TD3 : Unification et analyse de vivacité

Exercice 1

Les paires de termes suivants sont-elles unifiables ? Si oui, donner un unificateur le plus général possible.

- $X \to X$ et $\text{int} \to Y$
- $X \to X$ et $tab[Y] \to Z$
- $X \to X$ et $tab[Y] \to Y$
- $(X \to Y) \to (Z \to Z)$ et $((W \to Z) \to V) \to W$

Exercice 2

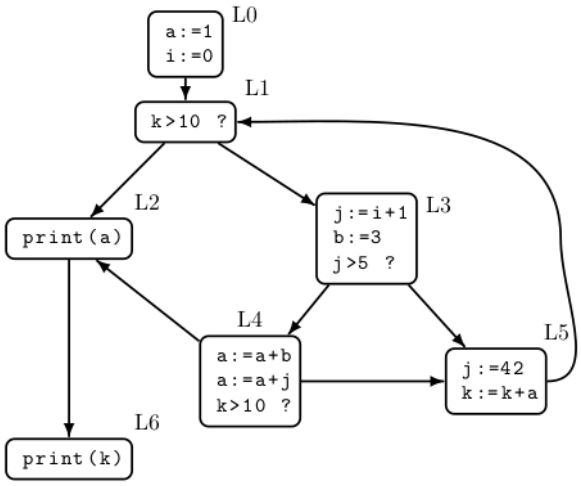
Donner le type le plus général des fonctions suivantes.

```
auto f(auto x, auto y, auto z){
    if (x=={y})
        return {z}+x;
    else
        return {z,y};
}
```

```
auto g(auto a, auto b) {
    while (a != {})
        b := f(b, a, g(a,b));
    return b[0];
}
```

Exercice 3

On considère le graphe de contrôle suivant :



On rappelle qu'une variable est *tuée* par un bloc si elle apparaît à gauche d'une affectation. Elle est *générée* par un bloc si sa valeur est utilisée dans le bloc (avant d'être possiblement tuée). Une variable v est *vivante à la sortie* d'un bloc B s'il existe un chemin de B vers une utilisation de v qui ne passe pas par une redéfinition de v.

1. Quelles sont les variables générées et tuées par chaque bloc ?

2. Pour chaque bloc, donner la liste des variables vivantes à sa sortie.

Exercice 4

On considère le programme suivant :

```
while (d>0) {
    a:=b+c;
    d:=d-b;
    e:=a+f;
    if (e>0) {
        f:=a+d;
        b:=d+f;
    } else {
        e:=a-c;
    }
    b:=a+c;
}
```

1. Dessiner son graphe de contrôle.

2. Pour chaque bloc, calculer LV_{entry} et LV_{exit}.

Exercice 5

On considère l'expression $E = ((n + 1) * n) + (2 * n)$.

1. En supposant que la variable n est stockée dans la mémoire à l'adresse stockée dans le stack pointer SP, écrire un code intermédiaire calculant E. On écrira le code le plus naïf possible, qui n'essaye pas de réutiliser des registres, et qui utilise un nouveau registre pour stocker la valeur de chaque sous-expression.

2. Donner la liste des registres vivants en entrée de chaque instruction.

3. Dessiner le graphe de conflits, puis le colorer.

4. On suppose que R_2, R_3, R_4 sont réservés pour gérer les variables en mémoire, que R_5 contient le stack pointer SP, et que le processeur a seulement deux autres registres utilisables R_0 et R_1. Transformer le code intermédiaire en code assembleur.