Algorithmes et Complexité
Projet

Le devoir doit être effectué par groupe de 1 *à* 2 *personnes, chaque personne appartenant à exactement un groupe.*

1 Travail à effectuer

Description

Le devoir est à rendre sur `arche` pour le **2 janvier 2023**.

Certaines questions nécessitent d'écrire un programme Java. Pour vous aider, des classes sont fournies. Il est bien entendu possible de modifier le code fourni.

Le projet étant un projet d'algorithmique et non pas de développement logiciel, la majeure partie de la notation portera sur l'implémentation correcte des algorithmes et les réponses aux questions.

2 Evaluation

Le but de cet exercice est de simplifier des programmes et (surtout) de vérifier les simplifications. Dans toute la suite, un programme est une liste d'instructions. Une instruction est soit une affectation simple, soit une opération arithmétique parmi $+, *, -$. Voici un exemple de programme

```
x=1
y=x
x=2*y
z=x-y
x=z*4
```

Le *résultat* d'un programme est la valeur de la variable `x` à la fin du programme. (On définit le résultat comme valant 0 s'il n'y a pas de variables `x`).

Pour vous aider, les fichiers Java donnés permettent de convertir un programme (une chaîne de caractères, ou un fichier) en une liste d'instructions. Regardez en particulier le fichier `Toto.java`. Attention ! Le parseur n'est pas parfait. Il suppose que les instructions soient séparées par des retours à la ligne ou des points-virgules.

Q 1) Ecrire un évaluateur pour un programme. *Rendu de la première question : Un programme Java, qui prend en entrée un fichier, et qui renvoie l'entier correspondant.*

Q 2) Les entiers en java sont des entiers 32 bits. Donner un programme de maximum 10 lignes pour lequel vous n'obtenez pas la bonne réponse car le résultat est trop grand. (Au contraire de Java, Python a des entiers de taille arbitraire, vous pouvez donc tester votre liste d'instructions avec Python). *Rendu de la deuxième question : un programme (une liste d'instruction), testé avec le rendu de la première question, et une explication de pourquoi le résultat est trop grand.*

3 Simplification et évaluation abstraite

Le but de cette partie 2 est d'écrire un optimiseur pour les programmes. Par exemple, le programme suivant :

```
x = 1
w = 4
y = x - x
x = y * w
```

peut être optimisé en

```
x = 0
```

Le problème est qu'on ne peut pas évaluer l'expression car le résultat est trop grand (cf. question précédente).

L'idée est d'évaluer les instructions ligne par ligne et d'éliminer les opérations inutiles, mais en ne gardant qu'une valeur approchée (on dit abstraite) des nombres : lors de l'évaluation on retient, pour chaque variable, si elle vaut 0, 1 ou une valeur inconnue.

Par exemple, après l'évaluation des deux premières lignes, on retiendra que x vaut 1 et que w vaut une valeur inconnue (car "4" est un nombre trop grand). Lors de l'évaluation de la ligne 3, on retiendra que y vaut 0, ce qui permettra de simplifier la dernière ligne.

L'inconvénient est qu'on ne peut pas traiter tous les programmes. Regardons le programme suivant :

```
x = 4
w = 4
y = w - x
x = y * w
```

On ne pourra pas optimiser le programme car on ne se rend pas compte que y vaut 0 : En effet x et w valent tous les deux "valeur inconnue" donc quand on soustrait, on obtient une valeur inconnue.

Q 3) Ecrire un simplificateur. Le programme prend un fichier en entrée et renvoie un fichier simplifié. *Rendu de cette question : un programme, des fichiers de test, et une explication des optimisations traitées et non traitées. Cette question est très ouverte, lâchez-vous !.*

3.1 Test des simplifications

Le but maintenant est d'écrire un programme qui vérifie que l'optimiseur fonctionne correctement : Le problème comme on l'a vu est que l'évaluation est impossible car les nombres sont trop grands.

L'idée dans cette partie est de faire l'évaluation modulo un nombre premier p de 30 bits tiré aléatoirement. Comme on fait l'évaluation modulo p, les nombres deviennent petits. Et on montrera dans la partie suivante qu'on n'a peu de chance de se tromper.

Q 4) Ecrire une fonction évaluateur qui prend un programme (une liste d'instructions) et un nombre entier p et qui évalue le programme modulo p. Voir ci-dessous pour comment multiplier deux nombres.

Une des difficultés des opérations est que si x et y sont entre 0 et $2^{30} - 1$ alors $(x \times y)$ mod p ne donne pas le bon résultat : En effet le produit $x \times y$ nécessite plus de 32 bits, donc sera tronqué lors du calcul.

On pourra utiliser le fragment de code suivant pour calculer le produit de deux nombres :

```
int multiply(int x, int y, int p)
{
   int result = 0;
   while (y)
     {
        if (y & 1) result = (result + x) % p;
        y >>= 1;
        x = (2*x)%p;
     }
   return result;
}
```

Q 5) Avec la fonction précédente, écrire un programme Java qui prend deux programmes (listes d'instructions) en entrée, qui tire un nombre premier au hasard, et renvoie Vrai si les deux programmes donnent le même résultat modulo p. Pour tirer un nombre premier au hasard, on pourra utiliser la classe `RandomPrime` fournie. *Rendu de la question : Un programme Java, qui prend en entrée deux fichiers, et qui dit si les deux fichiers donnent le même résultat modulo p.*

4 Partie théorique

On suppose dans toute cette partie que les seuls entiers autorisés quand on écrit un programme sont 0 et 1 (pour simplifier) : Si on veut construire $x = 5$ par exemple, on peut faire $x = 1; y = x + x; y = x + x; y = x + y$ par exemple.

Q 6) Montrer que le résultat d'un programme de taille n est inférieur à 2^{2^n}.

Pour la suite, on admettra les deux résultats suivant :

- Si $x \leq 2^k$ alors il existe au maximum k nombres premiers p tel que $x = 0 \mod p$.
- Il y a à peu près 50 millions de nombres premiers inférieurs à 2^{30}.

Q 7) On se donne deux programmes p_1, p_2 de 20 instructions. Montrer que :

- Si p_1 et p_2 calculent le même nombre, alors le programme répondra toujours Vrai.
- Si p_1 et p_2 calculent des nombres différents, le programme a une probabilité $> 1/2$ de répondre Faux.

Q 8) Est-ce que le résultat est encore vrai pour des programmes de 24 instructions ? 28 ? *Note : Le fait qu'on ne puisse traiter que des programmes de 24∼28 instructions peut paraître frustrant, mais il faut se souvenir qu'on utilise des entiers Java qui sont sur 4 octets et donc très petits. En utilisant des entiers de taille variable, on peut montrer qu'on peut traiter des programmes avec n instructions en un temps polynomial en n, ce qui est pas mal vu que les programmes peuvent représenter des nombres de taille* 2^{2^n}.

5 Recherche documentaire

On cherche maintenant à savoir si deux *fonctions* sont équivalentes. Contrairement aux programmes, on a maintenant des entrées, et les fonctions s'écrivent par exemple :

```
def f(a,b):
  x = a + b
  y = x * 2
  z = a + x
  x = b - y
```

On ne peut pas utiliser ce qu'on a fait précédemment, car, pour tester que deux fonctions sont équivalentes, il faudrait tester toutes les valeurs possibles des arguments, et il y en a une infinité ! !

Q 9) Cherchez sur Internet des documents sur "Polynomial Identity Testing" ou "Schwartz-Zippel". Résumez en une vingtaine de lignes ce que vous avez compris, et expliquez comment on pourrait implémenter tout ça. Vous devrez mentionner explicitement les références que vous avez utilisés, et au moins deux d'entre elles doivent être hors Wikipedia.