Chapitre 1

Ensembles

1.1 Spécifications

Les opérations élémentaires nécessaires sont:

- ensemble inclu(x, e)
- ensemble retire(x, e)
- **booleen** exist(x, e)
- **typeT** nimporte(e)
- **booleen** vide(e)
- ensemble ensemblevide()

que nous définirons par les axiomes suivants:

$$
\begin{array}{ll}
(1) & \text{vide(ensemblevide())} = \textbf{vrai} \\
(2) & \text{vide(inclu}(x, e)) = \textbf{faux} \\
(3) & \text{exist}(x,\text{ensemblevide())} = \textbf{faux} \\
(4) & \text{exist}(x,\text{inclu}(y, e)) = \\
 & \qquad\qquad \textbf{si } (x == y) \textbf{ alors vrai} \\
 & \qquad\qquad\qquad\qquad \textbf{sinon } \text{exist}(x, e)) \\
(5) & \text{retire}(x,\text{ensemblevide())} = \text{ensemblevide()}
\end{array}
$$

(6) retire(x,inclu(y, e)) =
si ($x == y$) **alors** retire(x, e)
sinon inclu(y,retire(x, e))

(7) **si non** vide(e) **alors**
exist(nimporte(e),e) = **vrai**

Nous supposerons que cette spécification est complète et consistante et proposons quelques implantations (voir listings)
Nous pouvons maintenant créer les opérations de la couche immédiatement supérieure à savoir:

- ensemble union($e1, e2$)
- ensemble intersection($e1, e2$)
- ensemble complement(e, E)
- **integer** cardinal(e)

et pourquoi pas:

- ensemble_d_ensembles parties(e)

1.2 Applications immédiates

- compter le nombre d'éléments positifs dans un ensemble d'entiers.
- créer l'ensemble des entiers pairs pris dans e.
- créer les ensembles de "clusters" d'un ensemble de points c'est-à-dire qu'il faut créer tous les ensembles e_i tels que tout point de e_i est situé à une distance inférieure à D de tous les autres éléments de e_i et que tout point de e situé à une distance inférieure à D soit dans e_i.

1.3 Dynamique-Statique

La programmation de tout ça est évidemment simple si on ne tient pas compte de la notion "Dynamique-Statique", il suffit en général de transcrire dans le langage souhaité les spécifications "étudiées pour".
Revenons à nos ensembles, on voit qu'il y a un problème pour *exist()* et *retire()*, l'un se servant de l'autre suivant la manière de programmer et, surtout pour *retire()*, on voit qu'il faut introduire une nouvelle fonction qui doit fournir par le même appel n'importe quel élément de e et l'ensemble obtenu par exclusion de cet élément de e.

1.3.1 Programmons,

booleen exist(x, e)
  si vide(e) **alors vrai**
    sinon y=nimporte(e)
      ($y == x$ ou exist(x,retire(y, e)))

...jusque là pas de problème si ce n'est que *retire()* ne doit pas utiliser *exist()*...et encore c'est pas sûr.

ensemble retire(x, e)
  si vide(e) **alors** ensemblevide()
    sinon (y, r) = exclusion(e)
      Si $x == y$ **alors** r
        sinon inclu(y,retire(x, r))

Il a fallu rajouter la fonction *exclusion(e)* qui fournit deux résultats, un élément et un ensemble.

1.3.2 Exercices

Ensembles des parties de e.

Il faut évidemment créer une structure d'ensemble d'ensembles.

Nous supposerons qu'elle existe. Nous utiliserons les mêmes notations pour Exist(e, ee), Retire(e, ee), Inclu(e, ee), Vide(ee) et Nimporte(ee) en mettant la majuscule. Nous noterons Ensemble_d_ensembles_Vide() la fonction qui crée un ensemble vide d'ensembles.

On écrit alors simplement (mais en réfléchissant beaucoup) le programme suivant:

```
Ensemble_d_ensembles Epe(e)

    ensemble e

    Epe(e) = si Vide(e)
                 alors Inclu(ensemblevide()
                             ,Ensemble_d_ensembles_Vide())
                 sinon
                     x = nimporte(e)
                     r = retire(x, e)
                     Union(Epe(r),inclupartout(y,Epe(r))



Ensemble_d_ensembles inclupartout(y, Ee)

    float y,Ensemble_d_ensembles Ee

    si Vide(Ee) alors Ensemble_d_ensembles_Vide()
                sinon
                    e = nimporte(Ee)
                    Inclu(inclu(y, e),inclupartout(y,retire(e, Ee)))
```