ÉCOLE POLYTECHNIQUE
Promotion 2004
Année 2005-2006

Informatique Fondamentale (INF 431)
Contrôle Classant 2
28 juin 2006

Les deux problèmes sont indépendants. On demande des réponses simples et rigoureuses. On tiendra compte de la qualité de la rédaction.
Dans la section **1.1** *on demande explicitement des solutions en Java. Dans les autres sections, il suffira de donner des algorithmes comme dans le cours.*
Les notes de cours, de PC et de DT sont autorisées ainsi que le poly et les copies de transparents des amphis.

1 Expressions booléennes

1.1 Programmes

Voici quelques exemples d'expressions booléennes, de type complètement parenthésé. Dans toutes ces expressions, les parenthèses sont obligatoires :
`((True) Or ((False) And (Not (True)))),`
`(False),`
`(((False) Or ((False) Or ((False) Or (True)))) And (True))`.
On représente une expression booléenne en utilisant une structure d'arbre dont les nœuds sont étiquetés "And", "Or" (auquels cas le nœud a 2 fils) ou "Not" (auquel cas le nœud a un seul fils) et dont les feuilles sont étiquetées par des valeurs de vérité "True" ou "False".

Question 1 : Préliminaires. Quelles sont les valeurs de vérité (Vrai ou Faux) des trois expressions ci-dessus ? Pour chacune des expressions ci-dessus, dessiner l'arbre correspondant, arbre dont la lecture infixe donne cette expression. ◇

Solution. True, False, True

Question 2 Définir une classe abstraite `Terme` qui définit un nœud de l'arbre, et des classes concrètes pour les différents types de noeuds.

Solution.
```
abstract class Terme {
}

abstract class OperateurBinaire extends Terme {
        Terme fg,fd;
        OperateurBinaire(Terme g,Terme d) {
                super();
                fg = g;
                fd = d;
```

```
        }
}

class And extends OperateurBinaire {
        And(Terme g,Terme d) {
                super(g,d);
        }
}

class Or extends OperateurBinaire {
        Or(Terme g,Terme d) {
                super(g,d);
        }
}

class Not extends Terme {
        Terme f;
}

class Valeur extends Terme {
        boolean value;
}
```

Question 3 Ecrire une méthode de la classe `Terme` qui évalue l'arbre correspondant et retourne la valeur "True" ou "False" du résultat. ◇

Solution.

```
abstract class Terme {
  // version statique
  static boolean eval(Terme t) {
  if ( t instanceof And )
    return eval(((And)t).fg) && eval(((And)t).fd);
  if ( t instanceof Or )
    return eval(((Or)t).fg) || eval(((Or)t).fd);
  if ( t instanceof Not )
    return !eval(((Not)t).f);
  if ( t instanceof Valeur )
    return ((Valeur)t).value;
  return false; // n'arrive jamais !
  }
  // version objet
  abstract boolean eval();
  // test
  public static void main(String[] args) {
    Terme t;
    t = new Or(new Valeur(true), new And(new Valeur(false), new
                                    Not(new Valeur(true))));
                          // ((True) Or ((False) And (Not (True))))
    System.out.println(eval(t));
    System.out.println(t.eval());
  }
}
abstract class OperateurBinaire extends Terme {
```

```
  Terme fg,fd;
  OperateurBinaire(Terme g,Terme d) {
    super();
    fg = g;
    fd = d;
  }
}
class And extends OperateurBinaire {
  And(Terme g,Terme d) {
    super(g,d);
  }
  // realisation version objet
  boolean eval() {
    return fg.eval() && fd.eval();
  }
}
class Or extends OperateurBinaire {
  Or(Terme g,Terme d) {
    super(g,d);
  }
  // realisation version objet
  boolean eval() {
    return fg.eval() || fd.eval();
  }
}
class Not extends Terme {
  Terme f;
  Not(Terme t) {
    f = t;
  }
  // realisation version objet
  boolean eval() {
    return !f.eval();
  }
}
class Valeur extends Terme {
  boolean value;
  Valeur(boolean b) {
    value = b;
  }
  // realisation version objet
  boolean eval() {
    return value;
  }
}
```

1.2 Analyse syntaxique

Question 4 : Expressions non-parenthésées. Voici quelques exemples d'expressions booléennes non parenthésées. Notez que dans cette question du problème, les expressions ne comportent pas de parenthèses, comme ci-dessous :

```
True And False Or Not True;
False;
False Or Not True And True Or True
```

(a) Evaluation. On rappelle les priorités d'évaluation des opérations booléennes : `Not` a la plus grande priorité, puis `And`, puis `Or`. Quelles sont les valeurs de vérité (Vrai ou Faux) des trois expressions ci-dessus ?

(b) Grammaire. Définir un alphabet et une grammaire engendrant des expressions booléennes du type ci-dessus. Dériver la première expression ci-dessus avec votre grammaire.

(c) Ambiguité. Si votre grammaire est ambiguë, le démontrer en donnant une expression et deux dérivations différentes de l'expression avec votre grammaires ; si elle ne l'est pas, le démontrer.

(d) Quels sont les inconvénients de l'ambiguïté pour une grammaire ?

Solution. (a) False, False, True
(b) Unique variable `E`, Terminaux `True, False, And, Or, Not`, règles de production `E -> True | False | E And E | E Or E | Not E`.
`E -> E And E -> True And E -> True And E Or E -> True and False Or E -> True And False Or Not E -> True And False Or Not True`
(c) `True And True And True` peut être dérivée de deux manières, selon l'ordre `(True And True) And True` ou bien `True And (True And True)`. Donc la grammaire est ambiguë.
(d) Euh... il y a plusieurs arbres de dérivation possibles, donc la personne qui a écrit le texte pouvait avoir une dérivation en tête mais en fait l'algorithme va en sortir une autre... Le sens de l'expression risque d'être différent selon l'analyse qui est faite.

Question 5 : Ajout de parenthèses. On reprend désormais les expressions booléennes de type complètement parenthésé vues dans la première partie. Définir un alphabet et une grammaire engendrant des expressions booléennes de ce type.

Pensez-vous que votre grammaire soit ambiguë ? On ne vous demande pas de preuve. ◇

Solution. Unique non-terminal `E`, variables `(, ), True, False, And, Or, Not`, règles
`E -> (True) | (False) | ( E And E ) | ( E Or E ) | ( Not E )`
Cette grammaire n'est pas ambiguë. □

Question 6 Supposons que l'expression soit dans un tableau `t` dont les entrées sont : parenthèse ouvrante, parenthèse fermante, booléen `True` ou `False`, ou opérateur `And`, `Or` ou `Not`. Donner un algorithme qui, étant donné `t` et un entier `i` tel que `t[i]='('`, calcule l'entier `j` tel que `t[j]=')'` soit la parenthèse fermante correspondant à `t[i]`. L'expression est supposée être bien formée. ◇

Solution.
```
methode correspondance(i)
  k <-- i
  compteur <-- 1
  tant que le compteur est non nul:
     k++
     si t[k]=='('
     alors    compteur++
     sinon si  t[k]==')'
                alors compteur--
  retourner k
```

Question 7 : Algorithme. En déduire un algorithme récursif qui prend une expression booléenne `t` de type complètement parenthésé et supposée correcte syntaxiquement et qui l'évalue. ◇

Solution. Retourner l'evaluation de t[0...longueur(t)], avec:

```
Evaluation de t[i..j]:// Ceci suppose que t[i]='(' et que t[j]=')' est
                      //  la parenthese fermante correspondante
  si  j==i+2
    alors retourner t[i+1]
    sinon
      si t[i+1]=='Not'
        alors retourner NOT (evaluation de t[i+2..k-1]
        sinon
          soit k la parenthese fermante correspondant a t[i+1].
          b1 <-- evaluation de t[i+1..k]
          x<-- t[k+1]
          b2 <-- evaluation de t[k+2..j-1]
          si x=='And' alors retourner (b1 AND b2) sinon retourner (b1 OU b2)
```

Question 8 : Commentaires. Pensez-vous qu'il soit plus facile de tester les expressions de type parenthésé ou non-parenthésé ? Pourquoi ? ◇

Solution. Parenthésé est beaucoup plus facile, sinon on ne sait jamais où on en est. □

1.3 Robustesse d'expressions booléennes

On reprend les arbres de la Question 2. Non seulement on souhaite avoir la valeur booléenne de l'arbre, mais de plus il peut être important de connaître la "robustesse" de cette valeur. Est-elle si sensible aux données qu'il suffit de changer la valeur de vérité de quelques-unes des variables pour affecter le résultat ? On définit la *robustesse* d'un arbre comme le nombre minimum de feuilles dont on doit changer la valeur pour changer la valeur de vérité de l'arbre.

Question 9 : Exemple. On considère l'arbre dont le parcours préfixe donne

$$[\texttt{And}[\texttt{Or}[\texttt{And}[\texttt{True}, \texttt{False}], \texttt{And}[\texttt{False}, \texttt{False}]], \texttt{Or}[\texttt{True}, \texttt{True}]]].$$

Quelle est la valeur de cet arbre ? Quelle est la robustesse de cet arbre ? Justifier. ◇

Solution. False.
Robustesse 1, grâce au changement du premier False en True. □

Question 10 : Propriété structurelle. Etant donné un arbre A, soit $v(A)$ la valeur de vérité de A et soit $r(A)$ la robustesse de A. Ecrire une expression exprimant $r(A)$ en fonction de l'étiquette de la racine de A, et des valeurs et robustesses du ou des sous-arbres de A. ◇

Solution. Si A est une feuille, alors $r(A) = 1$.
Si la racine de A est un Not est que le fils est B, alors $r(A) = r(B)$.
Si la racine de A est un And et que les fils soient B, C, alors,
si $v(A) = v(B) = v(C) = False$ alors $r(A) = r(B) + r(C)$,
si $v(A) = v(B) = False \neq v(C)$ alors $r(A) = r(B)$,
si $v(A) = v(C) = False \neq v(B)$ alors $r(A) = r(C)$,
si $v(A) = v(B) = v(C) = True$ alors $r(A) = \min(r(B), r(C))$.
Si la racine de A est un Or et que les fils soient B, C, alors,
si $v(A) = v(B) = v(C) = True$ alors $r(A) = r(B) + r(C)$,
si $v(A) = v(B) = True \neq v(C)$ alors $r(A) = r(B)$,
si $v(A) = v(C) = True \neq v(B)$ alors $r(A) = r(C)$,
si $v(A) = v(B) = v(C) = False$ alors $r(A) = \min(r(B), r(C))$. □

Question 11 : Algorithme. En déduire un algorithme de complexité linéaire qui prend en entrée un arbre A et donne en résultat l'entier $r(A)$. $\diamond$

Solution. Algorithme récursif calculant les valeurs est les robustesse de bas en haut dans l'arbre, en passant un temps constant à chaque noeud. Pour ne pas avoir a recalculer les valeurs des fils, il faut les memoriser.

```
ValeurEtRobustesse(a,(vg,rg),(vd,rd)) retourner une paire (booleen, entier)
// a est l'arbre. la premiere valeur retournee est l'evaluation de a,
// la seconde est sa robustesse. Si a est une feuille, les autres
// parametres n'ont pas de sens.
// Si a est un Not, (vg,rg) est le resultat de la methode pour le fils de a, et les
// parametres n'ont pas de sens. Sinon, (vg,rg) est le resultat du fils gauche et (v
// est le resultat du fils droit.
Si a est une feuille alors retourner (a.valeur, 1)
Si a est un arbre NOT alors retourner (NOT(vg),rg)
Si a est un arbre AND alors
  v<-- vg AND vd
  si vg==false et vd==false alors retourner (v,rg+rd)
  si vg==false et vd==true alors retourner (v,rg)
  si vd==false et vg==true alors retourner (v,rd)
  sinon retourner (v,min(rg,rd))
Si a est un arbre OR alors
    ...  // de facon symetrique
```

2 Protocole de diffusion dans les réseaux

On considère un réseau défini par un graphe connexe $G = (V, E)$ non orienté dont les sommets représentent les postes (ou stations) et les arêtes représentent les liens de communication. On suppose que chaque poste connaît tous ses voisins et que les liens sont permanents.
Un poste a le droit d'envoyer plusieurs messages en même temps et de recevoir plusieurs messages en même temps.
Le poste 0 désire diffuser un message m à l'ensemble du réseau.

2.1 Protocole de diffusion en profondeur

On considère un premier protocole de diffusion P_1. Initialement, le poste 0 envoie le message m à un de ses voisins.

(a) Si le poste u reçoit le message m pour la première fois, provenant d'un certain voisin v, il renvoie m vers un de ses autres voisins. Quand u reçoit un acquittement ack(m) de ce voisin, il renvoie m à un autre voisin (différent de v), s'il le peut, etc. Au bout du compte, il envoie le message m à tous ses voisins sauf v, et quand tous ses voisins autres que v ont acquitté m, il renvoie un acquittement vers v.

(b) Si un poste qui a déjà reçu le message m d'un voisin v le reçoit encore d'un autre voisin w, il envoie un acquittement ack(m) à w. S'il le reçoit du même voisin alors qu'il a déjà acquitté, il répète l'acquittement.

(c) Si un poste qui a envoyé le paquet m vers un de ses voisins w n'en reçoit pas l'acquittement ack(m) de ce même voisin w, après un certain délai D il retransmet le paquet m à w.

Sur le réseau suivant : $V = \{0, 1, 2\}$, $E = \{\{0, 1\}, \{0, 2\}\}$, en supposant les transmissions de messages à date multiple d'entiers de l'unité de temps (par exemple, la seconde) on a le tableau des échanges suivants où $m(i, j)$ signifie "envoi du paquet m de i vers j" et $a(i, j)$ signifie "envoi de

l'acquittement ack(m) de i vers j". Dans cet exemple on suppose des transmissions sans perte ni erreur et que les opérations locales aux postes sont instantanées

temps	événements
1	$m(0,1)$
2	$a(1,0)$
3	$m(0,2)$
4	$a(2,0)$

Dans toutes les questions sauf les Questions 14 et 18, on suppose que le délai D est infiniment grand (ce qui revient à dire qu'on n'exécute pas les actions du (c)).

Question 12 : Etude détaillée de P_1 sur un exemple. Dans les mêmes hypothèses et le même format que dans l'exemple ci-dessus donner le tableau d'exécution du protocole sur le réseau suivant : $V = \{0,1,2,3,4,5\}$, $E = \{\{0,1\},\{0,5\},\{1,2\},\{1,5\},\{2,3\},\{3,1\},\{3,4\}\}$. On supposera qu'un sommet pouvant choisir entre plusieurs de ses voisins choisira toujours celui de plus petit numéro. Présenter l'exécution sous la forme suivante : chaque ligne correspond à un instant dans le temps ; sur cette ligne, vous écrirez quels sommets envoient quoi (m ou ack(m)) à quels autres sommets.

Quels sommets ont reçu le message à la fin ? Combien de temps prend l'algorithme ? Pour une arête donnée, combien de fois le message m a-t-il transité sur l'arête, au minimum, au maximum ? Quel est le nombre total de communications du message m, sommé sur toutes les arêtes du graphe ? Quelle a été la durée minimum entre l'envoi du message sur un lien et son acquittement sur le même lien ? Quelle a été la durée maximum ?

Solution. Le tableau est le suivant :

temps	événements
1	$m(0,1)$
2	$m(1,2)$
3	$m(2,3)$
4	$m(3,1)$
5	$a(1,3)$
6	$m(3,4)$
7	$a(4,3)$
8	$a(3,2)$
9	$a(2,1)$
10	$m(1,3)$
11	$a(3,1)$
12	$m(1,5)$
13	$m(5,0)$
14	$a(0,5)$
15	$a(5,1)$
16	$a(1,0)$
17	$m(0,5)$
18	$a(5,0)$

Tous les sommets reçoivent le message, l'algorithme de diffusion prend 18 unités de temps. Sur chaque arête le message a transité au minimum une fois et au maximum deux fois. Il y a eu 18 transmission de paquets. La durée minimum entre l'émission et son acquittement est d'une unité de temps, par exemple sur le lien $(5,0)$, la durée maximum est de 16 unité de temps sur le lien $(0,1)$. □

Question 13 On appelle E_c l'ensemble des arêtes (u,v) telles que le sommet u est le premier sommet du réseau à délivrer le message m au sommet v. Donner cet ensemble E_c pour l'exemple de la question 12. Que forme l'ensemble E_c sur le graphe (V,E) ? Quel est le nombre de communications du message m dans les deux sens sur une arête de E_c en l'absence de perte ? ◇

Solution. On a $E_c = \{(0,1),(1,2),(2,3),(3,4),(1,5)\}$. E_c forme un arbre couvrant en profondeur du graphe (V,E). Le message m ne passe qu'une fois et dans un seul sens sur chaque arête de E_c. □

Question 14 : Propriété de base. Un *canal fini* est un canal où, même s'il peut quelquefois y avoir des pertes, tout message finit par passer au bout d'un nombre fini de retransmissions. Un *poste obstiné* est un poste qui répétera indéfiniment un message m sur un lien tant qu'il n'a pas reçu l'acquittement ack(m) sur ce lien.
Démontrer qu'avec un canal fini sur les liens et avec des postes obstinés, si le graphe du réseau est connexe alors chacun des postes finit par recevoir le paquet. ◇

Solution. On raisonne par l'absurde. Soit $V_r \subset V$ l'ensemble des sommets qui ont reçu le message m. Si $V - V_r \neq \emptyset$ alors comme le graphe est connexe il existe $u \in V_r$ et $v \in V - V_r$ tel que $\{u,v\} \in E$. Le sommet u essaye d'envoyer le message m vers tous ses voisins sauf celui de qui il a reçu le message m. Le premier de ces voisins v_1 finit par recevoir le message m et u reçoit l'acquittement, sinon le message m serait répété indéfiniment sur le lien (u, v_1), donc en vertu de la propriété du canal fini, il serait reçu et acquitté indéfiniment, le sommet u recevant donc en retour une infinité de messages ack(m). Donc chacun de ces voisins finit par recevoir le message m, donc le sommet v reçoit le message v. □

Question 15 : Efficacité du protocole. On suppose qu'aucun message n'est perdu. Donner une borne supérieure du nombre total de communications du message m au cours de l'exécution du protocole ? Donner une borne supérieure du temps pris par le protocole ? Quel graphe donne le nombre de communications du message m et le temps minimum d'exécution pour un réseau à n sommets ? On ne vous demande pas de preuves mais seulement de bons exemples en fonction des paramètres du graphe. ◇

Solution. Le message m passe au maximmum deux fois sur une arête, une fois dans chaque sens. Donc le nombre maximum de transmission du message m est $2|E|$. Comme le message m est acquitté deux fois sur la même arête, le temps maximum d'éxécution est $4|E|$ unités de temps. Sur chaque arête de l'arbre couvrant E_c le message m ne passe qu'une fois et n'est acquitté qu'une fois. Donc l'arbre est la forme du graphe qui minimise le nombre de communications, $n-1$, et la durée d'exécution de l'algorithme, $2(n-1)$ unités de temps. De manière générale le nombre total de comunications est $2|E| - |V| + 1$ et la durée $4|E| - 2|V| + 2$ unités de temps. □

2.2 Protocole parallèle

On étudie maintenant un protocole à diffusion *en parallèle* P_2 qui fonctionne de la manière suivante :
Initialement, le poste 0 envoie le message m à *tous* ses voisins en parallèle.

(a) Si le poste u reçoit le message m pour la première fois, provenant d'un certain voisin v, il renvoie m en parallèle vers tous ses voisins sauf v. S'il reçoit la première fois le message m de deux voisins ou plus simultanément, il choisit comme premier voisin celui qui a le plus petit indice et renverra le message m vers les autres

(b) A chaque fois qu'un poste reçoit le message m d'un voisin v, y compris la première fois, il envoie aussitôt (en parallèle à ses autres actions) un acquittement ack(m) à v. S'il le reçoit d'un voisin qu'il a déjà acquitté, il répète l'acquittement.

(c) (Règle inchangée.) Si un poste qui a envoyé le paquet m vers un de ses voisins w n'en reçoit pas l'acquittement ack(m) de ce même voisin w, après un certain délai D, il retransmet le paquet m à w.

Question 16 : Etude détaillée de P_2 sur un exemple. Exécuter le protocole P_2 sur l'exemple de la question 12. Répondre aux mêmes questions de détail. On suppose que les deux sens des liens peuvent être utilisés simultanément. ◇

Solution. On a le tableau suivant :

temps	événements
1	$m(0,1)$, $m(0,5)$
2	$a(1,0)$, $a(5,0)$, $m(1,2)$, $m(1,3)$, $m(1,5)$, $m(5,1)$
3	$a(2,1)$, $a(3,1)$, $a(5,1)$, $a(1,5)$, $m(2,3)$, $m(3,2)$, $m(3,4)$
4	$a(3,2)$, $a(2,3)$, $a(4,3)$

Ce sont les mêmes réponses que pour la question 12 sauf que l'arbre couvrant est en largeur et c'est $E_c = \{(0,1),(0,5),(1,2),(1,3),(3,4)\}$ et le temps d'éxécution est ramassé sur 4 unités de temps. □

Question 17 : Comparaison. Comparer le protocole P_2 au protocole P_1 : quel avantage y voyez-vous ? Est-ce significatif en général ? Voyez-vous un inconvénient à P_2 ? ◇

Solution. Le protocole P_2 s'exécute plus rapidement que le protocole P_1 grâce aux transmissions parallèles. L'avantage est significatif dans la mesure où la durée d'éxécution n'est plus proportionnelle au nombre de liens, mais au diamètre du graphe. L'inconvénient du protocole P_2 est que la source du message m n'est plus informée quand la diffusion est terminée (de toute mamière cette propriété n'est vraie qu'en l'absence de perte de message). Dans le protocole P_1 la source est informée quand son dernier voisin envoie son acquittement ack(m). □

Question 18 : Délais et retransmissions. On reprend l'exemple de la question 12, avec cette différence que maintenant *on suppose qu'on a un délai* $D = 4$. Ré-exécuter le protocole P_1 ; quelle différence remarquez-vous ? Ré-exécuter le protocole P_2 : quelle différence remarquez-vous ?

Pour chacun de ces deux protocoles, que conseilleriez-vous au concepteur responsable du choix de la valeur de D ? ◇

Solution. Le déroulement du protocole P_1 est modifié. Au temps 4 le sommet 0 renvoie le message vers 1, celui l'acquitte immédiatement . Le sommet 0 envoie le message à son dernier voisin 5 qui l'acquitte alors que la procédure de diffusion n'est pas terminée. Le déroulement du protocole P_2 reste inchangé.

Pour chacun des deux protocoles la valeur de D doit être supérieure au temps d'aller et retour d'un message sur un lien. □