Concours commun Mines-Ponts, France

Session 2026, filière MPI

Informatique 1

Durée : 3 heures

Ce que traite le sujet

- Section 1 : implémentation de quelques fonctions utiles
- Section 2 : automates finis sur Sigma_n
- Section 3 : opérations arithmétiques
- Section 4 : arithmétique de Presburger
- Section 5 : retour sur les automates

Sujet reproduit sans retouche à partir de la page suivante.

Résumé des parties, chapitres de cours liés et autres années :
prepahelp.com/annales/mines/mpi/2026/info1

Prepa Help ne revendique aucun droit sur le sujet, propriété de l'organisateur du concours.

A2026 – INFO I MPI

ÉCOLE NATIONALE DES PONTS et CHAUSSÉES,
ISAE-SUPAERO, ENSTA,
TÉLÉCOM PARIS, MINES PARIS - PSL,
MINES SAINT-ÉTIENNE, MINES NANCY,
IMT ATLANTIQUE, ENSAE PARIS,
CHIMIE PARISTECH - PSL.

Concours Mines-Télécom,
Concours Centrale-Supélec (Cycle International).

CONCOURS 2026

PREMIÈRE ÉPREUVE D'INFORMATIQUE

Durée de l'épreuve : 3 heures

L'usage de la calculatrice ou de tout dispositif électronique est interdit.

Les candidats sont priés de mentionner de façon apparente sur la première page de la copie :

INFORMATIQUE I - MPI

Cette épreuve concerne uniquement les candidats de la filière MPI.

L'énoncé de cette épreuve comporte 9 pages de texte.

Si, au cours de l'épreuve, un candidat repère ce qui lui semble être une erreur d'énoncé, il le signale sur sa copie et poursuit sa composition en expliquant les raisons des initiatives qu'il est amené à prendre.

Les sujets sont la propriété du GIP CCMP. Ils sont publiés sous les termes de la licence Creative Commons Attribution - Pas d'Utilisation Commerciale - Pas de Modification 3.0 France.

Tout autre usage est soumis à une autorisation préalable du Concours commun Mines-Ponts.

Préliminaires

L'épreuve est composée d'un problème unique, comportant 30 questions. Le problème est divisé en cinq sections. Dans la première section (page 2), nous implémentons plusieurs fonctions mathématiques. Dans la deuxième section (page 3), nous étudions des automates sur un alphabet particulier et nous nous intéressons à une implémentation. Dans la troisième section (page 5), nous nous intéressons à la reconnaissabilité de certains langages sous cet alphabet. Dans la quatrième section (page 5), nous nous intéressons à l'arithmétique de Presburger et ses formules logiques. Enfin, dans la cinquième et dernière section (page 7), nous cherchons à décider de la satisfiabilité de ces formules à l'aide des automates précédemment étudiés.

Dans tout l'énoncé, un même identificateur écrit dans deux polices de caractères différentes désigne la même entité, mais du point de vue mathématique pour la police en italique (par exemple n) et du point de vue informatique pour celle en romain avec espacement fixe (par exemple `n`).

Des rappels portant sur les règles de la déduction naturelle sont fournis en annexe.

Travail attendu

Pour répondre à une question, il est permis de réutiliser le résultat d'une question antérieure, même sans avoir réussi à établir ce résultat.

Il faudra coder des fonctions à l'aide du langage de programmation OCaml exclusivement. Les fonctions des modules `List` et `Array` peuvent être utilisées sans explication. Il est cependant recommandé de s'en tenir aux fonctions les plus courantes afin de rester compréhensible.

Le barème tient compte de la clarté et de la concision des programmes. Nous recommandons de choisir des noms de variables intelligibles ou encore de structurer de longs codes par des blocs ou par des fonctions auxiliaires dont on décrit le rôle.

Introduction

On note $\mathbb{N}$ l'ensemble des entiers naturels. Pour tout couple d'entiers a, b tel que $a \leqslant b$, on note $[\![a, b]\!]$ l'ensemble des entiers de a à b compris. Pour tout ensemble E, on note $\mathcal{P}(E)$ l'ensemble des parties de E, c'est-à-dire l'ensemble des sous-ensembles de E. Pour tout ensemble fini E, on note $|E|$ son cardinal.

Pour tout mot m, on note $|m|$ la taille de m, qui correspond à son nombre de lettres.

Dans tout le sujet, pour tout entier n positif, on notera Σ_n l'alphabet composé des vecteurs colonnes de taille n à valeurs dans $\{0, 1\}$. Par exemple,

$$\Sigma_2 = \left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \end{pmatrix} \right\}.$$

Pour tout entier i dans $[\![1, n]\!]$, et pour tout mot m sur Σ_n, on note $\pi_i(m)$ la projection de m sur la composante i, c'est-à-dire, le mot sur $\{0, 1\}$ composé du terme à la i-ème ligne sur chaque lettre de m.

Par abus de notation, on s'autorise à confondre la lettre 0 et 1 de l'alphabet $\{0, 1\}$ avec les entiers 0 et 1. Pour tout mot u de taille $|u| = k$ sur $\{0, 1\}$ tel que $u = u_0 \dots u_{k-1}$, on va noter $v(u)$ la valeur suivante :

$$v(u) = \sum_{j=0}^{k-1} 2^j u_j$$

Autrement dit, $v(u)$ est l'entier dont u en est une de ses représentations en binaire telles que le bit de poids faible soit à gauche, donc à l'inverse du sens habituel.

Pour un mot m sur Σ_n et un entier i entre 1 et n, on note $v_i(m) = v(\pi_i(m))$.

Considérons par exemple le mot m suivant :

$$m = \begin{pmatrix}0\\1\\1\end{pmatrix}\begin{pmatrix}1\\0\\1\end{pmatrix}\begin{pmatrix}1\\0\\0\end{pmatrix}\begin{pmatrix}0\\0\\0\end{pmatrix}\begin{pmatrix}1\\1\\1\end{pmatrix}\begin{pmatrix}0\\1\\1\end{pmatrix}\begin{pmatrix}1\\0\\1\end{pmatrix}\begin{pmatrix}1\\0\\0\end{pmatrix}$$

On a $\pi_2(m) = 10001100$, $v_1(m) = 2 + 4 + 16 + 64 + 128 = 214$, $v_2(m) = 49$ et $v_3(m) = 115$.

Dans ce sujet, un *automate fini non déterministe à un seul état initial*, qu'on s'autorise à appeler simplement automate, est défini comme un tuple $\mathcal{A} = (\Sigma, Q, q_0, F, \delta)$ où :

— Σ est un alphabet fini ;

— Q est un ensemble fini d'états ;

— q_0 est un état de Q appelé état initial ;

— F est un sous-ensemble de Q appelé ensemble des états finaux ;

— δ est une fonction de $Q \times \Sigma$ dans $\mathcal{P}(Q)$ appelée fonction de transition.

On dit qu'un automate $\mathcal{A} = (\Sigma, Q, q_0, \delta, F)$ est déterministe *complet* lorsque pour chaque sommet $q \in Q$, et chaque lettre $a \in \Sigma$, on a $|\delta(q, a)| = 1$.

Pour deux états p et q de Q, et pour toute lettre a de Σ, on note $p \xrightarrow{a} q$ si $q \in \delta(p, a)$. On dit qu'il existe une transition de p à q étiquetée par a. On prend comme convention graphique d'indiquer l'état initial par une flèche entrante et les états finaux en les entourant.

Pour un automate $\mathcal{A}$ on note $\mathcal{L}(\mathcal{A})$ le langage reconnu par $\mathcal{A}$.

Dans tout le sujet, on fera l'hypothèse simplificatrice qu'en OCaml, on dispose d'une mémoire infinie et que les variables peuvent stocker des valeurs aussi grandes que l'on souhaite.

1. Implémentation de quelques fonctions utiles

Commençons par implémenter plusieurs fonctions mathématiques qui nous seront utiles dans la suite.

☐ 1 – Écrire une fonction `pow2 : int -> int` qui prend en entrée un entier n et renvoie 2^n. On attend une fonction récursive de complexité temporelle en $O(\log(n))$.

On représente en OCaml les lettres de Σ_n à l'aide du type `int array` en utilisant des tableaux de dimension n dont les éléments sont à valeurs dans $\{0, 1\}$.

☐ 2 – Exhiber une bijection entre $[\![0, 2^n - 1]\!]$ et Σ_n.

À présent, on souhaite implémenter notre bijection.

☐ 3 – Écrire une fonction `decompose : int -> int -> int array` qui prend en entrée `x` et `n` tels que $x \in [\![0, 2^n - 1]\!]$ et qui renvoie l'élément de Σ_n associée par la bijection précédemment définie. On vérifiera que l'entrée est valide à l'aide d'une assertion.

2. Automates finis sur Σ_n

Considérons les automates $\mathcal{A}_1$ et $\mathcal{A}_2$ suivants :

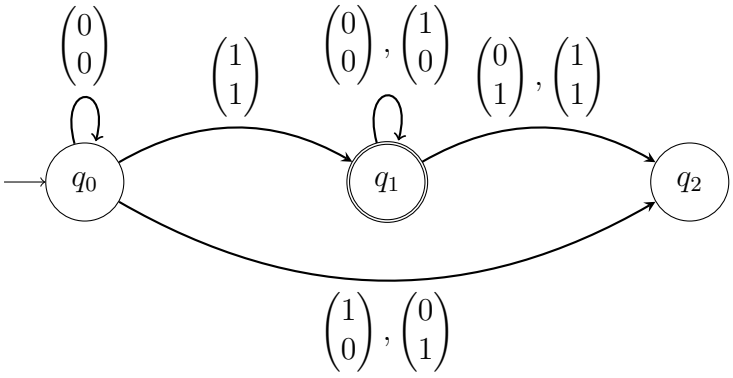


FIGURE 1 – Représentation de l'automate $\mathcal{A}_1$

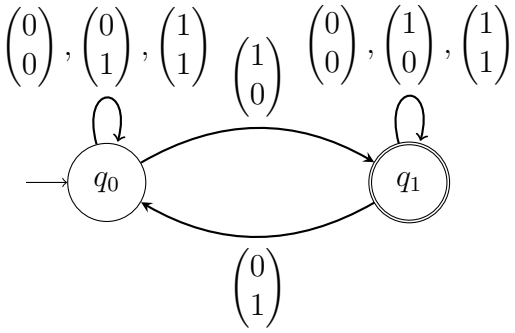


FIGURE 2 – Représentation de l'automate $\mathcal{A}_2$

Soit E l'ensemble des mots m sur Σ_2 tels que $v_2(m)$ est la plus grande puissance de 2 qui divise $v_1(m)$.

☐ 4 – Démontrer rigoureusement que $\mathcal{L}(\mathcal{A}_1) = E$.

☐ 5 – Montrer que pour tout $m \in \mathcal{L}(\mathcal{A}_1), v_1(m) \equiv v_2(m)[2 \times v_2(m)]$.

☐ 6 – Décrire sans justifier le langage $\mathcal{L}(\mathcal{A}_2)$ en donnant une condition nécessaire et suffisante pour qu'un mot m sur Σ_2 soit dans le langage qui fait intervenir $v_1(m)$ et $v_2(m)$.

☐ 7 – Proposer sans justifier un automate à 3 états qui reconnaît le langage L défini par :

$$L = \{m \in \Sigma_3^* \mid v_1(m) = v_3(m) \text{ ou } v_2(m) = v_3(m)\}$$

.

Pour implémenter les automates, on utilise le type `automate` suivant en OCaml. Un automate sur Σ_n à k états est représenté en utilisant un entier différent dans $[\![0, k-1]\!]$ pour chaque état. L'état initial est toujours l'état 0.

```
type automate = {
    k : int ;
    n : int ;
    finaux : bool array ;
    delta : int -> int array -> int list ;
}
```

Ici, `k` correspond donc au nombre d'états de l'automate, `n` correspond à n, `finaux` est un tableau de booléens de taille k qui indique si un état est final, et `delta` est la fonction de transition prenant en paramètre un état q et une lettre a, représentée par un tableau d'entiers de taille n, et renvoie la liste des états de $\delta(q, a)$

☐ 8 – Écrire une fonction `est_deterministe_complet : automate -> bool` qui vérifie si un automate $\mathcal{A} = (\Sigma_n, Q, q_0, \delta, F)$ est déterministe complet. On attend une complexité temporelle en $O(n \times |Q| \times |\Sigma_n|)$. Justifier la complexité obtenue.

On se propose maintenant d'implémenter l'algorithme de déterminisation des automates.

☐ 9 – Soit $\mathcal{A} = (\Sigma, Q, q_0, \delta, F)$ un automate fini non déterministe. Donner la construction formelle d'un automate déterministe reconnaissant $\mathcal{L}(\mathcal{A})$.

☐ 10 – Écrire une fonction `union : int array -> automate -> int array -> int array` qui prend en entrée les paramètres `etats`, `a` et `lettre`, tels que :

- `a` soit un automate à k états ;
- `etats` soit un tableau d'entiers de taille k à valeur dans $\{0, 1\}$;
- `lettre` soit une lettre représentée par un tableau d'entiers.

Cette fonction doit renvoyer un tableau `t` de taille k, tel que `t.(i)=1` s'il existe un q tels que `etats.(q)=1` et $q \xrightarrow{\texttt{lettre}} i$ est une transition de `a`, et `t.(i)=0` sinon.

☐ 11 – En déduire une fonction `determinise : automate -> automate` qui prend en entrée un automate et renvoie un automate déterministe complet reconnaissant le même langage.
Indication : On pourra utiliser la fonction `decompose` *plusieurs fois.*

On appelle *complémentaire* d'un langage L sur un alphabet Σ le langage $\overline{L}$ des mots sur Σ n'appartenant pas à L.

☐ 12 – Écrire une fonction `complementaire : automate -> automate` qui prend en entrée un automate et renvoie un automate reconnaissant son langage complémentaire.

☐ 13 – Considérons deux automates $\mathcal{A}_1 = (\Sigma_n, Q_1, q_1, \delta_1, F_1)$ et $\mathcal{A}_2 = (\Sigma_n, Q_2, q_2, \delta_2, F_2)$. Donner la construction formelle d'un automate ayant pour ensemble d'états $Q_1 \times Q_2$ et reconnaissant $\mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2)$. Ici $Q_1 \times Q_2$ est le produit cartésien de Q_1 et Q_2.

On suppose dans la suite disposer d'une fonction OCaml `intersection : automate -> automate -> automate` qui prend en entrée deux automates et renvoie un automate reconnaissant l'intersection de leurs langages.

3. Opérations arithmétiques

On s'intéresse à présent aux opérations arithmétiques en binaire et à la possibilité de les représenter à l'aide d'automates sur Σ_3.

☐ 14 – Donner, dans un premier temps sans justifier, un automate à deux états reconnaissant le langage $L_+ = \{m \in \Sigma_3^* \mid v_1(m) + v_2(m) = v_3(m)\}$.

☐ 15 – Démontrer que l'automate proposé dans la question précédente reconnaît bien L_+.

☐ 16 – Le langage $L_\times = \{m \in \Sigma_3^* \mid v_1(m) \times v_2(m) = v_3(m)\}$ est-il reconnaissable ? Justifier.

4. Arithmétique de Presburger

L'arithmétique de Presburger est l'arithmétique usuelle des entiers et de l'addition, mais sans la multiplication.

Soit $\mathcal{V}$ un ensemble infini de variables.

L'ensemble des *termes* est défini inductivement de la façon suivante :

— pour tout $x \in \mathcal{V}$, x est un terme ;

— pour deux termes t_1 et t_2, $t_1 + t_2$ est un terme.

L'ensemble des *formules de Presburger*, qu'on s'autorise à appeler formules, est ensuite défini inductivement de la façon suivante :

— pour t_1 et t_2 deux termes, $t_1 = t_2$ est une formule ;

— pour φ une formule, $\neg\varphi$ est une formule ;

— pour φ_1 et φ_2 deux formules $\varphi_1 \to \varphi_2$, $\varphi_1 \wedge \varphi_2$ et $\varphi_1 \vee \varphi_2$ sont des formules ;

— pour φ une formule et x une variable qui n'apparaît pas juste après un quantificateur dans φ, alors $\exists x.\varphi$ et $\forall x.\varphi$ sont des formules.

L'évaluation des formules suit le sens habituel de l'opérateur d'égalité $=$, et des opérateurs logiques. Les quantificateurs ont leur sens usuel avec des variables prises à valeurs dans l'ensemble $\mathbb{N}$. On utilise des parenthèses pour repérer les sous-formules lorsqu'il y a ambiguïté et pour indiquer les priorités d'ordre des opérations usuelles.

Pour toute formule φ, et toute sous-formule $\forall x.\psi$ ou $\exists x.\psi$ de φ, on dit que toutes les occurrences de x dans ψ sont liées. Les occurrences non liées de x dans φ sont dites libres. Si φ possède des occurrences libres de x, on dit que x est une variable libre de φ.

Dans la formule $\exists x.\ \forall y.\ (z = t + y \wedge z = x) \vee (\exists w.\ z = t + w)$, les variables x, y et w sont liées, et les variables libres sont z et t.

Pour $\overline{x}$ un vecteur de variables, on notera souvent $\varphi(\overline{x})$ pour signifier que $\overline{x}$ est l'ensemble des variables libres de φ. Une assignation est une fonction des variables libres de φ vers $\mathbb{N}$.

Pour φ une formule de Presburger et a une assignation de ses variables libres, on note $[\![\varphi]\!]_a$ la valeur de vérité de φ dans l'assignation a.

Par exemple, considérons la formule $\varphi_{\leqslant}(x, y) := \exists z.\ y = x + z$, où $:=$ est un opérateur de définition. Ici, z est une variable liée, et y et x sont des variables libres. Pour un x et un y donnés, cette formule est vraie si et seulement s'il existe un entier z dans $\mathbb{N}$ tel que $y = x + z$, donc si et seulement si $x \leqslant y$.

Pour deux formules φ et φ' ayant le même ensemble de variables libres, on dit qu'elles sont équivalentes, si pour toute assignation de ces variables libres, elles ont la même valeur de vérité.

☐ 17 – Donner une formule de Presburger $div_2(x)$ qui est vraie si et seulement si x est divisible par 2.

☐ 18 – Pour tout entier $n \in \mathbb{N}$, donner une formule de Presburger $f_n(x)$ qui est vraie si et seulement si $x = n$.
Attention, n n'est pas un terme et ne peut donc pas apparaître dans votre formule sous cette forme.

On dit qu'une formule est *sous forme simple* si elle vérifie les propriétés suivantes :

(i) Elle ne contient pas les opérateurs $\vee$ et $\rightarrow$.

(ii) Elle ne contient pas le quantificateur $\forall$.

(iii) Pour toute sous-formule de la forme $t_1 = t_2$, t_1 est une variable dans $\mathcal{V}$ et
- soit t_2 est une variable dans $\mathcal{V}$;
- soit $t_2 = x + y$ avec x et y deux variables de $\mathcal{V}$.

☐ 19 – Montrer que toute formule de Presburger φ est équivalente à une formule sous forme simple.

On veut à présent montrer que certaines formules sont des tautologies à l'aide de la déduction naturelle.

☐ 20 – Prouver à l'aide de la déduction naturelle le séquent suivant où $\varphi(x)$ désigne une formule faisant apparaître la variable libre x et $\psi(y)$ désigne une formule faisant apparaître la variable libre y.

$$\vdash (\varphi(x) \vee \psi(y)) \rightarrow (\neg\varphi(x) \rightarrow \psi(y))$$

Pour φ une formule, x une variable, et t un terme, on note $\varphi[t/x]$ la formule φ où toutes les occurrences libres de x ont été remplacées par le terme t. Pour pouvoir effectuer des preuves sur des formules plus complexes, on introduit les règles suivantes sur l'égalité pour x une variable, φ une formule et t un terme :

$$\frac{}{\Gamma \vdash t = t}\, ax_{=} \qquad\qquad \frac{\Gamma, x = t \vdash \varphi[t/x]}{\Gamma, x = t \vdash \varphi}\, sub_{=}$$

De plus, on sait que l'addition est associative et pour simplifier les preuves, on s'autorise à utiliser cette propriété dans nos arbres. Par exemple, pour t_1, t_2 et t_3 des termes, on peut considérer que $t_1 + (t_2 + t_3)$ est le même terme que $(t_1 + t_2) + t_3$ ainsi que $t_1 + t_2 + t_3$.

☐ 21 – Donner une formule sans variable libre de Presburger exprimant le fait que l'égalité est transitive sur les variables et la prouver en utilisant la déduction naturelle.

☐ 22 – Prouver à l'aide de la déduction naturelle le séquent suivant.

$$\vdash \forall x.\forall y.\forall z.\ (\exists a.\ \exists b.\ x = y + a \wedge y = z + b) \to (\exists c.\ x = z + c)$$

5. Retour sur les automates

On s'intéresse au problème suivant.
TAUTOLOGIE-PRESBURGER

- **Entrée :** Une formule de Presburger φ sans variable libre.
- **Sortie :** La valeur de vérité de cette formule.

On rappelle la définition du problème SAT-CNF.
SAT-CNF

- **Entrée :** Une formule logique sans quantificateur φ sous forme normale conjonctive.
- **Sortie :** Un booléen indiquant si φ est satisfiable.

☐ 23 – Montrer que SAT-CNF se réduit polynomialement à TAUTOLOGIE-PRESBURGER.

On veut à présent montrer que le problème TAUTOLOGIE-PRESBURGER est décidable. Pour ce faire, on veut transformer une formule de Presburger en un automate et se ramener à la question de savoir si un langage est vide ou non.

Avant de faire le lien entre automate et formules, on doit d'abord s'intéresser au problème des différentes représentations des entiers dues aux zéros non significatifs.

Soit n un entier naturel non nul et i dans $[\![1, n]\!]$. Soit L un langage sur Σ_n. On note $L_{\exists i}$ le langage défini par :

$$L_{\exists i} = \{m \in \Sigma_n^* \mid \exists w \in L, \forall k \in [\![1, n]\!], k \neq i, \pi_k(m) = \pi_k(w)\}.$$

Autrement dit, il s'agit de l'ensemble des mots égaux à un mot de L sauf possiblement à la ligne i.

☐ 24 – Soit $L = \left\{ \varepsilon\ , \begin{pmatrix}0\\1\\1\end{pmatrix}, \begin{pmatrix}0\\0\\0\end{pmatrix}\begin{pmatrix}1\\1\\1\end{pmatrix} \right\}$, donner $L_{\exists 2}$.

☐ 25 – Écrire une fonction `automate_existe : automate -> int -> automate` qui prend en entrée un automate $\mathcal{A}$ et un entier i et qui renvoie un automate reconnaissant $\mathcal{L}(\mathcal{A})_{\exists i}$.

Soit n un entier supérieur ou égal à 3. Soit, i, j et k des entiers de $[\![1, n]\!]$.

On note $L_{i,j,k}$ le langage $\{m \in \Sigma_n^* \mid v_i(m) + v_j(m) = v_k(m)\}$. On suppose disposer d'une fonction `automate_somme : int -> int -> int -> int -> automate` telle que `automate_somme n i j k` renvoie un automate reconnaissant $L_{i,j,k}$.

On note $L_{=i,j}$ le langage $\{m \in \Sigma_n^* \mid v_i(m) = v_j(m)\}$. On suppose disposer d'une fonction `automate_egalite : int -> int -> int -> automate` telle que `automate_egalite n i j` renvoie un automate reconnaissant $L_{=i,j}$.

On suppose également disposer d'une fonction `sans_zero : automate -> automate` qui prend en entrée un automate et qui renvoie une copie de cet automate dans laquelle les états finaux sont les états permettant d'accéder à un état final de l'automate initial par une suite de transitions étiquetées par la lettre $\begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix}$.

Considérons une formule sous forme simple φ utilisant n variables qu'on note $\{x_1, \ldots, x_n\}$. Soit k le nombre de variables libres de φ. Soit une fonction f strictement croissante de $[\![1, k]\!]$ dans $[\![1, n]\!]$ telle que les $\{x_{f(1)}, \ldots, x_{f(k)}\}$ est l'ensemble des variables libres de φ. Pour tout mot m de Σ_n, on note a^m l'assignation des variables libres de φ définie pour i dans $[\![1, k]\!]$ par $a^m(x_{f(i)}) = v_{f(i)}(m)$. On note L_φ le langage

$$L_\varphi = \{m \in \Sigma_n^* \mid [\![\varphi]\!]_{a^m} \text{ est vrai } \}.$$

Intéressons-nous à l'implémentation des formules de Presburger sous forme simple. On représente les variables par des entiers. On utilise les types OCaml suivants :

```
type terme = V of int | Plus of int*int

type formule =
    |Existe of int * formule
    |Et of formule * formule
    |Non of formule
    |Egale of terme * terme
```

☐ 26 – Expliquer en langage courant le principe d'un algorithme qui prend en entrée une formule φ sous forme simple, l'indice de la plus grande variable apparaissant dans φ, et qui renvoie un automate reconnaissant L_φ.

☐ 27 – Faites tourner à la main votre algorithme en détaillant bien les étapes et automates intermédiaires sur la formule $\varphi(x_2) := \neg\exists x_1.\ \neg(x_1 = x_2)$ pour renvoyer l'automate correspondant

☐ 28 – Écrire une fonction `automate_formule : formule -> int -> automate` qui prend en entrée une formule φ sous forme simple et l'indice de la plus grande variable apparaissant dans φ, et qui renvoie un automate reconnaissant L_φ.

☐ 29 – Démontrer la correction de votre algorithme.

☐ 30 – En déduire une fonction `est_vraie : formule -> int -> bool` qui détermine pour une formule sous forme simple supposée sans variable libre et le plus grand indice de ses variables si elle est vraie.

A. Annexe : Règles de la déduction naturelle

On présente les règles de la déduction naturelle suivantes.
Les arbres de preuves doivent être effectués à partir de l'ensemble de règles fourni ci-dessous.

$$\frac{}{\Gamma, A \vdash A}\,\text{ax} \qquad \frac{\Gamma \vdash B}{\Gamma, A \vdash B}\,\text{aff} \qquad \frac{\Gamma \vdash \bot}{\Gamma \vdash A}\,\bot$$

$$\frac{}{\Gamma \vdash A \vee \neg A}\,\text{t.e.} \qquad \frac{\Gamma, \neg A \vdash \bot}{\Gamma \vdash A}\,\text{RAA}$$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B}\,\to_i \qquad \frac{\Gamma \vdash A \to B \quad \Gamma \vdash A}{\Gamma \vdash B}\,\to_e$$

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B}\,\wedge_i \qquad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A}\,\wedge_e^g \qquad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B}\,\wedge_e^d$$

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B}\,\vee_i^g \qquad \frac{\Gamma \vdash B}{\Gamma \vdash A \vee B}\,\vee_i^d \qquad \frac{\Gamma \vdash A \vee B \quad \Gamma, A \vdash C \quad \Gamma, B \vdash C}{\Gamma \vdash C}\,\vee_e$$

$$\frac{\Gamma, A \vdash \bot}{\Gamma \vdash \neg A}\,\neg_i \qquad \frac{\Gamma \vdash \neg A \quad \Gamma \vdash A}{\Gamma \vdash \bot}\,\neg_e$$

$$\frac{\Gamma,\ A,\ B \vdash C}{\Gamma,\ A \wedge B \vdash C}\,\text{cut}$$

$$\frac{\Gamma \vdash A \quad x \notin FV(\Gamma)}{\Gamma \vdash \forall x.A}\,\forall_i \qquad \frac{\Gamma \vdash \forall x.A}{\Gamma \vdash A[t/x]}\,\forall_e$$

$$\frac{\Gamma \vdash A[t/x]}{\Gamma \vdash \exists x.A}\,\exists_i \qquad \frac{\Gamma \vdash \exists x.A \quad \Gamma, A \vdash B \quad x \notin FV(\Gamma) \cup FV(B)}{\Gamma \vdash B}\,\exists_e$$

$$\frac{\Gamma, A \vdash B \quad x \notin FV(\Gamma) \cup FV(B)}{\Gamma, \exists x.A \vdash B}\,\exists_e'$$

Où $FV(A)$ désigne l'ensemble des variables libres dans une formule A et $FV(\Gamma)$ l'ensemble des variables libres dans un ensemble de formules Γ. De plus, $A[t/x]$ correspond à la formule A où les occurrences d'une variable libre x ont été remplacées par t.

Fin de l'épreuve

Travailler les chapitres de ce sujet

Prepa Help suit le programme de MPI chapitre par chapitre : cours détaillé, méthodes, pièges, exercices et problèmes corrigés pas à pas, tirés des concours. L'essai gratuit donne accès aux chapitres offerts de ta filière, sans carte bancaire.

Chapitres liés à ce sujet

- Langage OCaml : expressions, types, filtrage, récursivité
- Récursivité
- Architecture de l'ordinateur et représentation des nombres

Ouvrir l'essai gratuit

Scanne le code ou clique sur le bouton.

prepahelp.com/#essai

Une question sur le concours ou sur ta filière : **écrire sur WhatsApp**

Les autres années et épreuves : prepahelp.com/annales/mines/mpi