Corrigé Examen final – TPGO / 2CS – ESI 2012/2013

1) Expliquez pourquoi la programmation dynamique est-elle qualifiée de « méthode ascendante » ?

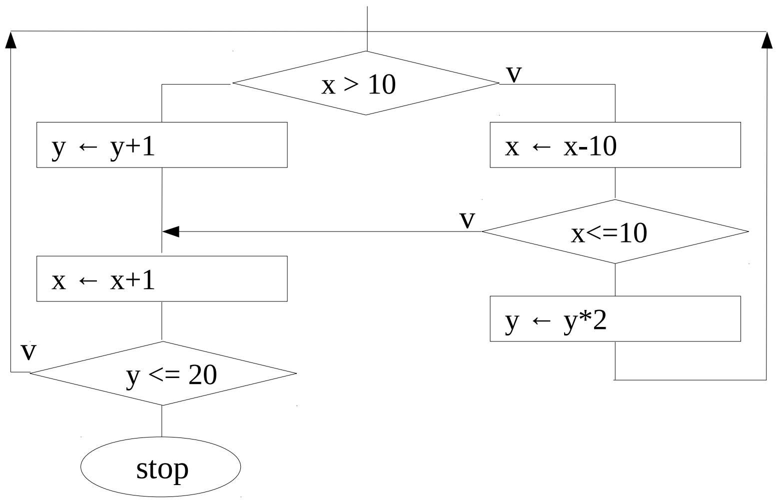
Car les sous-problèmes de l'arbre de décomposition sont considérés dans un ordre ascendant, des plus simples aux plus complexes.

2) Quand est-ce que l'exploration d'un espace de recherche nécessite t-elle un parcours en largeur ?

La parcours en largeur d'un espace de recherche devient une nécessité, lorsque cet espace est infini ou alors lorsqu'on est intéressé par trouver l'état solution le plus proche de l'état initial en terme de nombre de transitions.

3) En utilisant le théorème de Böhm et Jacopini, donnez une forme structurée du programme suivant :

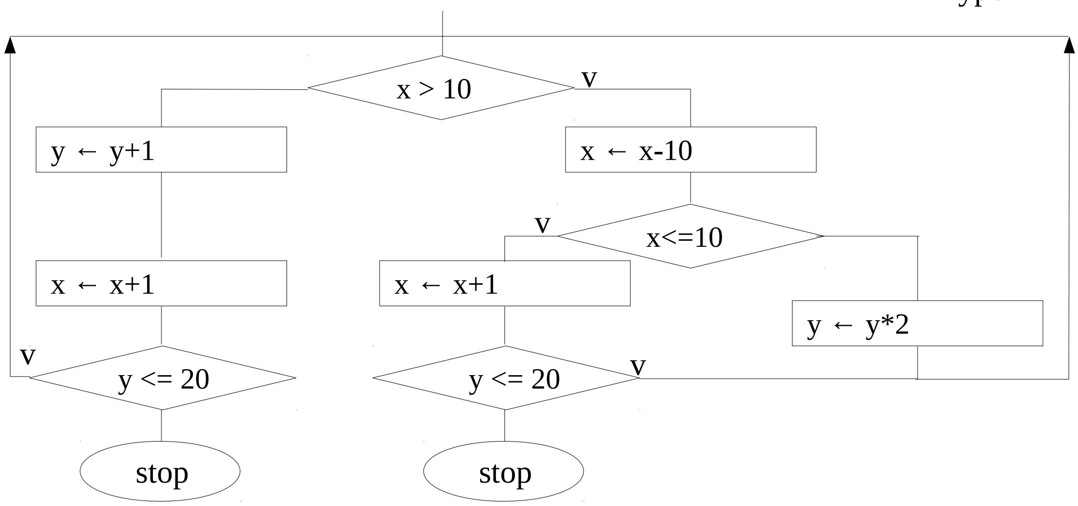
etiq1 : SI (x > 10) Allerà etiq2 ;
y ← y+1 ;
etiq4 : x ← x+1 ;
SI (y <= 20) Allera etiq1 ;
Allerà etiq3 ;
etiq2 : x ← x-10 ;
SI (x<=10) Allerà etiq4 ;
*y ← y*2 ;*
Allerà etiq1 ;
etiq3 : stop.



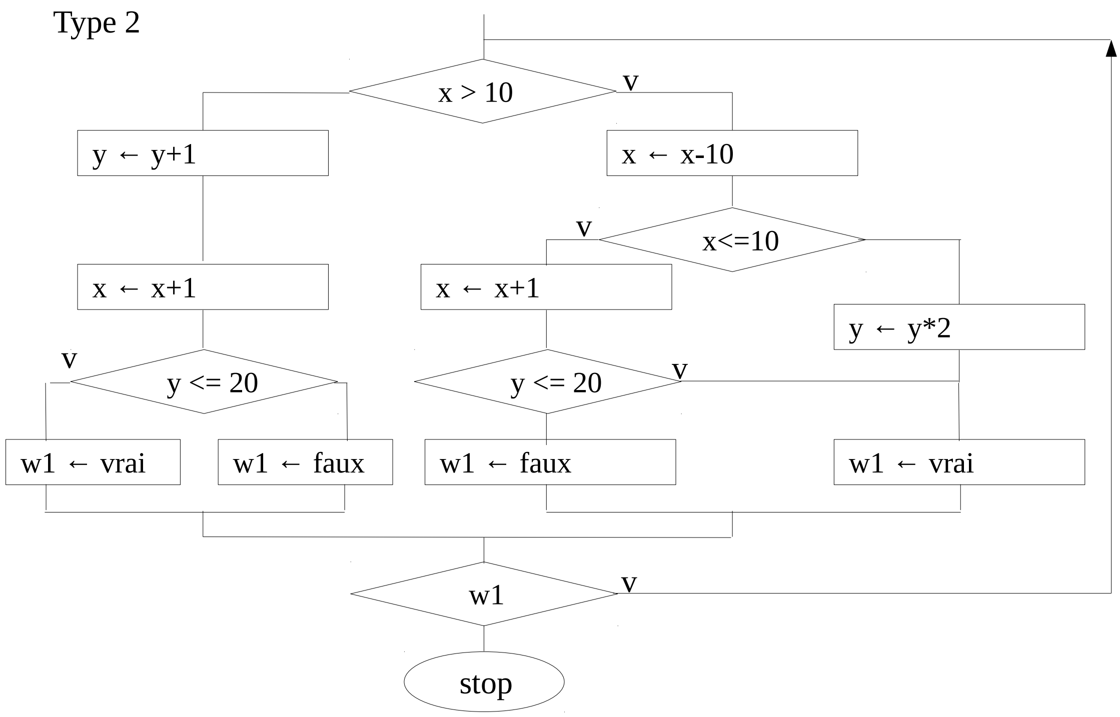
Type 3

On le transforme en type 2 en dupliquant les parties communes :

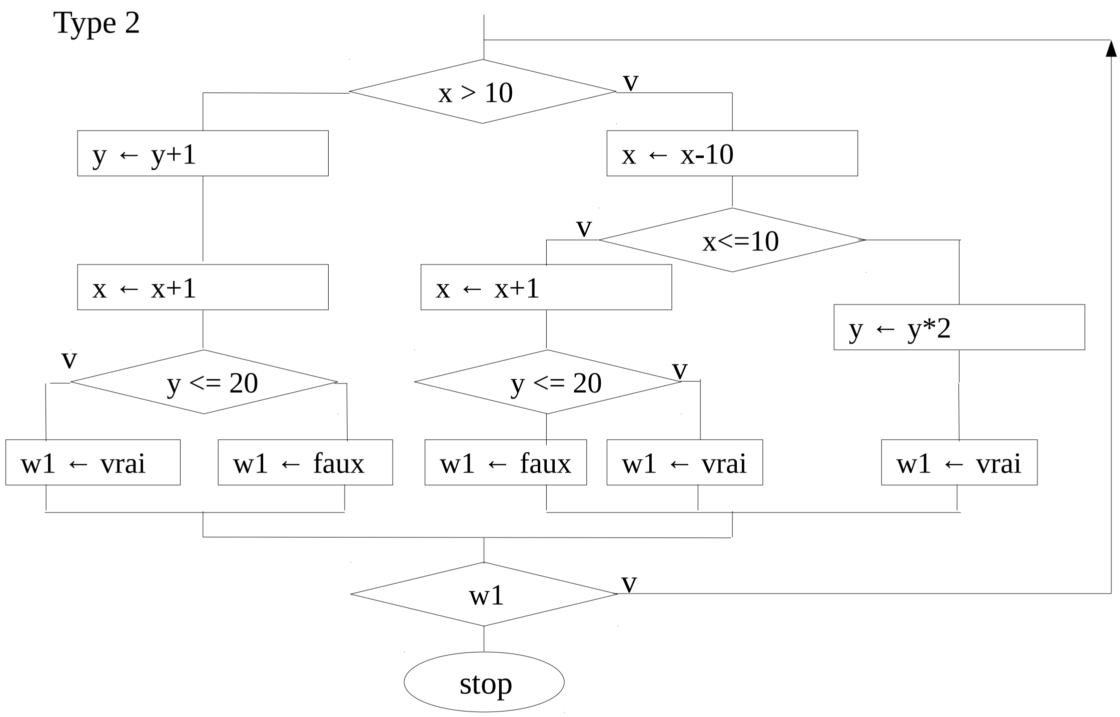
Type 2



On applique la transformation relative au type 2 :



Le sous-organigramme relatif au test « x <= 10 » est de type 3, on le transforme en type 2 en dupliquant la partie commune « l'affectation w1 ← vrai ».
Nous obtenons alors la forme finale de l'organigramme structuré :



4) Donnez les plus faibles préconditions nécessaires pour que les énoncés suivants, soient des théorèmes dans le système formel de Hoare :

a- *E { x ← y-x ; z ← y-1 ; y ← x+3 ; x ← z-y } (x = 3)*

L'énoncé (a) est vrai par SEQ si les deux énoncés suivants sont vrais :
(a1) E { x ← y-x ; z ← y-1 ; y ← x+3 } F et (a2) F { x ← z-y } (x = 3)
L'énoncé (a2) est vrai par AFF en choisissant F : (z-y = 3)
L'énoncé (a1) est vrai par SEQ si les deux prémisses ci-dessous sont vraies aussi :
(a3) E { x ← y-x ; z ← y-1 } G et (a4) G { y ← x+3 } F
L'énoncé (a4) est un axiome AFF en choisissant G : (z-x = 6)
L'énoncé (a3) est vrai par SEQ, si les deux énoncés ci-dessous sont aussi vrais :
(a5) E { x ← y-x } H et (a6) H { z ← y-1 } G
L'énoncé (a6) est un axiome AFF pour H : (y-x = 7)
L'énoncé (a5) est un axiome AFF pour E : (x=7)
Donc **la plus faible précondition (E)** nécessaire pour que l'énoncé (a) soit vrai est : **(x=7)**

b- *E { SI (x > y) z ← 2 ; SINON x ← x+1 ; SI (y = x) z ← z-1 FSI ; x ← y+1 FSI ; } (x=z)*

L'énoncé (b) est vrai par CND2, si les deux prémisses suivantes sont vraies :
(b1) E & (x > y) { z ← 2 } (x = z) et
(b2) E & (x ≤ y) { x ← x+1 ; SI (y = x) z ← z-1 FSI ; x ← y+1} (x = z)
L'énoncé (b1) est vrai par SEQ si les deux énoncés ci-dessous sont aussi vrais :
(b3) E & (x ≤ y) { x ← x+1 ; SI (y = x) z ← z-1 FSI } F et
(b4) F { x ← y+1} (x = z)
L'énoncé (b4) est un axiome AFF en choisissant pour F, le prédicat : (y+1 = z)
L'énoncé (b3) est vrai par SEQ si les deux énoncés ci-dessous sont aussi vrais :
(b5) E & (x ≤ y) { x ← x+1 } G et (b6) G { SI (y = x) z ← z-1 FSI } F
L'énoncé (b6) est vrai par CND1 si ses deux prémisses sont vraies aussi :
(b7) G & (y = x) { z ← z-1 } F et (b8) G & (y ≠ x) ⇒ F
L'énoncé (b7) est vrai par IMP1 si les deux énoncés suivants sont vrais :
(b9) G & (y = x) ⇒ H et (b10) H { z ← z-1 } F
L'énoncé (b10) est vrai par AFF (axiome) en prenant comme plus faible précondition H : (y+2 = z)
Il faut maintenant choisir la plus faible précondition G rendant les implications (b8) et (b9) vraies :
G : (x=y & y+2 = z) ou (y ≠ x & y+1 = z)
L'énoncé (b5) E & (x ≤ y) { x ← x+1 } G est vrai par IMP1 si ses deux prémisses sont vraies aussi :
(b11) E & (x ≤ y) ⇒ I et (b12) I { x ← x+1 } G
L'énoncé (b12) est vrai par AFF en choisissant pour I la condition : (x+1=y & y+2 = z) ou (y ≠ x+1 & y+1 = z)

L'énoncé (b1) E & (x > y) { z ← 2 } (x = z) est vrai par IMP1, si ses deux prémisses sont vraies :
(b13) E & (x > y) ⇒ J et (b14) J { z ← 2 } (x = z)
L'énoncé (b14) est vrai par AFF pour la condition J : (x = 2)

Maintenant, il ne reste qu'à choisir la plus faible précondition (E) rendant vrai les deux implications (b11) et (b13) en même temps.
E : (x>y & x=2) ou (x+1=y & y+2=z) ou (x ≤ y & y ≠ x+1 & y+1=z)

c- *E {x ← a ; y ← b ; TQ (y>1) x ← x*2 ; y ← y div 2 FTQ } (x = ab)*

L'énoncé (c) est vrai par SEQ, si ses deux prémisses le sont aussi :
(c1) E {x ← a ; y ← b } F et (c2) F { TQ (y>1) x ← x*2 ; y ← y div 2 FTQ } (x = ab)
L'énoncé (c2) est vrai par IMP2, si les deux énoncés suivants sont vrais :
(c3) F { TQ (y>1) x ← x*2 ; y ← y div 2 FTQ } (F & y ≤ 1) et
(c4) (F & y ≤ 1) ⇒ (x = ab)
// F étant l'invariant de boucle à trouver
L'énoncé (c3) est vrai par ITE si l'énoncé suivant est vrai aussi :
(c5) (F & y>1) { x ← x*2 ; y ← y div 2 } F
Cet énoncé (c5) est vrai par SEQ, si ses deux prémisses le sont aussi :

(c6) (F & y>1) { x ← x*2 } G et (c7) G { y ← y div 2 } F

L'énoncé (c7) est vrai par AFF (axiome) en prenant G la condition : [y div 2 / y] F

L'énoncé (c6) est vrai par IMP1, si ses deux prémisses sont aussi vraies :

(c8) (F & y>1) ⇒ H et (c9) H{ x ← x*2 } G

L'énoncé (c9) est un axiome (AFF) en choisissant pour H la condition : [2x / x]G (donc [2x / x] [y div 2 / y] F)

L'énoncé (c1) E {x ← a ; y ← b } F est vrai par SEQ si les deux énoncés suivants sont vrais :

(c10) E { x ← a } I et (c11) I { y ← b } F

L'énoncé (c11) est un axiome (AFF) en choisissant I : [b / y] F

L'énoncé (c10) est aussi un axiome (AFF) en choisissant E : [a / x] I (c-a-d [a / x] [b / y] F)

Pour terminer la preuve, il ne reste qu'à déterminer le bon invariant (F) permettant de rendre vrai les implications :

(c4) (F & y ≤ 1) ⇒ (x = ab) et (c8) (F & y>1) ⇒ [2x / x] [y div 2 / y] F

Pour rendre vrai (c4) on peut choisir comme invariant F, la condition : (xy = ab & y ≥ 1)

Par contre ce choix ne permet pas de vérifier (c8) :

(xy = ab & y ≥ 1 & y>1) n'implique pas : (2x(y div 2) = ab & (y div 2) ≥ 1) car la condition 2x(y div 2) = ab ne peut pas être déduite à partir des antécédents de l'implication (xy = ab & y ≥ 1 & y>1), car y doit être un nombre pair (et ce à chaque itération).

Il faudrait alors rajouter dans l'invariant F une condition stipulant que y est une puissance de 2, ainsi on aura à chaque itération, y div 2 = y/2 et donc xy = 2x(y div 2).

L'invariant F devient alors : (xy = ab & y ≥ 1 & (∃k∈ N, y = 2^k))

L'implication (c4) reste vraie, car on a fait que rajouter une condition supplémentaire dans l'antécédent.

L'implication (c8) devient :

(xy = ab & y ≥ 1 & (∃k∈N, y = 2^k) & y > 1) ⇒ (2x(y div 2) = ab & (y div 2) ≥ 1 & (∃k∈N, y div 2 = 2^k))

Cette implication est maintenant vraie car tous les membres du conséquent de l'implication sont déductibles à partir de l'antécédent.

La plus faible précondition (E : [a / x] [b / y] F) est donc: **(b ≥ 1 & (∃k∈ N, b = 2^k))**

C'est-a-dire que **b doit être une puissance de 2**

5) Comment montrer qu'un programme fonctionnel P calcule partiellement une fonction f en utilisant le théorème du point fixe ?

Chaque programme fonctionnel (récursif) peut être écrit sous forme d'une équation à point fixe g = τ(g). Toutes les solutions (les points fixes de l'équation) sont des fonctions partiellement calculées par le programme. Donc pour montrer qu'un programme P calcule partiellement une fonction f, il suffit de vérifier si elle est solution de l'équation à point fixe associée à P, c-a-d vérifier si f = τ(f)

6) Montrez par la méthode du point fixe, que le programme fonctionnel ci-dessous calcule exactement le terme fib(n) de la suite de Fibonacci :

P(n) = [λn. SI (< n 2) 1 SINON (+ (P (- n 1)) (P (- n 2)))]

Pour cela nous devons calculer le plus petit point fixe de l'équation associé à P (c-a-d l'équation : P = τ(P)) et vérifier alors s'il s'agit bien de la fonction fib(n).

Le plus petit point fixe est la borne supérieure de la chaîne : $(\tau^n(\Omega))_{n \geq 0}$ (théorème du point fixe)

Dans notre cas :

τ représente l'expression : [λf. λn. SI (< n 2) 1 SINON (+ (f (- n 1)) (f (- n 2)))]

Ω représente la fonction nul part définie : Ω(n) est indéfinie pour tout n

Commençons par calculer $\tau^1(\Omega)$: (c-a-d τ appliquée à Ω)

$\tau^1(\Omega)$ = [λn. SI (< n 2) 1 SINON (+ (Ω (- n 1)) (Ω (- n 2)))] = [λn. SI (< n 2) 1]

La fonction $\tau^1(\Omega)$ n'est définie que pour n < 2 et elle vaut 1, c-a-d :

$\tau^1(\Omega)(0) = \tau^1(\Omega)(1) = 1$ et $\tau^1(\Omega)(n)$ indéfinie pour tout n >1

Calculons $\tau^2(\Omega)$: (c-a-d $\tau^1(\tau^1(\Omega))$)

$\tau^2(\Omega)$ = [λn. SI (< n 2) 1 SINON (+ ($\tau^1(\Omega)$ (- n 1)) ($\tau^1(\Omega)$ (- n 2)))]

comme le terme ($\tau^1(\Omega)$ (- n 1)) n'est défini que pour n-1 < 2 (c-a-d n<3) et le terme ($\tau^1(\Omega)$ (- n 2)) n'est défini

que pour n-2 < 2 (c-a-d n<4), la partie SINON (l'expression : (+ ($\tau^1(\Omega)$ (- n 1)) ($\tau^1(\Omega)$ (- n 2)))) ne sera définie que pour n<3 et aura pour valeur 2 (+ 1 1).
Donc $\tau^2(\Omega)(0) = \tau^2(\Omega)(1) = 1$ et $\tau^2(\Omega)(2) = 2$ et $\tau^2(\Omega)(n)$ indéfinie pour tout n > 2
Calculons $\tau^3(\Omega)$: (c-a-d $\tau^1(\tau^2(\Omega))$)
$\tau^3(\Omega)$ = [λn. SI (< n 2) 1 SINON (+ ($\tau^2(\Omega)$ (- n 1)) ($\tau^2(\Omega)$ (- n 2)))]
comme le terme ($\tau^2(\Omega)$ (- n 1)) n'est défini que pour n-1 < 3 (c-a-d n<4) et le terme ($\tau^2(\Omega)$ (- n 2)) n'est défini que pour n-2 < 3 (c-a-d n<5), la partie SINON ne sera définie que pour n<4 et aura pour valeur :
(+ ($\tau^2(\Omega)$ (- n 1)) ($\tau^2(\Omega)$ (- n 2))) avec n = 2 ou n = 3.
Dans le cas où n=2 la valeur du SINON sera 2 (+ 1 1) et dans le cas où n=3 la valeur du SINON sera 3 (+ 2 1)
Donc $\tau^3(\Omega)(0) = \tau^3(\Omega)(1) = 1$, $\tau^3(\Omega)(2) = 2$, $\tau^3(\Omega)(3) = 3$ et $\tau^3(\Omega)(n)$ indéfinie pour tout n > 3

Prenons comme hypothèse de récurrence que :
$\tau^k(\Omega)(n)$ = fib(n) pour tout n entre 0 et k et $\tau^k(\Omega)(n)$ indéfinie ailleurs.
Puis montrons que :
$\tau^{k+1}(\Omega)(n)$ = fib(n) pour tout n entre 0 et k+1 et $\tau^{k+1}(\Omega)(n)$ indéfinie ailleurs.
$\tau^{k+1}(\Omega) = \tau^1(\tau^k(\Omega))$ = [λn. SI (< n 2) 1 SINON (+ ($\tau^\kappa(\Omega)$ (- n 1)) ($\tau^\kappa(\Omega)$ (- n 2)))]
Le terme ($\tau^\kappa(\Omega)$ (- n 1)) n'est défini que pour n-1 < k+1 (c-a-d n < k+2) et le terme ($\tau^\kappa(\Omega)$ (- n 2)) n'est défini que pour n-2 < k+1 (c-a-d n < k+3).
La partie SINON ne sera alors définie que pour n<k+2 et d'après l'hypothèse de récurrence, aura pour valeur (fib(n-1) + fib(n-2)) (c-a-d fib(n))
Donc $\tau^{k+1}(\Omega)(n)$ = fib(n) pour tout n entre 0 et k+1 et $\tau^{k+1}(\Omega)(n)$ indéfinie ailleurs.
L'hypothèse de récurrence est donc correcte et le plus petit point fixe $\tau^{+\infty}(\Omega)$ est bien égal à la fonction fib(n) pour tout n.

7) Donnez un programme purement logique définissant un prédicat 'pair(X)' évalué à vrai si X représente un entier naturel pair.
On représentera les entiers naturels, en programmation logique, par l'équation à point fixe suivante :
Entier = {zéro OU s(Entier)}
ainsi l'entier 3 sera représenté par le terme structuré : s(s(s(zéro)))

```
pair( zéro ) .
pair( s(s(X)) ) :-  pair( X ) .
```