Leçon 4 : Principe d'induction

Prérequis : Principe de récurrence
Maîtrise d'OCamL (sauf type)

Niveau : MPII

I- Principe

I-1) Définition

Définition 1 : $(\mathcal{B}, (f_i)_{i \in I})$ est une signature sur X si :
- $\mathcal{B} \subset X$ (appelé cas de base)
- $f_i : X^{d(i)} \to X$ (appelé constructeur) avec $d(i) \in \mathbb{N}^*$ (appelé arité) ; f_i injectif et $\mathrm{Im}(f_j) \cap \mathrm{Im}(f_i) = \emptyset$ pour $i \neq j$ et $\mathrm{Im}(f_i) \cap \mathcal{B} = \emptyset$

Remarque 2 : On se contente souvent de dire que les constructeurs existent, sans donner leur définition (de même pour les cas de base et pour X)

Exemple 3 : On prend une constante Z et un constructeur d'arité 1 Succ.

Exemple 4 : On peut prendre les constructeurs $\oplus$, $\ominus$ et $\otimes$ d'arité 2, 1 et 2 et $\mathbb{N}$ comme cas de bases

Définition 5 : Un ensemble inductif est définit par une signature $(\mathcal{B}, (f_i)_{i \in I})$ de manière équivalente par :
(i) Le plus petit ensemble contenant $\mathcal{B}$ et stable par tous les f_i
(ii) $\bigcup_{k \in \mathbb{N}} T_k$ où $T_0 = \mathcal{B}$ et $T_{k+1} = T_k \cup \bigcup_{i \in I} f_i(T_k^{d(i)})$

Exemple 6 : L'ensemble inductif définit par l'exemple 3 peut-être une définition des entiers naturels

Exemple 7 : L'ensemble inductif $\mathcal{A}_{simp}$ définit par l'exemple 4 est l'ensemble des expressions arithmétiques simplifiées.

Remarque 8 : $\oplus(1,1) \neq \otimes(1,2) \neq \otimes(2,1)$. On s'intéresse à l'expression et non au résultat.

$\boxed{\text{Développement 1 : Construction d'un ensemble inductif (rq 2 + def 6)}}$

I-2) Induction Structurelle

On prendra maintenant $(\mathcal{B}, (f_i)_{i \in I})$ une signature

Propriété 9 : Soit E l'ensemble inductif construit par $(\mathcal{B}, (f_i)_{i \in I})$
Alors la donnée de fonction g_i (d'arité $d(i)$ et avec $\mathrm{Im}(g_i) \subset \mathrm{Dom}(g_j)$) et de $f(b)$ pour $b \in \mathcal{B}$ définit une unique fonction f sur E vérifiant :
$$\forall i \in I, \forall x_1, \dots, x_{d(i)} \in X, f(f_i(x_1, \dots, x_{d(i)})) = g_i(f(x_1), \dots, f(x_{d(i)}))$$

Exemple 10 : Sur $\mathcal{A}_{simp}$, on peut définir eval : $\mathcal{A}_{simp} \to \mathbb{N}$ par
$eval(a) = a$ pour $a \in \mathbb{N}$ $\qquad$ $eval(\otimes(a,b)) = eval(a) \times eval(b)$
$eval(\oplus(a,b)) = eval(a) + eval(b)$ $\qquad$ $eval(\ominus(a)) = -eval(a)$

Théorème 11 : (Induction structurelle)
Soit E l'ensemble inductif défini par $(\mathcal{B}, (f_i)_{i \in I})$, et P une propriété définie pour tout $x \in E$.
Alors $\begin{cases} (i)\ \forall b \in \mathcal{B}, P(b) \\ (ii)\ \forall i \in I, \forall x_1, \dots, x_{d(i)} \in X, (\forall j, P(x_j)) \Rightarrow P(f_i(x_1, \dots, x_{d(i)})) \end{cases}$
$\Rightarrow \forall x \in E, P(x)$

Remarque 12 : La récurrence est un cas particulier dans le cas de la définition des entiers par l'exemple 6.

Exemple 13 : On montre que $eval(e)$ pour $e \in \mathcal{A}_{simp}$ est multiple du pgcd des constantes apparaissant dans e.

Définition 14 : Soit E l'ensemble inductif défini par $(\mathcal{B}, (f_i)_{i \in I})$
On définit l'ordre structurel $\leqslant_s$ sur E comme la clôture transitive reflexive de $x_j \leqslant_s f_i(x_1, \dots, x_{d(i)})$

Propriété 15 : $\leqslant_s$ est bien une relation d'ordre

Théorème 16 : L'induction structurelle se réécrit
$$(\forall x\ (\forall y < x, P(x))) \Rightarrow \forall x, P(x)$$

Remarque 17 : C'est l'équivalent de la récurrence forte

I-3) En OCaml

Syntaxe 18 : En OCaml, on peut créer un type représentant un ensemble inductif avec cette syntaxe :

```
type t = Casdebase1 | Casdebase2 | ... |
         Constructeur1 of type1 | Constructeur of type2 | ...
```

où - Casdebasei peut soit être un type existant, soit une constante (nom commençant par une majuscule),
- Constructeuri est une étiquette (commencée par une majuscule)
- typei est un type OCaml (pouvant contenir t).

Exemple 19 : Pour définir les entiers de l'exemple 6, on peut écrire `type entier = Zero | Succ of entier`

Syntaxe 20 : Pour gérer les types, on peut utiliser le filtrage comme pour les listes

Exemple 21 : Pour l'addition sur notre type entier on peut écrire :

```
let rec ajouter x y = match y with
  | Zero -> x
  | Succ(y) -> ajouter (Succ(x)) z
```

Remarque 22 : La validité de cette définition vient de la propriété 9.

II- Structures de données inductives

II-1) Les listes chaînées

En OCaml, on peut définir des listes d'entier simplement chaînées par `type liste = V | Cons of int * liste`

Ainsi, une liste, c'est soit une liste vide, soit un entier et le reste de la liste.

Remarque 23 : Ici V est le cas de base et Cons le constructeur. Mais Cons est défini sur $\mathbb{N} \times \{liste\}$ et non $\{liste\}^2$. C'est un raccourci OCaml, où en réalité on définit un constructeur pour chaque premier argument, et donc on construit non pas Cons (x, ℓ) mais $Cons_x(\ell)$.

Remarque 24 : cela correspond au type int list d'OCaml ;-)

Exercice 25 : Définir inductivement la taille d'une liste chaînée.

II-2) Les arbres binaires

Définition 26 : Arbre binaire

Soit A un ensemble. On définit de manière inductive les arbres binaires sur A par :
- l'arbre vide E (cas de base)
- si $e \in A$ et g et d sont des arbres binaires, alors Noeud (e, g, d) est un arbre binaire

Ce qui en OCaml nous donne `type 'a arbre = E | Noeud of 'a * 'a arbre * 'a arbre;;`

Exemple 27 : La hauteur d'un arbre binaire se calcule alors inductivement par

```
let rec hauteur arb = match arb with
  | E -> 0
  | Noeud (e, g, d) -> 1 + max (hauteur g) (hauteur d)
```

Exercice 28 : Prouver par induction structurelle la terminaison de la fonction hauteur.

Exercice 29 : Donner la définition inductive de la taille d'un arbre binaire (son nombre d'éléments).

II-3) Les arbres généraux

Définition 30 : Un arbre général est un noeud (la racine) et une liste d'arbre (ses fils).

On voudrait alors définir le type arbre par (pour les arbres d'entier)

`type arbre = Noeud of int * arbre_liste`

Il faut alors définir le type arbre_liste, par

`type arbre_liste = Vide | Cons of arbre * arbre_liste`

Ne rien écrire hors du cadre.

On remarque que chaque type a besoin de l'autre pour exister.
On écrit alors

```
type arbre = Noeud of int * arbre_liste
and type arbre_liste = Vide | Cons arbre * arbre_liste
```

Remarque 31 : On passe souvent cela sous le tapis grâce au polymorphisme qui définit des manières de construire des types et non des types directement.

Développement 2 : Equivalence des arbres binaires et des arbres généraux.

III - Ensembles inductifs

III - 1) Formules propositionelles

Définition 32 : (formule propositionelle)
Soit V un ensemble de variables et la signature :
- $\mathcal{B} = \{\bot, \top\} \cup V$
- le constructeur $\neg$ d'arité 1
- les constructeurs $\vee$, $\wedge$ et $\rightarrow$ d'arité 2 (en forme infixe).

Les formules propositionnelles forment l'ensemble inductif défini par cette signature.

Exercice 33 : Définir inductivement le nombre de variables présent dans la formule.

Définition 33 : On appelle valuation (ou distribution de vérité) toute fonction $v : V \rightarrow \{0, 1\}$

Définition 34 : Évaluation d'une formule.
Soit v une valuation. La fonction d'évaluation d'une formule $[\cdot]_v : F \rightarrow \{0, 1\}$ se définit inductivement par :

$[\top]_v = 1$ $\quad [\bot]_v = 0$ $\quad [x]_v = v(x)$ pour $x \in V$

$[\neg F]_v = -[F]_v$ $\quad [\ell_1 \vee \ell_2]_v = \max([\ell_1]_v, [\ell_2]_v)$

$[\ell_1 \wedge \ell_2]_v = \min([\ell_1]_v, [\ell_2]_v)$

$$[\ell_1 \rightarrow \ell_2]_v = \begin{cases} 0 & \text{si } [\ell_1]_v = 0 \text{ et } [\ell_2]_v = 1 \\ 1 & \text{sinon} \end{cases}$$

Exercice 35 : Définir l'équivalence entre formules et montrer par induction structurelle que, à équivalence près, on peut ne garder que les constructeurs $\neg$ et $\vee$.

III - 2) Langages

Définition 36 : Soit Σ un ensemble fini et non vide d'élément (appelé alphabet). On définit inductivement l'ensemble des mots sur Σ, Σ^* par la signature :
- ε (le mot vide) comme cas de base
- Si $a \in \Sigma$ et $w \in \Sigma^*$, $w.a \in \Sigma^*$

Remarque 37 : On aurait aussi pu prendre comme définition de Σ^*, $\bigcup_{n \geq 0} \Sigma^n$

Définition 38 : La concaténation de deux mots $w_1, w_2 \in \Sigma^*$ se définit inductivement comme :
- $\text{concat}(w_1, \varepsilon) = w_1$
- $\text{concat}(w_1, w_2.a) = \text{concat}(w_1, w_2).a$

Exercice 39 : Montrer par induction structurelle que $\text{concat}(w_1, \text{concat}(a, w_2)) = \text{concat}(w_1 a, w_2)$