Graphes et outils logiques
`http://www.lri.fr/~blsk/GOL`

TD8 - Termes

Exercice 1 *Sandwichs.*
On modélise des sandwichs comme des termes. Un sandwich peut être juste composé d'une tranche de pain. On peut aussi ajouter à un sandwich l'un des ingrédients suivants : beurre, fromage, jambon. Finalement on peut aussi combiner deux sandwichs pour faire un nouveau sandwich double.

1. Proposer une signature pour représenter les sandwichs : on introduira des symboles pour représenter un sandwich composé uniquement de pain, pour l'opération correspondant à l'ajout d'un ingrédient et finalement la formation d'un sandwich double. On indiquera l'arité de chaque symbole.
2. En utilisant les symboles introduits précédemment, construire le terme qui correspond à un sandwich double composé d'un sandwich beurre-jambon et un sandwich fromage.
3. Construire une fonction récursive `prix` (on donnera les équations récursives) qui calcule le prix d'un sandwich, sachant que la base de pain est facturée 1€, le beurre 0.2€, le jambon et le fromage 1.5€. Quel est le prix du sandwich de la question précédente ?
4. Construire une fonction récursive `nbpain` qui calcule le nombre de tranches de pain sur un sandwich.
5. Énoncer le principe d'induction sur les sandwichs.
6. Utiliser ce principe pour montrer que le prix d'un sandwich est toujours supérieur au nombre de tranches de pain dans le sandwich.

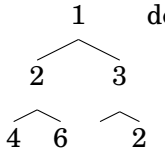
Correction :

1. *On considère l'ensemble de sortes* $\{sdw, ing\}$ *où* sdw *est la sorte correspondant à un sandwich et* ing *est un ingrédient. Un ingrédient est représenté par le symbole* $beurre$, $jambon$ *ou* $fromage$, *chacun d'arité* ing.
   Un sandwich réduit à une tranche de pain sera représenté par le symbole $pain$ *d'arité* sdw. *Le symbole* add *d'arité* $sdw \times ing \to sdw$ *dénotera l'ajout d'un ingrédient à un sandwich, tandis que le symbole* $comb$ *d'arité* $sdw \times sdw \to sdw$ *dénotera l'opération de combinaison de deux sandwichs.*
   Remarque : la solution présentée ici utilise deux sortes, l'une pour les ingédients, l'autre pour les sandwichs. On peut également utiliser une seule sorte avec des symboles $fromage$, $jambon$ *et* $beurre$ *d'arité* 1 *au lieu du symbole* add.
2. $comb(add(add(pain, beurre), jambon), add(pain, fromage))$
3. *On définit la fonction* $prix : sdw \to \mathbb{R}$ *récursivement de la façon suivante.*
   $prix(pain) = 1$
   $prix(add(s, beurre)) = 0,2 + prix(s)$
   $prix(add(s, jambon)) = 1,5 + prix(s)$
   $prix(add(s, fromage)) = 1,5 + prix(s)$
   $prix(comb(s,t)) = prix(s) + prix(t)$
   Le prix du sandwich précédent est égal à $(1 + 0,2 + 1,5) + (1 + 1,5) = 5,2$€.

4. *On définit la fonction* $nbpain : sdw \to \mathbb{N}$ *récursivement de la façon suivante.*
   $nbpain(pain) = 1$
   $nbpain(add(s,i)) = nbpain(s)$
   $nbpain(comb(s,t)) = nbpain(s) + nbpain(t)$
5. *Le principe d'induction sur les sandwichs est le suivant. Pour toute propriété P,*
   Si
   (a) $P(pain)$
   (b) $\forall s,i, P(s) \Rightarrow P(add(s,i))$
   (c) $\forall s,t, P(s) \Rightarrow P(t) \Rightarrow P(comb(s,t))$
   alors $\forall s, P(s)$
6. *Pour un sandwich s, on pose* $P(s)$ *la propriété* $prix(s) \geq nbpain(s)$*. On montre cette propriété par induction sur les sandwichs.*
   — *Pour* $s = pain$*, on a* $prix(pain) = 1$ *et* $nbpain(pain) = 1$ *donc la propriété est vérifiée.*
   — *Soit s un sandwich et i un ingrédient. On suppose* $P(s)$*, c'est-à-dire que* $prix(s) \geq nbpain(s)$*. Montrons* $P(add(s,i))$*. On raisonne par cas sur l'ingrédient i.*
   On a $prix(add(s,beurre)) = 0,2 + prix(s)$ *et* $nbpain(add(s,beurre)) = nbpain(s)$*, or* $prix(s) \geq nbpain(s)$*par hypothèse d'induction, donc* $prix(add(s,beurre)) \geq nbpain(add(s,beurre))$*.*
   On a $prix(add(s,jambon)) = 1,5 + prix(s)$ *et* $nbpain(add(s,jambon)) = nbpain(s)$ *donc par hypothèse d'induction* $prix(add(s,jambon)) \geq nbpain(add(s,jambon))$*. (Raisonnement analogue pour* $i = fromage$*.)*
   — *Soient s et t deux sandwichs. On suppose* $P(s)$ *et* $P(t)$*. Montrons* $P(comb(s,t))$*.*
   On a $prix(comb(s,t)) = prix(s) + prix(t)$ *et* $nbpain(comb(s,t)) = nbpain(s) + nbpain(t)$*, donc par hypothèse d'induction,* $prix(comb(s,t)) \geq nbpain(comb(s,t))$*.*

Exercice 2 *Arbres binaires de recherche.*

On rappelle que la signature des arbres binaires contenant des entiers est formée d'une constante `leaf` et d'un symbole ternaire `node` d'arité $\texttt{tree} \times \mathbb{N} \times \texttt{tree} \to \texttt{tree}$.

1. Donner les équations pour définir une fonction récursive `infix` qui étant donné un arbre binaire t, renvoie le mot composé des entiers contenus dans l'arbre parcouru de manière infixe : on parcourt d'abord le sous-arbre gauche, puis on ajoute l'entier du nœud puis le mot correspondant au parcours du sous-arbre droit.
   Par exemple, l'arbre  donnera le mot : 426132.

2. Donner des règles d'inférence pour définir la relation $\texttt{In}(n,t)$ qui représente le fait que l'entier n apparaît dans un nœud de l'arbre t.

On s'intéresse aux *arbres binaires de recherche* qui vérifient que dans chaque sous-arbre de la forme $\texttt{node}(l,n,r)$, tous les entiers stockés dans le sous-arbre l sont inférieurs ou égaux à n qui est lui-même inférieur ou égal à tous les entiers stockés dans le sous-arbre r.

3. On note $\texttt{elt}(n)$ l'arbre $\texttt{node}(\texttt{leaf}, n, \texttt{leaf})$ qui contient un seul entier n. Est-ce un arbre binaire de recherche ? Les termes suivants sont-ils des arbres binaires de recherche ?

$$\texttt{node}(\texttt{elt}(1), 2, \texttt{elt}(3)) \qquad \texttt{node}(\texttt{leaf}, 2, \texttt{node}(\texttt{elt}(1), 3, \texttt{leaf}))$$

4. Écrire deux termes différents qui correspondent à des arbres binaires de recherche contenant les valeurs 1, 2, 3 et 4. Donner la valeur de la fonction `infix` pour ces arbres.
5. Donner des règles d'inférence pour définir la relation $\texttt{Abr}(t)$ qui représente le fait que l'arbre t est un arbre binaire de recherche.
6. Démontrer que si l'arbre t est un arbre binaire de recherche alors le mot $\texttt{infix}(t)$ est trié, c'est-à-dire que les entiers apparaissent en ordre croissant.

La structure d'arbre binaire de recherche permet de chercher efficacement un élément.

7. Lorsque l'on recherche un élément n dans un arbre binaire de recherche de la forme $\texttt{node}(g,x,d)$, est-il utile de fouiller l'ensemble de l'arbre ? dans quel cas faut-il chercher dans g et dans quel cas dans d ?
8. Proposer des équations récursives pour une fonction `inabr` telle que $\texttt{inabr}(n,t)$ teste si l'entier n est présent dans l'arbre t, en supposant que t est un arbre binaire de recherche.
9. Montrer par récurrence sur t que

$$\forall nt,\ \texttt{inabr}(n,t) = \texttt{true} \implies \texttt{In}(n,t)$$

10. Montrer par récurrence sur la définition de la relation `Abr` que

$$\forall nt,\ \texttt{Abr}(t) \implies (\texttt{In}(n,t) \implies \texttt{inabr}(n,t) = \texttt{true})$$

On peut également définir une procédure efficace pour vérifier qu'un arbre binaire est bien un arbre binaire de recherche.

11. Donner des équations récursives pour définir une fonction `verif` telle que $\texttt{verif}(min,max,t)$ renvoie vrai si et seulement si t est un arbre binaire de recherche tel que tous les entiers x dans t vérifient $min \leq x \leq max$.
12. Comment faut-il choisir min et max pour que $\texttt{verif}(min,max,t)$ teste si t est un arbre binaire de recherche ?
13. Démontrer par récurrence sur t que la fonction `verif` est correcte, c'est-à-dire que

$$\forall m_1 m_2 t,\ \texttt{verif}(m_1,m_2,t) = \texttt{true} \implies (\texttt{Abr}(t) \wedge (\forall x,\ \texttt{In}(x,t) \implies m_1 \leq x \leq m_2))$$

Correction :

1. *Un cas pour chaque forme de terme.*

$$infix(leaf) = \epsilon \qquad infix(node(l,e,r)) = infix(l)e\,infix(r)$$

2. *Un cas pour chaque endroit où peut se trouver n.*

$$\frac{}{In(n,node(l,n,r))} \qquad \frac{In(n,l)}{In(n,node(l,e,r))} \qquad \frac{In(n,r)}{In(n,node(l,e,r))}$$

3. *Le singleton* $node(leaf,n,leaf)$ *est un arbre binaire de recherche ainsi que le premier arbre* $node(elt(1),2,elt(3))$. *Le dernier arbre* $node(leaf,2,node(elt(1),3,leaf))$ *n'est pas un arbre binaire de recherche car le sous-arbre à droite de* 2 *contient* 1.
4. *Exemples :* $node(node(elt(1),2,elt(3)),4,leaf)$ *et* $node(elt(1),2,node(elt(3),4,leaf))$. *Dans tous les cas la fonction* $infix$ *donne* 1234.

5. *On commence par définir deux relations $Inf(n,t)$ et $Sup(n,t)$ qui sont vraies lorsque n est plus petit (Inf) ou plus grand (Sup) que tous les éléments dans l'arbre t. Cela peut se faire avec des règles d'inférence ou encore sous les formes $Inf(n,t) \equiv \forall x, In(x,t) \Rightarrow n \leq x$ et $Sup(n,t) \equiv \forall x, In(x,t) \Rightarrow x \leq n$.*

$$\frac{}{Abr(leaf)} \qquad \frac{Abr(l) \quad Abr(r) \quad Inf(e,r) \quad Sup(e,l)}{Abr(node(l,e,r))}$$

6. *On montre que $Abr(t)$ implique que $infix(t)$ est la suite triée des éléments de t par induction sur la définition de la relation $Abr(t)$. Mettre l'accent sur le fait qu'on a besoin de toutes les conditions, en particulier on a les mêmes éléments dans t et $infix(t)$ et c'est ce qui permet d'affirmer que $Inf(n,t)$ implique que n est inférieur à tous les éléments de $infix(t)$.*
   Induction sur $Abr(t)$ pour démontrer la propriété $P(t) \equiv infix(t)$ est la suité triée des éléments de t.
   — *Cas $Abr(leaf)$ avec $t = leaf$: dans ce cas $infix(t) = infix(leaf) = \epsilon$, qui est la suite triée vide, et justement t ne contient pas d'éléments.*
   — *Cas $Abr(node(l,e,r))$ avec $t = node(l,e,r)$ et les hypothèses $Abr(l)$, $Abr(r)$, $Inf(e,r)$ et $Sup(e,l)$. Par hypothèse de récurrence on a $P(l)$ et $P(r)$, à savoir $infix(l)$ et $infix(r)$ sont respectivement les listes triées d'éléments de l et de r. Dans ce cas $infix(t) = infix(l) \cdot e \cdot infix(r)$ et on vérifie que :*
      — *$infix(t)$ contient exactement les éléments de t : vrai car $infix(l)$ et $infix(r)$ contiennent les éléments de l et r.*
      — *$infix(t)$ est triée : vrai car deux éléments consécutifs de la liste sont toujours en ordre croissant. En effet, si on prend deux éléments dans l, par hypothèse de récurrence sur $Abr(l)$ ils sont triés ; si on prend deux éléments dans r, par hypothèse de récurrence sur $Abr(r)$ ils sont triés ; si on prend le dernier élément de l et e, par hypothèse $Sup(e,l)$ ils sont triés, et si on prend e et le premier élément de r, par hypothèse $Inf(e,r)$ ils sont encore triés.*
7. *On cherche à gauche (resp. à droite) lorsque $n < x$ (resp. $x < n$). Pas besoin de chercher à gauche ni à droit si $n = x$.*
8. *On sépare le cas $node$ en trois pour tenir compte des critères de la question précédente.*

$$\begin{aligned}
inabr(n, leaf) &= false \\
inabr(n, node(g,n,d)) &= true \\
inabr(n, node(g,x,d)) &= inabr(n,g) \qquad \text{si } n < x \\
inabr(n, node(g,x,d)) &= inabr(n,d) \qquad \text{si } x < n
\end{aligned}$$

9. *Dans la récurrence sur t, on ajoute un raisonnement par cas pour s'intéresser aux manières dont $inabr(n,t)$ peut valoir $true$ (inversion).*
10. *Dans la récurrence sur Abr, on ajoute un raisonnement par cas sur les manières dont $In(n,t)$ peut être vérifiée (inversion) et on élimine les cas incompatibles avec les hypothèses de Abr.*
11. *Lorsque l'on vérifie les deux sous-arbres d'un nœud, on met à jour les bornes en se servant de la valeur de ce nœud.*

$$\begin{aligned}
verif(min, max, leaf) &= true \\
verif(min, max, node(g,x,d) &= min \leq x \leq max \wedge verif(min,x,g) \wedge verif(x,max,d)
\end{aligned}$$

12. *Les bornes min et max doivent être respectivement un minorant et un majorant de l'ensemble des éléments. Cela fonctionne à coup sûr si on s'autorise $-\infty$ et $+\infty$. De manière plus subtile on peut choisir les éléments à l'extrême gauche et à l'extrême droite de l'arbre.*
13. *Récurrence sur t, avec inversion de la condition $verif(m_1, m_2, t) = true$.*