ÉCOLE POLYTECHNIQUE – ÉCOLES NORMALES SUPÉRIEURES

CONCOURS D'ADMISSION 2013 FILIÈRE MP SPÉCIALITÉ INFO

COMPOSITION D'INFORMATIQUE – A – (XULC)

(Durée : 4 heures)

L'utilisation des calculatrices **n'est pas autorisée** pour cette épreuve.
Le langage de programmation choisi par le candidat doit être spécifié en tête de la copie.

⋆ ⋆ ⋆

Logique temporelle

On étudie dans ce problème un formalisme logique, la *logique temporelle*, permettant de définir des formules auxquelles sont associés des langages de mots. Ainsi, pour toute formule φ de la logique temporelle et pour tout mot u on définira la propriété que le mot u satisfait la formule φ, et à toute formule φ on associera l'ensemble L_φ des mots qui satisfont φ. L'objet principal de ce problème est de s'intéresser aux propriétés de ces langages L_φ.

La partie I introduit la logique temporelle et donne des exemples de formules. La partie II introduit une forme normale pour les formules. La partie III est consacrée à montrer que pour toute formule l'ensemble de mots associés est un langage rationnel. Enfin, la partie IV étudie d'une part problème de la satisfiabilité d'une formule (étant donnée une formule φ, existe-t-il un mot satisfaisant φ ?) et d'autre part l'expressivité des formules.

Les parties peuvent être traitées indépendamment. Néanmoins, chaque partie utilise des notations et des fonctions introduites dans les parties précédentes.

La complexité, ou le coût, d'un programme P (fonction ou procédure) est le nombre d'opérations élémentaires (addition, soustraction, multiplication, division, affectation, etc...) nécessaires à l'exécution de P. Lorsque cette complexité dépend d'un paramètre n, on dira que P a une complexité en $\mathcal{O}(f(n))$, s'il existe $K > 0$ tel que la complexité de P est au plus $Kf(n)$, pour tout n. Lorsqu'il est demandé de garantir une certaine complexité, le candidat devra justifier cette dernière si elle ne se déduit pas directement de la lecture du code.

Partie I. Préliminaires

Un **alphabet** est un ensemble fini A dont les éléments sont appelés **lettres**. Un **mot** sur un alphabet A est une suite finie d'éléments de A ; on notera ε le **mot vide** (c'est-à-dire la suite de longueur nulle) et on définira la **longueur** $|u|$ d'un mot non vide $u = a_0 \cdots a_{\ell-1}$ comme valant ℓ.

Si A est un alphabet, on notera A^* l'ensemble des mots sur A et $A^+ = A^* \setminus \{\varepsilon\}$ l'ensemble des mots non vides sur A.

Dans la suite, les lettres d'un mot de longueur ℓ sont indicées de 0 à $\ell - 1$.

En Caml, nous représenterons les mots à l'aide du type `string`. Les fonctions suivantes pourront être utilisées dans la suite.

- Si `mot` est de type `string` et `i` est de type `int` alors `mot.[i]` est de type `char` et donne la lettre d'indice `i` de `mot`. Par exemple : `"bonjour".[3]` renvoie `'j'`.
- `string_length: string -> int` renvoie la longueur d'un mot.

En Pascal, nous représenterons les mots à l'aide du type `string`. Les fonctions suivantes pourront être utilisées dans la suite.

- Si `mot` est de type `string` et `i` est de type `integer` alors `mot[i]` est de type `char` et donne la lettre d'indice `i` de `mot` **avec la convention que la première lettre est d'indice 1**. Par exemple : `mot:='bonjour';mot[4]` renvoie `'j'`.
- Si `mot` est de type `string`, `length(mot)` donne la longueur de `mot`.

Dans tout le problème on se fixe un alphabet fini A (on pourra imaginer qu'il s'agit des lettres minuscules de l'alphabet usuel). On souhaite définir des ensembles de mots sur l'alphabet A à l'aide de formules logiques. Pour cela, on définit, pour chaque lettre $a \in A$, un prédicat p_a, qui permettra de tester si la lettre à une position donnée est un a. Pour construire des formules à partir de ces prédicats, on utilise les connecteurs Booléeens $\vee$ (ou), $\wedge$ (et) et $\neg$ (non) ainsi que les **connecteurs temporels X** (juste après), **G** (désormais), **F** (un jour) et **U** (jusqu'à).

Les **formules de la logique temporelle** sont alors construites par induction comme suit. Si p_a est un prédicat, et si φ, ψ sont des formules de la logique temporelle, toutes les formules seront construites selon la syntaxe suivante :

1. **VRAI**
2. p_a
3. $(\neg\varphi)$
4. $(\varphi \vee \psi)$
5. $(\varphi \wedge \psi)$
6. $(\mathbf{X}\varphi)$
7. $(\mathbf{G}\varphi)$
8. $(\mathbf{F}\varphi)$
9. $(\varphi\, \mathbf{U}\, \psi)$

Toutes les formules seront construites de la façon précédente, en omettant les parenthèses si celles-ci sont inutiles. Par exemple, $\mathbf{X}p_b$, $p_a\, \mathbf{U}\, p_b$ et $\mathbf{F}(\mathbf{G}p_a)$ sont des formules.

Nous allons maintenant définir le **sens** (ou la sémantique) des formules.

Soit un mot u et soit une formule de la logique temporelle φ. On définit, pour tout $i \geq 0$, la propriété "*le mot u satisfait la formule φ à la position i*", ce qui sera noté $(u, i) \models \varphi$, comme suit. Si $i \geq |u|$, on n'a pas $(u, i) \models \varphi$: une formule ne peut être vraie qu'à une position du mot ; en particulier le mot vide ne satisfait **aucune formule** (pas même la formule **VRAI**). Si $i \leq |u| - 1$, on note $u = a_0 \cdots a_{|u|-1}$ et on raisonne alors par induction sur la structure de φ.

1. $(u, i) \models \mathbf{VRAI}$.
2. $(u, i) \models p_a$ si et seulement si $a_i = a$.
3. $(u, i) \models (\neg\varphi)$ si et seulement si l'on n'a pas $(u, i) \models \varphi$.
4. $(u, i) \models (\varphi \vee \psi)$ si et seulement si $(u, i) \models \varphi$ ou $(u, i) \models \psi$.
5. $(u, i) \models (\varphi \wedge \psi)$ si et seulement si $(u, i) \models \varphi$ et $(u, i) \models \psi$.
6. $(u, i) \models (\mathbf{X}\varphi)$ si et seulement si $(u, i + 1) \models \varphi$. Notez que si $i = |u| - 1$, alors on ne peut avoir $(u, i) \models (\mathbf{X}\varphi)$.
7. $(u, i) \models (\mathbf{G}\varphi)$ si et seulement si $(u, j) \models \varphi$ pour tout $i \leq j \leq |u| - 1$.
8. $(u, i) \models (\mathbf{F}\varphi)$ si et seulement si $\exists j$ tel que $i \leq j \leq |u| - 1$ et $(u, j) \models \varphi$.
9. $(u, i) \models (\varphi \,\mathbf{U}\, \psi)$ si et seulement si $\exists j$ tel que $i \leq j \leq |u| - 1$, $(u, j) \models \psi$ et $(u, k) \models \varphi$ pour tout k tel que $i \leq k < j$.

On notera $(u, i) \not\models \varphi$ si et seulement si l'on n'a pas $(u, i) \models \varphi$. On notera $u \models \varphi$ (et on dira alors que φ est vraie pour u) le fait que $(u, 0) \models \varphi$, et on notera $u \not\models \varphi$ le fait que $(u, 0) \not\models \varphi$.

Par exemple :
- $(aaabcbab, 2) \models \mathbf{X}p_b$ car la lettre d'indice 3 de *aaabcbab* est un b.
- $aaabcbab \models p_a \,\mathbf{U}\, p_b$ car la lettre d'indice 3 est un b et toutes les lettres qui la précèdent sont des a.
- $aaabcbab \not\models \mathbf{F}(\mathbf{G}p_a)$ car il n'existe pas d'indice tel qu'à partir de cet indice il n'y ait plus que des a (en effet, la dernière lettre est un b).

Question 1 On pose $u = bbbcbbaa$. Pour chacun des énoncés ci-dessous dire s'il est vrai en justifiant brièvement votre réponse.

(a) $(u, 4) \models \mathbf{G}(p_a \vee p_b)$
(b) $(u, 2) \models \mathbf{X}(\mathbf{G}(p_a \vee p_c))$
(c) $(u, 1) \models \mathbf{F}(\mathbf{G}(p_a \vee p_b))$
(d) $u \models (p_a \vee p_b)\,\mathbf{U}(p_a \vee p_c)$

Question 2 Écrire une formule φ telle que $u \models \varphi$ si et seulement si u contient un a suivi plus tard d'un b. Par exemple on aura $ccacccba \models \varphi$ tandis que $ccacccaa \not\models \varphi$.

Question 3 Écrire une formule Fin telle que $(u, i) \models \text{Fin}$ si et seulement si $i = |u| - 1$ (c'est-à-dire si et seulement si i est l'indice de la dernière lettre de u).

Dans la suite, on pourra utiliser Fin comme une boîte noire.

Question 4 Écrire une formule φ telle que $u \models \varphi$ si et seulement si u se termine par un a.

Question 5 Écrire une formule φ telle que $u \models \varphi$ si et seulement si $u = ababab \cdots ab$, c'est-à-dire si et seulement s'il existe $k \geq 1$ tel que $u = u_0 u_1 \cdots u_{2k-1}$ et, pour tout i tel que $0 \leq i \leq 2k - 1$ on a $u_i = a$ si i est pair et $u_i = b$ si i est impair.

Question 6 Pour cette question, on pose $A = \{a, b, c\}$. Soit $\varphi = \mathbf{F}(p_a \wedge \mathbf{X}(\mathbf{G}(\neg p_a)) \wedge \mathbf{F}(p_b \wedge \mathbf{X}p_c))$. Décrire un automate fini (pouvant être non-déterministe) reconnaissant le langage $L_\varphi = \{u \in A^+ \mid \varphi \models u\}$.

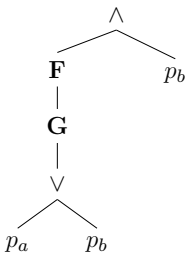
Deux formules φ et ψ sont **équivalentes**, ce que l'on note $\varphi \equiv \psi$, si pour tout mot u, on a $u \models \varphi$ si et seulement si $u \models \psi$. Autrement dit, les formules φ et ψ sont vraies pour les mêmes mots.

Question 7 Soient φ et ψ deux formules quelconques. Montrer que l'on a $\varphi \,\mathbf{U}\, \psi \equiv \psi \vee (\varphi \wedge (\mathbf{X}(\varphi \,\mathbf{U}\, \psi)))$.

Partie II. Normalisation de formules

Afin de manipuler des formules de la logique temporelle, on va représenter ces dernières par des arbres. Outre les prédicats $(p_a)_{a\in A}$ et la formule **VRAI** nous avons des connecteurs unaires ($\neg$, **X**, **G** et **F**) et des connecteurs binaires ($\vee$, $\wedge$ et **U**). À la manière des expressions arithmétiques, on représente une formule de la logique temporelle par un arbre dont les feuilles sont étiquetées soit par un prédicat p_a soit par **VRAI**, et dont les nœuds internes sont étiquetés par un connecteurs unaire (un tel nœud aura alors un seul fils) ou par un connecteur binaire (un tel nœud aura alors deux fils). Dans la suite, on confondra fréquemment une formule avec sa représentation par un arbre.

Par exemple, la formule $(\mathbf{F}(\mathbf{G}(p_a \vee p_b))) \wedge p_b$ sera représentée par l'arbre ci-dessous :



Le type des formules est défini comme suit :

```
        (* Caml *)
type formule =
   |VRAI
   |Predicat of char
   |NON of formule
   |ET of formule * formule
   |OU of formule * formule
   |X of formule
   |G of formule
   |F of formule
   |U of formule * formule;;
```

```
                  { Pascal }
Type
   typeFormule = (VRAI,Predicat,NON,ET,OU,X,G,F,U);
   formule     = ^rformule;
   rformule    = Record
                   case t : typeFormule of
                     VRAI      : ();
                     Predicat  : (c : char);
                     NON,X,G,F : (h : formule);
                     U,ET,OU   : (d,g : formule);
                   End;
```

En Pascal, on pourra utiliser sans les programmer les fonctions suivantes :

```
Function FVrai() : formule;
```

```
Function FP(c: Char) : formule;
Function FUn(t: typeFormule; h: formule) : formule;
Function FBin(t: typeFormule; g,d: formule) : formule;
```

La fonction `FVrai` renvoie la formule **VRAI**. La fonction `FP` appelée avec une lettre `'c'` renvoie la formule p_c. La fonction `FUn` appelée avec `NON` (resp. `X`,`G` et `F`) et une formule φ renvoie la formule $\neg\varphi$ (resp. $\mathbf{X}\varphi$, $\mathbf{G}\varphi$ et $\mathbf{F}\varphi$). La fonction `FBin` appelée avec `U` (resp. `ET` et `OU`) et deux formules φ et ψ renvoie la formule $\varphi U\psi$ (resp. $\varphi \wedge \psi$ et $\varphi \wedge \psi$). Par exemple, `FUn(X,FBin(U,FVrai(),FP('a')))` renvoie la formule $\mathbf{X}(\mathbf{VRAI}\,\mathbf{U}\,p_a)$.

On définit la taille d'une formule par le nombre de nœuds de l'arbre qui la représente. En particulier, la formule **VRAI** et les formules réduites à un prédicat p_a sont de taille 1.

Question 8 Écrire une fonction `taille` qui prend en argument une `formule` et renvoie sa taille.

```
(* Caml *)  taille: formule -> int
{ Pascal }  taille(phi: formule) : integer;
```

Question 9 Donner, pour toute formule φ n'utilisant pas le connecteur **F**, une formule équivalente à $\mathbf{F}\varphi$ qui n'utilise pas le connecteur temporel **F**. En déduire une fonction `normaliseF` qui prend en entrée une `formule` quelconque et renvoie une `formule` équivalente qui n'utilise pas le connecteur **F**. Quelle est la complexité de votre algorithme ?

```
(* Caml *)  normaliseF: formule -> formule
{ Pascal }  normaliseF(phi: formule) : formule;
```

Une formule qui utilise comme seuls connecteurs temporels le **X** et le **U** sera qualifiée de **normalisée**.

Question 10 Montrer que toute formule est équivalente à une formule normalisée. Donner une fonction `normalise` de complexité linéaire qui prend en entrée une `formule` quelconque et renvoie une `formule` équivalente normalisée.

```
(* Caml *)  normalise: formule -> formule
{ Pascal }  normalise(phi: formule) : formule;
```

Dans toute la suite du problème, on supposera que l'on travaille avec des **formules normalisées**.

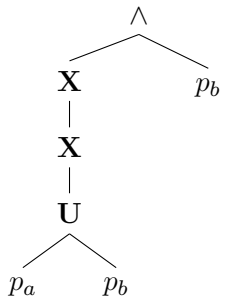
Question 11 Écrire une fonction récursive `veriteN` qui prend en argument un mot u, un indice i et une `formule` (normalisée) φ et détermine si $(u, i) \models \varphi$. Justifier que votre fonction termine et indiquer si elle est exponentielle ou polynomiale en la taille de ses arguments.

```
(* Caml *)  veriteN: formule -> string -> int -> bool
{ Pascal }  veriteN(phi: formule; v: string; i: integer) : boolean;
```

Partie III. Rationalité des langages décrits par des formules

Le but de cette partie est de montrer qu'étant donnée une formule φ, l'ensemble des mots pour lesquels φ est vraie est un langage rationnel.

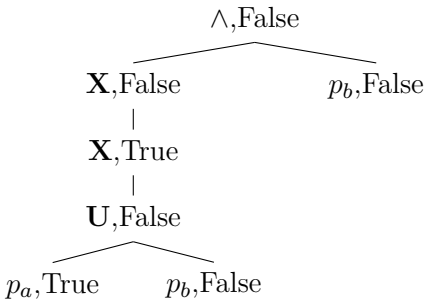
Soit φ une formule représentée par un arbre t_φ. Une sous-formule de φ est une formule représentée par un sous-arbre de t_φ. Par exemple si l'on considère la formule $\varphi = (\mathbf{X}(\mathbf{X}(p_a \,\mathbf{U}\, p_b))) \wedge p_b$, représentée par l'arbre ci-dessous,



l'ensemble de ses sous-formules est :

$$\{\ (\mathbf{X}(\mathbf{X}(p_a \,\mathbf{U}\, p_b))) \wedge p_b\ ,\ \mathbf{X}(\mathbf{X}(p_a \,\mathbf{U}\, p_b))\ ,\ \mathbf{X}(p_a \,\mathbf{U}\, p_b)\ ,\ p_a \,\mathbf{U}\, p_b\ ,\ p_b\ ,\ p_a\ \}$$

On souhaite représenter des ensembles de sous-formules d'une formule donnée. Pour cela, on va utiliser des arbres dont les nœuds sont étiquetés (en plus des symboles de la logique) par des booléens. Par exemple pour la formule $(\mathbf{X}(\mathbf{X}(p_a \,\mathbf{U}\, p_b))) \wedge p_b$ représentée ci-dessus, l'ensemble de sous-formules $\{\ p_a\ ,\ \ \mathbf{X}(p_a \,\mathbf{U}\, p_b)\}$ sera représenté par l'arbre ci-dessous.



Une sous-formule ψ appartient à l'ensemble représenté par un arbre de type `ensemble` s'il existe un nœud de l'arbre étiqueté par True et dont le sous-arbre "correspond" à la formule ψ (c'est-à-dire qu'après suppression des booléens, le sous-arbre est égal à l'arbre codant ψ). Le type `ensemble` est défini comme suit.

```
(* Caml *)
type ensemble ==  eformule * bool
and eformule =
| AVRAI
| APredicat of char
| ANON of ensemble
| AET of ensemble * ensemble
| AOU of ensemble * ensemble
| AX of ensemble
| AU of ensemble * ensemble;;
```

```
{ Pascal }
ensemble  = ^rensemble;
rensemble =
  Record
   b      : Boolean;
   case t : typeFormule of
    VRAI : ();
    Predicat : (c : Char);
    NON,X,G,F : (h : ensemble);
    U,ET,OU   : (d,g : ensemble);
   End;
```

On rappelle que l'on ne travaille plus qu'avec des formules normalisées.

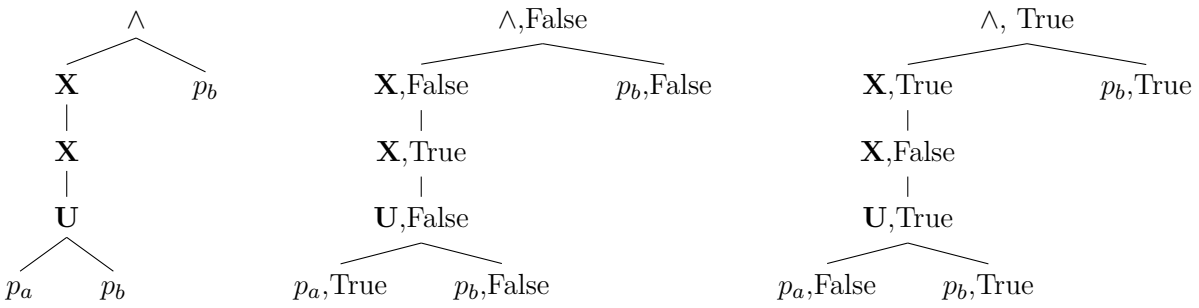
Question 12 Écrire une fonction `initialise` qui prend en argument une `formule` φ et renvoie un `ensemble` représentant l'ensemble vide de sous-formules de φ.

```
(* Caml *)  initialise: formule -> ensemble
{ Pascal }  initialise(phi: formule) : ensemble;
```

Question 13 Soient u un mot, φ une formule et S l'ensemble des sous-formules de φ vraies pour u. Soit a une lettre. Montrer que l'ensemble S' des sous-formules de φ vraies pour $a \cdot u$ peut être déterminé uniquement en fonction de a et de S (en particulier indépendamment de u).

Question 14 En vous basant sur la question 13, écrire une fonction `maj` qui prend en argument un `ensemble` S (représentant un ensemble de sous-formules d'une formule φ) et une lettre a et renvoie un `ensemble` S' tel que : pour tout mot u si S représente l'ensemble $\{\psi \mid u \models \psi$ et ψ sous-formule de $\varphi\}$ des sous-formules de φ vraies pour u alors S' représente l'ensemble $\{\psi \mid a \cdot u \models \psi$ et ψ sous-formule de $\varphi\}$ des sous-formules de φ vraies pour $a \cdot u$. Justifier la terminaison et préciser la complexité (dans la taille de la formule) de votre fonction.

Par exemple si l'on reprend l'exemple de la formule $\varphi = (\mathbf{X}(\mathbf{X}(p_a \,\mathbf{U}\, p_b))) \wedge p_b$ (arbre à gauche ci-dessous) et de l'`ensemble` S (au centre ci-dessous), alors `maj` φ `'b'` devra renvoyer l'`ensemble` S' (à droite ci-dessous).



```
(* Caml *)  maj: ensemble -> char -> ensemble
{ Pascal }  maj(phi: ensemble; c: Char) : ensemble;
```

Question 15 Écrire une fonction `sousFormulesVraies` qui prend en argument une `formule` φ et un mot et renvoie un `ensemble` décrivant les sous-formules ψ de φ telles que $u \models \psi$. Votre fonction devra avoir une complexité polynomiale dans la taille de la formule et dans la taille du mot. Justifier la terminaison et préciser la complexité (dans la taille de la formule et la taille du mot).

```
(* Caml *)  sousFormulesVraies: formule -> string -> ensemble
{ Pascal }  sousFormulesVraies(phi: formule; v: string) : ensemble;
```

Question 16 En déduire une fonction `veriteBis` qui teste si une formule donnée est vraie à la première position d'un mot donné.

```
(* Caml *)  veriteBis: formule -> string -> bool
{ Pascal }  veriteBis(phi: formule; v: String) : Boolean;
```

Soit φ une formule. On associe à φ un langage de mots $L_\varphi \subseteq A^+$ en posant

$$L_\varphi = \{u \in A^+ \mid u \models \varphi\}$$

Soit un mot $u = a_0 \cdots a_{\ell-1}$. On note $\widetilde{u}$ le mot miroir de u : $\widetilde{u} = a_{\ell-1}a_{\ell-2}\cdots a_0$. Soit un langage $L \subseteq A^+$, on notera $\widetilde{L} = \{\widetilde{u} \mid u \in L\}$.

Question 17 En vous inspirant de la fonction `maj` de la question 14, montrer que pour toute formule φ, le langage $\widetilde{L_\varphi}$ est reconnu par un automate déterministe complet. Donner un majorant du nombre d'états d'un tel automate.

Question 18 En déduire que pour toute formule φ, le langage L_φ est reconnu par un automate fini. Donner un majorant du nombre d'états d'un tel automate (automate qui pourra être pris non-déterministe).

Partie IV. Satisfiabilité et expressivité

On rappelle que désormais les formules considérées sont normalisées (elles n'utilisent donc ni le **F** ni le **G**). Dans toute cette partie, on considérera que $A = \{a, b\}$ sauf mention explicite.

Le but de cette partie est dans un premier temps d'écrire un programme qui prend en entrée une `formule` φ et renvoie `true` si et seulement s'il existe $u \in A^+$ tel que $u \models \varphi$. Dans un second

temps, on montre qu'il existe un langage accepté par un automate fini qui ne peut être décrit par aucune formule de la logique temporelle.

Question 19 Soit $\mathcal{A}$ un automate fini déterministe sur l'alphabet A. Décrire informellement un algorithme qui calcule la liste des états atteignables depuis l'état initial (c'est-à-dire l'ensemble des états q tels qu'il existe un mot qui lu depuis l'état initial termine dans q).

Question 20 Soit φ une formule satisfiable, c'est-à-dire pour laquelle existe un mot $u \in A^+$ tel que $u \models \varphi$. Soit u_{min} un plus court u tel que $u \models \varphi$. Majorer la longueur de u_{min} en fonction de la taille de φ.

Pour la question suivante, on pourra utiliser des listes de couples formés d'un ensemble et d'une chaîne de caractères. En Caml, une telle liste aura le type `(ensemble*string) list`. En Pascal, une telle liste aura le type `listeEnsembles` donné par :

```
listeEnsembles  = ^rlisteEnsembles;
rlisteEnsembles = Record
   Mot : string;
   Element : ensemble;
   Suivant : listeEnsembles;
 End;
```

En Pascal, on pourra utiliser sans la programmer une fonction `cons(mot : string; ens : ensemble; L : listeEnsembles): listeEnsembles` qui renvoie la liste obtenue en ajoutant le couple `(mot,ens)` en tête de la liste `L`. L'appel à cette fonction a un coût constant.

Question 21 Écrire une fonction `satisfiable` qui prend en argument une `formule` φ et renvoie soit la chaîne `"Formule non satisfiable"` (Caml) ou la chaîne `'Formule non satisfiable'` (Pascal) s'il n'existe pas de $u \in A^+$ tel que $u \models \varphi$ et sinon renvoie un $u \in A^+$ tel que $u \models \varphi$. La complexité de votre fonction devra être en $\mathcal{O}(2^{\alpha|\varphi|^\beta})$ où α et β sont deux constantes que vous préciserez.
Il est permis de décomposer la réponse en quelques fonctions auxiliaires. On rappelle ici que l'on suppose que $A = \{a, b\}$. Donner dans un premier temps les idées clés de l'algorithme avant d'écrire le code.

```
(* Caml *)  satisfiable : formule -> string
{ Pascal }  satisfiable(phi: formule) : string;
```

Question 22 Pour cette question, on pose $A = \{a\}$ et on note a^i le mot formé de i lettres a, par exemple $a^4 = aaaa$. Montrer qu'il n'existe pas de formule φ telle que $L_\varphi = \{a^{2i} \mid i \geq 1\}$. On pourra montrer que pour toute formule φ, il existe $N \geq 0$ tel que l'on ait l'une des deux situations suivantes :

- pour tout $n \geq N$, $a^n \models \varphi$.
- pour tout $n \geq N$, $a^n \not\models \varphi$.

* *
*