public TransformInstrResult transform(Assign a) {
        Instr newInstr = ...;
        Node n = cfg.node(a);
        cfg.updateInstr(n, newInstr); // Mise à jour du CFG
        return new TransformInstrResult(newInstr);
}
```

Il est essentiel de mettre à jour le CFG afin de garder le lien entre les nœuds et les instructions au cours de l'analyse.

La méthode transform doit être redéfinie pour toutes les instructions impactées par l'optimisation. Les implémentations par défaut dans la classe Transform appliquent l'identité.

Partie 3: optimisation par simplification d'expression (fichier TP3ConstExprSimpl.java)

Le but de cette optimisation est de remplacer toutes les opérations arithmétiques constantes du programme par leur résultat. Par exemple

```
t.0 = Add(1 2)
```

se simplifie en

```
t.0 = 3
```

La classe hérite de Transform et doit simplifier les opérations arithmétiques, c'est-à-dire redéfinir les instructions Built-in, quand cela est possible.

Itérations

Les optimisations des parties 2 et 3 doivent être appliquées itérativement pour optimiser un programme. Le fichier TP3ConstSimplification.java se charge d'appeler les deux transformations jusqu'à ce qu'un point fixe soit atteint. Définissez dans le fichier TP3ConstExprSimpl.java, via la méthode `public boolean hasChanged()` la gestion de la condition pour réitérer les optimisations.

Partie 4: optimisation par élimination de code mort (fichier TP3DeadDefElim.java)

L'optimisation doit supprimer toutes les affectations (et uniquement les affectations) qui sont inutiles, c'est-à-dire pour lesquelles la variable n'est jamais utilisée. La classe hérite de Transform et peut comme les précédentes optimisations modifier les instructions. Retourner `null` comme nouvelle instruction supprime l'ancienne.
Pour cette analyse, vous n'avez besoin que des chaînes definition-usages, disponible via l'attribut `defUse` de la classe.

Nous vous invitons à tester le résultat final de ce TP sur le programme examples/rtl/TP3Challenge.rtl