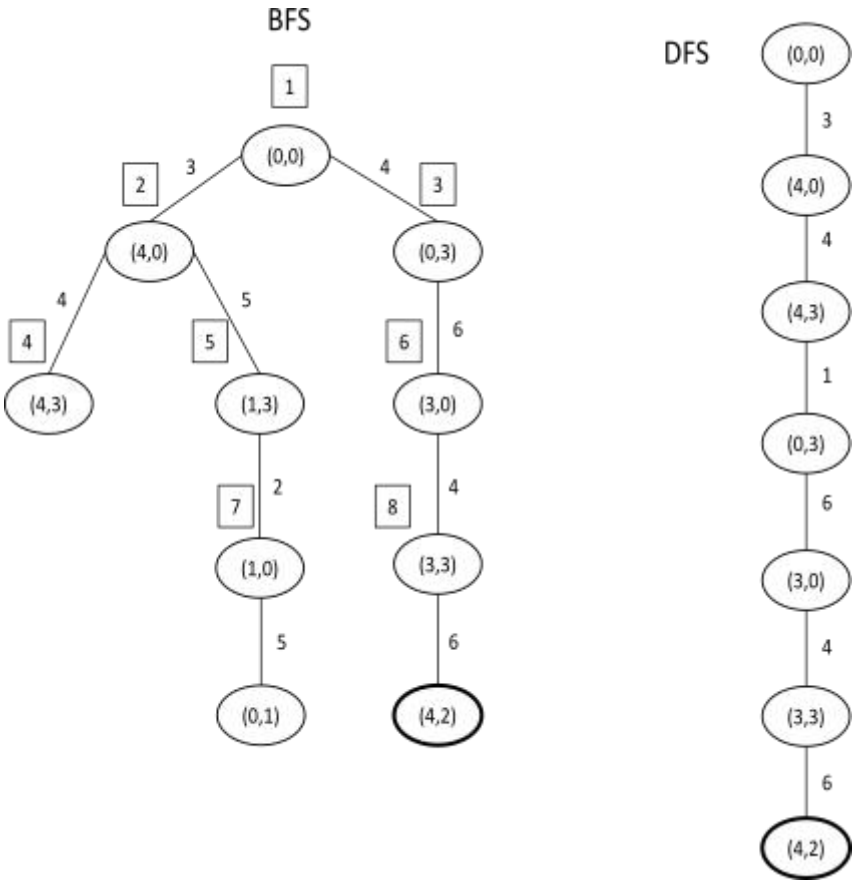
Corrigé Examen TPGO 2022-2023

I. Conception de programmes

Trace

La figure ci-après montre la trace des deux types de parcours. On part de la configuration initiale (0, 0) pour arriver à la configuration finale (4, 2). Les chiffres à coté des arcs indiquent les opérations effectuées. Pour DFS, l'ordre de parcours est de haut en bas. Pour BFS, l'ordre de parcours est mentionné par les chiffres à l'intérieur des carrés.



Pseudo-algorithme DFS

Nous utiliserons une pile. Soit L la liste des configurations existantes.

L := {}; Creerpile(P) // P vide

1. Empiler(P, (0, 0);
2. Si Pilevide(P) fin avec échec
3. Dépiler(P, (x, y) ; Si (x = d) ou (y=d) Fin avec succès
4.

- Générer les configuration possibles (a, b) à partir de (x, y) par l'application des règles de 1 à 6 dans l'ordre.
- Pour chaque (a, b) n'appartenant pas à L : Empiler(P, (a, b)); L := L + (a, b) Fpour

5. Aller 2

Pseudo-algorithme BFS

Nous utiliserons une file d'attente. Soit L la liste des configurations existantes.

L := {}; Creerfile(F); // F vide.

1. Enfiler(F, (0, 0);
2. Si Filevide(F) fin avec échec
3. Defiler(F, (x, y) ; Si (x = d) ou (y=d) Fin avec succès
4.

- Générer les configuration possibles (a, b) à partir de (x, y) par l'application des règles de 1 à 6 dans l'ordre.
- Pour chaque (a, b) n'appartenant pas à L : Enfiler(F, (a, b)); L := L + (a, b) Fpour

5. Aller 2

II. Complexité

Pseudo-algorithme qui colore un graphe avec 2 couleurs.

Nous devons visiter chaque nœud du graphe. Nous pouvons le faire soit avec DFS ou BFS. Initialement les nœuds ne sont pas colorés. Nous attribuons une couleur arbitraire pour le premier nœud visité.

Pour chaque nœud visité , nous devons vérifier 2 cas:

- 1 ou plusieurs nœuds adjacents à n ont déjà une même couleur. Nous attribuons à n la seconde couleur.
- 2 ou plusieurs nœuds adjacents à n ont déjà une couleur différente. Il n'est pas possible d'attribuer à n une couleur valide.

2-COLOR se réduit polynomialement à 2-SAT

On construit une instance de 2-SAT à partir d'un graphe 2-COLOR de la manière suivante:.

- à chaque sommet u on associe une variable booléenne xu égale à Vrai pour une couleur et égale à faux pour l'autre couleur.
- pour chaque couple (u, v) de sommets, u # v, on construit les deux clauses (xu || xv) && (¬xu || ¬xv). Notez que cette expressions n'est vraie que pour les valeurs de xu et xv contradictoires. Ceci garantit donc que les deux nœuds u et v sont de couleurs différentes.

l'instance de 2-SAT est donc le produit de toutes les clauses associées aux arêtes du graphe.

Si G est 2-coloriable, alors on obtient une formule vraie.

Il est facilement observable que L'algorithme de construction de l'instance est polynomial. Ainsi 2-COLOR $\leq_P$ 2-SAT.

D'après le théorème : Si L1 $\leq_P$ L2 et L2 $\in$ P, alors L1 $\in$ P (L1 et L2 étant deux problèmes)

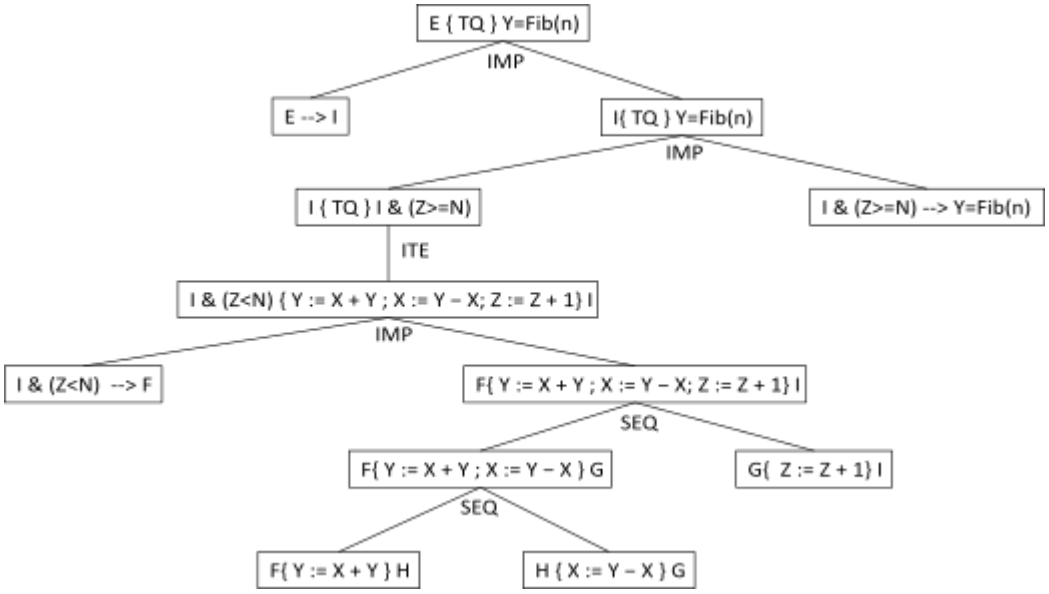
Comme 2-COLOR $\leq_P$ 2-SAT et 2-SAT est dans P, alors on déduit que 2-COLOR est dans P.

III Construction de programmes

Arbre de preuve

Posons E = ($X = 0 \wedge Y = 1 \wedge Z = 1 \wedge 1 \leq N \wedge N = n$)

I est l'invariant de boucle.



Preuve de programme

$I \equiv Y = fib(Z) \wedge X = fib(Z-1)$

I est un invariant de boucle si

I { Y := X + Y ; X := Y − X; Z := Z + 1) I

C'est à dire

G { Z := Z + 1 } I

H {X:= Y-Z} G

F{Y:=X + Y} H avec F = I

L'axiome de l'affectation donne

G ≡ Y=Fib(Z+1) et X=fib(Z)

H ≡ Y=Fib(Z+1) et Y-X=fib(Z)

I ≡ X+Y=Fib(Z+1) et (X+Y) - X= fib(Z)

I peut être simplifié comme suit:

I ≡ X+Y=Fib(Z+1) et Y= fib(Z)

I ≡ X+fib(Z)=Fib(Z+1) et Y= fib(Z)

I ≡ X=Fib(Z+1) - fib(Z) et Y= fib(Z)

$I \equiv X = fib(Z-1) \wedge Y = fib(Z)$

Nous aurons à prouver:

(1) $X = 0 \wedge Y = 1 \wedge Z = 1 \wedge 1 \leq N \wedge N = n \Rightarrow I$

(2) $\{I \wedge (Z < N)\}$ Y := X + Y ; X := Y − X; Z := Z + 1 {I }

(3) $(I \wedge \neg(Z < N)) \Rightarrow Y = fib(n)$

(1) $X = 0 \wedge Y = 1 \wedge Z = 1 \wedge 1 \leq N \wedge N = n \Rightarrow 1 = fib(1) \wedge 0 = fib(0)$

Le deuxième membre est vrai, l'implication est donc vraie.

(2) On a déjà montré

I { Y := X + Y ; X := Y − X; Z := Z + 1} I

Nous devons aussi prouver:

$I \wedge (Z < N) \Rightarrow I$
Ce qui est vraie.

(3) Bien que $Y = fib(Z) \wedge X = fib(Z - 1)$est un invariant de boucle, il ne permet pas de démontrer la post condition souhaitée.
Nous devons rajouter une condition de l'arrêt de la boucle Z = N.
L'invariant devient $Y = fib(Z) \wedge X = fib(Z - 1) \wedge Z \leq N \wedge N = n$
(3) est alors prouvée comme suit avec le nouvel invariant:
$(Y = fib(Z) \wedge X = fib(Z - 1) \wedge Z \leq N \wedge N = n)$ et $\neg(Z < N)) \Rightarrow Y = fib(n)$
$(Y = fib(Z) \wedge X = fib(Z - 1) \wedge (Z = n) \Rightarrow Y = fib(n)$
$(Y = fib(n) \wedge X = fib(n - 1) \Rightarrow Y = fib(n)$
Ce qui est vraie.

<u>Evaluation des lambda-expressions</u>
1. $(\lambda x. (x+y)) 3$
 => 3 + y

2. $(\lambda f. \lambda x. f (f x)) (\lambda y. y+1)$
 => $\lambda x. (\lambda y. y+1) ((\lambda y. y+1) x)$
 => $\lambda x. (\lambda y. y+1) (x+1)$
 => $\lambda x. (x+1)+1$

3. $\lambda x. (\lambda y. y x) (\lambda z. x z)$
 => $\lambda x. (\lambda z. x z) x$
 => $\lambda x. x x$

4. $(\lambda x. (\lambda y. y x) (\lambda z. x z)) (\lambda y. y y)$
 => $(\lambda x. x x) (\lambda y. y y)$
 => $(\lambda y. y y) (\lambda y. y y)$
 => $(\lambda y. y y) (\lambda y. y y)$
=> ...

<u>Rappel des définitions de variables libres et liées:</u>

Variables libres	Variables liées
Varlib(a) = {}	Varlié(a) = {}
Varlib(x) = {x}	Varlié(x) = {}
Varlib(XY)= Varlib(X) U Varlib(Y)	Varlié(XY)= Varlié(X) U Varlié(Y)
Varlib(λx.X)= Varlib(X) - {x}	Varlié(λx.X)= Varlié(X) U {x}

<u>Détermination des variables libres et liées</u>
E= $(\lambda f. \lambda x. f (f x)) (\lambda y. y+x)$
De la forme MN
Varlib(E) =Varlib[$(\lambda f. \lambda x. f (f x))$] U Varlib[$(\lambda y. y+x)$]

= Varlib[λ x. f (f x)] - {f} U Varlib[y+x] - {y}
= Varlib [f (f x)] - {x}- {f} U {y, x} - {y]
= {f, x} - {x} - {f} U { x}
= {} U {x}
= {x}

Varlie(E) =Varlie[(λ f. λ x. f (f x))] U Varlie[(λ y. y+x)]
= Varlie[λ x. f (f x)] + {f} U Varlie[y+x] + {y}
= Varlie [f (f x)] + {x} + {f} U {} + {y]
= {} + {x} +{f} U { y}
= {x, f, y}

Calcul de (λ f. λ x. f (f x)) (λ y. y+x)
Renommons la variable liée x en z
(λ f. λ z. f (f z)) (λ y. y+x)
=> λ z. [(λ y. y+x) ((λ y. y+x) z))]
=> λ z. z+x+x

Démonstrateur Prolog
Soit le programme Prolog:
E(X,[X|_]). (1)
E(X,[_|R]) :- E(X,R). (2)
SE([],_). (3)
SE([X|R],L) :- E(X,L),SE(R,L). (4)

Goal : - sous-ensemble([1,2,8],[1,3,5,2,8])
Le démonstrateur doit réfuter
Not SE([1,2,8], [1, 3, 5, 2, 8])
|
| [X|R] := [1,2,8] ; L := [1, 3, 5, 2, 8] (4) // Unification et règle utilisée
|
Not(E(1, [1, 3, 5, 2, 8], SE([2,8], [1, 3, 5, 2, 8])
|
| X:= 1 ; [X,_] := [1, 3, 5, 2, 8] (1)
|
Not (SE([2,8], [1, 3, 5, 2, 8])
|
| [X|R] := [2,8] ; L := [1, 3, 5, 2, 8] (4)
|
Not(E(2, [1, 3, 5, 2, 8], SE([8], [1, 3, 5, 2, 8])
|
| X:= 2; [_|R]:= [1, 3, 5, 2, 8] (2)
|
Not(E(2, [3, 5, 2, 8], SE([8], [1, 3, 5, 2, 8])

|

| X:= 2 ; [_|R]:= [3, 5, 2, 8] (2)

|

Not(E(2, [5, 2, 8], SE([8], [1, 3, 5, 2, 8])

|

| X:= 2 ; [_|R]:= [5, 2, 8] (2)

|

Not(E(2, [2, 8], SE([8], [1, 3, 5, 2, 8])

|

| X:=2; [X,_] := [2, 8] (1)

|

Not(SE([8], [1, 3, 5, 2, 8])

|

| [X|R] := [8] ; L := [1, 3, 5, 2, 8] (4)

|

Not(E(8, [1, 3, 5, 2, 8], SE([], [1, 3, 5, 2, 8])

|

| X:= 8 ; [_|R]:= [1, 3, 5, 2, 8] (2)

|

Not(E(8, [3, 5, 2, 8], SE([], [1, 3, 5, 2, 8])

|

| X:= 8; [_|R]:= [3, 5, 2, 8] (2)

|

Not(E(8, [5, 2, 8], SE([], [1, 3, 5, 2, 8])

|

| X:= 8 ; [_|R]:= [5, 2, 8] (2)

|

Not(E(8, [2, 8], SE([], [1, 3, 5, 2, 8])

|

| X:= 8; [_|R]:= [2, 8] (2)

|

Not(E(8, [8], SE([], [1, 3, 5, 2, 8])

|

| X:=8; [X,_] := [8] (1)

|

Not(SE([], [1, 3, 5, 2, 8])

|

| _ := [1, 3, 5, 2, 8] (3)

|

Clause vide

Goal : - sous-ensemble(X,[1,3,5,2,8])

Prolog ne peut fonctionner. Aucune unification n’est permises avec les conséquents des règles.