INFORMATIQUE

Note : Toutes les réponses doivent être justifiées.

Note : Vous indiquerez, au début de votre copie, si vous choisissez de rédiger vos programmes en PASCAL ou en CAML.

Partie I - Algorithmique

Dans cette partie, on s'intéresse aux arbres bicolores qui sont une classe particulière des arbres binaires de recherche.

I.A - Rappels et Notations

I.A.1) Définitions

Un *arbre* est un triplet $\mathcal{A} = \langle \mathcal{N}, n_0, p \rangle$ où :

— $\mathcal{N}$ est un ensemble fini dont les éléments sont appelés *nœuds*.

— n_0 est un élément particulier de $\mathcal{N}$ appelé *racine*.

— p est une fonction, nommée *père*, de $\mathcal{N} - \{n_0\}$ dans $\mathcal{N}$ ayant la propriété suivante :

$$\forall n \in \mathcal{N} - \{n_0\}, \exists x \in \mathbb{N}^* \mid p^x(n) = n_0$$

On rappelle que p^x désigne $\underbrace{p \circ \ldots \circ p}_{x}$; p^0 désigne la fonction identité.

Un *arbre vide* est un arbre dégénéré ayant un ensemble de nœuds vide (il n'a donc pas de racine).

Dans ce qui suit, on considère un arbre non vide $\mathcal{A} = \langle \mathcal{N}, n_0, p \rangle$ et les n_α sont des éléments de $\mathcal{N}$.

— Si $p(n_1) = n_2$,on dit que n_2 est le *père* de n_1, et que n_1 est un fils de n_2.

— Si $p^x(n_1) = n_2$, on dit que n_2 est un *ascendant* de n_1, et que n_1 est un *descendant* de n_2.

— Un chemin d'un nœud n_a à un nœud n_b est une suite de nœuds $(n_1 = n_a, n_2, \ldots, n_k = n_b)$ tel que $\forall i \in [1 \ldots k-1]$, $n_i = p(n_{i+1})$.

— Si $\forall n \in \mathcal{N} - \{n_0\}, p(n) \neq n_1$, on dit que n_1 est une *feuille*. Dans le cas contraire, on dit que n_1 est un *nœud interne*.

Filière MP

— Soit $\mathcal{F}_{n_1} = \left\{ (n \in \mathcal{N} - \{n_0\}) \middle| p(n) = n_1 \right\}$. $\mathcal{F}_{n_1}$ est l'ensemble des fils de n_1. On appelle *degré* de n_1 le cardinal de $\mathcal{F}_{n_1}$ et *degré maximal* de l'arbre le plus grand des degrés des nœuds.

— Soit $\mathcal{N}_{n_1} = \left\{ n \in \mathcal{N} \middle| \ \exists x \in \mathbb{N}, p^x(n) = n_1 \right\}$.

Si p_{n_1} est la restriction de p à $\mathcal{N}_{n_1}$, alors $\langle \mathcal{N}_{n_1}, n_1, p_{n_1} \rangle$ est un arbre, dit *sous-arbre* de $\mathcal{A}$ de racine n_1.

— La *profondeur* d'un nœud n est l'entier $x \in \mathbb{N}$ tel que $p^x(n) = n_0$. La *hauteur* d'un arbre est la profondeur maximale de ses nœuds.

— Un *arbre ordonné* est un arbre auquel on adjoint à chaque nœud un ordre total sur ses fils.

— Un *arbre n-aire* est un arbre ordonné dont chaque nœud a exactement n fils (certains des fils pouvant être des arbres vides).

— Un *arbre binaire* est un arbre 2-aire. On nomme classiquement *fils droit* et *fils gauche* les deux fils de chaque nœud.

Un *arbre étiqueté* est un quintuplet $\mathcal{A} = \langle \mathcal{N}, n_0, p, \varepsilon, e \rangle$ où :

— $\mathcal{N}$, n_0 et p sont définis comme ci-dessus.

— ε est un ensemble.

— e est une fonction de $\mathcal{N}$ dans ε.

Un arbre binaire étiqueté $\mathcal{A}$ est noté $\mathcal{A} = (i, \mathcal{A}_g, \mathcal{A}_d)$. C'est l'arbre dont la racine a pour étiquette $i \in \varepsilon$, et pour sous-arbre gauche (respectivement sous-arbre droit) $\mathcal{A}_g$ (respectivement $\mathcal{A}_d$).

Un *arbre binaire de recherche* est un arbre binaire étiqueté, tel qu'il existe un ordre total sur ε, et tel que si n est un nœud, $\mathcal{A}_d$ et $\mathcal{A}_g$ ses sous-arbres droit et gauche (éventuellement des arbres vides) alors :

— $\forall n_g \in \mathcal{A}_g, e(n_g) \leq e(n)$.

— $\forall n_d \in \mathcal{A}_d, e(n_d) \geq e(n)$.

On suppose connu du candidat l'ordre de grandeur de la complexité de l'opération de recherche d'un élément dans un arbre binaire de recherche ainsi que le principe d'insertion d'un élément dans un tel arbre.

I.A.2) Représentation

— Dans toute la partie algorithmique, on considère des arbres binaires étiquetés, et ε est l'ensemble des entiers $\mathbb{Z}$.

— Langage CAML : on n'utilisera pas ici la représentation classique d'un arbre sous forme de produit, car on souhaite pouvoir modifier un nœud sans en créer un nouveau. On utilisera donc un enregistrement.

- Un arbre binaire étiqueté sera représenté ainsi :

```
Type Noeud =
   mutable etiquette : int ;
   mutable gauche : ArbreBinaire ;
   mutable droit : ArbreBinaire ;
and ArbreBinaire =
      ArbreVide
   | Pointeur of Noeud
;;
```

- Les fonctions suivantes seront utilisées :

```
let est_vide = function
      ArbreVide  ->  true
   |  _           ->  false
;;
let noeud = function
     ArbreVide      ->  failwith "arbre vide"
   | Pointeur x     ->  x
;;
```

- Voici un exemple d'utilisation de ces fonctions : la fonction échangeant les fils gauche et droit d'un arbre sera écrite :

```
let echange_gauche_droit a =
   if est_vide a then a
   else let temp = (noeud a).gauche in
         (noeud a).gauche <- (noeud a).droit;
         (noeud a).droit <- temp;
         a
;;
```

— Langage PASCAL

- Un arbre binaire étiqueté sera représenté ainsi :

```
TYPE
   ArbreBinaire = ^Noeud ;
   Noeud = RECORD
              etiquette : INTEGER
              gauche, droit : ArbreBinaire
              END ;
```

- La fonction suivante sera utilisée :

```
FUNCTION est_vide (a : ArbreBinaire) : BOOLEAN ;
BEGIN
   est_vide := (a = NIL);
END ;
```

- Voici un exemple d'utilisation de cette fonction : la fonction échangeant les fils gauche et droit d'un arbre sera écrite :

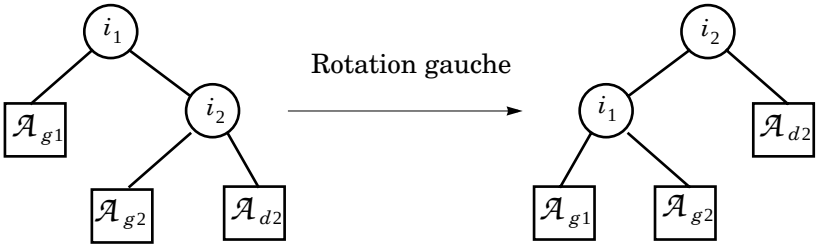
```
FUNCTION echange_gauche_droit (a : ArbreBinaire) : ArbreBinaire ;
VAR
   temp : ArbreBinaire ;
BEGIN
   IF (est_vide (a)) THEN
      echange_gauche_droit := a
   ELSE
   BEGIN
      temp := a^.gauche;
      a^.gauche := a^.droit;
      a^.droit := temp;
      echange_gauche_droit := a
   END ;
END ;
```

I.B - Rotations

Afin de pouvoir modifier la hauteur d'un arbre, on introduit, lorsqu'elles sont possibles, les opérations suivantes :

I.B.1) Rotation gauche

L'arbre obtenu en appliquant une rotation gauche à l'arbre $(i_1, \mathcal{A}_{g1}, (i_2, \mathcal{A}_{g2}, \mathcal{A}_{d2}))$ est l'arbre $(i_2, (i_1, \mathcal{A}_{g1}, \mathcal{A}_{g2}), \mathcal{A}_{d2})$.



a) Écrire la fonction *rotation_gauche*, recevant un ArbreBinaire comme seul argument, lui appliquant une rotation gauche, et retournant comme résultat l'arbre modifié. On veillera à n'appliquer la rotation que si celle-ci est possible. Dans le cas contraire, l'arbre non modifié sera alors retourné.

b) Montrer que si $\mathcal{A}$ est un arbre binaire de recherche, alors *rotation_gauche* $(\mathcal{A})$ est aussi un arbre binaire de recherche.

I.B.2) Rotation droite

La rotation droite appliquée à un arbre binaire est l'opération inverse de la rotation gauche.

Écrire la fonction *rotation_droite*, selon les mêmes principes que la fonction *rotation_gauche*. On fera un dessin au préalable.

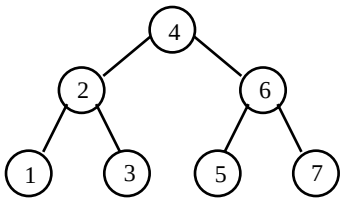
I.B.3) Doubles Rotations

Soit $\mathcal{A} = (i, \mathcal{A}_g, \mathcal{A}_d)$ un arbre binaire. La rotation gauche_droite de $\mathcal{A}$ est l'arbre :

$$rotation\_droite\ (i\ ,\ rotation\_gauche\ (\mathcal{A}_g)\ ,\ \mathcal{A}_d\ )$$

a) Écrire la fonction *rotation_gauche_droite*. Là aussi, l'opération ne sera effectuée que si elle est possible.

b) Dessiner le résultat obtenu après application de l'opération de rotation gauche-droite à l'arbre $\mathcal{A}_0$ suivant :



La rotation droite-gauche de $\mathcal{A} = (i, \mathcal{A}_g, \mathcal{A}_d)$ est, s'il est défini, l'arbre :

$$rotation\_gauche\,(i\,,\,\mathcal{A}_g\,,\,rotation\_droite(\mathcal{A}_d)\,)$$

c) Écrire la fonction *rotation_droite_gauche*.

d) Dessiner le résultat obtenu après application de l'opération de rotation droite-gauche à l'arbre $\mathcal{A}_0$ défini en b).

I.C - Arbres bicolores

Un arbre bicolore est un arbre binaire de recherche dont les nœuds sont colorés d'une couleur parmi deux selon des règles particulières énoncées ci-dessous. Si le rouge et noir sont classiquement utilisés, nous prendrons ici le blanc et noir. On choisira d'autre part de ne pas étiqueter les feuilles.

I.C.1) Nouvelle représentation

On modifie la représentation des arbres afin d'ajouter l'information de coloriage. On en profite pour ajouter dans un nœud un champ indiquant le père qui sera un arbre vide dans le cas de la racine. Les feuilles, non étiquetées, et qui ont une couleur fixe (voir ci-dessous) seront systématiquement représentées par des arbres vides. On supposera que les fonctions de rotation ont été corrigées pour tenir compte de cette nouvelle représentation.

- En CAML, un arbre binaire bicolore sera représenté ainsi :

```
type Couleur = Blanc | Noir
;;
type Noeud =
   mutable etiquette : int ;
   mutable couleur: Couleur;
   mutable gauche: ArbreBinaire;
   mutable droit: ArbreBinaire;
   mutable pere: ArbreBinaire;
and ArbreBinaire =
      ArbreVide
   | Pointeur of Noeud
;;
```

- En PASCAL, un arbre bicolore sera représenté ainsi :

```
TYPE
  TCouleur = (Blanc,Noir);
  ArbreBinaire = ^Noeud;
  Noeud = RECORD
    etiquette : INTEGER ;
    couleur : TCouleur ;
    gauche, droit, pere: ArbreBinaire
  END ;
```

Un arbre bicolore doit satisfaire aux contraintes suivantes (en plus d'être un arbre binaire de recherche) :

(A-1) Un nœud est soit blanc, soit noir.

(A-2) Une feuille est noire.

(A-3) La racine est noire.

(A-4) Le père d'un nœud blanc est noir.

(A-5) Tous les chemins partant d'un nœud donné et se terminant à une feuille contiennent le même nombre de nœuds noirs (sans prendre en compte le nœud de départ)

On nomme *hauteur noire* d'un nœud n appartenant à un arbre bicolore (on la note $hn(n)$) le nombre de nœuds noirs sur les chemins partant de ce nœud et aboutissant à une feuille (sans prendre en compte n). Cette fonction est bien définie grâce à (A-5). On nomme hauteur noire d'un arbre bicolore la hauteur noire de sa racine.

I.C.2) Hauteur d'un arbre bicolore

a) Montrer qu'un sous-arbre de racine n d'un arbre bicolore contient au moins $2^{hn(n)} - 1$ nœuds internes.

b) Montrer qu'un arbre bicolore comportant l nœuds internes a une hauteur au plus égale à $2\log_2(l+1)$.

c) Que peut-on dire de l'ordre de grandeur de la complexité de la recherche d'un élément dans un arbre bicolore ?

I.C.3) Insertion dans un arbre bicolore

Pour insérer un nouveau nœud n dans un arbre bicolore, on commence par l'insérer selon l'algorithme d'insertion dans un arbre binaire de recherche (on parlera d'insertion initiale), en lui affectant la couleur blanc. L'arbre obtenu ainsi n'est plus forcément bicolore, et il faut le modifier pour lui redonner cette pro-

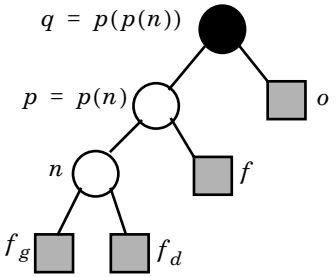
priété. L'algorithme utilisé est itératif, donc si au départ, le nœud courant n a deux fils qui sont des arbres vides, cette propriété n'est plus obligatoirement respectée aux itérations suivantes.

a) Dans le cas où l'arbre initial est vide, quelle modification faut-il apporter pour que l'arbre résultat soit de nouveau bicolore ?

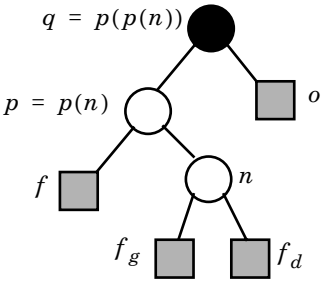
b) Dans le cas d'un arbre non vide, que peut-on dire de l'arbre résultat si le père de n après insertion initiale est noire ?

On se place maintenant dans le cas où le père de n après insertion initiale (ou après itération) est blanc. Le père de n n'est donc pas la racine, et a lui-même un père qui est noir.

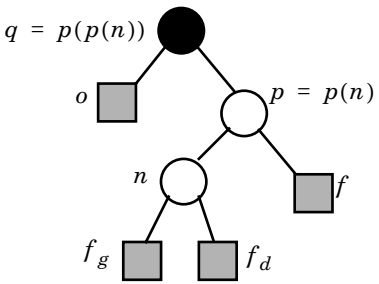
Les différents cas possibles sont donc (on note f_d et f_g les deux fils de n, p son père, q son grand père, f son frère et o son oncle) :



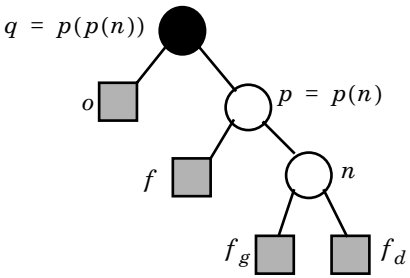
Cas n°1



Cas n°2



Cas n°3

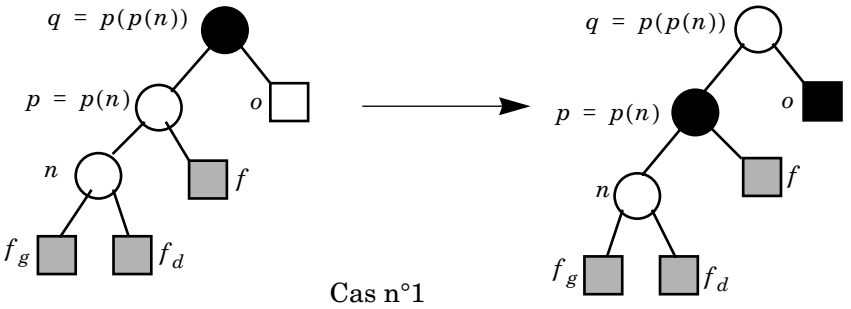


Cas n°4

On suppose qu'au départ, tous les nœuds respectent la propriété A-5 des arbres bicolores et que seuls n et p violent la condition (A-4).

c) Que peut-on dire de la couleur du frère f de n ?

On va modifier l'arbre afin de retrouver un arbre bicolore. Deux familles de modifications sont possibles, selon la couleur de o. On suppose tout d'abord que l'oncle o de n est blanc. Dans le cas n°1, on effectue la transformation suivante :



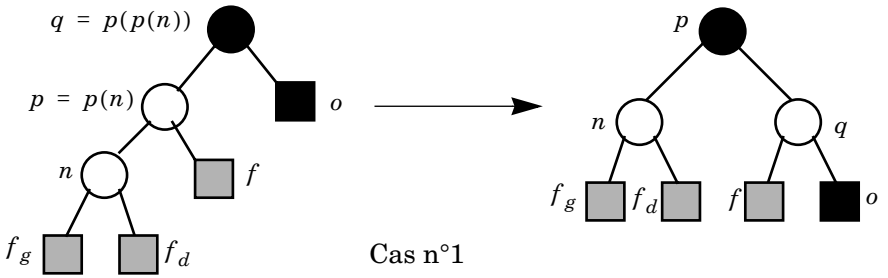
Cas n°1

d) À quelle condition simple l'arbre global obtenu est-il bicolore ?

e) Que faut-il faire si la condition n'est pas vérifiée afin de retrouver un arbre bicolore ?

f) Quelle transformation faut-il effectuer dans les cas n°2, 3 et 4 afin de retrouver un arbre bicolore, toujours en supposant que o est blanc ?

On suppose maintenant que l'oncle o de n est noir. Dans le cas n°1, on effectue la transformation suivante :



Cas n°1

g) Quelle est cette transformation ?

h) Montrer que l'arbre global obtenu est bicolore.

i) Montrer que dans le cas n°2, si on effectue la même transformation, on ne peut pas trouver une coloration des nœuds p, q, n telle que les contraintes (A-4) et (A-5) soient vérifiées.

j) Dessiner et nommer la transformation à effectuer dans le cas n°2 qui permette de conserver la hauteur noire vue du père du sous-arbre.

k) Dessiner et nommer une transformation à effectuer dans le cas n°3.

l) Dessiner et nommer la transformation à effectuer dans le cas n°4.

On suppose écrite la fonction d'insertion dans un arbre binaire de recherche. Cette fonction retourne l'arbre binaire dont la racine est colorée en blanc et est étiquetée par l'élément qui vient d'être inséré.

- En CAML, la fonction a le profil suivant :

```
inserer_arbre_binaire : ArbreBinaire -> int -> ArbreBinaire = <fun>
```

- En PASCAL, la fonction est déclarée ainsi :

```
FUNCTION inserer_arbre_binaire (a : ArbreBinaire;
                                val : INTEGER) : ArbreBinaire;
```

m) Écrire une fonction `inserer_arbre_bicolore`, ayant le même profil que la fonction `inserer_arbre_binaire`, mais réalisant l'insertion dans un arbre bicolore.

TSVP

Partie II - Logique et automates

Quand on considère un système (pris au sens large du terme), la logique permet d'exprimer les propriétés des états du système. La logique temporelle permet d'exprimer l'évolution de l'état d'un système. Cette évolution est considérée comme une suite d'états.

II.A - Définitions et notations

On considère un ensemble non vide $E = \{e_1, \ldots, e_n\}$.

Une propriété p des éléments de E est un sous ensemble de E.

On dira que $e \in E$ vérifie la propriété p quand $e \in p$.

On notera V la propriété satisfaite par tous les éléments de E, c'est à dire celle qui correspond à l'ensemble E.

On notera F la propriété satisfaite par aucun élément de E, c'est à dire celle qui correspond à l'ensemble $\varnothing$.

Soit $\mathcal{P} = \{p_1, p_2, \ldots, p_k\}$ un ensemble de propriétés de E.

L'ensemble des formules de logique temporelle basées sur $\mathcal{P}$ est définie comme suit à partir des propriétés, avec les opérateurs booléens $\vee$, $\wedge$, $\neg$ (ou, et, non) et les opérateurs temporels $\bigcirc$ (juste après), $\Box$ (désormais), $\Diamond$ (finalement), $\mathfrak{I}$ (jusqu'à).

Si p est une propriété, et si ϕ et ψ sont des formules de logique temporelle :

- p
- $(\phi \wedge \psi)$
- $(\phi \vee \psi)$
- $(\neg\phi)$
- $(\bigcirc \phi)$
- $(\phi \mathfrak{I} \psi)$
- $\Diamond(\phi)$
- $\Box(\phi)$

sont des formules de logique temporelle.

Toute formule de logique temporelle est construite à partir des règles précédentes.

Pour simplifier l'écriture des formules, on négligera les parenthèses quand il n'y a pas d'ambiguïté possible. Par exemple, on écrira $\phi \wedge \psi$ au lieu de $(\phi \wedge \psi)$, et $\Box(\phi \wedge \psi)$ au lieu de $\Box((\phi \wedge \psi))$.

On utilisera aussi le connecteur $\rightarrow$, $\phi \rightarrow \psi$ étant l'abréviation de $(\neg\phi) \vee \psi$.

Soit n un entier strictement positif et $\sigma = (\sigma_0, \sigma_1, \ldots, \sigma_{n-1})$ une suite finie d'éléments de E. Par définition, la *longueur* de la suite σ et n et est notée $|\sigma|$.

Soit σ une suite non vide, et i un entier naturel.

On définit la relation "**la suite σ satisfait la formule de logique temporelle ϕ au rang i**", ce qui sera noté $(\sigma,i) \models \phi$ par induction sur la composition des formules par les règles suivantes :

(R-1) $(\sigma,i) \models p$ si et seulement si $i < |\sigma|$ et σ_i vérifie la propriété p ;

(R-2) $(\sigma,i) \models (\phi \vee \psi)$ si et seulement si $(\sigma,i) \models \phi$ ou $(\sigma,i) \models \psi$;

(R-3) $(\sigma,i) \models (\phi \wedge \psi)$ si et seulement si $(\sigma,i) \models \phi$ et $(\sigma,i) \models \psi$;

(R-4) $(\sigma,i) \models (\neg\phi)$ si et seulement si $i < |\sigma|$ et (σ, i) ne satisfait pas ϕ ;

(R-5) $(\sigma,i) \models (\bigcirc\phi)$ si et seulement si $(\sigma,i+1) \models \phi$;

(R-6) $(\sigma,i) \models (\Box\phi)$ si et seulement si pour tout j tel que $i \le j < |\sigma|$, $(\sigma,j) \models \phi$;

(R-7) $(\sigma,i) \models (\Diamond\phi)$ si et seulement s'il existe j tel que $i \le j < |\sigma|$ et $(\sigma,j) \models \phi$;

(R-8) $(\sigma,i) \models (\phi \Im \psi)$ si et seulement s'il existe j tel que $i \le j < |\sigma|$ et $(\sigma,j) \models \psi$ et pour tout k tel que $i \le k < j$, $(\sigma,k) \models \phi$.

On notera $\sigma \models \phi$, le fait que $(\sigma,0) \models \phi$. On notera $(\sigma,i) \not\models \phi$ quand σ ne satisfait pas la formule ϕ au rang i.

Exemples :

$E = \{e_0, e_1, e_2, e_3\}$, $p = \{e_1, e_3\}$,

$\sigma = (e_1, e_1, e_3, e_3, e_2, e_3, e_3, e_0, e_2, e_2, e_0)$,

$\sigma \models p$, $(\sigma,2) \models p$, $(\sigma, 4) \not\models p$, $(\sigma, 15) \not\models p$,

$(\sigma, 23) \not\models V$, $(\sigma, 6) \models V$,

$(\sigma, 2) \models p \wedge \bigcirc p$,

$(\sigma,1) \not\models \Box \neg p$,

$(\sigma, 2) \models \Diamond \Box \neg p$.

On dit que ϕ a comme conséquence ψ, ce qu'on note $\phi \Rightarrow \psi$ si seulement si pour tout E, tout ensemble $\mathcal{P}$ de propriétés sur E, toute suite σ et tout i, si $(\sigma,i) \models \phi$, alors $(\sigma,i) \models \psi$

Exemple : $\Box\phi \Rightarrow \Diamond\phi$.

On dit que ϕ et ψ sont équivalentes si $\phi \Rightarrow \psi$ et $\psi \Rightarrow \phi$, on notera cela $\phi \Leftrightarrow \psi$.

II.B - Satisfaction de formules

II.B.1) Pour l'ensemble $E = \{e_0, e_1, e_2\}$, les propriétés $p = \{e_0, e_1\}$ et $q = \{e_0, e_2\}$, et la suite $\sigma = (e_1, e_1, e_1, e_2, e_1, e_1, e_0, e_0)$, les formules suivantes sont-elles satisfaites ?

- $(\sigma,4) \models \Box p$
- $(\sigma,2) \models \bigcirc(\Box q)$
- $(\sigma,1) \models \Diamond(\Box p)$
- $(\sigma,1) \models p \Im q$

II.B.2) Soit p une propriété. Quelle sont les suites σ vérifiant $\sigma \models \Box(\bigcirc p)$?

II.B.3) Donner une formule $Dern$ telle que $(\sigma,i) \models Dern$ si et seulement si σ_i est le dernier élément de la suite, c'est-à-dire si $i = |\sigma| - 1$. $Dern$ pourra être utilisée par la suite.

II.B.4) Soit p une propriété. Donner, avec justification préalable, une formule $Pair$, dépendant de p, telle que $\sigma \models Pair$ si et seulement si la propriété p est vérifiée pour tous les éléments σ_{2*i} tel que $0 \leq 2*i < |\sigma|$ et seulement ceux-là.

II.B.5) Les équivalences suivantes sont-elles satisfaites ?

- $\Box \Diamond \phi \Leftrightarrow \Diamond \Box \phi$
- $\Diamond \phi \Leftrightarrow \Diamond \Diamond \phi$
- $(\Box \phi) \vee (\Box \psi) \Leftrightarrow \Box(\phi \vee \psi)$
- $(\Diamond \phi) \wedge (\Box \psi) \Leftrightarrow \Diamond(\phi \wedge \Box \psi)$
- $(\Diamond \phi) \wedge (\Box \psi) \Leftrightarrow \Box (\Diamond \phi \wedge \psi)$
- $(\phi \Im \psi) \Im \psi \Leftrightarrow \phi \Im \psi$

Si oui le démontrer, si non donner un contre exemple.

II.B.6) Montrer que l'opérateur $\Diamond$ peut s'exprimer à l'aide de l'opérateur $\Im$ et de V, plus précisément, que pour toute formule ϕ où apparaît $\Diamond$, il existe une formule ψ équivalente où n'apparaît pas $\Diamond$, mais où apparaîssent $\Im$ et V.

II.B.7) Montrer que toute formule de logique temporelle est équivalente à une formule où n'apparaissent comme opérateurs temporels que $\bigcirc$ et $\Im$.

II.C - Rappels de définitions et de notations sur les automates finis.

Un automate fini non déterministe $\mathcal{A} = (E, e_0, F, A, \delta)$ est défini ainsi :

- E est l'ensemble (fini) des états ;
- e_0 est un élément distingué de E appelé état initial ;
- F est un sous ensemble de E appelé ensemble des états finals ;
- A est l'alphabet d'entrée ;
- δ est une application de $E \times A$ dans $P(E)$, appelée fonction de transition.

Soit $u = u_0 u_1 \ldots u_{n-1}$ un mot de A^* :

- Une suite d'états $s = s_0 s_1 \ldots s_n$ est compatible pour u quand $s_0 = e_0$, et $s_{i+1} \in \delta(s_i, u_i)$, pour tout i tel que $0 \leq i \leq n-1$.

- Une suite compatible est acceptante quand elle se termine par un état final.

- Un mot u est accepté par $\mathcal{A}$ s'il existe une suite acceptante pour u.

- Le langage accepté (on dit aussi reconnu) par l'automate $\mathcal{A}$ est l'ensemble des mots acceptés par $\mathcal{A}$.

Pour chaque $a \in A$, on définit la propriété $p_a = \{a\}$, c'est-à-dire que pour $a' \in A$, a' vérifie la propriété p_a si et seulement si $a' = a$. On aura donc : pour le mot $u = u_0 u_1 \ldots u_{n-1}$, $(u,i) \models p_a$ si et seulement si $i \leq n-1$ et $u_i = a$.

Dans cette partie, on verra sur des exemples les liens entre la logique temporelle et les automates finis.

On se contentera d'une justification sommaire des automates et des formules demandés.

II.C.1) Soit L le langage défini par l'expression régulière :

$$L = \{a, b, c\}^* \cdot a \cdot \{b, c\}^* \cdot bc \cdot \{b, c\}^*$$

Donner un automate non déterministe reconnaissant ce langage et donner une formule de logique temporelle ϕ basée sur $\{p_a, p_b, p_c\}$, telle que $\sigma \models \phi$ si et seulement si $\sigma \in L$.

II.C.2) Soit l'ensemble $A = \{a, b, c\}$. On pose $P = \{p_a, p_b, p_c\}$.

Pour chacun des langages L_i $(1 \leq i \leq 3)$ suivants, donner une expression rationnelle e_i décrivant L_i, puis un automate fini $\mathcal{A}_i$ reconnaissant L_i :

$$L_1 = \{\sigma \in A^* | \sigma \models (p_a \rightarrow \bigcirc p_b)\}$$

$$L_2 = \{\sigma \in A^* | \sigma \models \Box(p_a \rightarrow \bigcirc p_b)\}$$

$$L_3 = \{\sigma \in A^* | \sigma \models \Diamond(\Box(p_a \rightarrow \bigcirc p_b))\}$$

••• FIN •••