

1 Exercice de logique : logique propositionnelle tri-valuée.

1. $P_1 = \neg(\neg x \wedge \neg y)$.

Avec x et y qui valent I et V : $x \vee y$ vaut V et P_1 vaut $\neg I \wedge F = \neg F = V$.

Avec x et y qui valent I et F : $x \vee y$ vaut I et P_1 vaut $\neg I \wedge V = \neg I = I$.

Avec x et y qui valent I : $x \vee y$ vaut I et P_1 vaut $\neg I \wedge I = \neg I = I$.

P_1 et $x \vee y$ sont donc aussi équivalents dans $\mathcal{L}_3$.

2. Dans $\mathcal{L}_2$, on prend $P_2 = \neg x \vee y$.

Avec $x = I$ et $y = I$: $x \Rightarrow y$ vaut V et P_2 vaut $I \vee I = I$.

Et ce n'est plus équivalent dans $\mathcal{L}_3$.

3.

x	y	$x \Rightarrow y$	$(\neg x \Rightarrow y)$	$(x \Rightarrow y) \wedge ((\neg x \Rightarrow y))$	P_3
V	V	V	V	V	V
V	F	F	V	F	V
V	I	I	V	I	V
F	V	V	V	V	V
F	F	V	F	F	V
F	I	V	I	I	V
I	V	V	V	V	V
I	F	I	I	I	I
I	I	V	V	V	I

Ce n'est pas une tautologie (si on définit bien une tautologie comme toujours vraie, et pas jamais fausse !)

4. On note l'équivalence logique par $\equiv$. $(x \vee y) \Rightarrow y \equiv (\neg(x \vee y) \vee z) \equiv (\neg x \wedge \neg y) \vee z$ et ceci est une forme normale disjonctive.

On prend donc $P_4 = (\neg x \wedge \neg y) \vee z$.

Avec $x = y = z = I$: $(x \vee y) \Rightarrow z$ vaut V tandis que P_4 vaut I : elles ne sont pas équivalentes dans $\mathcal{L}_3$.

5. Pour $x = V$ et $y = I$: $x \Rightarrow y$ vaut I et $\neg y \Rightarrow \neg x$ vaut I .

Pour $x = F$ et $y = I$: $x \Rightarrow y$ vaut V et $\neg y \Rightarrow \neg x$ vaut V .

Pour $x = I$ et $y = I$: $x \Rightarrow y$ vaut V et $\neg y \Rightarrow \neg x$ vaut V .

Pour $x = I$ et $y = V$: $x \Rightarrow y$ vaut V et $\neg y \Rightarrow \neg x$ vaut V .

Pour $x = I$ et $y = F$: $x \Rightarrow y$ vaut I et $\neg y \Rightarrow \neg x$ vaut I .

Donc la contraposition est encore vraie dans $\mathcal{L}_3$.

6. $\neg(\neg x) \equiv x$ impose $\neg I = I$.

Il suffit de déterminer $I \wedge x$ pour tout x pour connaître entièrement $\wedge$ par commutativité.

$I \wedge F$, par croissance, doit être entre $F \wedge F = F$ et $V \wedge F = F$ donc $I \wedge F = F$.

Si $I \wedge V = F$ alors $I \wedge I = F$ par croissance et on en déduit les tables de $\vee$ et $\Rightarrow$ par $x \vee y \equiv P_1$ et $x \Rightarrow y \equiv (\neg x \vee y)$. D'où la première possibilité.

Si $I \wedge V = I$, on peut avoir $I \wedge I = I$ ou $I \wedge I = F$ en respectant la croissance, d'où deux nouvelles possibilités.

Si $I \wedge V = V$, $I \wedge I$ peut être F , I ou V , d'où trois possibilités.

On a en tout six logiques possibles.

2 Algorithmique : coloration d'un graphe.

2.4 Détermination des voisins des sommets.

1.

```
let rec insere L s = match L with
  | [] -> [s]
  | t::q when s < t -> s::L
  | t::q when s = t -> L
  | t::q -> t::(insere q s) ;;
```
2. Le pire des cas se produit lorsque s majore la liste L . Dans ce cas, on effectue $O(|L|)$ opérations $::$ (où $|L|$ désigne la longueur de L).
3. On écrit une fonction auxiliaire qui stocke dans un accumulateur la deuxième extrémité de chacune des arêtes dont une extrémité est s .

```
let voisins G s =
  let rec aux accu = function
    | [] -> accu
    | t::q when t.a = s -> aux (insere accu t.b) q
    | t::q when t.b = s -> aux (insere accu t.a) q
    | _::q -> aux accu q
  in aux [] G.A ;;
```

2.5 Un algorithme de bonne coloration d'un graphe.

4. 0 est coloré 1, 1 est coloré 1, 2 est coloré 2, 3 est coloré 2, 4 est coloré 3, 5 est coloré 3.
Mais on peut colorier avec tous les sommets pairs de la couleur 1 et tous les sommets impairs de la couleur 2.
5. On écrit une fonction auxiliaire `voisins_plus_petits` qui extrait de la liste de tous les voisins ceux qui sont de numéro inférieur strictement à i .

La deuxième fonction auxiliaire `premier_choix` est telle que `premier_choix 1 q` renvoie le premier numéro de couleur qui ne fait pas partie de la liste q .

```
let rec voisins_plus_petits i = function
  | t::q when t < i -> t::(voisins_plus_petits i q)
  | _ -> [] ;;

let rec premier_choix i = function
  | q when mem i q -> premier_choix (i+1) q
  | _ -> i;;
```

```

let coloration G = let couleurs = make_vect G.n 0 in
  for i=0 to G.n - 1 do
    let vois = voisins G i in
    let v = voisins_plus_petits i vois in
    let c = map (fun x -> couleurs.(x)) v in
    couleurs.(i) <- premier_choix 1 c
  done ;
couleurs ;;

```

Dans la fonction `coloration`, `c` est construit, à partir de la liste des voisins de `i` de numéro strictement inférieur, en remplaçant chacun des voisins par son numéro de couleur, de façon à choisir la première couleur disponible pour l'attribuer à `i`.

2.6 Définition du nombre chromatique de G .

6. L'ensemble $E_c(G)$ est non vide : il existe en effet une bonne coloration à $n(G)$ couleurs en donnant au sommet i la couleur $i + 1$.

Notons $\text{NBC}(G) = \min E_c(G)$. Il reste à montrer que $E_c(G) = \{p \mid p \geq \text{NBC}(G)\}$.

Soit p tel que $\text{NBC}(G) + 1 \leq p \leq n(G) - 1$. Partons d'une coloration de G avec $\text{NBC}(G)$ couleurs ; choisissons $p - \text{NBC}(G)$ sommets, et colorions-les avec les couleurs $\text{NBC}(G) + 1, \dots, p$, et on a une bonne coloration à p couleurs, donc p appartient à $E_c(G)$.

Pour $p > n(G)$, les couleurs supplémentaires n'apparaissent pas explicitement.

7. $\text{NBC}(G) = 1$ puisqu'on peut donner la même couleur à tous les sommets.

$f_c(G, p)$ est le nombre de façon d'affecter arbitrairement une des couleurs à chacun des sommets, donc le nombre d'applications de $[1, n(G)]$ dans $[1, p]$ soit $p^{n(G)}$.

8. $\text{NBC}(G) = n(G)$ car les couleurs des sommets doivent être deux à deux distinctes et pour $p < n(G)$, $f_c(G, p) = 0$.

Pour $p \geq n(G)$: $f_c(G, p) = p(p-1) \dots (p - n(G) + 1) = \frac{p!}{(p - n(G))!}$ car on choisit la couleur

du premier sommet (p choix) puis celle du deuxième parmi les $p - 1$ couleurs restantes ...

La première formule est la mieux adaptée à la suite du problème, et reste vraie pour $p < n(G)$.

9. $\text{NBC}(G_{ex1}) = 3$ car 0, 3 et 4 doivent avoir trois couleurs différentes, et 1 et 2 peuvent être de la couleur de 0.

Pour $p < 3$: $f_c(G, p) = 0$.

Pour $p \geq 3$: 3 et 2 peuvent être de n'importe quelle couleur : p choix chacun. 0 et 1 doivent être d'une couleur différente de celle de 3 : $p - 1$ choix. 4 est de toute couleur sauf celles de 0 et 3 : $p - 2$ choix.

Ainsi : $f_c(G, p) = p^2(p-1)^2(p-2)$ pour $p \geq 3$ et cette formule reste valable pour $p < 3$.

2.7 Les applications H et K .

10. `let prem_voisin G s = hd (voisins G s) ;;`
 puisque `voisins G s` est rangé par ordre croissant.
11. `let prem_ni G =`
`let rec non_isole s = if (voisins G s = []) then non_isole (s+1) else s`
`in non_isole 0 ;;`
12. On écrit une fonction auxiliaire `filtre` qui prend un prédicat sous forme d'une fonction `p : 'a -> bool` et une liste, et renvoie la sous-liste constituée des éléments qui ne vérifient pas le prédicat. On l'utilise ensuite dans H pour éliminer les arêtes entre s_1 et s_2 (avec `p = f`).

```
let rec filtre p = function
| [] -> []
| t::q when p t -> filtre p q
| t::q -> t :: (filtre p q) ;;
```

```
let H G = let s1 = prem_ni G in let s2 = prem_voisin G s1 in
  let f {a=x ; b = y} = (x=s1 && y=s2) || (x=s2 && y=s1) in
  let Aprime = filtre f G.A in {n = G.n ; A = Aprime} ;;
```

13. La fonction auxiliaire `g` renumérote correctement les états.

```
let g s1 s2 = function
  | s when s < s2 -> s
  | s when s = s2 -> s1 ;
  | s -> s-1 ;;
```

```
let K G = let s1 = prem_ni G in let s2 = prem_voisin G s1 in
  let f {a=x;b=y} = {a = g s1 s2 x ; b = g s1 s2 y} in
  let Gprime = H G in {n = G.n - 1 ; A = map f Gprime.A} ;;
```

2.8 Fonction $f_c(G, p)$ et polynôme chromatique.

14. Toute bonne coloration de G est une bonne coloration de $H(G)$ puisqu'on ne fait que supprimer des arêtes.
15. $BC(H(G), p) \setminus BC(G, p)$ est l'ensemble des colorations de $H(G)$ qui ne sont pas des colorations de G , i.e. des colorations telles que s_1 et s_2 aient même couleur.
 À toute bonne coloration de $H(G)$ donnant les mêmes couleurs à s_1 et s_2 , on associe une bonne coloration de $K(G)$ puisque les autres arêtes ne changent pas, et réciproquement.
 Ainsi les cardinaux de $BC(K(G), p)$ et de $BC(H(G), p) \setminus BC(G, p)$ sont égaux.
16. Passage aux cardinaux dans l'égalité précédente.
17. Pour un graphe G sans arêtes : question 7.
 Sinon, on applique la formule précédente.

18. La terminaison est assurée par le fait que $H(G)$ et $K(G)$ ont tous deux strictement moins d'arêtes que G .

19. Pour un graphe G sans arêtes : $f_c(G, p) = p^{n(G)}$.

On travaille par induction avec la formule précédente : $f_c(H(G), p)$ est un polynôme de degré $n(H(G)) = n(G)$ et $f_c(K(G), p)$ est un polynôme de degré $n(K(G)) = n(G) - 1$ donc $f_c(G, p)$ est un polynôme de degré $n(G)$.

2.9 Calcul du polynôme $P_c(G, p)$ et de $\text{nbc}(G)$.

```
20. let rec difference P Q =
    let dp = vect_length P - 1 and dq = vect_length Q - 1 in
    let dr = max dp dq in let R = make_vect dr 0 in
    for i = 0 to dp-1 do R.(i) <- P.(i) done;
    for j = 0 to dq-1 do R.(j) <- R.(j) - Q.(j) done;
    R ;;
```

Remarque : la fonction ci-dessus est un peu compliquée car il faut tenir compte du cas où Q serait de degré plus grand que P . Pour le problème que l'on est en train de résoudre par contre, on travaillera toujours avec $\deg P > \deg Q$ et on pourrait utiliser une version plus simple de `difference` :

```
let difference P Q =
    let dq = vect_length Q - 1 in
    let R = copy_vect P in
    for i = 0 to dq do
        R.(i) <- R.(i) - Q.(i)
    done ;
    R ;;
```

21. La fonction `simple` initialise le cas d'un graphe sans arête.

```
let simple G = let n = G.n in let p = make_vect (n+1) 0 in
    p.(n) <- 1 ;
p;;

let rec Pc G =
    if G.A = [] then simple G else difference (Pc (H G)) (Pc (K G)) ;;
```

22. On applique l'algorithme de Horner :

```
let eval P x = let d = vect_length P - 1 in
    let rec aux y = function
        | (-1) -> y
        | i -> aux (P.(i) + x * y) (i-1)
    in aux 0 d ;;
```

23. $\text{nbc}(G)$ est le premier entier p pour lequel $f_c(G, p)$ est non nul.

```
let nbc G =  
  let rec aux P p = if eval P p = 0 then aux P (p+1) else p in  
  aux (Pc G) 1 ;;
```