CONCOURS COMMUNS POLYTECHNIQUES

EPREUVE SPECIFIQUE - FILIERE MP

INFORMATIQUE

Durée : 3 heures

*Les calculatrices **sont interdites**.*

* * *

NB : Le candidat attachera la plus grande importance à la clarté, à la précision et à la concision de la rédaction.
Si un candidat est amené à repérer ce qui peut lui sembler être une erreur d'énoncé, il le signalera sur sa copie et devra poursuivre sa composition en expliquant les raisons des initiatives qu'il a été amené à prendre.

* * *

PRÉAMBULE : Les trois parties qui composent ce sujet sont indépendantes et peuvent être traitées par les candidats dans un ordre quelconque.

Pour les candidats ayant utilisé le langage CaML dans le cadre des enseignements d'informatique, la partie III (Algorithmique et programmation en CaML) se situe en page 5.

Pour les candidats ayant utilisé le langage PASCAL dans le cadre des enseignements d'informatique, la partie III (Algorithmique et programmation en PASCAL) se situe en page 12.

Partie I : Logique et calcul des propositions

Les universités anglophones nord-américaines sont célèbres pour leurs associations estudiantines, les fraternités, ou sororités. Celles-ci développent souvent une culture du secret.
Vous venez de rejoindre la fraternité $\Gamma\alpha\chi$. Lors de votre initiation, vous avez appris les règles gouvernant les discussions pour que seuls les membres puissent en comprendre le contenu. Lorsque plusieurs membres prennent la parole sur un sujet donné, soit ils disent tous la vérité, soit ils mentent tous. Si quelqu'un intervient sur plusieurs sujets, il peut avoir un rôle différent pour chaque sujet.
Pour être définitivement admis, vous devez démontrer votre maîtrise de ces règles. Vous participez à une discussion avec trois membres de la fraternité que nous appellerons A, B et C. Ceux-ci vous indiquent comment rejoindre la salle d'intronisation. Si vous n'arrivez pas à rejoindre directement cette salle, vous ne serez pas admis.
Un premier sujet est abordé concernant la pièce dans laquelle se tiendra la cérémonie :

A : « La cérémonie se tiendra dans le gymnase »

C : « Non, elle ne se tiendra pas dans cette pièce »

A : « Ou alors dans le réfectoire »

Nous noterons G et R les variables propositionnelles associées à la pièce où se tiendra la cérémonie.
Nous noterons A_1 et C_1 les formules propositionnelles correspondant aux affirmations de A et C sur le premier sujet.
Puis, un second sujet est traité concernant les escaliers qui permettent de rejoindre la pièce où se tiendra la cérémonie.

A : « Le premier escalier conduit à l'intronisation »

C : « Tu as raison »

A : « Le troisième escalier y conduit aussi »

B : « Si le deuxième escalier y conduit, alors le troisième n'y conduit pas »

C : « Le deuxième n'y conduit pas »

Nous noterons E_1, E_2, E_3 les variables propositionnelles correspondant au fait que le premier, le deuxième, le troisième escalier conduisent à la salle de cérémonie.
Nous noterons A_2, B_2 et C_2 les formules propositionnelles correspondant aux affirmations de A, B et C sur le second sujet.

Question I.1 *Représenter le premier sujet abordé sous la forme d'une formule du calcul des propositions dépendant des formules A_1, et C_1.*

Question I.2 *Représenter les informations données par les participants sous la forme de deux formules du calcul des propositions A_1 et C_1 dépendant des variables G et R.*

Question I.3 *En utilisant le calcul des propositions (résolution avec les formules de De Morgan), déterminer dans quelle pièce vous devez vous rendre pour rejoindre la cérémonie.*

Question I.4 *Représenter le second sujet abordé sous la forme d'une formule du calcul des propositions dépendant des formules A_2, B_2 et C_2.*

Question I.5 *Représenter les informations données par les participants sous la forme de trois formules du calcul des propositions A_2, B_2 et C_2 dépendant des variables E_1, E_2 et E_3.*

Question I.6 *En utilisant le calcul des propositions (résolution avec les tables de vérité), déterminer l'escalier que vous devez suivre pour rejoindre la cérémonie.*

Question I.7 *En admettant que les trois participants aient menti, pouviez vous prendre d'autres escaliers ? Si oui, le ou lesquels ?*

Partie II : Automates et langages

Le but de cet exercice est l'étude des propriétés de l'opération $\ominus$ de composition de deux automates.

1 Automate fini complet déterministe

Pour simplifier les preuves, nous nous limiterons au cas des automates finis complets déterministes. Les résultats étudiés s'étendent au cadre des automates finis complets quelconques.

1.1 Représentation d'un automate fini complet déterministe

Déf. II.1 (Automate fini complet déterministe) Soit l'alphabet X (un ensemble de symboles), soit Λ le symbole représentant le mot vide ($\Lambda \notin X$), soit $X^\star$ l'ensemble contenant Λ et les mots composés de séquences de symboles de X (donc $\Lambda \in X^\star$) ; un automate fini complet déterministe sur X est un quintuplet $A = (Q, X, i, T, \delta)$ composé de :
- un ensemble fini d'états : Q ;
- un état initial : $i \in Q$;
- un ensemble d'états terminaux : $T \subseteq Q$;
- une fonction totale de transition confondue avec son graphe : $\delta \subseteq Q \times X \mapsto Q$.

Pour une transition $\delta(o, e) = d$ donnée, nous appelons o l'origine de la transition, e l'étiquette de la transition et d la destination de la transition.

1.2 Représentation graphique d'un automate

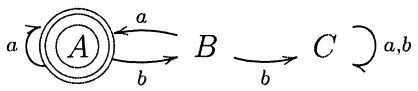
Les automates peuvent être représentés par un schéma suivant les conventions :
- les valeurs de la fonction totale de transition δ sont représentées par un graphe orienté dont les nœuds sont les états et les arêtes sont les transitions ;
- un état initial est entouré d'un cercle (i) ;
- un état terminal est entouré d'un double cercle $((t))$;
- un état qui est à la fois initial et terminal est entouré d'un triple cercle $(((it)))$;
- une arête étiquetée par le symbole $e \in X$ va de l'état o à l'état d si et seulement si $\delta(o, e) = d$.

Exemple II.1 L'automate $\mathcal{E}_1 = (Q_1, X, i_1, T_1, \delta_1)$ avec :

$$\begin{aligned} Q_1 &= \{A, B, C\} \\ X &= \{a, b\} \\ i_1 &= A \\ T_1 &= \{A\} \end{aligned}$$

$$\begin{aligned} \delta_1(A, a) &= A, & \delta_1(A, b) &= B, \\ \delta_1(B, a) &= A, & \delta_1(B, b) &= C, \\ \delta_1(C, a) &= C, & \delta_1(C, b) &= C, \end{aligned}$$

est représenté par le graphe suivant :

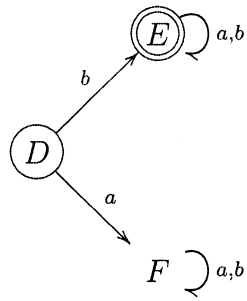


Exemple II.2 L'automate $\mathcal{E}_2 = (Q_2, X, i_2, T_2, \delta_2)$ avec :

$$\begin{aligned} Q_2 &= \{D, E, F\} \\ X &= \{a, b\} \\ i_2 &= D \\ T_2 &= \{E\} \end{aligned}$$

$$\begin{array}{ll} \delta_2(D,a) = F, & \delta_2(D,b) = E, \\ \delta_2(E,a) = E, & \delta_2(E,b) = E, \\ \delta_2(F,a) = F, & \delta_2(F,b) = F, \end{array}$$

est représenté par le graphe suivant :



1.3 Langage reconnu par un automate fini complet déterministe

Soit $\delta^\star$ l'extension de δ à $Q \times X^\star \mapsto Q$ définie par :

$$\forall q \in Q,\ \delta^\star(q, \Lambda) = q$$

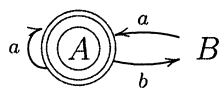
$$\left\{ \begin{array}{l} \forall e \in X, \\ \forall m \in X^\star, \\ \forall o \in Q, \\ \forall d \in Q, \end{array} \right. \quad \delta^\star(o, e.m) = d \Leftrightarrow \exists q \in Q,\ (\delta(o,e) = q) \wedge (\delta^\star(q,m) = d)$$

Soit X un alphabet, un langage sur X est un sous-ensemble de $X^\star$.
Le langage sur X reconnu par un automate fini complet déterministe est :

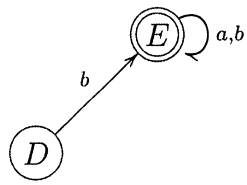
$$L(A) = \{m \in X^\star \mid \exists d \in T,\ \delta^\star(i,m) = d\}$$

Notons que certains états et transitions ne sont pas utiles dans la description d'un langage car ils ne permettent pas d'aller d'un état initial à un état terminal.

Exemple II.3 Le sous-automate de $\mathcal{E}_1$ (exemple II.1) composé des états et transitions utiles est représenté par le graphe suivant :



Exemple II.4 Le sous-automate de $\mathcal{E}_2$ (exemple II.2) composé des états et transitions utiles est représenté par le graphe suivant :



Question II.1 *Donner, sans les justifier, deux expressions régulières ou ensemblistes représentant les langages sur $X = \{a, b\}$ reconnus par les automates $\mathcal{E}_1$ de l'exemple II.1 et $\mathcal{E}_2$ de l'exemple II.2.*

2 Composition d'automates finis complets déterministes

2.1 Définition

Soit l'opération interne $\lhd$ sur les automates finis complets déterministes définie par :

Déf. II.2 (Composition d'automates finis complets déterministes) Soient $\mathcal{A}_1 = (Q_1, X, i_1, T_1, \delta_1)$ et $\mathcal{A}_2 = (Q_2, X, i_2, T_2, \delta_2)$ deux automates finis complets déterministes, l'automate $\mathcal{A} = \mathcal{A}_1 \lhd \mathcal{A}_2$ qui résulte de la composition de $\mathcal{A}_1$ et $\mathcal{A}_2$ est défini par :

$$\mathcal{A} = \mathcal{A}_1 \lhd \mathcal{A}_2 = (Q_1 \times Q_2, X, (i_1, i_2), T_1 \times (Q_2 \setminus T_2), \delta_{1\lhd 2})$$

$$\begin{cases} \forall o_1 \in Q_1, \\ \forall o_2 \in Q_2, \\ \forall x \in X, \quad \delta_{1\lhd 2}((o_1, o_2), x) = (d_1, d_2) \Leftrightarrow (\delta_1(o_1, x) = d_1) \wedge (\delta_2(o_2, x) = d_2) \\ \forall d_1 \in Q_1, \\ \forall d_2 \in Q_2, \end{cases}$$

Question II.2 *En considérant les exemples II.1 et II.2, construire le graphe représentant l'automate $\mathcal{E}_1 \lhd \mathcal{E}_2$ (seuls les états et les transitions utiles devront être construits).*

Question II.3 *Caractériser le langage reconnu par $\mathcal{E}_1 \lhd \mathcal{E}_2$, par une expression régulière ou ensembliste.*

2.2 Propriétés

Question II.4 *Montrer que si $\mathcal{A}_1$ et $\mathcal{A}_2$ sont des automates finis complets déterministes, alors $\mathcal{A}_1 \lhd \mathcal{A}_2$ est un automate fini complet déterministe.*

Question II.5 *Montrer que :*

$$\begin{cases} \forall o_1 \in Q_1, \\ \forall o_2 \in Q_2, \\ \forall m \in X^\star, \quad \delta^\star_{1\lhd 2}((o_1, o_2), m) = (d_1, d_2) \Leftrightarrow (\delta^\star_1(o_1, m) = d_1) \wedge (\delta^\star_2(o_2, m) = d_2) \\ \forall d_1 \in Q_1, \\ \forall d_2 \in Q_2, \end{cases}$$

Question II.6 *Soient $\mathcal{A}_1$ et $\mathcal{A}_2$ des automates finis complets déterministes, montrer que :*

$$m \in L(\mathcal{A}_1 \lhd \mathcal{A}_2) \Leftrightarrow m \in L(\mathcal{A}_1) \wedge m \notin L(\mathcal{A}_2)$$

Question II.7 *Quelle relation liant les langages reconnus par les automates $\mathcal{A}_1$, $\mathcal{A}_2$ et $\mathcal{A}_1 \lhd \mathcal{A}_2$ peut-on en déduire ?*

Partie III : Algorithmique et programmation en CaML

Cette partie doit être traitée par les candidats qui ont utilisé le langage CaML dans le cadre des enseignements d'informatique. Les fonctions écrites devront être récursives ou faire appel à des fonctions auxiliaires récursives. Elles ne devront pas utiliser d'instructions itératives (`for`, `while`, ...) ni de références.

1 Exercice : le tri fusion

L'objectif de cet exercice est d'étudier une implantation particulière d'un algorithme de tri d'une séquence d'entiers.

Déf. III.1 Une séquence s de taille n de valeurs v_i avec $1 \leq i \leq n$ est notée $\langle v_1, \ldots, v_n \rangle$. Une même valeur v peut figurer plusieurs fois dans s, nous noterons $card(v, s)$ le cardinal de v dans s, c'est-à-dire le nombre de fois que v figure dans s avec :

$$card(v, \langle v_1, \ldots, v_n \rangle) = card\{i \in \mathbb{N} \mid 1 \leq i \leq n, v = v_i\}$$

Une séquence d'entiers est représentée par le type `sequence` et une paire de séquences d'entiers est représentée par le type `paire` dont les définitions sont :

```
type sequence == int list;;

type paire == sequence * sequence;;
```

Soit le programme en langage CaML :

```
let rec eclater s =
  match s with
    | [] -> ([],[])
    | [e] -> ([e],[])
    | e1::e2::q ->
        let (p1,p2) = eclater q in
          (e1::p1,e2::p2);;

let rec fusionner s1 s2 =
  match s1,s2 with
    | (e1::q1,e2::q2) ->
        if (e1 <= e2)
        then e1::(fusionner q1 s2)
        else e2::(fusionner s1 q2)
    | (_,_) -> s1 @ s2;;

let rec trier s =
  match s with
    | [] -> []
    | [e] -> [e]
    | _ ->
        let (s1,s2) = eclater s in
          fusionner (trier s1) (trier s2);;
```

Soit la constante `exemple` définie et initialisée par :

```
let exemple = [ 4; 3; 2; 1];;
```

Question III.1 *Détailler les étapes du calcul de `(trier exemple)` en précisant pour chaque appel aux fonctions `eclater`, `fusionner` et `trier`, la valeur du paramètre et du résultat.*

Question III.2 *Soit la séquence* $\mathtt{p} = \langle p_1, \ldots, p_n \rangle$ *de taille* n*, soient les séquences* $\mathtt{a} = \langle a_1, \ldots, a_r \rangle$ *de taille* r *et* $\mathtt{b} = \langle b_1, \ldots, b_s \rangle$ *de taille* s *telles que* `(a,b) = (eclater p)`*, montrer que :*

$$\begin{array}{l} \forall i, 1 \leq i \leq n, card(p_i, p) = card(p_i, a) + card(p_i, b) \\ s = E(n/2) \\ r = n - E(n/2) \end{array}$$

Question III.3 *Soient les séquences* $\mathtt{a} = \langle a_1, \ldots, a_m \rangle$ *de taille* m *et* $\mathtt{b} = \langle b_1, \ldots, b_n \rangle$ *de taille* n*, soit la séquence* $\mathtt{r} = \langle r_1, \ldots, r_l \rangle$ *de taille* l *telle que* `r = (fusionner a b)`*, montrer que :*

$$\begin{array}{l} l = m + n \\ \forall k, 1 \leq k \leq l, card(r_k, r) = card(r_k, a) + card(r_k, b) \\ \left\{ \begin{array}{l} \forall i, 1 \leq i < m, a_i \leq a_{i+1} \\ \forall j, 1 \leq j < n, b_j \leq b_{j+1} \end{array} \right. \Rightarrow \forall k, 1 \leq k < l, r_k \leq r_{k+1} \end{array}$$

Question III.4 *Soit la séquence* $\mathtt{p} = \langle p_1, \ldots, p_m \rangle$ *de taille* m*, soit la séquence* $\mathtt{r} = \langle r_1, \ldots, r_n \rangle$ *telle que* `r = (trier p)`*, montrer que :*

$$\begin{array}{l} m = n \\ \forall i, 1 \leq i \leq m, card(p_k, r) = card(p_i, p) \\ \forall i, 1 \leq i < n, r_i \leq r_{i+1} \end{array}$$

Question III.5 *Montrer que le calcul des fonctions* `eclater`*,* `fusionner` *et* `trier` *se termine quelles que soient les valeurs de leurs paramètres respectant le type des fonctions.*

Question III.6 *Donner des exemples de valeurs du paramètre* `s` *de la fonction* `trier` *qui correspondent aux meilleur et pire cas en nombre d'appels récursifs effectués.*
Calculer une estimation de la complexité dans les meilleur et pire cas des fonctions `eclater`*,* `fusionner` *et* `trier` *en fonction du nombre de valeurs dans les séquences données en paramètre. Cette estimation ne prendra en compte que le nombre d'appels récursifs effectués.*

2 Problème : Représentation d'ensembles avec des intervalles

De nombreux algorithmes reposent sur la manipulation d'ensembles d'éléments ordonnés. Lorsque ces ensembles contiennent de nombreux éléments adjacents (aucune valeur n'existe entre les deux éléments), il est plus performant en terme de temps de calcul et d'occupation mémoire de manipuler des intervalles au lieu de valeurs singulières. Cela permet également de manipuler des ensembles infinis sous la forme d'un ensemble fini d'intervalles contenant un nombre de valeurs infinies (i.e. dans $\mathbb{Q}$ ou $\mathbb{R}$).
L'objectif de ce problème est de comparer deux implantations différentes d'un ensemble d'entiers, la première à base de listes triées d'intervalles, et la seconde à base d'arbres binaires d'intervalles.

Nous nous limiterons à des intervalles fermés de $\mathbb{R}$ dont les bornes sont des entiers $[min, max]$. L'extension à des intervalles ouverts et aux valeurs $-\infty$ et $+\infty$ ne pose pas de problème majeur.

2.1 Préliminaires : Représentation des intervalles

2.1.1 Définitions

Déf. III.2 (Intervalles disjoints) Deux intervalles sont disjoints si leur intersection est vide.

Déf. III.3 (Fusion d'intervalles) La fusion de deux intervalles a comme minimum le plus petit des minima des deux intervalles, et comme maximum le plus grand des maxima des deux intervalles. Cette opération correspond à l'union des intervalles si ceux-ci ne sont pas disjoints.

2.1.2 Représentation en CaML

Nous ferons l'hypothèse que les couples qui représentent des intervalles sont bien formés, c'est-à-dire que la valeur représentant le minimum est inférieure ou égale à la valeur représentant le maximum.

Un intervalle est représenté par le type `intervalle` équivalent à une paire de `int` (son minimum en premier et son maximum en second).

```
type bloc == int * int;;
```

Nous allons maintenant définir plusieurs opérations sur les intervalles.

2.1.3 Opérations sur la structure d'intervalle

La première opération est le test si deux intervalles sont disjoints.

Question III.7 *Écrire en CaML une fonction `disjoints` de type `intervalle -> intervalle -> bool` telle que l'appel `(disjoints i1 i2)` renvoie la valeur `true` si les intervalles `i1` et `i2` sont disjoints, et la valeur `false` sinon.*

La seconde opération est la fusion de deux intervalles.

Question III.8 *Écrire en CaML une fonction `fusion` de type `intervalle -> intervalle -> intervalle` telle que l'appel `(fusion i1 i2)` renvoie un intervalle correspondant à la fusion de `i1` et `i2`.*

2.2 Réalisation à base d'une liste triée d'intervalles

La réalisation la plus simple d'un ensemble de valeurs en utilisant des intervalles consiste à utiliser une liste d'intervalles. Pour réduire la complexité amortie de certaines opérations, nous utiliserons une liste triée d'intervalles d'entiers.

2.2.1 Définitions

Les algorithmes manipuleront des listes d'intervalles respectant certaines contraintes que nous appellerons liste bien formée.

Déf. III.4 (Liste bien formée d'intervalles) Une liste bien formée d'intervalles est une liste d'intervalles qui respecte les contraintes suivantes :
- les intervalles sont bien formés,
- les intervalles sont disjoints deux à deux,
- la liste est triée selon la relation d'ordre suivante : un intervalle i_1 est strictement plus petit qu'un intervalle i_2 si et seulement si le maximum de i_1 est strictement plus petit que le minimum de i_2.

Question III.9 *Montrer qu'une liste triée d'intervalles bien formés, est une liste bien formée d'intervalles.*

Question III.10 *Exprimer la définition III.4 sous la forme d'une propriété $LBFI(e)$ qui indique que $e = \langle v_1, \ldots, v_n \rangle$ est une liste bien formée d'intervalles (contenant les intervalles v_i).*

2.2.2 Représentation en CaML

Une liste triée d'intervalles est représentée par le type `liste` équivalent à une liste d'`intervalle`.

```
type liste == intervalle list;;
```

Nous allons maintenant définir plusieurs opérations sur les listes bien formées d'intervalles et estimer leur complexité.

2.2.3 Opérations sur la structure de liste bien formée d'intervalles

La première opération est l'ajout d'un intervalle dans une liste bien formée d'intervalles.

Question III.11 *Écrire en CaML une fonction `ajouter` de type `intervalle -> liste -> liste` telle que l'appel `(ajouter i l)` sur une liste bien formée d'intervalles `l` renvoie une liste bien formée d'intervalles contenant les intervalles contenus dans la liste `l` qui sont disjoints de `i`, et :*
- *soit le résultat de la fusion de `i` et des intervalles contenus dans la liste `l` qui ne sont pas disjoints de `i`,*
- *soit l'intervalle `i`, si tous les intervalles contenus dans la liste `l` lui sont disjoints.*

L'algorithme utilisé ne devra parcourir qu'une seule fois la liste. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.

Question III.12 *Donner des exemples de valeurs des paramètres `i` et `l` de la fonction `ajouter` qui correspondent aux meilleur et pire cas en nombre d'appels récursifs effectués.*
Calculer une estimation de la complexité dans les meilleur et pire cas de la fonction `ajouter` en fonction de la longueur de la liste `l`. Cette estimation ne prendra en compte que le nombre d'appels récursifs effectués.

La deuxième opération est le test d'appartenance d'une valeur dans une liste bien formée d'intervalles.

Question III.13 *Écrire en CaML une fonction `appartenir` de type `int -> liste -> bool` telle que l'appel `(appartenir v l)` sur une liste bien formée d'intervalles `l` renvoie la valeur `true` si la valeur `v` appartient à un des intervalles contenus dans la liste `l` et la valeur `false` sinon.*
L'algorithme utilisé ne devra parcourir qu'une seule fois la liste. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.

La troisième opération est la vérification qu'une liste d'intervalles est bien formée, c'est-à-dire respecte les contraintes de la définition III.4 formalisée par la propriété $LBFI(\mathtt{e})$ dans la question III.10.

Question III.14 *Écrire en CaML une fonction `verifier` de type `liste -> bool` telle que l'appel `(verifier l)` sur une liste d'intervalles `l` renvoie la valeur `true` si la liste `l` respecte les contraintes d'une liste bien formée d'intervalles, et la valeur `false` sinon.*
L'algorithme utilisé ne devra parcourir qu'une seule fois la liste. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.

2.3 Réalisation à base d'arbres binaires

L'utilisation de la structure d'arbre binaire pour représenter un ensemble à base d'intervalles permet de réduire la complexité en temps de calcul pour les différentes opérations.
Une liste est représentée par un arbre binaire dont les feuilles sont les intervalles.

2.4 Arbre binaire d'intervalles

2.4.1 Définition

Déf. III.5 (Arbre binaire d'intervalles) Un arbre binaire d'intervalles a est une structure qui peut :
- soit être vide notée $\emptyset$,
- soit être une feuille qui contient un intervalle notée $[min, max]$,
- soit un nœud qui contient un sous-arbre gauche (noté $\mathcal{G}(a)$) et un sous-arbre droit (noté $\mathcal{D}(a)$) qui sont tous deux des arbres binaires d'intervalles non vides.

L'ensemble des feuilles, respectivement des nœuds, est noté $\mathcal{F}$, respectivement $\mathcal{N}$. L'ensemble des arbres est $\mathcal{A} = \mathcal{F} \cup \mathcal{N}$.
L'ensemble des feuilles, respectivement des nœuds, d'un arbre a est noté $\mathcal{F}(a)$, respectivement $\mathcal{N}(a)$.

Déf. III.6 (Intervalle englobant) L'intervalle englobant d'un arbre d'intervalles est le plus petit intervalle bien formé qui contient tous les intervalles contenus dans les feuilles de l'arbre. L'intervalle englobant d'un arbre d'intervalles a est noté $\mathcal{I}(a)$.

Déf. III.7 (Parcours gauche-droite d'un arbre binaire) Le parcours gauche-droite d'un arbre binaire a, noté $\mathcal{E}(a)$, construit la séquence des feuilles de l'arbre dans l'ordre de gauche à droite. Il est défini par les équations :

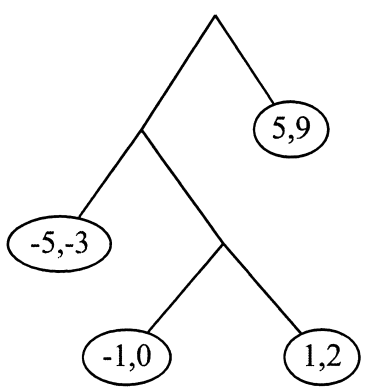
$$\mathcal{E}(a) = \begin{cases} \langle [min_a, max_a] \rangle & \text{si } a \in \mathcal{F} \\ \langle f_1, \ldots, f_{m+n} \rangle & \text{si } a \in \mathcal{N} \\ & \text{et } \langle f_1, \ldots, f_m \rangle = \mathcal{E}(\mathcal{G}(a)) \\ & \text{et } \langle f_{m+1}, \ldots, f_{m+n} \rangle = \mathcal{E}(\mathcal{D}(a)) \end{cases}$$

Déf. III.8 (Arbre bien formé d'intervalles) Un arbre bien formé d'intervalles est un arbre d'intervalles qui respecte les contraintes suivantes :
- les intervalles contenus dans les feuilles sont bien formés,
- les intervalles contenus dans les feuilles sont disjoints deux à deux,
- la liste d'intervalles obtenue par le parcours gauche-droite de l'arbre est bien formée.

Question III.15 *Soit a un arbre d'intervalles bien formés. Montrer que si pour tous les nœuds de l'arbre, le maximum de l'intervalle englobant du fils gauche est strictement plus petit que le minimum de l'intervalle englobant du fils droit, alors l'arbre est bien formé.*

Question III.16 *Exprimer la définition III.8 sous la forme d'une propriété $ABFI(a)$ qui indique que a est un arbre bien formé d'intervalles. Donner deux formes à cette définition, une qui s'appuie sur le parcours gauche-droite, et une qui s'appuie uniquement sur la structure de l'arbre.*

Exemple III.1 (Arbres binaires d'intervalles) Voici un exemple d'arbres binaires d'intervalles :



2.4.2 Représentation des arbres binaires d'intervalles en CaML

Un arbre binaire de blocs est représenté par le type CaML :

```
type arbre = Vide
           | Feuille of int * int
           | Noeud of arbre * arbre;;
```

Dans l'appel `Feuille( min, max)`, les paramètres `min` et `max` sont respectivement les valeurs minimum et maximum de l'intervalle contenu dans la feuille.
Dans l'appel `Noeud( fg, fd)`, les paramètres `fg` et `fd` sont respectivement le fils gauche et le fils droit de la racine de l'arbre créé.

Exemple III.2 Le terme suivant

```
Noeud(
   Noeud(
      Feuille( -5, -3 ),
      Noeud(
         Feuille( -1, 0),
         Feuille( 1, 2)
      )
   ),
   Feuille( 5, 9)
)
```

est alors associé au troisième arbre binaire de blocs représenté graphiquement dans l'exemple III.1.

2.4.3 Opérations sur la structure d'arbre d'intervalles

La première opération consiste à déterminer l'intervalle englobant d'un arbre d'intervalles.

Question III.17 *Écrire en CaML une fonction `englobant` de type `arbre -> intervalle` telle que l'appel `(englobant a)` renvoie l'intervalle englobant de l'arbre d'intervalles bien formé non vide `a`. L'algorithme utilisé ne devra parcourir qu'une seule fois l'arbre. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

La deuxième opération est la vérification qu'un arbre d'intervalles est bien formé, c'est-à-dire qu'il respecte les contraintes de la définition III.8.

Question III.18 *Écrire en CaML une fonction `verifier` de type `arbre -> bool` telle que l'appel `(verifier a)` sur un arbre d'intervalles `a` renvoie la valeur `true` si `a` est un arbre d'intervalles bien formé, et la valeur `false` sinon. L'algorithme utilisé ne devra parcourir qu'une seule fois l'arbre. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

La troisième opération est le test d'appartenance d'une valeur dans un arbre d'intervalles bien formé.

Question III.19 *Écrire en CaML une fonction `appartenir` de type `int -> arbre -> bool` telle que l'appel `(appartenir v a)` sur un arbre d'intervalles bien formé `a` renvoie la valeur `true` si la valeur `v` appartient à un des intervalles contenus dans les feuilles de `a` et la valeur `false` sinon.*
L'algorithme utilisé ne devra parcourir qu'une seule fois l'arbre. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.

La quatrième opération est le découpage d'un arbre bien formé d'intervalles selon une valeur.

Question III.20 *Écrire en CaML une fonction* `decouper` *de type* `int -> arbre -> arbre * arbre * arbre` *telle que l'appel* `(decouper v a)` *sur un arbre d'intervalles bien formé* `a` *renvoie un triplet composé :*
- *d'un arbre bien formé contenant les intervalles strictement inférieur à la valeur* `v`*,*
- *d'un arbre bien formé contenant les intervalles contenant la valeur* `v`*,*
- *d'un arbre bien formé contenant les intervalles strictement supérieur à la valeur* `v`*.*

L'algorithme utilisé ne devra parcourir qu'une seule fois l'arbre. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.

La dernière opération est l'ajout d'un intervalle dans un arbre d'intervalles bien formé.

Question III.21 *Écrire en CaML une fonction* `ajouter` *de type* `intervalle -> arbre -> arbre` *telle que l'appel* `(ajouter i a)` *sur un arbre d'intervalles bien formé* `a` *renvoie un arbre d'intervalles bien formé contenant les intervalles contenus dans les feuilles de* `a` *qui sont disjoints de* `i`*, et :*
- *soit le résultat de la fusion de* `i` *et des intervalles contenus dans les feuilles de* `a` *qui ne sont pas disjoints de* `i`*,*
- *soit l'intervalle* `i`*, si tous les intervalles contenus dans les feuilles de* `a` *lui sont disjoints.*

L'algorithme utilisé ne devra parcourir qu'une seule fois l'arbre. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.

Question III.22 *Donner des exemples de valeurs des paramètres* `i` *et* `a` *de la fonction* `ajouter` *qui correspondent aux meilleur et pire cas en nombre d'appels récursifs effectués.*
Calculer une estimation de la complexité dans les meilleur et pire cas de la fonction `ajouter` *en fonction de la profondeur de l'arbre* `a`*. Cette estimation ne prendra en compte que le nombre d'appels récursifs effectués.*

Partie III : Algorithmique et programmation en PASCAL

Cette partie doit être traitée par les candidats qui ont utilisé le langage PASCAL dans le cadre des enseignements d'informatique. Les fonctions écrites devront être récursives ou faire appel à des fonctions auxiliaires récursives. Elles ne devront pas utiliser d'instructions itératives (`for`, `while`, `repeat`, ...).

1 Exercice : le tri fusion

L'objectif de cet exercice est d'étudier une implantation particulière d'un algorithme de tri d'une séquence d'entiers.

Déf. III.1 Une séquence s de taille n de valeurs v_i avec $1 \leq i \leq n$ est notée $\langle v_1, \ldots, v_n \rangle$. Une même valeur v peut figurer plusieurs fois dans s, nous noterons $card(v, s)$ le cardinal de v dans s, c'est-à-dire le nombre de fois que v figure dans s avec :

$$card(v, \langle v_1, \ldots, v_n \rangle) = card\{i \in \mathbb{N} \mid 1 \leq i \leq n, v = v_i\}$$

Une séquence d'entiers est représentée par le type `SEQUENCE`. Nous supposons prédéfinies les constantes et les fonctions suivantes dont le calcul se termine quelles que soient les valeurs de leurs paramètres. Elles pourront éventuellement être utilisées dans les réponses aux questions :
- `NIL` représente la séquence vide d'entiers ;

- `FUNCTION S_Inserer(e:INTEGER;s:SEQUENCE):SEQUENCE;` renvoie une séquence d'entiers composée d'un premier entier `e` et du reste de la séquence contenu dans `s` ;
- `FUNCTION S_Tete(s:SEQUENCE):INTEGER;` renvoie le premier entier de la séquence `s`. Cette séquence ne doit pas être vide ;
- `FUNCTION S_Queue(s:SEQUENCE):SEQUENCE;` renvoie le reste de la séquence `s` privée de son premier entier. Cette séquence ne doit pas être vide ;
- `FUNCTION S_Juxtaposer(s1,s2:SEQUENCE):SEQUENCE;` renvoie une séquence composée des éléments de la séquence `s1` dans le même ordre que dans `s1` suivis des éléments de la séquence `s2` dans le même ordre que dans `s2`.

Une paire de séquence d'entiers est représentée par le type `PAIRE`. Nous supposons prédéfinies les constantes et les fonctions suivantes dont le calcul se termine quelles que soient les valeurs de leurs paramètres. Elles pourront éventuellement être utilisées dans les réponses aux questions :

- `FUNCTION P_Creer(s1,s2:SEQUENCE):PAIRE;` renvoie une paire de séquences dont le premier élément est `s1` et le second élément `s2`,
- `FUNCTION P_Premier(p:PAIRE):SEQUENCE;` renvoie la première séquence contenue dans la paire de séquences `p`,
- `FUNCTION P_Second(p:PAIRE):SEQUENCE;` renvoie la seconde séquence contenue dans la paire de séquences `p`.

Soit le programme en langage PASCAL :

```
FUNCTION eclater(s : SEQUENCE):PAIRE;
VAR e1, e2 : INTEGER; q : SEQUENCE; p : PAIRE;
BEGIN
  IF (s = NIL) THEN
     eclater := P_Creer( NIL, NIL)
  ELSE BEGIN
     e1 := S_Tete( s ); q := S_Queue( s );
     IF (q = NIL) THEN
        eclater := P_Creer( S_Inserer( e1, NIL), NIL)
     ELSE BEGIN
        e2 := S_Tete( q ); q := S_Queue( q );
        p := eclater( q );
        eclater := P_Creer( S_Inserer( e1, P_Premier( p )),
                            S_Inserer( e2, P_Second( p )))
     END
  END
END;

FUNCTION fusionner(s1,s2 : SEQUENCE):SEQUENCE;
VAR e1, e2   : INTEGER; q1, q2 : SEQUENCE;
BEGIN
   IF (s1 <> NIL) AND (s2 <> NIL) THEN BEGIN
      e1 := S_Tete( s1 ); q1 := S_Queue( s1 );
      e2 := S_Tete( s2 ); q2 := S_Queue( s2 );
      IF (e1 <= e2) THEN
         fusionner := S_Inserer( e1, fusionner( q1, s2))
      ELSE
         fusionner := S_Inserer( e2, fusionner( s1, q2))
   END ELSE
      fusionner := S_Juxtaposer( s1, s2)
END;
```

```
FUNCTION trier(s : SEQUENCE) : SEQUENCE;
VAR p : PAIRE;
BEGIN
   IF (s = NIL) THEN
      trier := NIL
   ELSE BEGIN
      IF (S_Queue(s) = NIL) THEN
         trier := s
      ELSE BEGIN
         p := eclater( s );
         trier := fusionner( trier( P_Premier( p )),
                                    trier( P_Second( p )))
      END
   END
END;
```

Soit la constante `exemple` définie et initialisée par :

```
CONST exemple : SEQUENCE
   = S_Ajouter( 4,  S_Ajouter( 3,
                        S_Ajouter( 2, S_Ajouter(1, NIL))));
```

Question III.1 *Détailler les étapes du calcul de* `trier(exemple)` *en précisant pour chaque appel aux fonctions* `eclater`, `fusionner` *et* `trier`, *la valeur du paramètre et du résultat.*

Question III.2 *Soit la séquence* $\mathtt{p} = \langle p_1, \ldots, p_n \rangle$ *de taille* n, *soient les séquences* $\mathtt{a} = \langle a_1, \ldots, a_r \rangle$ *de taille* r *et* $\mathtt{b} = \langle b_1, \ldots, b_s \rangle$ *de taille* s *telles que* `c = eclater(p), a = P_Premier(c)` *et* `b = P_Premier(c)`, *montrer que :*

$$\begin{array}{l} \forall i, 1 \leq i \leq n, card(p_i, p) = card(p_i, a) + card(p_i, b) \\ s = E(n/2) \\ r = n - E(n/2) \end{array}$$

Question III.3 *Soient les séquences* $\mathtt{a} = \langle a_1, \ldots, a_m \rangle$ *de taille* m *et* $\mathtt{b} = \langle b_1, \ldots, b_n \rangle$ *de taille* n, *soit la séquence* $\mathtt{r} = \langle r_1, \ldots, r_l \rangle$ *de taille* l *telle que* `r = fusionner(a,b)`, *montrer que :*

$$\begin{array}{l} l = m + n \\ \forall k, 1 \leq k \leq l, card(r_k, r) = card(r_k, a) + card(r_k, b) \\ \left\{ \begin{array}{l} \forall i, 1 \leq i < m, a_i \leq a_{i+1} \\ \forall j, 1 \leq j < n, b_j \leq b_{j+1} \end{array} \right. \Rightarrow \forall k, 1 \leq k < l, r_k \leq r_{k+1} \end{array}$$

Question III.4 *Soit la séquence* $\mathtt{p} = \langle p_1, \ldots, p_n \rangle$ *de taille* n, *soit la séquence* $\mathtt{r} = \langle r_1, \ldots, r_n \rangle$ *telle que* `r = trier(p)`, *montrer que :*

$$\begin{array}{l} m = n \\ \forall i, 1 \leq i \leq m, card(p_k, r) = card(p_i, p) \\ \forall i, 1 \leq i < n, r_i \leq r_{i+1} \end{array}$$

Question III.5 *Montrer que le calcul des fonctions* `eclater`, `fusionner` *et* `trier` *se termine quelles que soient les valeurs de leurs paramètres respectant le type des fonctions.*

Question III.6 *Donner des exemples de valeurs du paramètre* `s` *de la fonction* `trier` *qui correspondent aux meilleur et pire cas en nombre d'appels récursifs effectués.*
Calculer une estimation de la complexité dans les meilleur et pire cas des fonctions `eclater`, `fusionner` *et* `trier` *en fonction du nombre de valeurs dans les séquences données en paramètre. Cette estimation ne prendra en compte que le nombre d'appels récursifs effectués.*

2 Problème : Représentation d'ensembles avec des intervalles

De nombreux algorithmes reposent sur la manipulation d'ensembles d'éléments ordonnés. Lorsque ces ensembles contiennent de nombreux éléments adjacents (aucune valeur n'existe entre les deux éléments), il est plus performant en terme de temps de calcul et d'occupation mémoire de manipuler des intervalles au lieu de valeurs singulières. Cela permet également de manipuler des ensembles infinis sous la forme d'un ensemble fini d'intervalles contenant un nombre de valeurs infinies (i.e. dans $\mathbb{Q}$ ou $\mathbb{R}$).
L'objectif de ce problème est de comparer deux implantations différentes d'un ensemble d'entiers, la première à base de listes triées d'intervalles, et la seconde à base d'arbres binaires d'intervalles.

Nous nous limiterons à des intervalles fermés de $\mathbb{R}$ dont les bornes sont des entiers $[min, max]$. L'extension à des intervalles ouverts et aux valeurs $-\infty$ et $+\infty$ ne pose pas de problème majeur.

2.1 Préliminaires : Représentation des intervalles

2.1.1 Définitions

Déf. III.2 (Intervalles disjoints) Deux intervalles sont disjoints si leur intersection est vide.

Déf. III.3 (Fusion d'intervalles) La fusion de deux intervalles a comme minimum le plus petit des minima des deux intervalles, et comme maximum le plus grand des maxima des deux intervalles. Cette opération correspond à l'union des intervalles si ceux-ci ne sont pas disjoints.

2.1.2 Représentation en PASCAL

Nous ferons l'hypothèse que les couples qui représentent des intervalles sont bien formés, c'est-à-dire que la valeur représentant le minimum est inférieure ou égale à la valeur représentant le maximum.

Un intervalle est représenté par le type de base `INTERVALLE`.
Nous supposons prédéfinies les constantes et les fonctions suivantes dont le calcul se termine quelles que soient les valeurs de leurs paramètres. Elles pourront éventuellement être utilisées dans les réponses aux questions :
- `FUNCTION I_Creer(min,max:INTEGER):INTERVALLE;` renvoie un intervalle dont le minimum est `min` et le maximum est `max`,
- `FUNCTION I_Minimum(i:INTERVALLE):INTEGER;` renvoie le minimum de l'intervalle `i`,
- `FUNCTION I_Maximum(i:INTERVALLE):INTEGER;` renvoie le maximum de l'intervalle `i`.

Nous allons maintenant définir plusieurs opérations sur les intervalles.

2.1.3 Opérations sur la structure d'intervalle

La première opération est le test si deux intervalles sont disjoints.

Question III.7 *Écrire en PASCAL une fonction* `disjoints(i1:INTERVALLE;i2:INTERVALLE):BOOLEAN;` *telle que l'appel* `disjoints(i1,i2)` *renvoie la valeur* `TRUE` *si les intervalles* `i1` *et* `i2` *sont disjoints, et la valeur* `FALSE` *sinon.*

La seconde opération est la fusion de deux intervalles.

Question III.8 *Écrire en PASCAL une fonction* `fusion(i1,i2:INTERVALLE):INTERVALLE;` *telle que l'appel* `fusion(i1,i2)` *renvoie un intervalle correspondant à la fusion de* `i1` *et* `i2`.

2.2 Réalisation à base d'une liste triée d'intervalles

La réalisation la plus simple d'un ensemble de valeurs en utilisant des intervalles consiste à utiliser une liste d'intervalles. Pour réduire la complexité amortie de certaines opérations, nous utiliserons une liste triée d'intervalles d'entiers.

2.2.1 Définitions

Les algorithmes manipuleront des listes d'intervalles respectant certaines contraintes que nous appellerons liste bien formée.

Déf. III.4 (Liste bien formée d'intervalles) Une liste bien formée d'intervalles est une liste d'intervalles qui respecte les contraintes suivantes :
- les intervalles sont bien formés,
- les intervalles sont disjoints deux à deux,
- la liste est triée selon la relation d'ordre suivante : un intervalle i_1 est strictement plus petit qu'un intervalle i_2 si et seulement si le maximum de i_1 est strictement plus petit que le minimum de i_2.

Question III.9 *Montrer qu'une liste triée d'intervalles bien formés, est une liste bien formée d'intervalles.*

Question III.10 *Exprimer la définition III.4 sous la forme d'une propriété* $LBFI(e)$ *qui indique que* $e = \langle v_1, \ldots, v_n \rangle$ *est une liste bien formée d'intervalles (contenant les intervalles* v_i*).*

2.2.2 Représentation en PASCAL

Une liste triée d'intervalles est représentée par le type de base `LISTE` correspondant à une liste d'`INTERVALLE`.
Nous supposons prédéfinies les constantes et les fonctions suivantes dont le calcul se termine quelles que soient les valeurs de leurs paramètres. Elles pourront éventuellement être utilisées dans les réponses aux questions :
- `NIL` représente la liste vide d'intervalles ;
- `FUNCTION L_Inserer(i:INTERVALLE;l:LISTE):LISTE;` renvoie une liste d'intervalles composée d'un premier intervalle `i` et du reste de la liste contenu dans `l` ;
- `FUNCTION L_Premier(l:LISTE):INTERVALLE;` renvoie le premier intervalle de la liste `l`. Cette liste ne doit pas être vide ;
- `FUNCTION L_Reste(l:LISTE):LISTE;` renvoie le reste de la liste `l` privée de son premier intervalle. Cette liste ne doit pas être vide ;
- `FUNCTION L_Juxtaposer(l1,l2:LISTE):LISTE;` renvoie une liste composée des intervalles de la liste `l1` dans le même ordre que dans `l1` suivis des intervalles de la liste `l2` dans le même ordre que dans `l2`.

Nous allons maintenant définir plusieurs opérations sur les listes bien formées d'intervalles et estimer leur complexité.

2.2.3 Opérations sur la structure de liste bien formée d'intervalles

La première opération est l'ajout d'un intervalle dans une liste bien formée d'intervalles.

Question III.11 *Écrire en PASCAL une fonction* `ajouter(i:INTERVALLE;l:LISTE):LISTE;` *telle que l'appel* `ajouter(i,l)` *sur une liste bien formée d'intervalles* `l` *renvoie une liste bien formée d'intervalles contenant les intervalles contenus dans la liste* `l` *qui sont disjoints de* `i`*, et :*
- *soit le résultat de la fusion de* `i` *et des intervalles contenus dans la liste* `l` *qui ne sont pas disjoints de* `i`*,*
- *soit l'intervalle* `i`*, si tous les intervalles contenus dans la liste* `l` *lui sont disjoints.*

L'algorithme utilisé ne devra parcourir qu'une seule fois la liste. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.

Question III.12 *Donner des exemples de valeurs des paramètres* `i` *et* `l` *de la fonction* `ajouter` *qui correspondent aux meilleur et pire cas en nombre d'appels récursifs effectués.*
Calculer une estimation de la complexité dans les meilleur et pire cas de la fonction `ajouter` *en fonction de la longueur de la liste* `l`*. Cette estimation ne prendra en compte que le nombre d'appels récursifs effectués.*

La deuxième opération est le test d'appartenance d'une valeur dans une liste bien formée d'intervalles.

Question III.13 *Écrire en PASCAL une fonction* `appartenir(v:INTEGER;l:LISTE):BOOLEAN;` *telle que l'appel* `appartenir(v,l)` *sur une liste bien formée d'intervalles* `l` *renvoie la valeur* `TRUE` *si la valeur* `v` *appartient à un des intervalles contenus dans la liste* `l` *et la valeur* `FALSE` *sinon.*
L'algorithme utilisé ne devra parcourir qu'une seule fois la liste. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.

La troisième opération est la vérification qu'une liste d'intervalles est bien formée, c'est-à-dire respecte les contraintes de la définition III.4 formalisée par la propriété $LBFI(\mathrm{e})$ dans la question III.10.

Question III.14 *Écrire en PASCAL une fonction* `verifier(l:LISTE):BOOLEAN;` *telle que l'appel* `verifier(l)` *sur une liste d'intervalles* `l` *renvoie la valeur* `TRUE` *si la liste* `l` *respecte les contraintes d'une liste bien formée, et la valeur* `FALSE` *sinon.*
L'algorithme utilisé ne devra parcourir qu'une seule fois la liste. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.

2.3 Réalisation à base d'arbres binaires

L'utilisation de la structure d'arbre binaire pour représenter un ensemble à base d'intervalles permet de réduire la complexité en temps de calcul pour les différentes opérations.
Une liste est représentée par un arbre binaire dont les feuilles sont les intervalles.

2.4 Arbre binaire d'intervalles

2.4.1 Définition

Déf. III.5 (Arbre binaire d'intervalles) Un arbre binaire d'intervalles a est une structure qui peut :
- soit être vide notée $\emptyset$,
- soit être une feuille qui contient un intervalle notée $[min, max]$,

– soit un nœud qui contient un sous-arbre gauche (noté $\mathcal{G}(a)$) et un sous-arbre droit (noté $\mathcal{D}(a)$) qui sont tous deux des arbres binaires d'intervalles non vides.

L'ensemble des feuilles, respectivement des nœuds, est noté $\mathcal{F}$, respectivement $\mathcal{N}$. L'ensemble des arbres est $\mathcal{A} = \mathcal{F} \cup \mathcal{N}$.
L'ensemble des feuilles, respectivement des nœuds, d'un arbre a est noté $\mathcal{F}(a)$, respectivement $\mathcal{N}(a)$.

Déf. III.6 (Intervalle englobant) L'intervalle englobant d'un arbre d'intervalles est le plus petit intervalle bien formé qui contient tous les intervalles contenus dans les feuilles de l'arbre. L'intervalle englobant d'un arbre d'intervalles a est noté $\mathcal{I}(a)$.

Déf. III.7 (Parcours gauche-droite d'un arbre binaire) Le parcours gauche-droite d'un arbre binaire a, noté $\mathcal{E}(a)$, construit la séquence des feuilles de l'arbre dans l'ordre de gauche à droite. Il est défini par les équations :

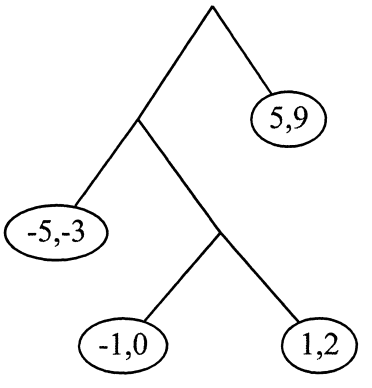
$$\mathcal{E}(a) = \begin{cases} \langle [min_a, max_a] \rangle & \text{si } a \in \mathcal{F} \\ \langle f_1, \ldots, f_{m+n} \rangle & \text{si } a \in \mathcal{N} \\ & \text{et } \langle f_1, \ldots, f_m \rangle = \mathcal{E}(\mathcal{G}(a)) \\ & \text{et } \langle f_{m+1}, \ldots, f_{m+n} \rangle = \mathcal{E}(\mathcal{D}(a)) \end{cases}$$

Déf. III.8 (Arbre bien formé d'intervalles) Un arbre bien formé d'intervalles est un arbre d'intervalles qui respecte les contraintes suivantes :
– les intervalles contenus dans les feuilles sont bien formés,
– les intervalles contenus dans les feuilles sont disjoints deux à deux,
– la liste d'intervalles obtenue par le parcours gauche-droite de l'arbre est bien formée.

Question III.15 *Soit a un arbre d'intervalles bien formés. Montrer que si pour tous les nœuds de l'arbre, le maximum de l'intervalle englobant du fils gauche est strictement plus petit que le minimum de l'intervalle englobant du fils droit, alors l'arbre est bien formé.*

Question III.16 *Exprimer la définition III.8 sous la forme d'une propriété $ABFI(a)$ qui indique que a est un arbre bien formé d'intervalles. Donner deux formes à cette définition, une qui s'appuie sur le parcours gauche-droite, et une qui s'appuie uniquement sur la structure de l'arbre.*

Exemple III.1 (Arbres binaires d'intervalles) Voici un exemple d'arbres binaires d'intervalles :



2.4.2 Représentation des arbres binaires d'intervalles en PASCAL

Un arbre binaire d'intervalles est représenté par le type de base `ARBRE`.
Nous supposons prédéfinies les constantes et les fonctions suivantes dont le calcul se termine quelles que soient les valeurs de leurs paramètres. Elles pourront éventuellement être utilisées dans les réponses aux questions :
– `NIL` représente l'arbre binaire d'intervalles vide ;

- `FUNCTION Feuille(i:INTERVALLE):ARBRE;` est une fonction qui renvoie une feuille contenant l'intervalle `i` ;
- `FUNCTION EstFeuille(a:ARBRE):BOOLEAN;` est une fonction qui renvoie la valeur `TRUE` si l'arbre `a` est une feuille contenant un intervalle, et la valeur `FALSE` sinon ;
- `FUNCTION Intervalle(a:ARBRE):INTEGER;` est une fonction qui renvoie l'intervalle contenu par la racine de l'arbre `a` si `a` est une feuille ;
- `FUNCTION Noeud(fg:ARBRE;fd:ARBRE):ARBRE;` est une fonction qui renvoie un nœud dont le fils gauche et le fils droit de la racine sont respectivement `fg` et `fd` ;
- `FUNCTION EstNoeud(a:ARBRE):BOOLEAN;` est une fonction qui renvoie la valeur `TRUE` si l'arbre `a` est un nœud, et la valeur `FALSE` sinon ;
- `FUNCTION Gauche(a:ARBRE):ARBRE;` est une fonction qui renvoie le fils gauche de la racine de l'arbre `a` si `a` est un nœud ;
- `FUNCTION Droit(a:ARBRE):ARBRE;` est une fonction qui renvoie le fils droit de la racine de l'arbre `a` si `a` est un nœud.

Exemple III.2 L'appel suivant

```
Noeud(
   Noeud(
      Feuille( I_Creer( -5, -3 )),
      Noeud(
         Feuille( I_Creer( -1, 0)),
         Feuille( I_Creer( 1, 2))
      )
   ),
   Feuille( I_Creer( 5, 9))
)
```

renvoie l'arbre binaire de blocs représenté graphiquement par le troisième arbre dans l'exemple III.1.

2.4.3 Opérations sur la structure d'arbre d'intervalles

La première opération consiste à déterminer l'intervalle englobant d'un arbre d'intervalles.

Question III.17 *Écrire en PASCAL une fonction `englobant(a:ARBRE):INTERVALLE;` telle que l'appel `englobant(a)` renvoie l'intervalle englobant de l'arbre d'intervalles bien formé non vide `a`. L'algorithme utilisé ne devra parcourir qu'une seule fois l'arbre. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

La deuxième opération est la vérification qu'un arbre d'intervalles est bien formé, c'est-à-dire qu'il respecte les contraintes de la définition III.8.

Question III.18 *Écrire en PASCAL une fonction `verifier(a:ARBRE):BOOLEAN;` telle que l'appel `verifier(a)` sur un arbre d'intervalles `a` renvoie la valeur `TRUE` si `a` est un arbre d'intervalles bien formé, et la valeur `FALSE` sinon. L'algorithme utilisé ne devra parcourir qu'une seule fois l'arbre. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

La troisième opération est le test d'appartenance d'une valeur dans un arbre d'intervalles bien formé.

Question III.19 *Écrire en PASCAL une fonction `appartenir(v:INTEGER;a:ARBRE):BOOLEAN;` telle que l'appel `appartenir(v,a)` sur un arbre d'intervalles bien formé `a` renvoie la valeur `TRUE` si la valeur `v` appartient à un des intervalles contenus dans les feuilles de `a` et la valeur `FALSE` sinon.*

L'algorithme utilisé ne devra parcourir qu'une seule fois l'arbre. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.

La quatrième opération est le découpage d'un arbre bien formé d'intervalles selon une valeur.
Un triplet d'arbres d'intervalles est représenté par le type de base `TRIPLET`.
Nous supposons prédéfinies les constantes et les fonctions suivantes dont le calcul se termine quelles que soient les valeurs de leurs paramètres. Elles pourront éventuellement être utilisées dans les réponses aux questions :

- `FUNCTION T_Creer(a1,a2,a3:ARBRE):TRIPLET;` renvoie un triplet dont les éléments sont `a1`, `a2` et `a3`,
- `FUNCTION T_Un(t:TRIPLET):ARBRE;` renvoie le premier élément de `t`,
- `FUNCTION T_Deux(t:TRIPLET):ARBRE;` renvoie le deuxième élément de `t`,
- `FUNCTION T_Trois(t:TRIPLET):ARBRE;` renvoie le troisième élément de `t`.

Question III.20 *Écrire en PASCAL une fonction* `decouper(v:INTEGER;a:ARBRE):TRIPLET;` *telle que l'appel* `decouper(v,a)` *sur un arbre d'intervalles bien formé* `a` *renvoie un triplet composé :*

- *d'un arbre bien formé contenant les intervalles strictement inférieur à la valeur* `v`,
- *d'un arbre bien formé contenant les intervalles contenant la valeur* `v`,
- *d'un arbre bien formé contenant les intervalles strictement supérieur à la valeur* `v`.

L'algorithme utilisé ne devra parcourir qu'une seule fois l'arbre. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.

La dernière opération est l'ajout d'un intervalle dans un arbre d'intervalles bien formé.

Question III.21 *Écrire en PASCAL une fonction* `ajouter(i:INTERVALLE;a:ARBRE):ARBRE;` *telle que l'appel* `ajouter(i,a)` *sur un arbre d'intervalles bien formé* `a` *renvoie un arbre d'intervalles bien formé contenant les intervalles contenus dans les feuilles de* `a` *qui sont disjoints de* `i`, *et:*

- *soit le résultat de la fusion de* `i` *et des intervalles contenus dans les feuilles de* `a` *qui ne sont pas disjoints de* `i`,
- *soit l'intervalle* `i`, *si et seulement si tous les intervalles contenus dans les feuilles de* `a` *lui sont disjoints.*

L'algorithme utilisé ne devra parcourir qu'une seule fois l'arbre. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.

Question III.22 *Donner des exemples de valeurs des paramètres* `i` *et* `a` *de la fonction* `ajouter` *qui correspondent aux meilleur et pire cas en nombre d'appels récursifs effectués.*
Calculer une estimation de la complexité dans les meilleur et pire cas de la fonction `ajouter` *en fonction de la profondeur de l'arbre* `a`. *Cette estimation ne prendra en compte que le nombre d'appels récursifs effectués.*

Fin de l'énoncé