ÉCOLE NATIONALE DES PONTS ET CHAUSSÉES,
ÉCOLES NATIONALES SUPÉRIEURES DE L'AÉRONAUTIQUE ET DE L'ESPACE,
DES TECHNIQUES AVANCÉES, DES TÉLÉCOMMUNICATIONS,
DES MINES DE PARIS, DES MINES DE SAINT-ÉTIENNE, DES MINES DE NANCY,
DES TÉLÉCOMMUNICATIONS DE BRETAGNE,
ÉCOLE POLYTECHNIQUE
(Filière T.S.I.)

CONCOURS D'ADMISSION 2005

ÉPREUVE D'INFORMATIQUE

Filière MP
(Durée de l'épreuve : 3 heures)

Sujet mis à la disposition des concours Cycle International, ENSTIM et TPE-EIVP.

Les candidats et les candidates sont priés de mentionner de façon apparente sur la première page de la copie : « INFORMATIQUE – Filière MP »

RECOMMANDATIONS AUX CANDIDATS ET CANDIDATES

- **L'énoncé de cette épreuve, y compris cette page de garde, comporte 9 pages.**
- **Si, au cours de l'épreuve, un candidat ou une candidate repère ce qui lui semble être une erreur d'énoncé, il ou elle le signale sur sa copie et poursuit sa composition en expliquant les raisons des initiatives qu'il ou elle a décidé de prendre.**
- **Tout résultat fourni dans l'énoncé peut être utilisé pour les questions ultérieures même s'il n'a pas été démontré.**
- **Il ne faut pas hésiter à formuler les commentaires qui semblent pertinents même lorsque l'énoncé ne le demande pas explicitement.**
- **L'utilisation d'une calculatrice ou d'un ordinateur est interdite.**

COMPOSITION DE L'ÉPREUVE

L'épreuve comporte deux parties indépendantes :

- un problème sur les automates, pages 2 et 3 (60 mn environ) ;
- un problème d'algorithmique, logique et programmation, pages 3 à 9 (120 mn environ).

Téléchargé gratuitement sur www.Doc-Solus.fr .

1. Problème sur les automates (60 mn environ)

Notations et terminologie

Un *alphabet* Σ est un ensemble fini d'éléments appelés *lettres*. Un *mot* sur Σ est une suite finie, éventuellement vide, de lettres de Σ. On note Σ^* l'ensemble des mots sur Σ. La *longueur* d'un mot est le nombre de lettres qui le composent. Un *langage* est une partie de Σ^*.
Si a appartient à Σ et si n désigne un entier positif, a^n est le mot constitué de n fois la lettre a ; a^0 est le mot vide.

Un *automate* $\mathcal{A}$ est décrit par une structure $<\Sigma, Q, T, I, F>$, où :

- Σ est un alphabet ;
- Q est un ensemble fini et non vide appelé *ensemble des états* de $\mathcal{A}$;
- $T \subseteq Q \times \Sigma \times Q$ est appelé l'*ensemble des transitions* ; étant donnée une transition $(p, a, q) \in T$, on dit qu'elle va de l'état p à l'état q et qu'elle est d'*étiquette* a ;
- $I \subseteq Q$ est appelé *ensemble des états initiaux* de $\mathcal{A}$;
- $F \subseteq Q$ est appelé *ensemble des états finals* de $\mathcal{A}$.

Début de l'énoncé du problème

Dans tout ce problème, l'alphabet Σ utilisé n'a que deux lettres notées a et b : $\Sigma = \{a, b\}$.
On note $\mathcal{P}(\Sigma^*)$ l'ensemble des parties de Σ^*, c'est-à-dire l'ensemble des langages sur Σ^*.
On se propose d'étudier plusieurs « transformations de langage », c'est-à-dire plusieurs fonctions de $\mathcal{P}(\Sigma^*)$ dans $\mathcal{P}(\Sigma^*)$.
On désigne par L_3 le langage des mots sur Σ dont la longueur est multiple de 3.

❒ 1 – Indiquer un automate déterministe reconnaissant L_3.

❒ 2 – Donner une expression rationnelle de L_3.

On considère une fonction notée Φ de $\mathcal{P}(\Sigma^*)$ dans $\mathcal{P}(\Sigma^*)$ définie de la façon suivante. Soit L une partie de Σ^* ; un mot v appartient à $\Phi(L)$ si et seulement si v est le plus long préfixe de la forme a^n d'un mot de L. Autrement dit, un mot v appartient à $\Phi(L)$ si et seulement si :

- d'une part il existe un entier $n \geq 0$ tel que $v = a^n$;
- d'autre part, soit $v \in L$, soit il existe un mot $u \in L$ et un mot x sur Σ tel que $u = vbx$.

❒ 3 – Déterminer $\Phi(L_3)$.

❒ 4 – Donner un exemple de langage L sur Σ, non rationnel, tel que $\Phi(L)$ soit rationnel. Il n'est pas demandé de démontrer que le langage L proposé est non rationnel ; on indiquera ce que vaut $\Phi(L)$.

❒ 5 – Soit L un langage rationnel et $\mathcal{A} = <\Sigma, Q, T, I, F>$ un automate reconnaissant L. Décrire à partir de l'automate $\mathcal{A}$ un automate reconnaissant $\Phi(L)$ et justifier cette réponse.

On considère une deuxième fonction notée Γ de $\mathcal{P}(\Sigma^*)$ dans $\mathcal{P}(\Sigma^*)$ définie de la façon suivante. Soit L une partie de Σ^* ; un mot v appartient à $\Gamma(L)$ si et seulement s'il peut se déduire d'un mot u de L par insertion d'une occurrence de b dans ce mot ; cela revient à dire qu'un mot v appartient à $\Gamma(L)$ si et seulement s'il existe des mots x et y sur Σ tels que : $v = xby$ et le mot $u = xy$ est dans L.

❒ 6 – Déterminer $\Gamma(L_3)$.

❒ 7 – Soit L un langage rationnel et $\mathcal{A} = <\Sigma, Q, T, I, F>$ un automate reconnaissant L. Décrire à partir de l'automate $\mathcal{A}$ un automate $\mathcal{A}'$ reconnaissant $\Gamma(L)$. On donnera une description précise de l'automate $\mathcal{A}'$ mais il n'est pas demandé de justifier la réponse. Un automate $\mathcal{A}'$ décrit de façon trop imprécise sera considéré comme inexact.

Téléchargé gratuitement sur www.Doc-Solus.fr .

On considère une troisième fonction notée Ψ de $\mathcal{P}(\Sigma^*)$ dans $\mathcal{P}(\Sigma^*)$ définie de la façon suivante. Soit L une partie de Σ^*. Un mot v appartient à $\Psi(L)$ si et seulement s'il peut se déduire d'un mot u de L contenant au-moins une fois la lettre b par effacement de la première occurrence dans u de la lettre b ; cela revient à dire qu'un mot v appartient à $\Psi(L)$ si et seulement s'il existe un entier $n \geq 0$ et un mot x sur Σ tels que : $v = a^n x$ et le mot $u = a^n bx$ est dans L.

❒ 8 – Déterminer $\Psi(L_3)$.

❒ 9 – Donner un exemple de langage L sur Σ non rationnel invariant par Ψ, c'est-à-dire vérifiant $\Psi(L) = L$. On prouvera que le langage L proposé n'est pas rationnel. On prouvera la relation $\Psi(L) = L$.

❒ 10 – Soit L un langage rationnel et $\mathcal{A} = <\Sigma, Q, T, I, F>$ un automate reconnaissant L. Décrire, à partir de l'automate $\mathcal{A}$, un automate $\mathcal{A}'$ reconnaissant $\Psi(L)$. On donnera une description précise de l'automate $\mathcal{A}'$ mais il n'est pas demandé de justifier la réponse. Un automate $\mathcal{A}'$ décrit de façon trop imprécise sera considéré comme inexact.

FIN DU PROBLÈME SUR LES AUTOMATES

2. Problème d'algorithmique, logique et programmation (120 mn environ)

L'objectif de ce problème est l'étude d'un cas particulier, de complexité polynomiale, du problème de la satisfiabilité ; ce sujet ne sera abordé que dans la seconde partie. La première partie introduit un algorithme de graphes qui sera utilisé par la suite.

Préliminaire concernant la programmation : il faudra écrire des fonctions ou des procédures à l'aide d'un langage de programmation qui pourra être soit **Caml**, soit **Pascal**, tout autre langage étant exclu. **Indiquer en début de problème le langage de programmation choisi ; il est interdit de modifier ce choix au cours de l'épreuve**. Certaines questions du problème sont formulées différemment selon le langage de programmation ; cela est indiqué chaque fois que c'est nécessaire. Par ailleurs, lorsqu'un candidat ou une candidate écrira une fonction ou une procédure en langage de programmation, il ou elle précisera si nécessaire le rôle des variables locales et pourra définir des fonctions ou procédures auxiliaires qu'il ou elle explicitera. Enfin, lorsque le candidat ou la candidate écrira une fonction ou une procédure, il ou elle pourra faire appel à une autre fonction ou procédure définie dans les questions précédentes.
Dans l'énoncé du problème, un même identificateur écrit dans deux polices de caractères différentes désignera la même entité mais du point de vue mathématique pour une police (en italique) et du point de vue informatique pour l'autre (en romain) ; par exemple, G sera la représentation informatique du graphe G.

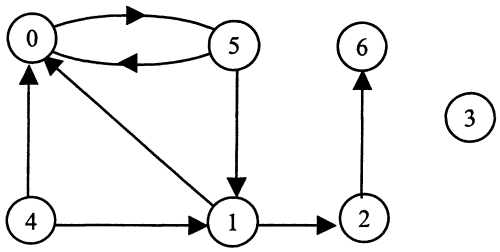
Définitions et notations concernant les graphes

- Un *graphe orienté* G est composé d'un ensemble fini d'éléments appelés *sommets* et d'une liste de couples de sommets, chacun de ces couples étant appelé *arc* ; dans ce problème, en notant n le nombre de sommets de G, l'ensemble des sommets sera toujours $\{0, 1, \ldots, n-1\}$.
- Si x et y sont deux sommets d'un graphe G et que (x, y) est un arc de G, on dira que y est un *fils* de x (on dit aussi que y est un successeur de x).
- Si x et y sont deux sommets d'un graphe G, on appelle *chemin* de x à y une suite de sommets $x_0, x_1, \ldots, x_p$ ($p \geq 0$) avec $x_0 = x$, $x_p = y$ et, pour tout i vérifiant $1 \leq i \leq p$, (x_{i-1}, x_i) est un arc de G ; on dit alors que y est un *descendant* de x ; on remarque qu'un sommet est descendant de lui-même.
- Une représentation graphique d'un graphe G consiste à associer à chaque sommet de G un cercle contenant le numéro du sommet et à tracer des flèches du cercle représentant un sommet aux cercles correspondant à ses fils.

Téléchargé gratuitement sur www.Doc-Solus.fr .

Exemple introductif : un graphe (nommé *Gex*) et une représentation de *Gex*

On considère le graphe *Gex* à 7 sommets, représenté ci-contre, dont l'ensemble des sommets est donc {0, 1, 2, 3, 4, 5, 6}.
Les arcs de ce graphe sont : (0, 5), (5, 0), (4, 0), (4, 1), (5, 1), (1, 0), (1, 2), (2, 6).
Le sommet 0 a pour fils le sommet 5.
Le sommet 1 a pour fils les sommets 0 et 2.
Le sommet 2 a pour fils le sommet 6.
Le sommet 3 n'a pas de fils.
Le sommet 4 a pour fils les sommets 0 et 1.
Le sommet 5 a pour fils les sommets 0 et 1.
Le sommet 6 n'a pas de fils.



Le graphe *Gex*

Indications pour la programmation

Caml : On définit les types suivants :

```
type Graphe = {nb_sommets : int; fils : int list vect};;

type Clause = {a : int; b : int};;

type Formule = {nb_var : int; clauses : Clause list};;
```

Ces trois types sont des types enregistrements (aussi appelés types produits). Une valeur de type enregistrement contient des champs. On peut accéder à un champ d'une valeur de type enregistrement en faisant suivre le nom de cette valeur d'un point puis du nom du champ considéré. Par exemple, si `G` est de type `Graphe`, `G` contient les champs `nb_sommets` et `fils`, auxquels on accède par `G.nb_sommets` et `G.fils`. Pour une valeur `G` de type `Graphe`, le champ `G.nb_sommets` est égal au nombre de sommets de `G`, le champ `G.fils` est un tableau (aussi appelé vecteur) de longueur `G.nb_sommets` et, pour un entier `s` vérifiant $0 \leq s \leq$ `G.nb_sommets - 1`, `G.fils.(s)` est la liste des fils du sommet `s`.

Le graphe *Gex* de l'exemple introductif est défini par :

```
let Gex = {nb_sommets = 7; fils = [|[5]; [0;2]; [6]; []; [0;1]; [0;1]; []|]};;
```

Fin des indications pour Caml

Pascal : Dans tout le problème, on supposera qu'on écrit les différentes fonctions dans un fichier contenant les définitions suivantes :

```
const
   MAX = 200;
   MAX_CLAUSES  = 100;

type Table = RECORD
      nb    : Integer;
      val   : array[0 ..  MAX - 1] of Integer;
   end;

type Graphe = RECORD
      nb_sommets  : Integer;
      fils        : array [0 ..  MAX - 1] of Table;
   end;

type Clause = RECORD
      a, b : Integer;
   end;
```

Téléchargé gratuitement sur www.Doc-Solus.fr .

```
type Formule = RECORD
      nb_var       : Integer;
      nb_clauses   : Integer;
      clauses      : array[0 .. MAX_CLAUSES - 1] of Clause;
   end;

type Valeurs = array[0 ..  MAX - 1] of Boolean;
```

On supposera que les graphes traités n'ont jamais plus de MAX sommets.

Les types `Table`, `Graphe`, `Clause` et `Formule` sont des types pour des enregistrements (RECORD). Un enregistrement contient des champs (quelquefois aussi appelés des membres) ; par exemple, une variable de type `Graphe` contient les champs `nb_sommets` et `fils` ; on peut accéder à un champ d'une variable de type enregistrement en faisant suivre le nom de cette variable d'un point puis du nom du champ considéré, comme on le voit dans la définition de `Gex` ci-dessous. Les variables de type enregistrement se manipulent comme toute autre variable : on peut définir des variables de type enregistrement, on peut affecter à une variable de type enregistrement la valeur d'une autre variable du même type, les variables de type enregistrement peuvent servir de paramètres à des fonctions ou procédures et peuvent être renvoyées par des fonctions.

Lorsqu'on a affaire à une variable `T` de type `Table`, le tableau `T.val` sert à contenir un ensemble de données entières et on convient que le champ `T.nb` indique toujours le nombre de données de cet ensemble ; on a toujours `T.nb` < `MAX` et les données contenues dans `T.val` se trouvent toujours dans les cases indicées de 0 à `T.nb - 1`. Pour une variable `G` de type `Graphe`, le champ `G.nb_sommets` est égal au nombre de sommets de `G` et, pour un entier `s` vérifiant `0 ≤ s ≤ G.nb_sommets - 1`, la table `G.fils[s]` contient les fils du sommet `s` : leur nombre est dans `G.fils[s].nb` et leurs numéros dans le tableau `G.fils[s].val`.

Ainsi, le graphe `Gex` de l'exemple introductif est défini par :

```
Gex.nb_sommets := 7;
Gex.fils[0].nb := 1;
Gex.fils[0].val[0] := 5;
Gex.fils[1].nb := 2;
Gex.fils[1].val[0] := 0;
Gex.fils[1].val[1] := 2;
Gex.fils[2].nb := 1;
Gex.fils[2].val[0] := 6;
Gex.fils[3].nb := 0;
Gex.fils[4].nb := 2;
Gex.fils[4].val[0] := 0;
Gex.fils[4].val[1] := 1;
Gex.fils[5].nb := 2;
Gex.fils[5].val[0] := 0;
Gex.fils[5].val[1] := 1;
Gex.fils[6].nb := 0;
```

Fin des indications pour Pascal

PREMIÈRE PARTIE

Descendants d'un sommet d'un graphe orienté

Il s'agit dans cette partie d'étudier un algorithme, nommé ici *calcul_descendants*, qui permet de déterminer les descendants d'un sommet dans un graphe donné. À titre d'exemples, remarquons que, dans le graphe *Gex* de l'exemple introductif, les descendants du sommet 0 sont les sommets 0, 1, 2, 5 et 6, les descendants du sommet 4 sont tous les sommets sauf le sommet 3, le sommet 2 a pour descendants les sommets 2 et 6, les sommets 3 et 6 n'ont pour descendants qu'eux-mêmes.

Considérons un graphe G ayant n sommets et considérons un sommet r ($0 \leq r \leq n-1$) de ce graphe. On suppose qu'on cherche les descendants du sommet r. Le principe de l'algorithme *calcul_descendants* est de « marquer »

Téléchargé gratuitement sur www.Doc-Solus.fr .

de proche en proche tous les descendants du sommet r, en commençant par le sommet r lui-même. Le marquage des descendants de r utilise un algorithme récursif appelé *marquage* décrit ci-dessous :

Marquage de G à partir de s
- *marquer s ;*
- *pour tout fils non marqué t de s, effectuer « Marquage de G à partir de t ».*

Pour effectuer dans G l'algorithme *calcul_descendants* à partir de r, on part d'une situation dans laquelle aucun sommet n'est marqué puis on effectue « Marquage de G à partir de r ». Après l'application de cet algorithme, les sommets marqués sont les descendants du sommet r.

❒ 11 – Appliquer l'algorithme ci-dessus au graphe *Gex* à partir du sommet 1 en détaillant toutes les opérations de l'algorithme.

❒ 12 – Il s'agit de programmer le calcul des descendants d'un sommet r donné dans un graphe G donné. On utilisera pour cela un tableau de variables booléennes nommé *marques* et destiné à contenir les « marques » des sommets. Plus précisément, après le déroulement de l'algorithme, si s est un sommet du graphe, la case d'indice s du tableau *marques* contiendra la valeur *vrai* (c'est-à-dire `true` pour le langage de programmation) si et seulement si le sommet s est descendant de r ; sinon, cette case contiendra la valeur *faux* (c'est-à-dire `false` pour le langage de programmation).
Pour répondre à la question ci-dessous, on sera amené à écrire aussi, en langage de programmation, une fonction récursive nommée *marquage* correspondant à l'algorithme décrit plus haut.

Caml : Écrire en Caml une fonction `calcul_descendants` telle que, si `G` est une valeur de type `Graphe` et `r` un entier représentant un sommet du graphe, `calcul_descendants G r` renvoie le tableau `marques` contenant les marques des sommets indiquant les descendants de `r`.

Pascal : Écrire en Pascal une fonction `calcul_descendants` telle que, si `G` est une variable de type `Graphe` et `r` un entier représentant un sommet du graphe, `calcul_descendants(G, r)` renvoie le tableau `marques`, de type `Valeurs`, contenant les marques des sommets indiquant les descendants de `r`.

❒ 13 – Prouver l'exactitude de l'algorithme *calcul_descendants*, c'est-à-dire que les sommets marqués après l'exécution de l'algorithme sont bien exactement les descendants de r.

❒ 14 – Indiquer la complexité dans le pire des cas de l'algorithme *calcul_descendants*. Cette complexité sera exprimée à l'aide du nombre n de sommets et du nombre m d'arcs du graphe G. On justifiera brièvement la réponse.

SECONDE PARTIE

Un problème de satisfiabilité

On appelle variable booléenne une variable qui ne peut prendre que les valeurs *vrai* ou *faux*. Si x est une variable booléenne, on appelle complémenté de x et on note $\overline{x}$ la négation de x qui vaut *vrai* si x vaut *faux* et inversement. On appelle *littéral* une variable booléenne ou son complémenté. Pour un littéral $\alpha = \overline{x}$, où x est une variable booléenne, on définit le complémenté $\overline{\alpha}$ de α comme étant égal à x. On a ainsi défini le complémenté de tout littéral.
On appelle *clause* une disjonction de littéraux et *longueur d'une clause* le nombre des littéraux qui composent cette clause. On appelle *formule logique sous forme normale conjonctive* une conjonction de clauses. Dans ce problème, on s'intéresse aux formules logiques sous forme normale conjonctive pour lesquelles **toutes les clauses sont de longueur 2**. On dira qu'une telle formule est sous forme *NC2*.
Lorsqu'on considérera une formule logique, on supposera que les littéraux d'une même clause sont différents et que toutes les clauses sont différentes.
Une formule logique est dite *satisfiable* s'il existe une façon d'attribuer des valeurs aux variables booléennes telle que la formule soit évaluée à *vrai*.
On représente le « ou logique » (disjonction) par le symbole $\vee$ et le « et logique » (conjonction) par le symbole $\wedge$.

Téléchargé gratuitement sur www.Doc-Solus.fr .

❒ 15 – Étant données trois variables booléennes x, y et z, on considère la formule :

$$F_1 = (x \vee y) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee z) \wedge (\overline{x} \vee \overline{z}).$$

F_1 est-elle satisfiable ? Justifier votre réponse.

❒ 16 – Étant données quatre variables booléennes x, y, z et t, on considère la formule :

$$F_2 = (x \vee y) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (t \vee \overline{z}) \wedge (y \vee \overline{t}) \wedge (x \vee \overline{y}).$$

F_2 est-elle satisfiable ? Justifier votre réponse.

On s'intéresse à l'histoire ci-dessous.
Quatre personnes, nommées X, Y, Z et T, peuvent être chacune soit *fiable*, soit *non fiable* : une personne *fiable* dit toujours la vérité ; une personne *non fiable* peut dire la vérité ou mentir. Chacune de ces personnes sait si les autres sont fiables ou non.

- La personne X dit : Z est fiable.
- La personne Y dit : Z est non fiable, T est fiable.
- La personne Z dit : Y est fiable, T est fiable.
- La personne T dit : X est non fiable, Y est fiable.

Par ailleurs, on sait que :

- X est fiable ou Y est fiable ou X et Y sont fiables.
- Z est fiable ou T est fiable ou Z et T sont fiables.

On définit quatre variables booléennes ; la variable booléenne x (resp. y, z, t) vaut *vrai* si X (resp. Y, Z, T) est fiable et *faux* si X (resp. Y, Z, T) n'est pas fiable.

❒ 17 – Exprimer, à l'aide des variables x, y, z, t et de leurs complémentés, sous forme *NC2*, les renseignements dont on dispose sur la fiabilité ou la non-fiabilité des quatre personnes X, Y, Z et T.

❒ 18 – Déterminer les personnes fiables et les personnes non fiables. On prouvera le résultat.

On va désormais numéroter à partir de 0 les variables booléennes d'une formule. Ainsi, si les variables sont x, y et z, on associe à x le numéro 0, à y le numéro 1 et à z le numéro 2. Par ailleurs, s'il y a p variables booléennes, on numérote les complémentés des variables à partir de p en respectant l'ordre choisi pour numéroter les variables ; avec l'exemple précédent, comme p vaut 3, on numérote $\overline{x}$ avec le numéro 3, $\overline{y}$ avec le numéro 4 et $\overline{z}$ avec le numéro 5. **Pour alléger les explications, on identifie désormais un littéral avec son numéro.**

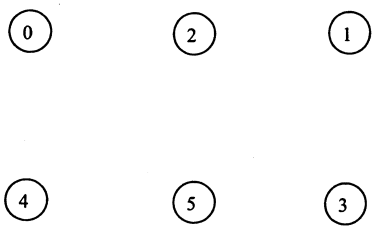
❒ 19 – On considère un entier compris entre 0 et $2p - 1$ identifiant un littéral α d'une formule dépendant de p variables booléennes. Il s'agit d'écrire une fonction, nommée *barre*, qui calcule le numéro du complémenté de α. Par exemple, si la formule possède trois variables, *barre*(0) doit valoir 3 et *barre*(5) doit valoir 2.

Caml : Écrire en Caml une fonction `barre` telle que si `alpha` est le numéro d'un littéral d'une formule dépendant de `p` variables, `barre alpha p` donne le numéro du complémenté de ce littéral.

Pascal : Écrire en Pascal une fonction `barre` telle que si `alpha` est le numéro d'un littéral d'une formule dépendant de `p` variables, `barre(alpha, p)` donne le numéro du complémenté de ce littéral.

À partir d'une formule logique F sous forme *NC2*, on construit un graphe orienté $G(F)$. Si la formule dépend de p variables, le graphe possède $2p$ sommets, les sommets 0, 1, 2, …, $2p - 1$, correspondant à ces variables et à leurs complémentés. À chaque clause $\alpha \vee \beta$, où α et β sont des littéraux, on fait correspondre deux arcs de $G(F)$: un de $\overline{\alpha}$ vers β et un de $\overline{\beta}$ vers α.

❒ 20 – On considère la formule :
$F_1 = (x \vee y) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee z) \wedge (\overline{x} \vee \overline{z})$.
Cette formule possède trois variables. On identifie x à 0, y à 1, z à 2, $\overline{x}$ à 3, $\overline{y}$ à 4, $\overline{z}$ à 5. Dessiner le graphe $G(F_1)$ associé à F_1 en disposant les sommets comme ci-contre.



Téléchargé gratuitement sur www.Doc-Solus.fr .

Indications pour Caml

Une formule logique sous forme *NC2* sera représentée par une valeur de type `Formule` ; ainsi la formule $(x \vee y) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee z) \wedge (\overline{x} \vee \overline{z})$ sera définie de la façon suivante :

```
{nb_var=3; clauses = [{a=0;b=1}; {a=3;b=2}; {a=4;b=2}; {a=3;b=5}]};;
```

Le premier champ indique le nombre de variables, le second champ indique la liste des clauses.

Fin des indications pour Caml

Indications pour Pascal

Une formule logique sous forme *NC2* sera représentée par une variable de type `Formule` ; si la variable utilisée pour coder la formule $(x \vee y) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee z) \wedge (\overline{x} \vee \overline{z})$ s'appelle `F`, la formule est définie par les instructions suivantes :

```
F.nb_var := 3;
F.nb_clauses := 4;
F.clauses[0].a := 0;
F.clauses[0].b := 1;
F.clauses[1].a := 3;
F.clauses[1].b := 2;
F.clauses[2].a := 4;
F.clauses[2].b := 2;
F.clauses[3].a := 3;
F.clauses[3].b := 5;
```

Le champ `nb_var` indique le nombre des variables booléennes et le champ `nb_clauses` le nombre de clauses. Le champ `clauses` est un tableau de variables de type `Clause` qui décrit les clauses.

On supposera que le nombre de variables booléennes ne dépasse pas la moitié de la constante `MAX` et que le nombre de clauses ne dépasse pas la constante `MAX_CLAUSES`.

Fin des indications pour Pascal

❒ 21 – Il s'agit d'écrire une fonction nommée *transformer* qui reçoit en paramètre une formule logique F sous forme *NC2* et qui construit $G(F)$.

Caml : Écrire en Caml une fonction `transformer` telle que, si `F` est une valeur de type `Formule`, alors `transformer F` renvoie une valeur de type `Graphe` correspondant au graphe $G(F)$ décrit plus haut.

Pascal : Écrire en Pascal une fonction `transformer` telle que, si `F` est une variable de type `Formule`, alors `transformer(F)` renvoie une valeur de type `Graphe` correspondant au graphe $G(F)$ décrit plus haut.

❒ 22 – Indiquer la complexité de la fonction *transformer* définie précédemment. On exprimera cette complexité en fonction du nombre p de variables et du nombre q de clauses de la formule F à laquelle est appliquée la fonction *transformer*.

❒ 23 – Par la suite, il sera utile de pouvoir retirer d'une formule logique F sous forme *NC2* toutes les clauses contenant un littéral donné α ou son complémenté. Dans la formule F' ainsi obtenue, une ou plusieurs variables ne figureront plus. Néanmoins, on considère que toutes les variables de F sont aussi variables de F', n'imposant donc pas que toutes les variables d'une formule figurent effectivement. Par exemple, si de $F_1 = (x \vee y) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee z) \wedge (\overline{x} \vee \overline{z})$, formule contenant trois variables, on décide de retirer le littéral $\overline{x}$ ou son complémenté, on obtient $F'_1 = (\overline{y} \vee z)$ et on considère encore que F'_1 dépend de trois variables, et les entiers correspondant aux variables restantes sont inchangés (à $\overline{y}$ correspond toujours le numéro 4 et à z le numéro 2).

Caml : Écrire en Caml une fonction `retirer` qui, si `F` est une valeur de type `Formule` ayant `F.nb_var` variables et si `alpha` est un entier compris entre 0 et `2×F.nb_var−1`, `retirer F alpha` renvoie une valeur de type `Formule` obtenue à partir de `F` en retirant les clauses contenant `alpha` ou son complémenté.

Pascal : Écrire en Pascal une fonction `retirer` qui, si `F` est une variable de type `Formule` ayant `F.nb_var` variables et si `alpha` est un entier compris entre 0 et `2×F.nb_var−1`, `retirer(F, alpha)` renvoie une valeur de type `Formule` obtenue à partir de `F` en retirant les clauses contenant `alpha` ou son complémenté.

Téléchargé gratuitement sur www.Doc-Solus.fr .

❒ 24 – Indiquer la complexité de la fonction *retirer* définie précédemment. On exprimera cette complexité en fonction du nombre p de variables et du nombre q de clauses de la formule F à laquelle est appliquée la fonction *retirer*.

❒ 25 – Soient α et β deux littéraux distincts d'une formule logique F sous forme *NC2*. Montrer que s'il existe un chemin de α à β dans le graphe $G(F)$, alors la formule F implique la formule « $\alpha \Rightarrow \beta$ ».

❒ 26 – Étant donnée une formule logique F sous forme *NC2*, montrer que, s'il existe un chemin de α à β dans le graphe $G(F)$, alors il existe aussi un chemin de $\overline{\beta}$ à $\overline{\alpha}$ dans $G(F)$.

❒ 27 – Étant donnée une formule logique F sous forme *NC2*, que peut-on dire de la formule F s'il existe dans le graphe $G(F)$ à la fois un chemin de α à $\overline{\alpha}$ et un chemin de $\overline{\alpha}$ à α?

❒ 28 – Étant donnés une formule logique F sous forme *NC2* et deux littéraux α et β de F, montrer que s'il n'existe pas de chemin de α à $\overline{\alpha}$ dans le graphe $G(F)$, alors β et $\overline{\beta}$ ne sont pas tous les deux descendants de α dans $G(F)$.

❒ 29 – Soit F une formule logique sous forme *NC2*. On considère un littéral α tel qu'il n'existe pas de chemin de α à $\overline{\alpha}$ dans le graphe $G(F)$. On note F' la formule obtenue à partir de F en retirant toutes les clauses contenant un littéral descendant de α dans $G(F)$ ou bien le complémenté d'un de ces descendants ; on retire donc en particulier les clauses contenant α ou $\overline{\alpha}$. Montrer que F est satisfiable si et seulement si F' l'est. En supposant que F' est satisfiable, indiquer, à l'aide des valeurs des variables figurant dans F' qui donnent la valeur *vrai* à F', des valeurs des variables figurant dans F qui donnent la valeur *vrai* à F.

❒ 30 – Soit F une formule logique sous forme *NC2*. Montrer que, si pour tout littéral α de F, il n'existe pas dans le graphe $G(F)$ à la fois un chemin de α à $\overline{\alpha}$ et un chemin de $\overline{\alpha}$ à α, alors F est satisfiable.

❒ 31 – Soit F une formule logique sous forme *NC2*. Décrire un algorithme s'appuyant sur les questions précédentes permettant de savoir si F est ou non satisfiable et qui, dans le cas où F est satisfiable, calcule des valeurs des variables pour lesquelles F vaut *vrai*. On utilisera une fonction récursive et on indiquera pourquoi le calcul se termine. On ne justifiera pas ici cet algorithme.

❒ 32 – Il s'agit de programmer l'algorithme, nommé *satisfiable*, décrit dans la question ❒ 31.

Caml : Écrire en Caml une fonction `satisfiable` telle que si :

- `F` est une valeur de type `Formule`,
- `solution` est un tableau pour $2 \times$ `F.nb_var` variables booléennes,

alors `satisfiable F solution`

- renvoie une valeur booléenne indiquant si la formule F décrite par `F` est ou non satisfiable,
- si F est satisfiable, remplit le tableau `solution` pour qu'il contienne, après exécution de la fonction, les valeurs des littéraux pour des valeurs des variables donnant la valeur *vrai* à F.

Pascal : Écrire en Pascal une fonction `satisfiable` telle que si :

- `F` est une valeur de type `Formule`,
- `solution` est un tableau de type `Valeurs`,

alors `satisfiable(F, solution)`

- renvoie une valeur booléenne indiquant si la formule F décrite par `F` est ou non satisfiable,
- si F est satisfiable, remplit le tableau `solution` pour qu'il contienne, après exécution de la fonction, les valeurs des littéraux pour des valeurs des variables donnant la valeur *vrai* à F.

❒ 33 – Indiquer la complexité dans le pire des cas de l'algorithme *satisfiable* définie précédemment. On exprimera cette complexité en fonction du nombre p de variables et du nombre q de clauses de la formule F à laquelle est appliquée la fonction *satisfiable*.

FIN DU PROBLÈME D'ALGORITHMIQUE, LOGIQUE ET PROGRAMMATION

Téléchargé gratuitement sur www.Doc-Solus.fr .