Corrigé de l'examen semestriel – TPGO – 2CS – ESI 2015/2016

1- Questions de cours
a) Peut-on avoir un problème de décision classé en même temps dans P et dans NP ?
Oui, car P est inclut ou égal à NP, donc tout problème dans P est aussi dans NP.

b) Comment peut-on étudier théoriquement la difficulté d'un problème général (c-a-d qui n'est pas forcément un problème de décision) ?
Dans certains cas c'est possible, par exemple :
Soit A le problème général dont on cherche à étudier la difficulté et soit B un problème de décision déjà classé NP-complet.
Si on arrive à exprimer une solution polynomiale de B en utilisant une solution de A (comme un sous-programme de B) considérée (par hypothèse) comme une action élémentaire, alors le problème général A sera au moins aussi difficile que B. Ce type de transformation est appelé réduction de Turing Polynomiale.
Car si un jour quelqu'un trouve une solution efficace (polynomiale) pour A, il en serait de même pour B et donc pour tout problème de décision de NP. Le problème A aurait alors la même propriété que les problèmes de décision NP-Complets (ce type de problèmes généraux sont dit NP-difficiles).

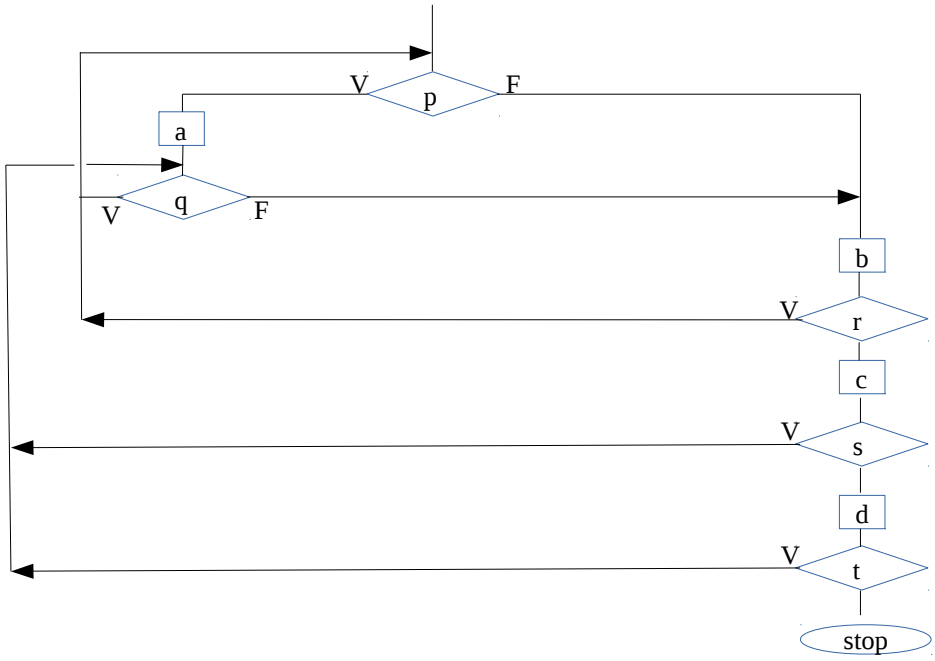
c) En λ-calcul, peut-on avoir des réductions infinies si on adopte uniquement l'ordre normal ?
Oui c'est possible d'avoir des réductions infinies en utilisant l'ordre normal, car il existe en lambda-calcul des expressions qui ne possèdent pas de forme normale.
Par exemple l'évaluation de l'expression (λx. x x y) (λx. x x y) ne termine jamais.

d) Dans la résolution descendante (celle utilisée dans Prolog), la partie hypothèse du programme logique n'est formée que par des clauses à un seul littéral positif. Quel est l'intérêt de cette contrainte ?
Le principal avantage à cela est la réduction de l'espace de recherche exploré par la résolution descendante. Puisque à chaque étape d'inférence, on ne considère que les alternatives où l'une des prémisses est issue de la dénégation.

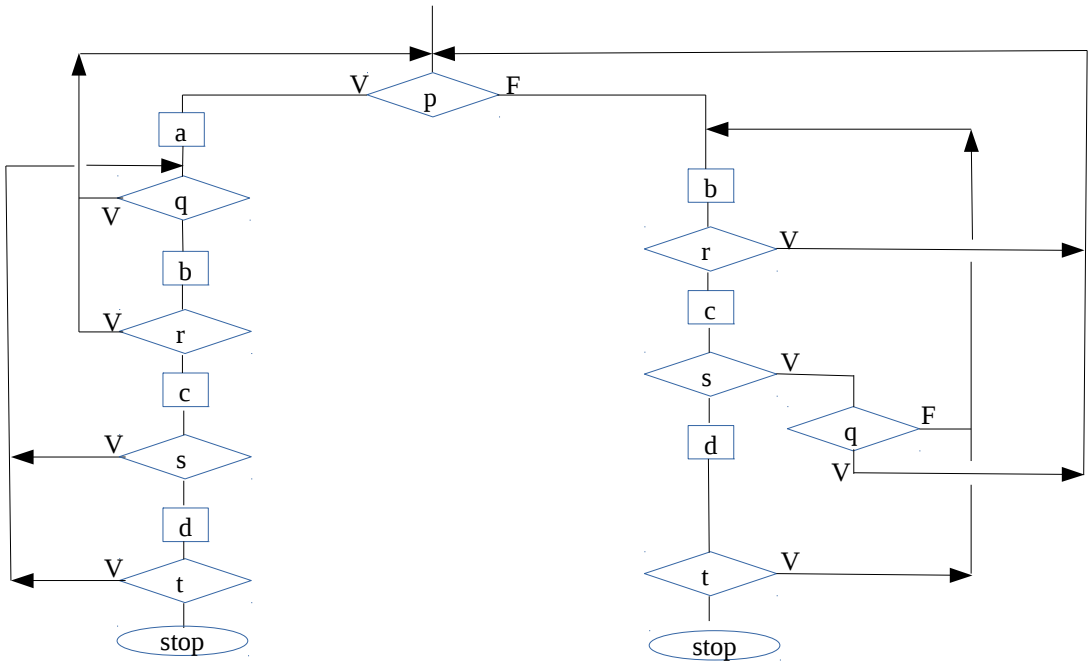
2- En utilisant le théorème de Bohm et Jacopini, donnez un programme structuré (D-algorithme), fonctionnellement équivalent au programme ci-dessous :

```
et1 :   SI p Aller à et3 ;   a ;
et2 :   SI q Aller à et1 ;
et3 :   b ;  SI r Aller à et1 ;   c ;   SI s Aller à et2 ;   d ;   SI t Aller à et2 ;
        stop
```

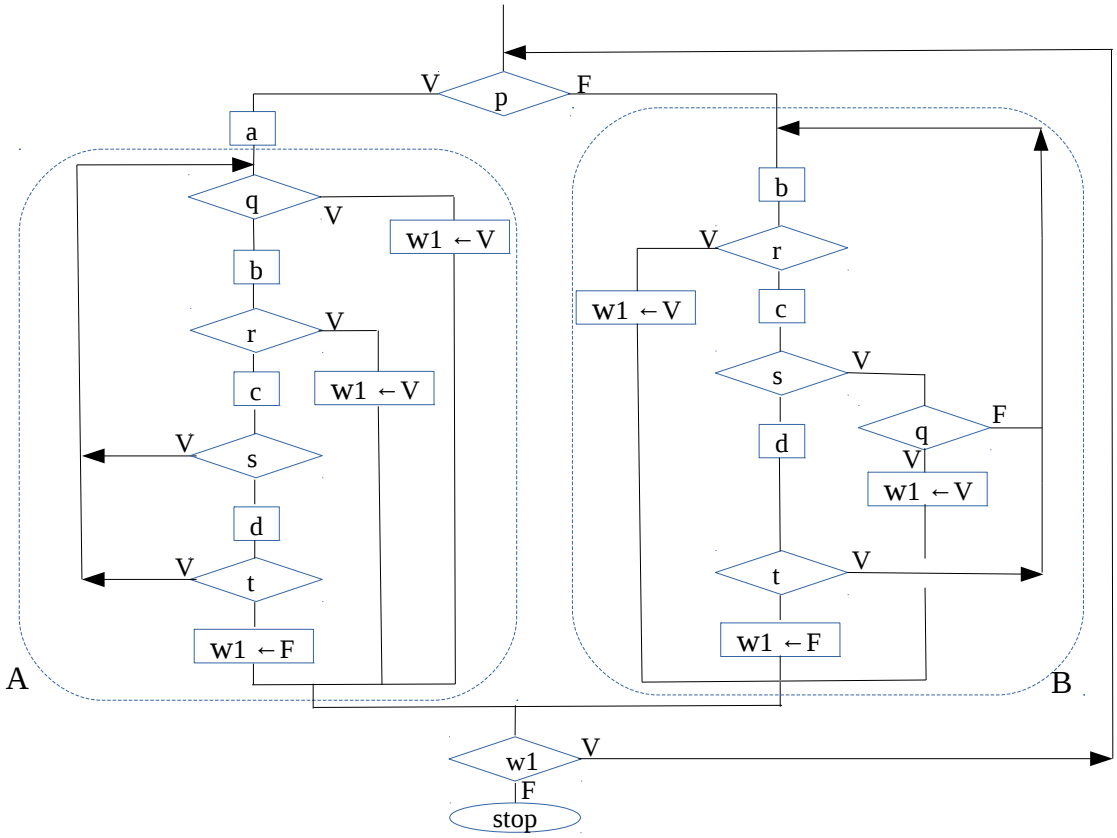
L'organigramme initial est comme suit :



C'est un type 3, on duplique les parties communes de parts et d'autres du test 'p' pour le transformer en type 2 :



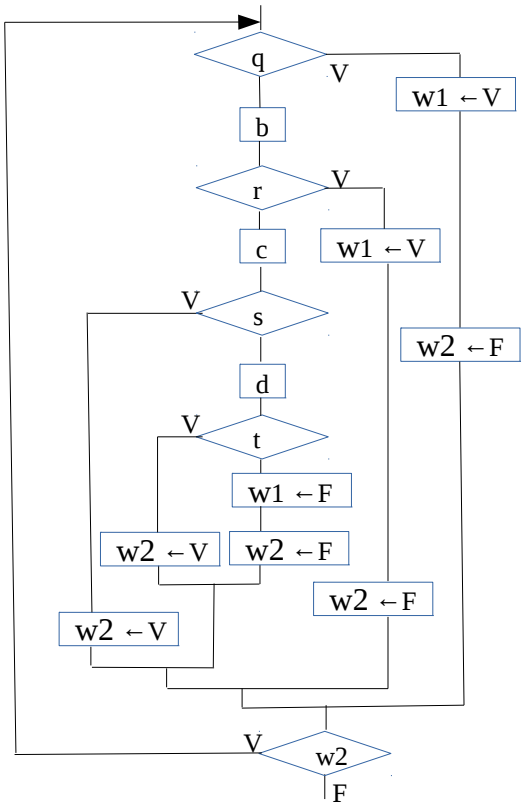
Après l'application de la transformation associée aux organigrammes de type 2, on obtient :



Il faut maintenant structurer les blocs A (qui commence par le test 'q') et B (qui commence par l'action 'b').

Le bloc A est de type 2, il sera transformé en A' comme suit :

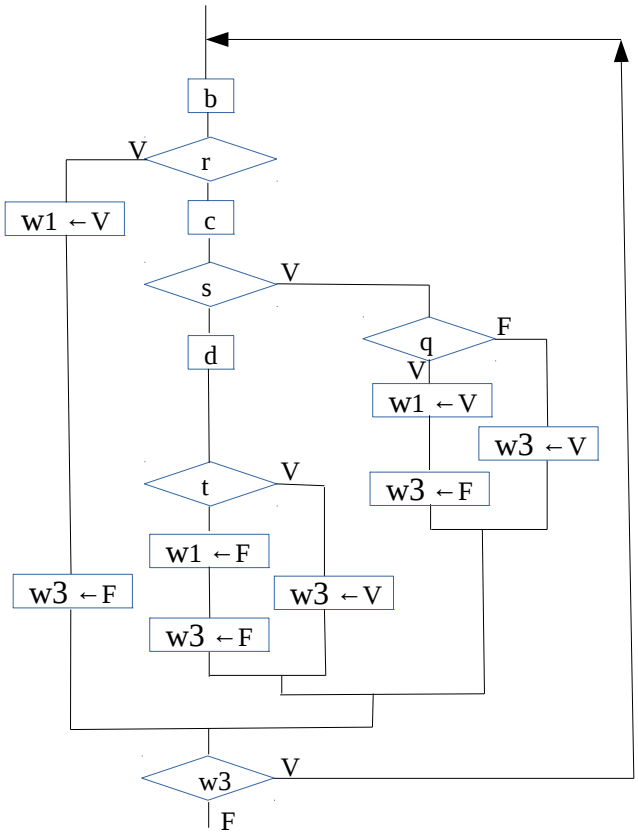
A'



Le résultat de cette transformation (A') est un bloc structuré.

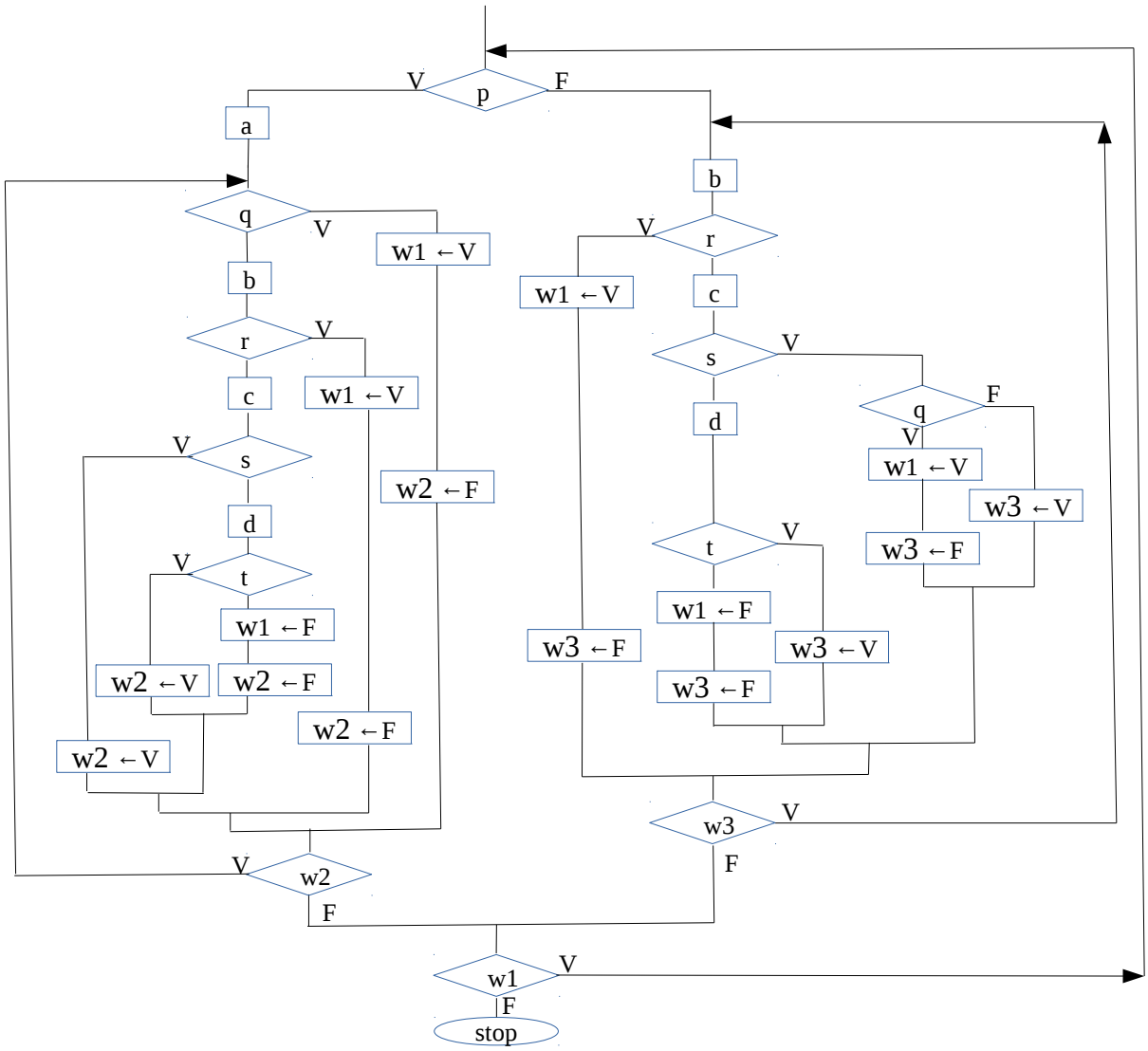
De même le bloc B est de type 1, il se transforme en B' comme suit :

B'



Le résultat de cette transformation (B') est un bloc structuré.

Le résultat final est donc :



3- Soit F(n,r) une procédure récursive et soit P son corps :

F(n : entier ; var r:entier) // n une entrée et r une sortie
SI (n = 0 OU n = 1) r ← 1
SINON SI (pair(n)) F(n/2 , r) SINON F(3*n+1, r) FSI
FSI

Montrez à l'aide du système formel de Hoare, que : (n ≥ 0) {P} (r = 1)

L'énoncé e1 :'(n ≥ 0) {P} (r = 1)' est vrai par 'CND2' si les 2 prémisses suivantes sont vraies aussi :
e2 : '(n ≥ 0 & (n=0 ou n=1)) { r ← 1} (r = 1)'
e3 : '(n ≥ 0 & (n<>0 & n<>1)) { SI (pair(n)) F(n/2 , r) SINON F(3*n+1, r) FSI} (r = 1)'

e2 est vrai par 'IMP1' car ses 2 prémisses sont vraies :
e4 : '(n ≥ 0 & (n=0 ou n=1)) => (1=1)' (l'implication est vraie car le conséquent est toujours vrai)
e5 : '(1=1) { r ← 1} (r = 1)' (axiome AFF)

- preuve de e3 : en utilisant 'CND2' il faudrait alors montrer les 2 énoncés suivant :
e5 : '(n ≥ 0 & (n<>0 & n<>1) & pair(n)) { F(n/2 , r) } (r = 1)'
e6 : '(n ≥ 0 & (n<>0 & n<>1) & impair(n)) { F(3*n+1 , r) } (r = 1)'

e5 est vrai par 'IMP1' car :
e7 : '(n ≥ 0 & (n<>0 & n<>1) & pair(n)) => (n/2 ≥ 0)' (l'antécédent n ≥ 0 suffit pour rendre l'implication vraie)
e8 : '(n/2 ≥ 0{ F(n/2 , r) } (r = 1)' (vrai par hypothèse sur les appels internes)

e6 est vrai par 'IMP1' car :
e9 : '(n ≥ 0 & (n<>0 & n<>1) & impair(n)) => ((3n+1) ≥ 0)' (l'antécédent n ≥ 0 suffit pour rendre l'implication vraie)
e10 : '((3n+1) ≥ 0{ F(3*n+1 , r) } (r = 1)' (vrai par hypothèse sur les appels internes)

L'énoncé e1 :'(n ≥ 0) {P} (r = 1)' est donc vrai.

4- Un ensemble E peut être représenté en Prolog par une liste de ses éléments (l'ordre des éléments dans la liste n'a pas d'importance).
- Définir un prédicat SousEns(E1, E2), évalué à vrai si E1 est un sous-ensemble de E2.

```
/* Prédicat pour vérifier l'appartenance d'un élément à une liste */
app( X, [X|_] ).
app( X, [_|L] ) :- app( X, L ).

/* Prédicat pour vérifier si le 1er argument est un sous-ensemble du 2e argument */
sousens( [], _ ).
sousens( [X|L], E ) :- app( X, E ), supp( X, E, E1 ), sousens( L, E1 ).

/* Prédicat pour la suppression d'un élément dans un ensemble :
   le 3e argument est le résultat de la suppression du 1er argument depuis l'ensemble représenté par le 2e argument */
supp( X, [], [] ).
supp( X, [X|L], L ).
supp( X, [Y|L], [Y|L1] ):- X =\= Y, supp( X, L, L1 ).
```

- Comment générer l'ensemble des parties d'un ensemble donné E ?

Il suffit de mentionner un but (sousens) contenant une variable en 1er argument : sousens(X, [….])

5- En utilisant la théorème du point fixe, établir la fonction exactement calculée par le programme fonctionnel suivant :
f = λn. SI (n = 0) Alors 0 Sinon SI (pair(n)) (f(n-2)*(n-1) + 2*n-1) / (n+1) FSI FSI

Posons $\tau(f)$ = λn. [SI (n = 0) 0 [SI (pair(n)) (f(n-2)*(n-1) + 2*n-1) / (n+1)]]

Pour trouver la fonction exactement calculée par le programme, on calcule le plus petit point fixe f_0 de l'équation :
$f = \tau(f)$
D'après le théorème du point fixe, on sait que $f_0 = Sup\ (\tau^n(\Omega))_{n \geq 0}$
Calculons $\tau^1(\Omega)$:
$\tau^1(\Omega)$ = λn. [SI (n = 0) 0 [SI (pair(n)) (Ω(n-2)*(n-1) + 2*n-1) / (n+1)]]
= λn. [SI (n = 0) 0]

Calculons $\tau^2(\Omega)$:
$\tau^2(\Omega)$ = λn. [SI (n = 0) 0 [SI (pair(n)) ($\tau^1(\Omega)$(n-2)*(n-1) + 2*n-1) / (n+1)]]
= λn. [SI (n = 0) 0 [SI (pair(n) & n = 2) (0*1 + 2*2 – 1) / 3]]
= λn. [SI (n = 0) 0 [SI (n = 2) 1]]

Calculons $\tau^3(\Omega)$:
$\tau^3(\Omega)$ = λn. [SI (n = 0) 0 [SI (pair(n)) ($\tau^2(\Omega)$(n-2)*(n-1) + 2*n-1) / (n+1)]]
= λn. [SI (n = 0) 0 [SI (pair(n) & n = 2) (0*1 + 2*2 – 1) / 3 [SI (pair(n) & n = 4) (1*3 + 2*4 – 1)/5]]]
= λn. [SI (n = 0) 0 [SI (n = 2) 1 [SI (n = 4) 2]]]

Montrons par récurrence sur k, que $\tau^k(\Omega)$ = λn. [SI (pair(n) & n ≤ 2(k-1)) n/2]

Calculons $\tau^{k+1}(\Omega)$:
$\tau^{k+1}(\Omega)$ = λn. [SI (n = 0) 0 [SI (pair(n)) ($\tau^k(\Omega)$(n-2)*(n-1) + 2*n-1) / (n+1)]]
= λn. [SI (n = 0) 0 [SI (pair(n) & (n-2) ≤ 2(k-1)) ((n-2)*(n-1)/2 + 2*n - 1) / (n+1)]]
= λn. [SI (n = 0) 0 [SI (pair(n) & n ≤ 2k) n/2]] (l'hypothèse de récurrence est donc vraie)

Le plus petit point fixe $f_0 = \tau^{+\infty}(\Omega)$ = **λn. [SI (pair(n)) n/2]**