4 Equations, fonctions et spécification

Nous allons illustrer dans ce paragraphe comment le formalisme pourtant encore abstrait que nous avons introduit jusqu'à présent permet néanmoins de pouvoir déjà parler de *preuves de programmes*

Les booléens et l'alternative Enrichissons au préalable un peu notre langage par le symbole de type B (pour les boléens) les constantes $true$ et $false$, la fonction unaire not, les fonctions binaires *and* et *or* et la fonction trenaire d'alternative if_then_else dont nous utiliserons la notation *mixfixée.* Soit D un symbole de type, on se donne les axiomes suivants que nous appelerons respectivement théories B et IF :

Les booléens	
BT1 :	$\mathrm{B}true$
BT2 :	$\mathrm{B}false$
BT3 :	$\forall b.\mathrm{B}b \Rightarrow \mathrm{B}\,not(b)$
BT4 :	$\forall b_1.\mathrm{B}b_1 \Rightarrow \forall b_2.\mathrm{B}b_2 \Rightarrow \mathrm{B}\,et(b_1, b_2)$
BT5 :	$\forall b_1.\mathrm{B}b_1 \Rightarrow \forall b_2.\mathrm{B}b_2 \Rightarrow \mathrm{B}\,ou(b_1, b_2)$
B1 :	$not(true) = false$
B2 :	$not(false) = true$
B3 :	$\forall b.\mathrm{B}b \Rightarrow and(true, b) = b$
B4 :	$\forall b.\mathrm{B}b \Rightarrow and(false, b) = false$
B5 :	$\forall b.\mathrm{B}b \Rightarrow or(true, b) = true$
B4 :	$\forall b.\mathrm{B}b \Rightarrow or(false, b) = b$

L'alternative (paramétrique)	
IFT1 :	$\forall b.\mathrm{B}b \Rightarrow \forall e_1.Da \Rightarrow \forall e_2.De_2 \Rightarrow D(if\ b\ then\ e_1\ else\ e_2)$
IF1 :	$\forall e_1.Da \Rightarrow \forall e_2.De_2 \Rightarrow (if\ true\ then\ e_1\ else\ e_2) = e_1$
IF1 :	$\forall e_1.Da \Rightarrow \forall e_2.De_2 \Rightarrow (if\ false\ then\ e_1\ else\ e_2) = e_2$

Le minimun d'une liste d'entiers Donnons nous la théorie des entiers et des listes d'entiers que l'on obtient en particularisant D à N dans le schéma de théorie L_D et en lui adjoignant les théories P, B et IF.

Nous utiliserons l'abréviation $A \Leftrightarrow B$ pour $(A \Rightarrow B) \wedge (B \Rightarrow A)$.
Pour tout symbole de type D, nous nous autoriserons désormais l'abréviation $\forall x : D.A$ pour $\forall x.Dx \Rightarrow A$.

Nous appellerons *axiome équationnel* ou *équation conditionnelle* toute formule close de laforme

$$\forall x_1.D_1x_1 \Rightarrow ..\forall x_n.D_nx_n \Rightarrow A \Rightarrow t = u$$

où D_1, ..,D_n sont des symboles de types et A une formule sans quatificateurs.

Nous voulons définir, spécifier et prouver la correction d'une fonction de recherche du minimum d'une liste d'entiers. La technique employée consiste à

1. introduire un symbole pour la fonction visée,
2. définir le *type* de la fonction (ce qui donne ses conditions de terminaison)
3. définir ses valeurs par des axiomes équationnels,
4. définir et prouver les autres propriétés attendues de la fonction.

Posons $n > m ::= \neg(n \leq m)$. Soit le symbole de fonction $lmin$ et les axiomes équationnels suivants[12]

lminT :	$\forall x : \mathrm{LN}.\mathrm{N}\, lmin(x)$
lmin1 :	$lmin([]) = 0$
lmin2 :	$\forall n : \mathrm{N}n.\forall x : \mathrm{L}_{\mathbb{N}}.n \leq lmin(x) \Rightarrow lmin(n :: x) = n$
lmin3 :	$\forall n : \mathrm{N}n.\forall x : \mathrm{L}_{\mathbb{N}}.n > lmin(x) \Rightarrow lmin(n :: x) = lmin(x)$

Nous pouvons d'ores et déjà nous assurer que les axiomes équationnels lmin1, lmin2, lmin3 permettent effectivement d'établir l'axiome de typage lminT[13].

Montrons $\forall x : \mathrm{L}_{\mathbb{N}}.\mathrm{N}\, lmin(x)$ par récurence structurelle sur x
Si $x = []$, il faut montrer $\mathrm{N}\, lmin([])$.
En utilisant l'équation lmin1, cela revient à montrer N0.
Or c'est un des axiomes de P.
Si $x = n :: y$ avec $\mathrm{N}n$, $\mathrm{L}_{\mathbb{N}}y$ et l'hypothèse de récurence $\mathrm{N}\, lmin(y)$.
Il faut montrer : $\mathrm{N}\, lmin(a :: y)$.
On raisonne par cas selon que $n \leq lmin(y)$ ou $n > lmin(y)$:
Si $n \leq lmin(y)$, on peut utiliser lmin2.
Il faut alors montrer $\mathrm{N}n$, ce qui est l'une de nos hypothèses.
Si $n > lmin(y)$, on peut utiliser lmin3.
Il faut alors montrer $\mathrm{N}\, lmin(y)$, ce qui est notre hypothèse de récurrence.

EXERCICE (12) : Formaliser la démonstration ci-dessus (en déduction naturelle, par exemple)

En montrant que les équations décrivant $lmin$ satisfont son axiome de typage, nous avons prouvé une spécification minimale de notre fonction. Ça n'est pas suffisant pour garantir que $lmin$ calcule bien la valeur attendue d'une fonction de recherche du minimum d'une liste : l'équation $\forall x : \mathrm{L}_{\mathbb{N}}.lmin(x) = 2384$ permet aussi de démontrer lminT, mais personne ne nous achètera une telle définition pour celle de la fonction attendue.

Il nous faut donc préciser et formaliser ce que nous attendons, pour pouvoir vérifier ensuite que lmin1, lmin2 et lmin3 décrivent la bonne fonction.
De la spécification en langue naturelle ≪le minimum d'une liste≫ duex mots sont à retenir.

1. le premier est évident, c'est ≪minimum≫L'entier cherché devra donc inférieur (ou égal) à tout élément de la liste.
2. le second se cache sous l'abréviation ≪d'≫L'entier recherché doit lui-même appartenir à la liste.

Avant de formuler nos deux *requisit*, nous devons définir la nouvelle notion introduite par notre analyse : ≪être élément de≫ ou ≪appartenir à≫ une liste.

Intuitivement, un élément a appartient à une liste x si et seulement si, ou bien la liste commence par a, ou bien a appartient au reste de la liste. De plus, rien n'appartient à la liste vide. Posons donc

THÉORIE AXIOMATIQUE DES D-LISTES (suite)	
L_D6 :	$\forall a : D.\neg(a \in [])$
L_D7 :	$\forall a : D.\forall x : \mathrm{L}_D.a \in (a :: x)$
L_D8 :	$\forall a : D.\forall x : \mathrm{L}_D.a \in x \Rightarrow a \in (a :: x)$

Puisque nous avons défini $lmin$ comme une fonction, dont nous avons montré qu'elle est partout définie sur les listes, nous pouvons désigné ≪l'élément recherché≫ par le terme $lmin(x)$, où x est la liste à traiter. Les propriétés à satisfaire s'écriront donc :

1. $\forall x : \mathrm{L}_{\mathbb{N}}.\forall n : \mathrm{N}.n \in x \Rightarrow lmin(x) \leq n$

[12]Nous nommerons désormais les axiomes équationnels avec le nom des fonctions concernées

[13]Les puristes objecterons que puisque lminT est démontrable à partir de équation, sa présence est superflue. Certes, mais nous ne cherchons pas à obtenir une axiomatisation minimale de nos programmes; nous cherchons à formuler et à obtenir par preuve un maximum de garantie de bon fonctionnement de ceux-ci.

2. $\forall x : \mathrm{L}_{\mathbf{N}}.lmin(x) \in x$

La seconde propriété dégagée pose la question (décidemment récurente) du cas de la liste vide : il n'y a pas de minimum dans la liste vide.

Nous avons commis ici l'erreur commune du programmeur : mettre la charrue avant les bœufs; décrire *comment* calculer avant d'avoir précisé *quoi* calculer. Jetons donc (pas trop loin tout de même) notre fonction *lmin* et (re)commençons par la spécification.
Soient n un entier et x une liste d'entiers, on définit la formule $LMIN[n, x]$ où n et x sont deux variables libres :

$$LMIN(n, x) \;::= (n \in x) \wedge (\forall m : \mathrm{N}.m \in x \Rightarrow n \leq n)$$

Ceci posé, n'allons pas encore rechercher notre définiton de *lmin*, mais posont nous au préalable la question de savoir si notre spécification est réalisable ? A savoir, la formule

$$\forall x : \mathrm{L}_{\mathbf{N}}.\exists n : \mathrm{N}.LMIN(n, x)$$

est-elle valide ? Nous avons vu que non et ce, indépendament de quelque fonction que ce soit : il faut supposer que x n'est pas vide. Nous amendons dons notre spécification en ce sens et cherchons à montrer :

$$\forall x : \mathrm{L}_{\mathbf{N}}.x \neq [] \Rightarrow \exists n : \mathrm{N}.LMIN(n, x)$$

Par récurence sur x, montrons $x \neq [] \Rightarrow \exists n : \mathrm{N}.LMIN(n, x)$
  Si $x = []$ on a trivialement le résultat
  Si $x = p :: y$ avec $\mathrm{N}p$, $\mathrm{L}_{\mathbf{N}}y$ et l'hypothèse de récurence $y \neq [] \Rightarrow \exists n : \mathrm{N}.LMIN(n, y)$.
    On remarque que $p :: y \neg []$ est vrai, reste donc à montrer que $\exists n : \mathrm{N}.LMIN(n, p :: y)$
    On raisonne par cas sur y selon qu'il est égal à [] ou non :
      Si $y = []$, on prend p comme *témoin* pour n.
        Par définition de $LMIN$, il faut montrer que
        (a) $p \in (p :: [])$ et que *(b)* $\forall m : \mathrm{L}.m \in (p :: []) \Rightarrow p \leq m$
          (a) On résoud ce cas en utilisant l'axiome $\mathrm{L}_D 7$.
          (b) Pour résoudre ce cas, on remarque que le seul élément de $p :: []$ est p,
            ce qui nous donne bien $p \leq p$.
      Si $y \neq []$, par hypothèse de récurence, on a un entier n_0 tel que $LMIN(n_0, y)$.
        on raisonne alors par cas selon que $p \leq n_0$ ou non :
          Si $p \leq n_0$, on prend p pour témoin :
            Par l'axiome $\mathrm{L}_D 7$, on a $p \in (p :: y)$.
            De l'hypothèse $LMIN(n_0, y)$, on tire, pour tout $m \in y$, que $n_0 \leq m$, d'où $p \leq m$.
            Et comme $p \leq p$, on a bien que pour tout $m \in p :: y$, $p \leq m$
          Si $p > n_0$, alors $n_0 \leq p$ et on prend n_0 comme témoin :
            Par l'axiome $\mathrm{L}_D 8$ et l'hypothèse $LMIN(n_0, p :: y)$, on a $n_0 \in (p :: y)$.
            De l'hypothèse $LMIN(n_0, y)$ et de $n_0 \leq p$, on tire que pour tout $m \in p :: y$, $n_0 \leq m$.

CQFDNotre spécification est donc réalisable. Nous pouvons maintenant ressortir de notre chapeau (où, en fait, nous l'avions cachée) la fonction *lmin*. On peut remarquer que chacun des témoin que nous avons choisi pour n est le résultat de *lmin* :

- on a bien $lmin(p :: []) = p$ (preuve en exercice) ;
- si $p \leq n_0$, avec, par hypothèse de récurence $n_0 = lmin(y)$, on a bien $p = lmin(p :: y)$;
- si $p > n_0$, on a aussi $n_0 = lmin(p :: y)$.

La fonction *lmin* réalise donc bien notre spécification.

La preuve de réalisabilité nous a donné l'assurance que le problème posé par notre spécification est solvable. Elle peut donner plus : suggérer comment obtenir la solution; *ie* : donner la fonction pour calculer la valeur recherchée. Reprenons notre preuve en n'en retenant que la structure pour en *extraire* une fonction $lmin'(x)$:

  Si $x = []$, c'est une erreur
  sinon $x = p :: y$ et
    si $y = []$ alors $lmin'(p :: []) = p$
    sinon

soit $n_0 = lmin'(y)$,
si $p \leq n_0$ alors $lmin'(p :: y) = p$
sinon $lmin'(p :: y) = n_0$

On voit qu'il n'y a plus trés loin de la coupe aux lèvres.

Exercice (13) : Paraphraser la définition de $lmin'$ dans différents langages de programmation

La technique d'extraction d'une fonction à partir d'une preuve que nous avons présenté sur cet exemple est en fait parfaitement connue théoriquement : c'est un résultat trés important de la théorie de la démonstration qui met en rapport le formalisme des preuves (déduction naturelle, calcul des séquent) et une formalisation des *fonctions calculables* appelée λ-calcul.