ÉCOLES NORMALES SUPÉRIEURES

CONCOURS D'ADMISSION 2020

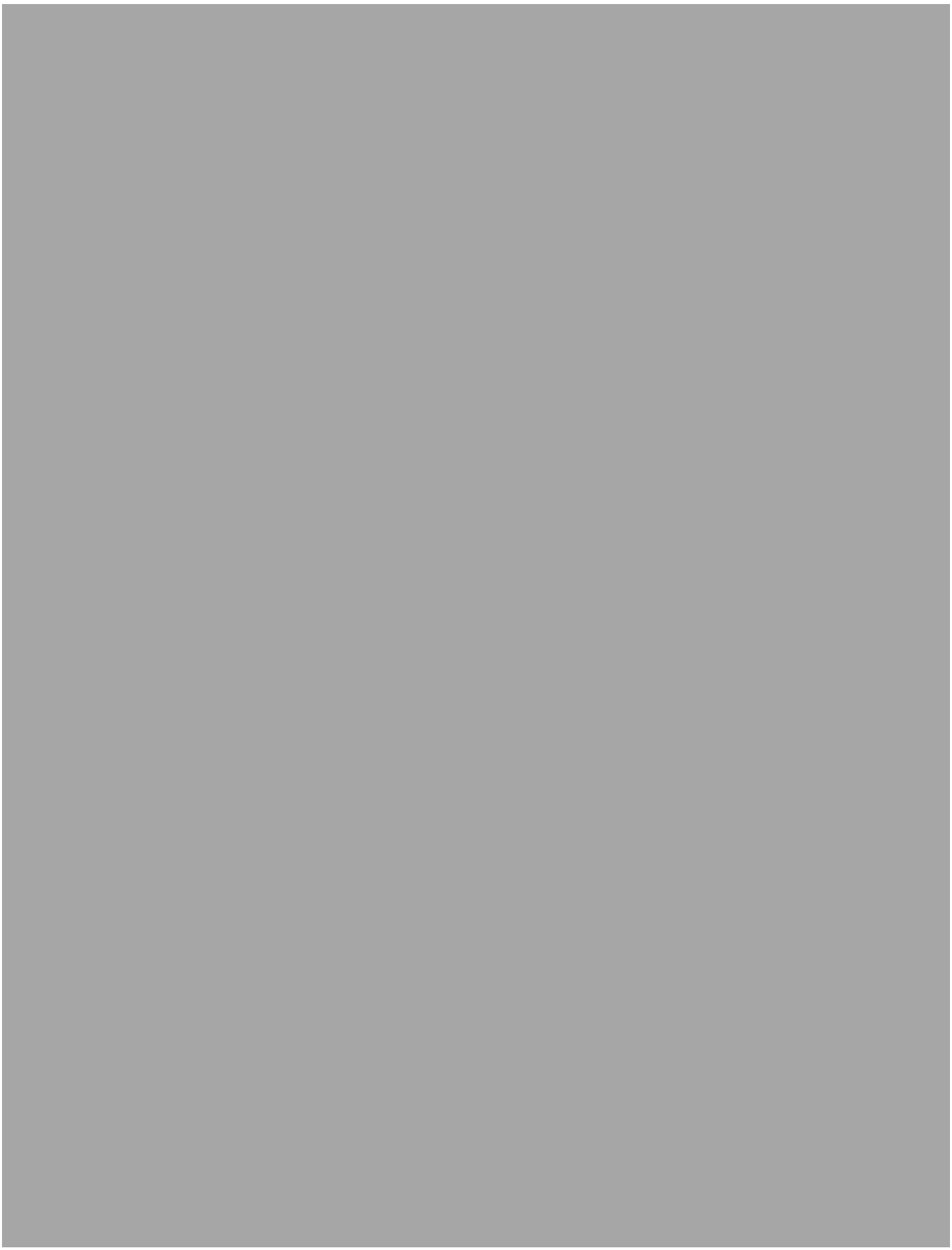
VENDREDI 24 AVRIL 2020 - 14h00 – 18h00

FILIERE MP

ÉPREUVE INFO-MATHS (ULCR)

Durée : 4 heures

L'utilisation des calculatrices n'est pas autorisée pour cette épreuve



Programmation fonctionnelle, sémantique et topologie

Le sujet comporte 13 pages, numérotées de 1 à 13.
Pour le traitement des trois dernières parties du sujet, il est recommandé de détacher la page 13 afin que son contenu soit facilement consultable.

Dans ce sujet, on considère un langage de programmation fonctionnelle minimaliste, proche mais différent d'OCaml. On définit rigoureusement la sémantique (signification) des programmes au moyen d'objets mathématiques usuels (e.g. entiers, fonctions, ordres partiels) et on l'utilise pour concevoir et prouver des programmes.

- La première partie introduit un système de types simples, ainsi que la notion d'ordre partiel complet. On y construit, pour chaque type, un certain ordre partiel complet.
- La deuxième partie introduit le langage de programmation, et définit l'interprétation des programmes.
- La troisième partie étudie des notions d'inspiration topologique issues de notre langage de programmation : on y parlera d'ouverts, de fermés et de compacts calculatoires.
- La quatrième partie porte sur la preuve de compacité de l'espace de Cantor : on y démontrera qu'il existe un programme p qui permet de déterminer, pour tout programme q, si q termine sur toute suite de bits.

Notations et définitions

- On note $\mathscr{P}(E)$ l'ensemble des parties d'un ensemble E.
- On note $E \uplus F$ l'union disjointe des ensembles E et F.
- Soit E un ensemble et F un ensemble de sous-ensembles de E, c'est à dire $F \subseteq \mathscr{P}(E)$. On note $\bigcap F$ l'intersection de tous les ensembles de F et $\bigcup F$ l'union de tous les ensembles de F. Quand F est vide, $\bigcap F = E$ et $\bigcup F = \emptyset$.
- La partie entière inférieure est notée $\lfloor \_ \rfloor$. En particulier, si $n \in \mathbb{N}$, alors $\lfloor n/2 \rfloor$ est le plus grand entier $m \in \mathbb{N}$ tel que $2m \leq n$.
- Un **ordre partiel** $(E, \leq_E)$ est un ensemble E équipé d'une relation binaire $\leq_E$ qui est réflexive, transitive et antisymétrique. Dans ce contexte, on notera $<_E$ l'ordre strict induit par $\leq_E$, dont on rappelle la définition : $x <_E y$ quand $x \leq_E y$ et $x \neq y$.
- Soit $(E, \leq_E)$ un ordre partiel et $(x_i)_{i \in \mathbb{N}}$ une suite d'éléments de E. On dit que $y \in E$ est un **majorant** de la suite quand $x_i \leq_E y$ pour tout $i \in \mathbb{N}$. On dit que y est une **borne supérieure** de la suite quand y est un majorant de la suite et qu'on a $y \leq_E z$ pour tout majorant z de la suite.

Partie I

On définit les **types** comme les expressions syntaxiques engendrées à partir des **types de base** unit, bool et nat au moyen de l'opérateur binaire $(\_ \Rightarrow \_)$, qui correspond intuitivement à l'opérateur `(_ -> _)` du langage OCaml. Par exemple, nat et $\mathsf{unit} \Rightarrow (\mathsf{bool} \Rightarrow \mathsf{bool})$ sont des types. Dans la suite, les types seront représentés par la lettre τ.

L'objectif de cette partie est de définir, pour tout type τ, un certain ensemble partiellement ordonné $(\llbracket\tau\rrbracket, \leq_\tau)$. Dans la partie suivante, les programmes de type τ seront interprétés comme des éléments de $\llbracket\tau\rrbracket$.

Étant donné un ordre partiel $(A, \leq_A)$, une **suite croissante** de A est une suite $(a_i)_{i\in\mathbb{N}}$ d'éléments de A telle que, pour tout $i \in \mathbb{N}$, $a_i \leq_A a_{i+1}$. On dit que $(A, \leq_A)$ est un **ordre partiel complet** quand $(A, \leq_A)$ est un ordre partiel et que toute suite croissante $(a_i)_{i\in\mathbb{N}}$ d'éléments de A admet une borne supérieure, notée $\sup^A_{i\in\mathbb{N}} a_i$.

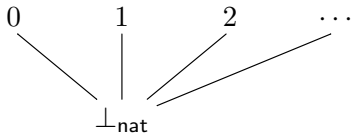
Étant donnés deux ordres partiels complets $(A, \leq_A)$ et $(B, \leq_B)$, une application $f : A \to B$ est dite **croissante** quand, pour tous $a_1 \in A$ et $a_2 \in A$ tels que $a_1 \leq_A a_2$, on a $f(a_1) \leq_B f(a_2)$. L'application f est dite **continue** quand elle est croissante et que, pour toute suite croissante $(a_i)_{i\in\mathbb{N}}$ de A, on a $f(\sup^A_{i\in\mathbb{N}} a_i) = \sup^B_{i\in\mathbb{N}} f(a_i)$.

Définissons dans un premier temps $(\llbracket\tau\rrbracket, \leq_\tau)$ pour les types de base. À cet effet, on se donne des constantes distinctes $\bot_{\mathsf{nat}}$, $\bot_{\mathsf{bool}}$, $\bot_{\mathsf{unit}}$, tt, ff et uu. Les trois dernières correspondent intuitivement aux valeurs OCaml `true`, `false` et `()`. Les constantes $\bot_\tau$ serviront à représenter l'absence de résultat des programmes de type τ dont l'évaluation ne termine pas. On définit :

$$\mathbb{B} \stackrel{def}{=} \{\mathsf{tt}, \mathsf{ff}\} \qquad \mathbb{U} \stackrel{def}{=} \{\mathsf{uu}\}$$

$$\llbracket\mathsf{nat}\rrbracket \stackrel{def}{=} \mathbb{N} \uplus \{\bot_{\mathsf{nat}}\} \qquad \llbracket\mathsf{bool}\rrbracket \stackrel{def}{=} \mathbb{B} \uplus \{\bot_{\mathsf{bool}}\} \qquad \llbracket\mathsf{unit}\rrbracket \stackrel{def}{=} \mathbb{U} \uplus \{\bot_{\mathsf{unit}}\}$$

Pour chaque type de base $b \in \{\mathsf{unit}, \mathsf{nat}, \mathsf{bool}\}$ on définit enfin $\leq_b$ comme suit : pour tous $e_1, e_2 \in \llbracket b\rrbracket$, $e_1 \leq_b e_2$ ssi $e_1 = e_2$ ou $e_1 = \bot_b$. Il est clair que cela définit des ordres partiels ; inutile de le vérifier dans les réponses aux questions. L'ordre partiel $\leq_{\mathsf{nat}}$ peut être représenté graphiquement comme suit :



Question 1.1. On souhaite vérifier que $(\llbracket\mathsf{nat}\rrbracket, \leq_{\mathsf{nat}})$ est un ordre partiel complet :

a. Caractériser les suites croissantes de $(\llbracket\mathsf{nat}\rrbracket, \leq_{\mathsf{nat}})$.

b. En déduire que toute suite croissante de $(\llbracket\mathsf{nat}\rrbracket, \leq_{\mathsf{nat}})$ admet une borne supérieure.

On admet que $(\llbracket\mathsf{unit}\rrbracket, \leq_{\mathsf{unit}})$ et $(\llbracket\mathsf{bool}\rrbracket, \leq_{\mathsf{bool}})$ sont aussi des ordres partiels complets.

Étant donnés deux types τ_1 et τ_2 pour lesquels on suppose avoir construit les ordres partiels complets $(\llbracket\tau_1\rrbracket, \leq_{\tau_1})$ et $(\llbracket\tau_2\rrbracket, \leq_{\tau_2})$, on définit $\llbracket\tau_1 \Rightarrow \tau_2\rrbracket$ comme l'ensemble des applications continues de $\llbracket\tau_1\rrbracket$ dans $\llbracket\tau_2\rrbracket$:

$$\llbracket\tau_1 \Rightarrow \tau_2\rrbracket \quad \stackrel{def}{=} \quad \{f : \llbracket\tau_1\rrbracket \to \llbracket\tau_2\rrbracket \mid f \text{ est continue}\}$$

On définit ensuite la relation $\leq_{\tau_1 \Rightarrow \tau_2}$ en posant, pour tous $f, g \in [\![\tau_1 \Rightarrow \tau_2]\!]$:

$$f \leq_{\tau_1 \Rightarrow \tau_2} g \quad \text{ssi} \quad f(e) \leq_{\tau_2} g(e) \text{ pour tout } e \in [\![\tau_1]\!]$$

Enfin, on définit $\bot_{\tau_1 \Rightarrow \tau_2}$ comme la fonction $f : [\![\tau_1]\!] \to [\![\tau_2]\!]$ telle que $f(e) = \bot_{\tau_2}$ pour tout $e \in [\![\tau_1]\!]$.

On admettra que $([\![\tau_1 \Rightarrow \tau_2]\!], \leq_{\tau_1 \Rightarrow \tau_2})$ est un ordre partiel, et que $\bot_{\tau_1 \Rightarrow \tau_2}$ est son plus petit élément.

Question 1.2. On considère dans cette question le cas particulier où $\tau_1 = \mathsf{nat}$ et $\tau_2 = \mathsf{unit}$.

a. Montrer que toute application croissante $f : [\![\mathsf{nat}]\!] \to [\![\mathsf{unit}]\!]$ est continue.

b. Combien y a-t-il d'éléments $f \in [\![\mathsf{nat} \Rightarrow \mathsf{unit}]\!]$ tels que $f(\bot_{\mathsf{nat}}) \neq \bot_{\mathsf{unit}}$?

c. Donner, sans justifier, une suite strictement croissante de $[\![\mathsf{nat} \Rightarrow \mathsf{unit}]\!]$, c'est à dire une suite $(f_i)_{i \in \mathbb{N}}$ telle que $f_i <_{\mathsf{nat} \Rightarrow \mathsf{unit}} f_{i+1}$ pour tout $i \in \mathbb{N}$.

Question 1.3. Soient τ_1 et τ_2 des types tels que $([\![\tau_1]\!], \leq_{\tau_1})$ et $([\![\tau_2]\!], \leq_{\tau_2})$ sont des ordres partiels complets.

a. Soit $(f_i)_{i \in \mathbb{N}}$ une suite croissante de $([\![\tau_1 \Rightarrow \tau_2]\!], \leq_{\tau_1 \Rightarrow \tau_2})$. Pour tout $e \in [\![\tau_1]\!]$, $(f_i(e))_{i \in \mathbb{N}}$ est une suite croissante de $([\![\tau_2]\!], \leq_{\tau_2})$ par définition de $\leq_{\tau_1 \Rightarrow \tau_2}$, et admet donc une borne supérieure.

Montrer que l'application $f' : [\![\tau_1]\!] \to [\![\tau_2]\!]$ définie par $f'(e) = \sup_{i \in \mathbb{N}}^{[\![\tau_2]\!]} f_i(e)$ est continue.

b. Montrer que $([\![\tau_1 \Rightarrow \tau_2]\!], \leq_{\tau_1 \Rightarrow \tau_2})$ est un ordre partiel complet.

Tout type étant construit (en un nombre fini d'étapes) à partir des types de base au moyen de l'opérateur $(\_ \Rightarrow \_)$, les définitions et résultats précédents nous donnent pour chaque type τ un ordre partiel complet $([\![\tau]\!], \leq_\tau)$.

Partie II

Nous introduisons dans cette partie les programmes et leur interprétation. On suppose un ensemble infini dénombrable de **variables**, noté $\mathcal{X}$ et dont les éléments seront représentés par les lettres x, y et z dans la suite. Les **programmes**, qui seront représentés par les lettres p et q, sont des expressions syntaxiques données par la grammaire suivante, où n dénote un entier arbitraire :

$$\begin{array}{rcl} p & ::= & \mathsf{uu} \\ & | & \mathsf{tt} \mid \mathsf{ff} \mid \mathsf{if}\ p_0\ \mathsf{then}\ p_1\ \mathsf{else}\ p_2 \mid p_1 = p_2 \\ & | & n \mid p_1 + p_2 \mid p_1 - p_2 \mid p_1 \times p_2 \mid p_0/2 \\ & | & x \mid (\mathsf{fun}\ x \to p_0) \mid (p_1\ p_2) \mid (\mathsf{rec}\ p_0) \\ & | & (p_1 \parallel_{\mathsf{u}} p_2) \end{array}$$

Cela signifie que l'ensemble des programmes est le plus petit ensemble tel que : les constantes uu, tt et ff sont des programmes ; tout $n \in \mathbb{N}$ est un programme ; tout $x \in \mathcal{X}$ est un programme ; si p_0, p_1 et p_2 sont des programmes alors $\mathsf{if}\ p_0\ \mathsf{then}\ p_1\ \mathsf{else}\ p_2$ aussi ; etc. Dans la plupart des cas, les constructions de programmes utilisent les mêmes notations qu'en OCaml, et la signification *intuitive* des constructions sera analogue à celle d'OCaml. Certaines autres constructions sont nouvelles. Dans tous les cas, une signification mathématique précise sera donnée plus loin.

Nous définissons ensuite une notion de typage pour nos programmes. Contrairement au cas du langage OCaml, un programme de notre langage admettra au plus un type. Afin de fixer l'unique type de chaque variable, on se donne une application t des variables dans les types. Nous supposerons que pour tout type τ il existe une infinité de variables x telles que $t(x) = \tau$. La **relation de typage** exprimant qu'un programme p est de type τ, notée $p : \tau$, est ensuite définie par les équivalences suivantes, pour tous programmes p_0, p_1, p_2, pour tout type τ, et pour tous $n \in \mathbb{N}$ et $x \in \mathcal{X}$:

$$\begin{array}{rcl} \mathsf{uu} : \tau & \Leftrightarrow & \tau = \mathsf{unit} \\ \mathsf{tt} : \tau & \Leftrightarrow & \tau = \mathsf{bool} \\ \mathsf{ff} : \tau & \Leftrightarrow & \tau = \mathsf{bool} \\ n : \tau & \Leftrightarrow & \tau = \mathsf{nat} \\ (\mathsf{if}\ p_0\ \mathsf{then}\ p_1\ \mathsf{else}\ p_2) : \tau & \Leftrightarrow & p_0 : \mathsf{bool},\ p_1 : \tau \text{ et } p_2 : \tau \\ (p_1 = p_2) : \tau & \Leftrightarrow & \tau = \mathsf{bool} \text{ et il existe } \tau' \in \{\mathsf{unit}, \mathsf{nat}, \mathsf{bool}\} \text{ tel que } p_1 : \tau' \text{ et } p_2 : \tau' \\ p_1 + p_2 : \tau & \Leftrightarrow & \tau = \mathsf{nat},\ p_1 : \mathsf{nat} \text{ et } p_2 : \mathsf{nat} \\ p_1 - p_2 : \tau & \Leftrightarrow & \tau = \mathsf{nat},\ p_1 : \mathsf{nat} \text{ et } p_2 : \mathsf{nat} \\ p_1 \times p_2 : \tau & \Leftrightarrow & \tau = \mathsf{nat},\ p_1 : \mathsf{nat} \text{ et } p_2 : \mathsf{nat} \\ p_0/2 : \tau & \Leftrightarrow & \tau = \mathsf{nat},\ p_0 : \mathsf{nat} \\ x : \tau & \Leftrightarrow & t(x) = \tau \\ (\mathsf{fun}\ x \to p_0) : \tau & \Leftrightarrow & \text{il existe } \tau' \text{ tel que } p_0 : \tau' \text{ et } \tau = (t(x) \Rightarrow \tau') \\ (p_1\ p_2) : \tau & \Leftrightarrow & \text{il existe } \tau' \text{ tel que } p_1 : (\tau' \Rightarrow \tau) \text{ et } p_2 : \tau' \\ (\mathsf{rec}\ p_0) : \tau & \Leftrightarrow & p_0 : (\tau \Rightarrow \tau) \\ (p_1 \parallel_{\mathsf{u}} p_2) : \tau & \Leftrightarrow & \tau = \mathsf{unit},\ p_1 : \mathsf{unit} \text{ et } p_2 : \mathsf{unit} \end{array}$$

On pourra par exemple vérifier que $(1 + (\mathsf{fun}\ x \to x)) : \tau$ est faux quel que soit τ. Ou encore, que $(\mathsf{fun}\ x \to (\mathsf{fun}\ y \to (1 + x))) : (\mathsf{nat} \Rightarrow t(y) \Rightarrow \mathsf{nat})$ est vrai à condition que $t(x) = \mathsf{nat}$.

Question 2.1. Soient x, y et z des variables quelconques, et τ un type. Indiquer, sans justifier, sous quelles conditions sur τ, $t(x)$, $t(y)$ et $t(z)$ on a $(\mathsf{fun}\ x \to \mathsf{if}\ x = y\ \mathsf{then}\ y\ \mathsf{else}\ z) : \tau$.

On considère désormais uniquement des programmes p bien typés, c'est à dire pour lesquels il existe τ tel que $p : \tau$. Quand p est un programme bien typé, on note $t(p)$ son unique type.

Pour donner un sens aux variables présentes dans un programme, on introduit la notion suivante : un **environnement** est une application $\mathcal{E} : \mathcal{X} \to \bigcup_\tau [\![\tau]\!]$ telle qu'on a $\mathcal{E}(x) \in [\![t(x)]\!]$ pour tout $x \in \mathcal{X}$. Si $\mathcal{E}$ est un environnement et $e \in [\![t(x)]\!]$, $\mathcal{E}[x \mapsto e]$ est l'environnement $\mathcal{E}'$ tel que $\mathcal{E}'(x) = e$ et $\mathcal{E}'(y) = \mathcal{E}(y)$ pour tout $y \neq x$.

L'**interprétation** d'un programme p dans un environnement $\mathcal{E}$, notée $[\![p]\!]^{\mathcal{E}}$, est définie par les clauses de la figure 1, page 13, qu'il n'est pas nécessaire de consulter immédiatement. Dans ces clauses, on écrit simplement $[\![p]\!]^{\mathcal{E}} = \bot$ pour $[\![p]\!]^{\mathcal{E}} = \bot_{t(p)}$. On admettra que, pour tout programme (bien typé) p et pour tout environnement $\mathcal{E}$, l'interprétation de p dans $\mathcal{E}$ est bien définie et appartient à $[\![t(p)]\!]$:

$$[\![p]\!]^{\mathcal{E}} \in [\![t(p)]\!]$$

La figure 1 comporte de nombreux cas, dont certains sont assez techniques, et les intuitions (parfois trompeuses) issues de la pratique d'OCaml ne seront pas suffisantes pour en comprendre l'ensemble : le candidat est encouragé à ne pas s'y attarder trop, mais à en approfondir les différents cas au fil de l'énoncé. Pour cela, il est conseillé de séparer la page de la figure du reste de l'énoncé pour la garder à portée de main.

Intuitivement, l'interprétation d'un programme représente le résultat de son évaluation. Pour les types de base, une évaluation n'a jamais pour résultat $\bot$, mais cet élément spécial représente au contraire le résultat indéfini d'une évaluation qui ne termine pas.

Considérons par exemple le programme $\mathsf{fun}\ x \to x + y$, de type $\mathsf{nat} \Rightarrow \mathsf{nat}$ si $x, y \in \mathcal{X}$ et $t(x) = t(y) = \mathsf{nat}$. Par définition de l'interprétation, $[\![\mathsf{fun}\ x \to x + y]\!]^{\mathcal{E}}$ est l'application définie par $[\![\mathsf{fun}\ x \to x + y]\!]^{\mathcal{E}}(e) = [\![x + y]\!]^{\mathcal{E}[x \mapsto e]}$ pour tout $e \in [\![\mathsf{nat}]\!]$. Si $e = \bot_{\mathsf{nat}}$ ou $\mathcal{E}(y) = \bot_{\mathsf{nat}}$, on vérifie $[\![\mathsf{fun}\ x \to x + y]\!]^{\mathcal{E}}(e) = \bot_{\mathsf{nat}}$. Sinon, $e \in \mathbb{N}$ et $\mathcal{E}(y) \in \mathbb{N}$, et l'on a $[\![\mathsf{fun}\ x \to x + y]\!]^{\mathcal{E}}(e) = e + \mathcal{E}(y)$.

On remarque dans cet exemple que la valeur de $\mathcal{E}$ sur les variables autres que y n'entre pas en jeu dans l'interprétation du programme. Cela correspond intuitivement au fait que seule la variable y fait référence à un objet extérieur au programme — on dit que cette variable est **libre**. À l'inverse, la variable x utilisée dans le sous-programme $x + y$ est une référence interne au programme, correspondant à l'argument de la fonction définie — l'occurrence de x dans $x + y$ est dite **liée** par la construction $\mathsf{fun}\ x \to \ldots$ englobante.

On dit que l'interprétation d'un programme q est **indépendante** de l'environnement quand $[\![q]\!]^{\mathcal{E}} = [\![q]\!]^{\mathcal{E}'}$ quels que soient $\mathcal{E}$ et $\mathcal{E}'$. On notera simplement $[\![q]\!]$ l'interprétation d'un tel programme dans un environnement arbitraire. Dans la suite du sujet, si on demande au candidat un programme q tel que $[\![q]\!]$ satisfasse une certaine propriété, on attend un programme dont l'interprétation soit indépendante de l'environnement — ce qui est garanti si le programme est sans variable libre.

Question 2.2. Soit une variable x telle que $t(x) = \mathsf{nat}$.

a. On considère le programme $p_e = \big(\mathsf{fun}\ x \to \big((2 \times (x/2)) = x\big)\big)$, de type $\mathsf{nat} \Rightarrow \mathsf{bool}$. Caractériser $\{n \in \mathbb{N} \mid [\![p_e]\!](n) = \mathsf{tt}\}$ en justifiant par le calcul de $[\![p_e]\!]$.

b. On définit l'application $f_c : \mathbb{N} \to \mathbb{N}$ par $f_c(n) = n/2$ si n est pair, $f_c(n) = 3n + 1$ sinon. Donner un programme $p_c : \mathsf{nat} \Rightarrow \mathsf{nat}$ tel que $[\![p_c]\!]$ coïncide avec f_c sur $\mathbb{N}$. *On utilisera directement p_e plutôt que de recopier sa définition.*

On s'intéresse maintenant à l'interprétation des programmes de la forme $\mathsf{rec}\ p_0$, permettant des définitions récursives de façon analogue au `let rec ...` d'OCaml. Un programme $\mathsf{rec}\ p_0$ est de type τ quand $p_0 : \tau \Rightarrow \tau$. Puisque $[\![p_0]\!]^{\mathcal{E}}$ est une application continue de $[\![\tau]\!]$ dans lui-même, la suite $(([\![p_0]\!]^{\mathcal{E}})^i(\bot_\tau))_{i\in\mathbb{N}}$ est croissante : on a en effet $\bot_\tau \leq_\tau [\![p_0]\!]^{\mathcal{E}}(\bot_\tau)$ par minimalité de $\bot_\tau$, puis $[\![p_0]\!]^{\mathcal{E}}(\bot_\tau) \leq_\tau ([\![p_0]\!]^{\mathcal{E}})^2(\bot_\tau)$ par croissance de $[\![p_0]\!]^{\mathcal{E}}$, et ainsi de suite. Toute suite croissante admettant une borne supérieure dans l'ordre partiel complet $([\![\tau]\!], \leq_\tau)$, l'interprétation $[\![\mathsf{rec}\ p_0]\!]^{\mathcal{E}}$ est bien définie.

Question 2.3. Soient $f, n \in \mathcal{X}$ des variables telles que $t(n) = \mathsf{nat}$ et $t(f) = (\mathsf{nat} \Rightarrow \mathsf{nat})$. On pose :

$$p \quad \stackrel{def}{=} \quad \mathsf{fun}\ f \to \mathsf{fun}\ n \to \mathsf{if}\ n = 0\ \mathsf{then}\ 1\ \mathsf{else}\ n \times f\ (n-1)$$

On pourra vérifier que p est bien typé, de type $(\mathsf{nat} \Rightarrow \mathsf{nat}) \Rightarrow (\mathsf{nat} \Rightarrow \mathsf{nat})$.

a. Soit $f \in [\![\mathsf{nat} \Rightarrow \mathsf{nat}]\!]$ et $e \in [\![\mathsf{nat}]\!]$. Exprimer $[\![p]\!](f)(e)$ en fonction de e et f.

b. Calculer $[\![p]\!]^{i+1}(\bot_{\mathsf{nat}\Rightarrow\mathsf{nat}})(e)$ pour tous $i \in \mathbb{N}$ et $e \in [\![\mathsf{nat}]\!]$.

c. En déduire que $[\![\mathsf{rec}\ p]\!](k) = k!$ pour tout $k \in \mathbb{N}$, et donner la valeur de $[\![\mathsf{rec}\ p]\!](\bot_{\mathsf{nat}})$.

Question 2.4. On reprend la définition de f_c de la question 2.2. La conjecture de Collatz énonce que, pour tout $n \in \mathbb{N}^*$, il existe $i \in \mathbb{N}$ tel que $f_c^i(n) = 1$. Autrement dit, la suite des itérées de f_c finirait toujours par atteindre 1, quelle que soit la valeur de départ non nulle.

a. Donner un programme $q : (\mathsf{nat} \Rightarrow \mathsf{unit}) \Rightarrow (\mathsf{nat} \Rightarrow \mathsf{unit})$ tel que, pour tous $i \in \mathbb{N}$ et $n \in \mathbb{N}^*$, $[\![q]\!]^{i+1}(\bot_{\mathsf{nat}\Rightarrow\mathsf{unit}})(n) = \mathsf{uu}$ ssi il existe $k \leq i$ tel que $f_c^k(n) = 1$. *On utilisera directement le programme p_c de la question 2.2 sans en recopier la définition.*

b. Donner un programme $p : \mathsf{nat} \Rightarrow \mathsf{unit}$ tel que, pour tout $n \in \mathbb{N}^*$, $[\![p]\!](n) = \mathsf{uu}$ ssi il existe $k \in \mathbb{N}$ tel que $f_c^k(n) = 1$. Justifier.

Soient τ un type et x une variable de type τ. On définit :

$$\mathsf{DIV}_\tau \stackrel{def}{=} \mathsf{rec}\ (\mathsf{fun}\ x \to x)$$

On vérifie aisément que $\mathsf{DIV}_\tau : \tau$ et $[\![\mathsf{DIV}_\tau]\!] = \bot_\tau$. Intuitivement, DIV_τ est défini récursivement comme un objet x égal à lui même. L'évaluation de DIV_τ déroule à l'infini cette définition, et ne termine donc pas.

Il existe donc des programmes $p : \tau_1 \Rightarrow \tau_2$ tels que $[\![p]\!](e) = \bot_{\tau_2}$ pour tout $e \in [\![\tau_1]\!]$. Si τ_2 est un type de base, l'évaluation de $(p\ q)$ ne termine donc pas, quel que soit $q : \tau_1$. Réciproquement, il existe des programmes $p : \tau_1 \Rightarrow \tau_2$ tels que $(p\ q)$ termine toujours, même quand $[\![q]\!] = \bot_{\tau_1}$. Par exemple, si x est une variable de type nat, le programme $(\mathsf{fun}\ x \to 42) : \mathsf{nat} \Rightarrow \mathsf{nat}$ a pour interprétation l'application qui envoie tout élément de $[\![\mathsf{nat}]\!]$ sur l'entier 42. Intuitivement, ce programme peut terminer même si l'évaluation de son argument ne termine pas, puisque cet argument n'est pas utilisé. Par contre, on pourra vérifier que $\mathsf{fun}\ x \to \mathsf{if}\ x = x\ \mathsf{then}\ 42\ \mathsf{else}\ 13$ a pour interprétation une application f telle que $f(\bot_{\mathsf{nat}}) = \bot_{\mathsf{nat}}$ — on notera aussi que cette application ne prend jamais la valeur 13, car $[\![x = x]\!]^{\mathcal{E}} \in \{\mathsf{tt}, \bot_{\mathsf{bool}}\}$ pour tout $\mathcal{E}$.

Soit τ un type. On dit qu'un élément $e \in [\![\tau]\!]$ est **calculable** quand il existe un programme $p : \tau$ tel que $[\![p]\!] = e$. Par extension, on dira qu'une application $f : [\![\tau_1]\!] \to [\![\tau_2]\!]$ est calculable quand il existe $p : \tau_1 \Rightarrow \tau_2$ tel que $[\![p]\!] = f$.

Question 2.5.

a. Montrer que tous les éléments de $[\![\mathsf{unit} \Rightarrow \mathsf{unit}]\!]$ sont calculables.

b. On admet que l'ensemble des programmes est dénombrable. Montrer qu'il existe des éléments non calculables dans $[\![\mathsf{nat} \Rightarrow \mathsf{unit}]\!]$.

Dans la suite du sujet, nous chercherons à reconnaître certains éléments de $[\![\tau]\!]$ au moyen de programmes de type $\tau \Rightarrow \mathsf{unit}$: ces programmes devront renvoyer uu quand un élément est reconnu, et ne pas terminer sinon. Dans ce cadre, des notions de conjonction et de disjonction sur le type unit seront utiles.

Question 2.6. Donner un programme $\mathsf{unit\_and} : \mathsf{unit} \Rightarrow \mathsf{unit} \Rightarrow \mathsf{unit}$ tel que, pour tout environnement $\mathcal{E}$ et pour tous $p : \mathsf{unit}$ et $q : \mathsf{unit}$,

$$[\![\mathsf{unit\_and}\ p\ q]\!]^{\mathcal{E}} = \mathsf{uu} \quad \text{ssi} \quad [\![p]\!]^{\mathcal{E}} = [\![q]\!]^{\mathcal{E}} = \mathsf{uu}.$$

Par la suite, on écrira simplement $p \,\&_{\mathsf{u}}\, q$ au lieu de $\mathsf{unit\_and}\ p\ q$.

On considère enfin l'opérateur $(\_ \|_{\mathsf{u}} \_)$ de disjonction sur unit. La définition de $[\![p_1 \|_{\mathsf{u}} p_2]\!]^{\mathcal{E}}$ en figure 1 indique que ce programme peut terminer (renvoyer uu) dès que l'un de ses deux sous-programmes termine. Intuitivement, on peut penser que les deux sous-programmes sont évalués en parallèle, et que uu est renvoyé dès que l'une de ces deux évaluations termine.

Question 2.7.

a. Soit x une variable telle que $t(x) = (\mathsf{bool} \Rightarrow \mathsf{unit})$. On pose :

$$p \quad \stackrel{def}{=} \quad \mathsf{fun}\ x \to \big((x\ \mathsf{tt}) \|_{\mathsf{u}} (x\ \mathsf{ff})\big)$$

Caractériser $\{f \in [\![\mathsf{bool} \Rightarrow \mathsf{unit}]\!] \mid [\![p]\!](f) = \mathsf{uu}\}$.

b. Soient x, p', et n des variables telles que $t(x) = t(p') = (\mathsf{nat} \Rightarrow \mathsf{unit})$ et $t(n) = \mathsf{nat}$. On pose :

$$p \quad \stackrel{def}{=} \quad \mathsf{fun}\ x \to \big(\big(\mathsf{rec}\ (\mathsf{fun}\ p' \to \mathsf{fun}\ n \to x\ n \|_{\mathsf{u}} p'\ (n+1))\big)\ 0\big)$$

Caractériser, sans justifier, $\{f \in [\![\mathsf{nat} \Rightarrow \mathsf{unit}]\!] \mid [\![p]\!](f) = \mathsf{uu}\}$.

Partie III

Il existe de nombreux liens entre topologie et sémantique des langages de programmation. Nous nous proposons ici de dériver des notions topologiques à partir de notre langage de programmation.

On dit que l'application $f : [\![\tau]\!] \to [\![\mathsf{unit}]\!]$ est la **fonction caractéristique** de $U \subseteq [\![\tau]\!]$ quand, pour tout $e \in [\![\tau]\!]$, $f(e) = \mathsf{uu}$ ssi $e \in U$. Un ensemble $U \subseteq [\![\tau]\!]$ est un **ouvert calculatoire** de $[\![\tau]\!]$ quand la fonction caractéristique de U est calculable. Un ensemble $U \subseteq [\![\tau]\!]$ est un **fermé calculatoire** de $[\![\tau]\!]$ quand son complémentaire dans $[\![\tau]\!]$ est un ouvert calculatoire de $[\![\tau]\!]$.

Par exemple, $[\![\mathsf{nat}]\!] \setminus \{\bot_{\mathsf{nat}}, 13\}$ est un ouvert calculatoire de $[\![\mathsf{nat}]\!]$. En effet, sa fonction caractéristique est l'interprétation du programme $\mathsf{fun}\ x \to \mathsf{if}\ x = 13\ \mathsf{then}\ \mathsf{DIV}_{\mathsf{unit}}\ \mathsf{else}\ \mathsf{uu}$.

Question 3.1. Soit τ un type. Montrer que $[\![\tau]\!]$ est à la fois un ouvert calculatoire et un fermé calculatoire de $[\![\tau]\!]$.

Question 3.2. Quels sont les ouverts calculatoires de $[\![\mathsf{unit}]\!]$?

Question 3.3. Soient τ et τ' des types, U un ouvert calculatoire de $[\![\tau']\!]$, et $f : [\![\tau]\!] \to [\![\tau']\!]$ une application calculable. Montrer que la pré-image $f^{-1}(U)$ de U par f est un ouvert calculatoire de $[\![\tau]\!]$.

Question 3.4. On considère un type τ quelconque. Utiliser les opérateurs $(\_ \|_{\mathsf{u}} \_)$ et $(\_ \&_{\mathsf{u}} \_)$ pour montrer les propriétés suivantes.

a. L'union de deux ouverts calculatoires de $[\![\tau]\!]$ est encore un ouvert calculatoire de $[\![\tau]\!]$.

b. L'intersection de deux ouverts calculatoires de $[\![\tau]\!]$ est encore un ouvert calculatoire de $[\![\tau]\!]$.

Pour la prochaine question on pourra utiliser le résultat suivant sans le justifier :

Propriété A. Soient τ_1 et τ_2 des types quelconques. On pose $\tau = (\tau_1 \Rightarrow \tau_2 \Rightarrow \mathsf{unit})$. Pour tous $p : \tau \Rightarrow \tau$, $e_1 \in [\![\tau_1]\!]$ et $e_2 \in [\![\tau_2]\!]$, on a

$$[\![\mathsf{rec}\ p]\!](e_1)(e_2) = \mathsf{uu} \quad \text{ssi} \quad \exists i \in \mathbb{N}.\ \big([\![p]\!]^i(\bot_\tau)\big)(e_1)(e_2) = \mathsf{uu}.$$

Question 3.5. On souhaite montrer qu'une union infinie calculable d'ouverts calculatoires est encore un ouvert calculatoire, dans le sens suivant. Soient $(U_i)_{i\in\mathbb{N}}$ une suite de parties de $[\![\tau]\!]$ et $p : \mathsf{nat} \Rightarrow \tau \Rightarrow \mathsf{unit}$ un programme tels que, pour tout $i \in \mathbb{N}$, $[\![p\ i]\!]$ est la fonction caractéristique de U_i. Montrer que $\cup_{i\in\mathbb{N}} U_i$ est un ouvert calculatoire de $[\![\tau]\!]$. *Indication : on pourra construire un programme intermédiaire* $\mathsf{rec}\ q : \mathsf{nat} \Rightarrow \tau \Rightarrow \mathsf{unit}$ *dont on montrera qu'il satisfait, pour tous* $k \in \mathbb{N}$ *et* $e \in [\![\tau]\!]$, $[\![\mathsf{rec}\ q]\!](k)(e) = \mathsf{uu}$ *ssi* $e \in \cup_{i\geq k} U_i$.

Un sous-ensemble $U \subseteq [\![\tau]\!]$ est **calculatoirement compact** dans $[\![\tau]\!]$ s'il existe un programme $\forall_U : (\tau \Rightarrow \mathsf{unit}) \Rightarrow \mathsf{unit}$ tel que, pour tout programme $p : \tau \Rightarrow \mathsf{unit}$,

$$[\![\forall_U\ p]\!] = \mathsf{uu} \quad \text{ssi} \quad [\![p]\!](e) = \mathsf{uu} \text{ pour tout } e \in U.$$

Par exemple, l'ensemble $E = \{40, 12\}$ est calculatoirement compact dans $[\![\mathsf{nat}]\!]$, comme en témoigne le programme $\forall_E = \mathsf{fun}\ f \to (f\ 40)\,\&_{\mathsf{u}}(f\ 12)$, où f est une variable quelconque de type $\mathsf{nat} \Rightarrow \mathsf{unit}$.

Question 3.6. Soit τ un type.

a. Montrer que $\emptyset$ est calculatoirement compact dans $[\![\tau]\!]$.

b. Soit $E \subseteq [\![\tau]\!]$ tel que $\bot_\tau \in E$. Montrer que E est calculatoirement compact dans $[\![\tau]\!]$.

c. En déduire que tout fermé calculatoire de $[\![\tau]\!]$ est calculatoirement compact dans $[\![\tau]\!]$.

Question 3.7. Soit $U \subseteq \mathbb{N}$. Montrer que U est calculatoirement compact dans $[\![\mathsf{nat}]\!]$ ssi U est fini. *Indication : on pourra écrire un certain programme p : $\mathsf{nat} \Rightarrow \mathsf{unit}$ comme la borne supérieure d'une suite croissante.*

Partie IV

Étant donnée une suite de booléens $s = (s_i)_{i\in\mathbb{N}} \in \mathbb{B}^{\mathbb{N}}$ on définit $f_s \in [\![\mathsf{nat} \Rightarrow \mathsf{bool}]\!]$ par $f_s(\bot_{\mathsf{nat}}) = \bot_{\mathsf{bool}}$ et $f_s(i) = s_i$ pour tout $i \in \mathbb{N}$. On définit l'espace de Cantor $\mathbb{C}$ comme l'ensemble de ces applications :

$$\mathbb{C} \quad \stackrel{def}{=} \quad \{f_s \mid s \in \mathbb{B}^{\mathbb{N}}\}$$

L'objectif de cette partie est de montrer que $\mathbb{C}$ est calculatoirement compact dans $[\![\mathsf{nat} \Rightarrow \mathsf{bool}]\!]$. On remarquera que $\mathbb{C}$ contient des éléments non calculables. Réciproquement, il existe des éléments calculables de $[\![\mathsf{nat} \Rightarrow \mathsf{bool}]\!]$ qui ne sont pas dans $\mathbb{C}$.

Dans la suite de l'énoncé, les programmes de type $\mathsf{nat} \Rightarrow \mathsf{bool}$ seront représentés par la lettre q. Les programmes de type $(\mathsf{nat} \Rightarrow \mathsf{bool}) \Rightarrow \mathsf{unit}$ seront représentés par la lettre p.

Pour $b \in \mathbb{B}$ et $s \in \mathbb{B}^{\mathbb{N}}$ on note $b{::}s$ la suite $s' \in \mathbb{B}^{\mathbb{N}}$ telle que $s'_0 = b$ et $s'_{n+1} = s_n$ pour tout $n \in \mathbb{N}$. Pour tout programme $q : \mathsf{nat} \Rightarrow \mathsf{bool}$ on définit l'opération analogue avec la même notation, où x est une variable de type nat :

$$b{::}q \quad \stackrel{def}{=} \quad \mathsf{fun}\ x \to \mathsf{if}\ x = 0\ \mathsf{then}\ b\ \mathsf{else}\ q\ (x-1)$$

Enfin, pour $p : (\mathsf{nat} \Rightarrow \mathsf{bool}) \Rightarrow \mathsf{unit}$ on définit, en prenant une variable y de type $\mathsf{nat} \Rightarrow \mathsf{bool}$:

$$p/b \quad \stackrel{def}{=} \quad \mathsf{fun}\ y \to p\ (b{::}y)$$

Pour $s \in \mathbb{B}^{\mathbb{N}}$ et $k \in \mathbb{N}$ on note $f_{s<k}$ l'application $f' \in [\![\mathsf{nat} \Rightarrow \mathsf{bool}]\!]$ telle que $f'(i) = s_i$ pour tout i tel que $0 \leq i < k$, et $f'(e) = \bot_{\mathsf{bool}}$ sinon.

Question 4.1. Soient un programme $p : (\mathsf{nat} \Rightarrow \mathsf{bool}) \Rightarrow \mathsf{unit}$ et $s \in \mathbb{B}^{\mathbb{N}}$ tels que $[\![p]\!](f_s) = \mathsf{uu}$. Montrer qu'il existe un entier k tel que $[\![p]\!](f_{s<k}) = \mathsf{uu}$. Dans la suite du sujet, on notera $k(s,p)$ le plus petit entier satisfaisant cette propriété.

Question 4.2. Soient un programme $p : (\mathsf{nat} \Rightarrow \mathsf{bool}) \Rightarrow \mathsf{unit}$, $b \in \mathbb{B}$ et $s \in \mathbb{B}^{\mathbb{N}}$ tels que $[\![p]\!](f_{b::s}) = \mathsf{uu}$. Montrer, pour tout $k \in \mathbb{N}$, que $[\![p/b]\!](f_{s<k}) = [\![p]\!](f_{b::s<1+k})$.

Question 4.3. Soit $p : (\mathsf{nat} \Rightarrow \mathsf{bool}) \Rightarrow \mathsf{unit}$ tel que, pour tout $f \in \mathbb{C}$, $[\![p]\!](f) = \mathsf{uu}$. On définit :

$$K(p) \quad \stackrel{def}{=} \quad \{k(s,p) \mid s \in \mathbb{B}^{\mathbb{N}}\}$$

a. Soit $s \in \mathbb{B}^{\mathbb{N}}$. On pose $i = k(s,p)$. Montrer que

$$K(((p/s_0)/s_1)\ldots/s_{i-1}) = \{0\}.$$

b. Montrer que, si $K(p)$ est non borné, alors il existe $b \in \mathbb{B}$ tel que $K(p/b)$ est encore non borné.

c. En déduire que $K(p)$ est borné.

La borne supérieure de $K(p)$ sera notée $|p|$ et appelée **module d'uniforme continuité** de p.

Question 4.4. Soit $p : (\mathsf{nat} \Rightarrow \mathsf{bool}) \Rightarrow \mathsf{unit}$ tel que, pour tout $f \in \mathbb{C}$, $[\![p]\!](f) = \mathsf{uu}$.

a. Soient $b \in \mathbb{B}$ et $s \in \mathbb{B}^{\mathbb{N}}$ tels que $k(b{::}s, p) > 0$. Montrer que $k(s, p/b) < k(b{::}s, p)$.

b. Montrer que, si $|p| > 0$, on a $|p/b| < |p|$ pour tout $b \in \mathbb{B}$.

Question 4.5. Montrer que $\mathbb{C}$ est calculatoirement compact dans $[\![\mathsf{nat} \Rightarrow \mathsf{bool}]\!]$.

$$
\begin{array}{rcl}
[\![c]\!]^{\mathcal{E}} & \stackrel{def}{=} & c \quad \text{pour tout } c \in \mathbb{N} \cup \mathbb{B} \cup \{\mathsf{uu}\} \\
[\![x]\!]^{\mathcal{E}} & \stackrel{def}{=} & \mathcal{E}(x) \\
[\![p_1 = p_2]\!]^{\mathcal{E}} & \stackrel{def}{=} & \left\{ \begin{array}{ll} \bot_{\mathsf{bool}} & \text{si } [\![p_1]\!]^{\mathcal{E}} = \bot \text{ ou } [\![p_2]\!]^{\mathcal{E}} = \bot \\ \mathsf{tt} & \text{si } [\![p_1]\!]^{\mathcal{E}} \neq \bot,\ [\![p_2]\!]^{\mathcal{E}} \neq \bot \text{ et } [\![p_1]\!]^{\mathcal{E}} = [\![p_2]\!]^{\mathcal{E}} \\ \mathsf{ff} & \text{si } [\![p_1]\!]^{\mathcal{E}} \neq \bot,\ [\![p_2]\!]^{\mathcal{E}} \neq \bot \text{ et } [\![p_1]\!]^{\mathcal{E}} \neq [\![p_2]\!]^{\mathcal{E}} \end{array} \right. \\
[\![\mathsf{if}\ p_0\ \mathsf{then}\ p_1\ \mathsf{else}\ p_2]\!]^{\mathcal{E}} & \stackrel{def}{=} & \left\{ \begin{array}{ll} [\![p_1]\!]^{\mathcal{E}} & \text{si } [\![p_0]\!]^{\mathcal{E}} = \mathsf{tt} \\ [\![p_2]\!]^{\mathcal{E}} & \text{si } [\![p_0]\!]^{\mathcal{E}} = \mathsf{ff} \\ \bot_{\tau} & \text{si } [\![p_0]\!]^{\mathcal{E}} = \bot,\ \text{où } \tau = t(p_1) = t(p_2) \end{array} \right. \\
[\![p_1 + p_2]\!]^{\mathcal{E}} & \stackrel{def}{=} & \left\{ \begin{array}{ll} \bot_{\mathsf{nat}} & \text{si } [\![p_1]\!]^{\mathcal{E}} = \bot \text{ ou } [\![p_2]\!]^{\mathcal{E}} = \bot \\ [\![p_1]\!]^{\mathcal{E}} + [\![p_2]\!]^{\mathcal{E}} & \text{si } [\![p_1]\!]^{\mathcal{E}} \in \mathbb{N} \text{ et } [\![p_2]\!]^{\mathcal{E}} \in \mathbb{N} \end{array} \right. \\
[\![p_1 \times p_2]\!]^{\mathcal{E}} & \stackrel{def}{=} & \left\{ \begin{array}{ll} \bot_{\mathsf{nat}} & \text{si } [\![p_1]\!]^{\mathcal{E}} = \bot \text{ ou } [\![p_2]\!]^{\mathcal{E}} = \bot \\ [\![p_1]\!]^{\mathcal{E}} \times [\![p_2]\!]^{\mathcal{E}} & \text{si } [\![p_1]\!]^{\mathcal{E}} \in \mathbb{N} \text{ et } [\![p_2]\!]^{\mathcal{E}} \in \mathbb{N} \end{array} \right. \\
[\![p_1 - p_2]\!]^{\mathcal{E}} & \stackrel{def}{=} & \left\{ \begin{array}{ll} \bot_{\mathsf{nat}} & \text{si } [\![p_1]\!]^{\mathcal{E}} = \bot \text{ ou } [\![p_2]\!]^{\mathcal{E}} = \bot \\ \max(0, [\![p_1]\!]^{\mathcal{E}} - [\![p_2]\!]^{\mathcal{E}}) & \text{si } [\![p_1]\!]^{\mathcal{E}} \in \mathbb{N} \text{ et } [\![p_2]\!]^{\mathcal{E}} \in \mathbb{N} \end{array} \right. \\
[\![p_0/2]\!]^{\mathcal{E}} & \stackrel{def}{=} & \left\{ \begin{array}{ll} \bot_{\mathsf{nat}} & \text{si } [\![p_0]\!]^{\mathcal{E}} = \bot \\ \lfloor [\![p_0]\!]^{\mathcal{E}}/2 \rfloor & \text{si } [\![p_0]\!]^{\mathcal{E}} \in \mathbb{N} \end{array} \right. \\
[\![p_1\ p_2]\!]^{\mathcal{E}} & \stackrel{def}{=} & [\![p_1]\!]^{\mathcal{E}}([\![p_2]\!]^{\mathcal{E}})
\end{array}
$$

$[\![\mathsf{fun}\ x \to p_0]\!]^{\mathcal{E}}$ est l'application $f : [\![t(x)]\!] \to [\![t(p_0)]\!]$ telle que $f(e) = [\![p_0]\!]^{\mathcal{E}[x \mapsto e]}$ pour tout $e \in [\![t(x)]\!]$

$$
[\![\mathsf{rec}\ p_0]\!]^{\mathcal{E}} \stackrel{def}{=} \sup_{i \in \mathbb{N}}^{[\![\tau]\!]}\big(([\![p_0]\!]^{\mathcal{E}})^i(\bot_{\tau})\big) \ \text{ où } \ \tau = t(\mathsf{rec}\ p_0)
$$

$$
[\![p_1 \parallel_{\mathsf{u}} p_2]\!]^{\mathcal{E}} \stackrel{def}{=} \left\{ \begin{array}{ll} \bot_{\mathsf{unit}} & \text{si } [\![p_1]\!]^{\mathcal{E}} = \bot \text{ et } [\![p_2]\!]^{\mathcal{E}} = \bot \\ \mathsf{uu} & \text{si } [\![p_1]\!]^{\mathcal{E}} = \mathsf{uu} \text{ ou } [\![p_2]\!]^{\mathcal{E}} = \mathsf{uu} \end{array} \right.
$$

FIGURE 1 – Définition de l'interprétation des programmes.