AFGL – TD 4

SÉMANTIQUE ALGÉBRIQUE

---

1 Entiers naturels

Soit le TAA définissant un type entier naturel :

TAA : Type Abstrait Algébrique

Exemple :

```
type bool
opérations
    true  : ⟶ bool        -- constante
    false : ⟶ bool        -- constante
    or : bool × bool ⟶ bool
axiomes
    or(a, b) = or(b, a)
    or(b, true) = true
    or(b, false) = b
```

```
type ent
utilise bool
opérations
    0    : ⟶ ent            -- constructeur
    suc  : ent ⟶ ent        -- constructeur
    pred : ent ⟶ ent
    nul  : ent ⟶ bool
```

Q1 *Compléter les axiomes de ce type.*

Axiome : vérité indémontrable qui doit être admise.

A part dans une autre version du formalisme intégrant les pré-conditions (voir exercice 3), les opérations sont des fonctions totales. Il faut donc ici en particulier prévoir une valeur pour `pred(0)`, par exemple en définissant une constante *indéfinie*.

```
indef : ⟶ ent
pred(0) = 0
ou
pred(0) = indef
pred(indef) = indef
suc(indef) = indef
```

```
nul(0) = true
nul(suc(n)) = false
pred(suc(n)) = n  attention : suc(pred(n)) = n n'est pas vrai !
```

Q2 *Enrichir ce type avec les opérations d'égalité et d'addition.*

Pour l'addition :

```
opérations
_ + _ : ent × ent ⟶ ent
axiomes
i + j = j + i
i + 0 = i      en particulier : 0 + 0 = 0
suc(i) + j = suc(i + j)
```

Pour l'égalité :

```
opérations
eq : ent × ent ⟶ bool
axiomes
eq(0, 0) = true
eq(suc(i), suc(j)) = eq(i, j)
eq(suc(i), 0) = false
eq(i, j) = eq(j, i)
```

Question subsidiaire : prouver par induction que eq(x, x) = true.

Soit E un ensemble dénombrable muni d'une bijection $f : \mathbb{N} \mapsto E$. Ici E est constitué de tous les successeurs (par composition) de 0.

Preuve par induction d'une propriété P : $\big(P(f(0)) \wedge \forall i \in \mathbb{N}, P(f(i)) \Rightarrow P(f(i+1))\big) \Rightarrow \forall e \in E\, P(e)$

Soit la bijection f définie par $f(0) = 0$ et $f(i+1) = \texttt{suc}(i)$.

De part les axiomes définis, on obtient directement :

- eq(0, 0) = true
- eq(i, i) = true ⇒ eq(suc(i), suc(i)) = true

Par induction, $\forall e \in \texttt{ent} - \{\texttt{indef}\}$, eq(x, x) = true.

On pourrait également prouver, par exemple, que :

- eq(x+suc(k), x) = false, par induction sur x,
- eq(x+k, y+k) = eq(x, y), par induction sur k.

Deux intérêts (dans le cadre de l'informatique pour systèmes critiques) :

- Prouver (par induction), les autres propriétés spécifiées à partir des axiomes.
- Implémenter des tests unitaires pour vérifier les axiomes.

2 Listes

On veut spécifier le TAA liste définissant les listes d'entiers. Les deux constructeurs sont `lv` (qui crée une liste vide) et `l` (qui construit une nouvelle liste à partir d'un entier et d'une liste existante). Ainsi, l'expression `l(8, l(3, l(2, lv)))` est une liste qui comprend les valeurs 8, 3 et 2.

Les opérations définissables sont `lg` (longueur d'une liste), `inv` (inversion d'une liste) et `conc` (concaténation de deux listes).

Q3 *Écrire le TAA liste.*

```
type liste
utilise ent
opérations
    lv   : → liste
    l    : ent × liste → liste
    lg   : liste → ent
    inv  : liste → liste
    conc : liste × liste → liste
axiomes
    lg(lv) = 0
    lg(l(v, g)) = suc(0) + lg(g)
    conc(lv, g) = g
    conc(l(v, k), g) = l(v, conc(k, g))
    inv(lv) = lv
    inv(l(v, g)) = conc(inv(g), l(v, lv))
```

3 Suites

Soit le TAA suite définisant le type suite :

```
type suite
utilise val, ent
opérations
    sCreer : → suite                        -- constructeur délivrant une suite vide
    sInserer : suite × val × ent → suite
    sValeur : suite × ent → val
    sLong : suite → ent
    sConcatener : suite × suite → suite
    sSupp : suite × ent → suite
opérations auxiliaires
    sAjout : suite × val → suite          -- constructeur
préconditions
    sInserer(s, v, i) : 1 ≤ i et i ≤ sLong(s) + 1
    sSupp(s, i) : 1 ≤ i et i ≤ sLong(s)
    sValeur(s, i) : 1 ≤ i et i ≤ sLong(s)
```

Q4 *Donner les axiomes du type suite.*

Opération auxiliaire : opération utilisée pour les préconditions ou les axiomes.

On suppose que la représentation des entiers ici est *naturelle*.

Solution avec ajout en queue de suite :

```
axiomes
    sLong(sCreer) = 0
    sLong(sAjout(s, v)) = suc(sLong(s))

    sConcatener(s, sCreer) = s
    sConcatener(s, sAjout(l, v)) = sAjout((sConcatener(s, l), v)

    sInserer(s, v, sLong(s) + 1) = sAjout(s, v)
    sInserer(sAjout(s, w), v, i) = sAjout(sInserer(s, v, i), w)

    sSupp(sInserer(s, v, i), i) = s

    sValeur(sInserer(s, v, i), i) = v
```

Solution avec ajout en tête de suite :

```
axiomes
    sLong(sCreer) = 0
    sLong(sAjout(s, v)) = suc(sLong(s))

    sConcatener(sCreer, s) = s
    sConcatener(sAjout(s, v), l) = sAjout((sConcatener(s, l), v)

    sInserer(s, v, 1) = sAjout(s, v)
    sInserer(sAjout(s, w), v, suc(i)) = sAjout(sInserer(s, v, i), w)

    sSupp(sInserer(s, v, i), i) = s

    sValeur(sInserer(s, v, i), i) = v
```

Remarque : ce n'est plus utilisé dans cette version de la correction, mais il est possible d'utiliser les implications logiques (⇒) pour définir des axiomes conditionnels.