CHAPITRE 1

Ensembles, relations, fonctions

Ce chapitre présente les notions élémentaires de la « théorie naïve » des ensembles. Le concept d'ensemble et les opérations classiques sur les ensembles sont introduits dans la section 1.1. Certains ensembles permettent de décrire les liens qui existent entre des éléments appartenant à d'autres ensembles : il s'agit de la notion de relation qui est détaillée dans la section 1.2. Enfin, certaines propriétés sont vérifiées par une classe particulière de relations : ces propriétés caractérisent les fonctions et les applications qui sont présentées dans la section 1.3. Les notions introduites dans ce chapitre sont largement utilisées en mathématiques et en informatique, et serviront de base dans la suite de l'ouvrage. Pour une présentation formelle et détaillée des fondements de la théorie des ensembles, on pourra consulter [Dev12, Kri07].

1.1 Ensembles

1.1.1 Définitions et représentations des ensembles

Intuitivement, un ensemble est la réunion, dans une même entité, de certains objets bien déterminés et est caractérisé par une relation d'appartenance, notée $\in$: pour signifier qu'un objet e est un élément appartenant à un ensemble E, on écrit $e \in E$. Un ensemble E peut être défini de deux manières différentes :

— soit en extension, en faisant figurer (dans n'importe quel ordre) entre accolades tous les éléments appartenant à E (chaque élément apparaissant une unique fois) :

$$E = \{e_1, e_2, \cdots\}$$

— soit en compréhension, en donnant une propriété caractéristique P des éléments appartenant à E (c-à-d une propriété vérifiée par tous les éléments de E et seulement par les éléments de E) :

$$E = \{x \mid P(x)\}$$

Un ensemble E qui contient un nombre fini d'éléments est un ensemble fini, et dans ce cas sa taille peut être caractérisée par le nombre d'éléments qu'il contient, appelé cardinal de E et noté $|E|$. L'ensemble vide, noté $\emptyset$, est l'ensemble qui ne contient aucun élément et l'ensemble $\{e\}$ qui contient un unique élément e est appelé un singleton.

Les relations d'appartenance à l'ensemble vide et à un singleton sont définies par :

$$\forall x \; x \notin \emptyset \qquad \forall x \; (x \in \{e\} \Leftrightarrow x = e)$$

Aucun objet n'appartient à l'ensemble vide et l'objet x appartient au singleton contenant l'unique élément e si et seulement si $x = e$.

Remarque 1.1 (Logique)
Nous venons ici d'utiliser les symboles logiques $\Leftrightarrow$ et $\forall$ qui seront présentés formellement dans les chapitres 6 et 7. Le symbole $\Leftrightarrow$ correspond à l'équivalence logique entre deux propriétés : $P_1 \Leftrightarrow P_2$ signifie que la propriété P_1 est vérifiée si et seulement si la propriété P_2 est aussi vérifiée. En d'autres termes, soit les propriétés P_1 et P_2 sont toutes les deux vraies, soit elles sont toutes les deux fausses. En fait, une équivalence logique désigne deux implications logiques : $P_1 \Leftrightarrow P_2$ signifie que $P_1 \Rightarrow P_2$ et $P_2 \Rightarrow P_1$. Le symbole $\Rightarrow$ de l'implication permet d'exprimer qu'à chaque fois qu'une propriété P_1 est vraie, une propriété P_2 est vraie aussi. En d'autres termes, $P_1 \Rightarrow P_2$ signifie qu'il n'est pas possible que la propriété P_1 soit vraie quand la propriété P_2 est fausse. Aussi, comme nous le verrons dans le chapitre 6, lors d'un raisonnement, utiliser une hypothèse de la forme $P_1 \Rightarrow P_2$ permet d'établir la propriété P_2 à condition de démontrer la propriété P_1. De même, pour démontrer l'implication $P_1 \Rightarrow P_2$, il suffit de supposer la propriété P_1 vraie (sans la démontrer) et de démontrer à partir de cette hypothèse la propriété P_2. Le symbole $\forall$ est un quantificateur logique qui signifie « pour tout » : si $P(e)$ exprime qu'une propriété P est vérifiée par un élément e (e est ici un paramètre de la propriété P), alors $\forall x \; P(x)$ signifie que quel que soit un élément e appartenant à l'univers du discours, cet élément vérifie la propriété P, c-à-d que $P(e)$ est vrai pour n'importe quel élément e choisi arbitrairement. Aussi, comme nous le verrons dans le chapitre 7, lors d'un raisonnement, si l'on dispose de l'hypothèse $\forall x \; P(x)$, on peut déduire de cette hypothèse que $P(e)$ est vrai pour un élément quelconque e. De même, pour démontrer que $\forall x \; P(x)$ est vrai, il faut considérer un élément e quelconque, sans rien supposer sur cet élément et démontrer la propriété $P(e)$. Enfn, on écrit souvent $\forall x \in E \; P(x)$ pour signifier que tous les éléments d'un ensemble E particulier vérifient la propriété P, ce qui peut s'exprimer par l'implication $\forall x \; (x \in E \Rightarrow P(x))$.

Dans toute la suite, $\mathbb{N}$ désigne l'ensemble des entiers naturels $(0, 1, 2, 3, \cdots)$, $\mathbb{Z}$ désigne l'ensemble des entiers relatifs $(\cdots, -2, -1, 0, 1, 2, \cdots)$, $\mathbb{Q}$ désigne l'ensemble des nombres rationnels (les nombres qui peuvent s'exprimer par un quotient n/m où n est un entier relatif et m est un entier relatif non nul), $\mathbb{R}$ désigne l'ensemble des nombres réels (qui peuvent être représentés par une partie entière et une liste finie ou infinie de décimales), et $\mathbb{B} = \{0, 1\}$ est l'ensemble des booléens (0 est souvent interprété par la valeur de vérité « $false$ » et 1 par la valeur de vérité « $true$ » et, par abus de notation, $\mathbb{B}$ sera aussi identifié à l'ensemble $\{false, true\}$).

Exemple 1.1 L'ensemble E contenant uniquement les deux éléments 2 et 3 peut être défini en extension par $E = \{2, 3\}$ et il peut être aussi défini en compréhension par :

$$E = \{x \in \mathbb{N} \mid x^2 - 5x + 6 = 0\}$$

L'ordre d'apparition des éléments dans la définition en extension d'un ensemble n'est pas discriminant : $\{2,3\}$ et $\{3,2\}$ désignent donc le même ensemble. Un même ensemble peut admettre plusieurs définitions en compréhension différentes. Par exemple, l'ensemble $E = \{2,3\}$ peut également être défini en compréhension par :

$$E = \{n \in \mathbb{N} \mid 2 \leq n \leq 3\}$$

Cet ensemble E est fini et son cardinal est $|E| = 2$.

Représentations d'un ensemble

La notion intuitive d'ensemble peut sembler mathématiquement simple mais elle est délicate à exprimer avec un langage de programmation : un ensemble regroupe des éléments sans indiquer dans quel ordre ces éléments sont considérés alors qu'un ordinateur stocke des informations à des adresses bien déterminées de sa mémoire, ce qui revient à considérer ces informations dans un certain ordre. Selon les propriétés qu'il vérifie et la manière avec laquelle il est défini, il existe plusieurs façons différentes d'implanter un ensemble avec un langage de programmation. On peut par exemple représenter un ensemble E par sa relation d'appartenance, qui peut être vue comme une fonction booléenne f_E (appelée fonction caractéristique de E) qui, étant donné un élément (pris dans un domaine de référence), renvoie le booléen *true* si et seulement si cet élément appartient à l'ensemble E (et renvoie $false$ sinon). Par exemple, l'ensemble $E = \{2,3\}$ peut être défini par la fonction caractéristique f_E telle que $f_E(x)$ renvoie *true* lorsque $x^2 - 5x + 6 = 0$ (il s'agit ici d'une définition en compréhension puisque l'on indique quelle propriété caractérise les éléments de E).

PYTHON

```
def is_in_E(x):
    return x**2 - (5*x) + 6 == 0
>>> is_in_E(1)
False
>>> is_in_E(2)
True
```

OCAML

```
# let is_in_E = function x -> ( (x*x) - (5*x) + 6 = 0 );;
val is_in_E : int -> bool = <fun>
# (is_in_E 1);;
- : bool = false
# (is_in_E 2);;
- : bool = true
```

Lorsque E est un ensemble fini contenant n éléments, il est également possible de le représenter en extension en utilisant une structure de données permettant de réunir un nombre fini d'éléments. On peut par exemple utiliser une liste $\langle e_1, \cdots, e_n \rangle$ contenant les n éléments de E dans un ordre correspondant à une certaine énumération de ses éléments. On choisit ici de représenter un ensemble fini par une liste dans laquelle un élément ne peut pas apparaître plus d'une fois (toutes les listes ne représentent donc pas un ensemble, par exemple la liste $\langle 1,1,1 \rangle$ ne correspond pas à la représentation

d'un ensemble par une liste sans « doublon »). Il existe bien sûr plusieurs énumérations possibles pour un ensemble fini, chacune correspondant à une permutation (sans répétition) de ses éléments. Par exemple, il existe six énumérations différentes des éléments de l'ensemble $E = \{1, 2, 3\}$ que l'on peut représenter par chacune des six listes suivantes :

$$\langle 1,2,3\rangle \quad \langle 1,3,2\rangle \quad \langle 2,1,3\rangle \quad \langle 2,3,1\rangle \quad \langle 3,1,2\rangle \quad \langle 3,2,1\rangle$$

Plus généralement, il existe $n! = 1 \times 2 \times \cdots \times n$ permutations différentes des éléments appartenant à un ensemble fini E de cardinal n. Toutes ces permutations représentent le même ensemble puisque l'ordre d'apparition des éléments dans la définition en extension d'un ensemble n'est pas discriminant. Par exemple :

$$\{1,2,3\} \quad \{1,3,2\} \quad \{2,1,3\} \quad \{2,3,1\} \quad \{3,1,2\} \quad \{3,2,1\}$$

désignent le même ensemble. Si l'on choisit une représentation par une liste « sans doublon » d'un ensemble fini, la relation d'appartenance à un ensemble se ramène à la relation d'appartenance à une liste et le calcul du cardinal d'un ensemble se ramène au calcul de la longueur d'une liste :

PYTHON
```
def is_in(x,e):
    for h in e:
        if x==h:
            return True
    return False

def card(e):
    return len(e)
```

Remarque Le langage PYTHON fournit un type prédéfini pour les ensembles. On peut par exemple écrire directement l'ensemble $\{1, 2, 3\}$:

```
>>> {1,2,3}
set([1, 2, 3])
```

Toutefois, le type de éléments d'un ensemble de type `set` ne peut pas être un ensemble (ou une liste) et le type `set` de PYTHON ne peut donc être utilisé pour manipuler des ensembles d'ensembles. Par exemple, le type `set` ne peut pas être utilisé pour calculer l'ensemble des parties d'un ensemble (défini page 16). Le type prédéfini `frozenset` atténue en partie ces contraintes.

OCAML
```
# let rec is_in x e = match e with
    [] -> false | h::t -> (x=h) || (is_in x t);;
val is_in : 'a -> 'a list -> bool = <fun>
# let card e = (List.length e);;
val card : 'a list -> int = <fun>
```

Toutefois, dans la définition de la fonction `is_in`, l'utilisation de l'égalité « atomique » (opérateur `==` de PYTHON, opérateur `=` de OCAML) pour tester si `x` et `h`

sont égaux peut poser un problème. En effet, considérons par exemple l'ensemble $E = \{\{0,1\},\{1,2,3\},\{2\}\}$, dont les éléments sont aussi des ensembles, et dont une représentation possible est la liste $\ell = \langle\langle 0,1\rangle, \langle 1,2,3\rangle, \langle 2\rangle\rangle$. Les éléments de E étant des ensembles, ils peuvent aussi admettre plusieurs représentations : par exemple, l'élément $\{0,1\}$ de E peut être représenté par la liste $\langle 0,1\rangle$ et aussi par la liste $\langle 1,0\rangle$. Dans le premier cas, la fonction `is_in` indiquera que $\{0,1\} \in E$ puisque $\langle 0,1\rangle$ est un élément de la liste ℓ, tandis que dans le deuxième cas, la fonction `is_in` indiquera que $\{0,1\} \notin E$ puisque la liste $\langle 1,0\rangle$ (différente de la liste $\langle 0,1\rangle$) n'apparaît pas dans ℓ. Pour résoudre ce problème, toutes les fonctions sur les ensembles que nous définirons seront paramétrées par la relation d'égalité sur les éléments de cet ensemble, ce qui permettra, par exemple, d'identifier les ensembles représentés par les listes $\langle 0,1\rangle$ et $\langle 1,0\rangle$. Aussi, la fonction caractéristique d'un ensemble représenté par une liste s'écrit :

PYTHON
```
def is_in(eq,x,e):
    for h in e:
        if eq(x,h):
            return True
    return False
```

OCAML
```
# let rec is_in eq x e = match e with
    [] -> false | h::t -> (eq x h) || (is_in eq x t);;
val is_in : ('a -> 'b -> bool) -> 'a -> 'b list -> bool = <fun>
```

Cette fonction peut également s'écrire un utilisant la fonctionnelle `List.exists` :

```
let is_in eq x e = (List.exists (function z -> (eq x z)) e);;
```

La relation d'égalité `eq` sur les éléments de la liste est passée en argument de la fonction `is_in` et est utilisée pour tester l'égalité entre `x` et `h`. Si les éléments d'un ensemble sont représentés par des valeurs d'un type sur lequel l'égalité atomique convient, alors on utilise simplement la relation d'égalité suivante :

PYTHON
```
def eq_atom(x,y):
    return x==y
```

OCAML
```
# let eq_atom x y = (x=y);;
val eq_atom : 'a -> 'a -> bool = <fun>
```

Lorsque ce n'est pas le cas, il faut définir une relation d'égalité adéquate. Par exemple, si les éléments d'un ensemble sont des couples d'éléments (x,y) où chaque x appartient à un ensemble E muni d'une relation d'égalité `eq1` sur ses éléments, et chaque y appartient à un ensemble F muni d'une relation d'égalité `eq2` sur ses éléments, alors la relation d'égalité sur les couples peut être définie par la fonction suivante :

PYTHON
```
def eq_pair(eq1,eq2,p1,p2):
    x1,x2=p1
    y1,y2=p2
    return eq1(x1,y1) and eq2(x2,y2)
```

OCAML
```
# let eq_pair eq1 eq2 (x1,x2) (y1,y2) = (eq1 x1 y1) && (eq2 x2 y2);;
val eq_pair : ('a -> 'b -> bool) -> ('c -> 'd -> bool) -> 'a * 'c -> 'b * 'd
              -> bool = <fun>
```

Etant données les relations d'égalité `eq1` et `eq2`, la fonction suivante permet d'obtenir directement la relation d'égalité sur les couples. Le résultat produit par cette fonction est une fonction et les langages qui permettent de manipuler ainsi des fonctions sont appelés des langages d'ordre supérieur.

PYTHON
```
def make_eq_pair(eq1,eq2):
    def _eq_pair(p1,p2):
        x1,x2 = p1
        y1,y2 = p2
        return eq1(x1,y1) and eq2(x2,y2)
    return _eq_pair
```

Avec le langage OCAML, il suffit d'écrire `(eq_pair eq1 eq2)` pour obtenir la fonction d'égalité sur les couples.

OCAML
```
# let make_eq_pair eq1 eq2  = (eq_pair eq1 eq2);;
val make_eq_pair : ('a -> 'b -> bool) -> ('c -> 'd -> bool) -> 'a * 'c -> 'b * 'd
                  -> bool = <fun>
```

On utilise ici le mécanisme d'application partielle du langage OCAML qui est souvent utilisé lorsque l'on programme dans un style fonctionnel et qui permet de ne fournir que les premiers arguments d'une fonction pour construire une autre fonction. Dans la suite, on ne définira pas de fonction en lui attribuant un nom pour les fonctions obtenues avec ce mécanisme. Ceci est aussi possible avec le langage PYTHON mais correspond à une pratique moins courante lorsque l'on programme avec ce langage.
Les fonctions caractéristiques de l'ensemble vide $\emptyset$ et du singleton $\{e\}$ s'écrivent simplement comme suit.

PYTHON
```
def is_in_emptyset(x):
    return False
def is_in_singleton(eq,elem,x):
    return eq(elem,x)
```

OCAML
```
# let is_in_emptyset = function x -> false;;
val is_in_emptyset : 'a -> bool = <fun>
# let is_in_singleton eq elem = function x -> (eq elem x);;
val is_in_singleton : ('a -> 'b -> 'c) -> 'a -> 'b -> 'c = <fun>
```

Ces deux ensembles sont des ensembles finis que l'on peut également représenter en extension respectivement par la liste vide et la liste $\langle e \rangle$:

PYTHON
```
emptyset = []
def singleton(e):
    return [e]
```

OCAML
```
# let emptyset = [];;
val emptyset : 'a list = []
# let singleton e = [e];;
val singleton : 'a -> 'a list = <fun>
```

Il existe bien d'autres façons de représenter un ensemble qui reposent en général sur les propriétés qu'il vérifie (par exemple, pour un ensemble fini totalement ordonné, une structure d'arbre permet une représentation plus efficace pour certaines opérations).

Inclusion, égalité Etant donnés deux ensembles A et B, on dit que A est inclus dans B, ou encore que A est un sous-ensemble de B, si et seulement si tous les éléments appartenant à A appartiennent aussi à B, ce que l'on note $A \subseteq B$.

$$A \subseteq B \Leftrightarrow (\forall x\ x \in A \Rightarrow x \in B)$$

Si A et B sont des ensembles finis, lorsque $A \subseteq B$, on a alors $|A| \leq |B|$.

Exemple 1.2 $\emptyset \subseteq \{0\} \subseteq \{0, 1, 2, 3\} \subseteq \mathbb{N} \subseteq \mathbb{Z} \subseteq \mathbb{Q} \subseteq \mathbb{R}$

Deux ensembles A et B sont égaux, ou identiques, s'ils contiennent exactement les mêmes éléments :

$$A = B \Leftrightarrow (A \subseteq B \text{ et } B \subseteq A)$$

Pour pouvoir vérifier qu'un ensemble A est inclus dans un ensemble B, il faut être capable de vérifier que tout élément de A appartient à B. Si A est un ensemble fini, il suffit alors de considérer une énumération de ses éléments pour procéder à cette vérification. Lorsque les ensembles sont représentés par des listes, il suffit donc de parcourir une liste représentant A et de vérifier que tous ses éléments appartiennent à la liste représentant B.

PYTHON
```
def is_subset(eq,la,lb):
    for a in la:
        if not(is_in(eq,a,lb)):
            return False
    return True
```

OCAML
```
# let is_subset eq la lb = (List.for_all (function x -> (is_in eq x lb)) la);;
val is_subset : ('a -> 'b -> bool) -> 'a list -> 'b list -> bool = <fun>
```

L'égalité de deux ensembles représentés par des listes se définit alors simplement à partir de la définition de l'inclusion.

PYTHON
```
def eq_set(eq,la,lb):
    return is_subset(eq,la,lb) and is_subset(eq,lb,la)
```

OCAML
```
# let eq_set eq la lb = (is_subset eq la lb) && (is_subset eq lb la);;
val eq_set : ('a -> 'a -> bool) -> 'a list -> 'a list -> bool = <fun>
```

Par exemple, la fonction `eq_set` appliquée aux listes $\langle 0, 1\rangle$ et $\langle 1, 0\rangle$ retourne le booléen *true* et permet donc de considérer le test de l'appartenance de l'élément $\{0, 1\}$ à l'ensemble $E = \{\{0, 1\}, \{1, 2, 3\}, \{2\}\}$ quelles que soient les listes utilisées pour représenter l'ensemble $\{0, 1\}$.

PYTHON
```
def eq_set_atom(la,lb):
    return eq_set(eq_atom,la,lb)
>>> is_in(eq_set_atom,[1,0],[[0,1],[1,2,3],[2]])
True
```

OCAML
```
# (is_in (eq_set eq_atom) [1;0] [[0;1];[1;2;3];[2]]);;
- : bool = true
```

Ici encore, avec le langage PYTHON on définit une fonction qui retourne la fonction permettant de tester l'égalité de deux ensembles dont les éléments sont comparables avec la relation d'égalité `eq` comme suit.

PYTHON
```
def make_eq_set(eq):
    def _eq_set(la,lb):
        return is_subset(eq,la,lb) and is_subset(eq,lb,la)
    return _eq_set
```

En utilisant le mécanisme d'application partielle du langage OCAML, on écrira simplement `(eq_set eq)` pour obtenir cette fonction d'égalité sur les ensembles.

1.1.2 Opérations sur les ensembles

Dans ce qui suit, on considère que l'on dispose d'un ensemble de référence (aussi parfois appelé domaine de référence, ou encore univers), noté **U**, contenant tous les éléments possibles pour définir des ensembles. Dans ce cadre, tout ensemble est un sous-ensemble de **U**.

Remarque 1.2 (Paradoxe de Russel)
Nous supposons ici que tous les ensembles sont des sous-ensembles de l'univers **U**. Toutefois, il n'est pas possible de définir l'ensemble de tous les ensembles. C'est le