

Induction.

1 Définitions inductives d'ensembles.

Dans ce cours, les ensembles définis par induction représenteront les données utilisées par les programmes. De plus, les notions d'ensembles et de types seront identiques : on identifiera :

$$\underbrace{n \in \text{nat}}_{\text{ensemble}} \longleftrightarrow \underbrace{n : \text{nat}}_{\text{type}}.$$

Exemple 1 (Types définis inductivement). Dans le code ci-dessous, on définit trois types : le type `nat` représentant les entiers naturels (construction de Peano); le type `nlist` représentant les listes d'entiers naturels; et le type `t` représentant les arbres binaires étiquetés par des entiers aux nœuds.

```
type nat = Z | S of nat
type nlist = Nil | Const of nat * nlist
type t1 = F | N of t1 * nat * t1
```

Code 1 | *Trois types définis inductivement*

Définition 1. La *définition inductive d'un ensemble* `t` est la donnée de k constructeurs C_1, \dots, C_k , où chaque C_i a pour argument un n_i -uplet dont le type est $u_1^i * u_2^i * \dots * u_{n_i}^i$. L'opération « $*$ » représente le produit cartésien, avec une notation « à la OCaml ». De plus, chaque u_j^i est, soit `t`, soit un type pré-existant.

Exemple 2.

```

type t2 =
  | F
  | N2 of (t * nlist * t)
  | N3 of (t * nat * t * nat * t)

```

Code 2 | *Un exemple de type*

Définition 2. Les *types algébriques* sont définis en se limitant à deux opérations :

- ▷ le produit cartésien « * » ;
- ▷ le « ou », noté « | » ou « + », qui correspond à la somme disjointe ;

et un type :

- ▷ le type `unit`, noté 1.

Exemple 3 (Quelques types algébriques...). ▷ Le type `bool` est alors défini par $1 + 1$.

- ▷ Le type « jour de la semaine » est alors défini par l'expression $1 + 1 + 1 + 1 + 1 + 1 + 1$.
- ▷ Le type `nat` vérifie l'équation $X = 1 + X$.
- ▷ Le type `nlist` vérifie l'équation $X = 1 + \text{nat} * X$.
- ▷ Le type `t option` est alors défini par $1 + t$.

Ces ensembles définis inductivement nous intéressent pour deux raisons :

- ▷ pour pouvoir calculer, c'est à dire définir des fonctions de `t` vers `t'` en faisant du *filtrage* (i.e. avec `match ... with`)
- ▷ raisonner / prouver des propriétés sur les éléments de `t` : « des preuves par induction ».

2 Preuves par induction sur un ensemble inductif.

Exemple 4. Intéressons nous à nat . Pour prouver $\forall x \in \text{nat}, \mathcal{P}(x)$, il suffit de prouver *deux* choses (parce que l'on a deux constructeurs à l'ensemble nat) :

1. on doit montrer $\mathcal{P}(0)$;
2. et on doit montrer $\mathcal{P}(S\ n)$ en supposant l'*hypothèse d'induction* $\mathcal{P}(n)$.

Remarque 1. Dans le cas général, pour prouver $\forall x \in \mathbf{t}, \mathcal{P}(x)$, il suffit de prouver les n propriétés (n est le nombre de constructeurs de l'ensemble \mathbf{t}), où la i -ème propriété s'écrit :

On montre $\mathcal{P}(\mathbf{C}_i(x_1, \dots, x_n))$ avec les hypothèses d'inductions $\mathcal{P}(x_j)$ lorsque $u_j^i = \mathbf{t}$.

Exemple 5. Avec le type \mathbf{t}_2 défini dans l'exemple 2, on a trois constructeurs, donc trois cas à traiter dans une preuve par induction. Le second cas s'écrit :

On suppose $\mathcal{P}(x_1)$ et $\mathcal{P}(x_3)$ comme hypothèses d'induction, et on montre $\mathcal{P}(\mathbf{N2}(x_1, k, x_3))$, où l'on se donne $k \in \text{nat}$.

Exemple 6. On pose la fonction `red` définie par le code ci-dessous.

```
let rec red k ℓ = match ℓ with
| Nil -> Nil
| Cons(x, ℓ) -> let ℓ'' = red k ℓ'' in
                  if x = k then ℓ''
                  else Cons(x, ℓ'')
```

Code 3 | Fonction de filtrage d'une liste

Cette fonction permet de supprimer toutes les occurrences de k dans une liste ℓ .

Démontrons ainsi la propriété

$$\forall \ell \in \text{nlist}, \underbrace{\forall k \in \text{nat}, \text{size}(\text{red } k \ell) \leq \text{size } \ell}_{\mathcal{P}(\ell)}.$$

Pour cela, on procède par induction. On a *deux* cas.

1. Cas Nil : $\forall k \in \text{nat}, \text{size}(\text{red } k \text{ Nil}) \leq \text{size Nil}$;
2. Cas Cons(x, ℓ') : on suppose

$$\forall k \in \text{nat}, \text{size}(\text{red } k \ell) \leq \text{size } \ell,$$

et on veut montrer que

$$\forall k \in \text{nat}, \text{size}(\text{red } k \text{ Cons}(x, \ell')) \leq \text{size Cons}(x, \ell'),$$

ce qui demandera deux sous-cas : si $x = k$ et si $x \neq k$.

3 Définitions inductives de relations.

Dans ce cours, les relations définies par inductions représenteront des propriétés sur des programmes.

Un premier exemple : notations et terminologies.

Une relation est un sous-ensemble d'un produit cartésien. Par exemple, la relation $\text{le} \subseteq \text{nat} * \text{nat}$ est une relation binaire. Cette relation représente \leq , « *lesser than or equal to* » en anglais.

Notation. On note $\text{le}(n, k)$ dès lors que l'on a $(n, k) \in \text{le}$.

Pour définir cette relation, on peut écrire :

Soit $\text{le} \subseteq \text{nat} * \text{nat}$ la relation qui vérifie :

1. $\forall n \in \text{nat}, \text{le}(n, n)$;
2. $\forall (n, k) \in \text{nat} * \text{nat}$, si $\text{le}(n, k)$ alors $\text{le}(n, S \ k)$.

mais, on écrira plutôt :

Soit $\text{le} \subseteq \text{nat} * \text{nat}$ la relation définie (inductivement) à partir des règles d'inférence suivantes :

$$\frac{}{\text{le}(n, n)} \mathcal{L}_1 \quad \frac{\text{le}(n, k)}{\text{le}(n, S\ k)} \mathcal{L}_2 .$$

Remarque 2. ▷ Dans la définition par règle d'inférence, chaque règle a *une* conclusion de la forme $\text{le}(\cdot, \cdot)$.

▷ Les *métavariabiles* n et k sont quantifiées universellement de façon implicite.

Définition 3. On appelle *dérivation* ou *preuve* un arbre construit en appliquant les règles d'inférence (ce qui fait intervenir l'*instantiation des métavariabiles*) avec des axiomes aux feuilles.

Exemple 7. Pour démontrer $\text{le}(2, 4)$, on réalise la dérivation ci-dessous.

$$\frac{\frac{\frac{}{\text{le}(2, 2)} \mathcal{L}_1}{\text{le}(2, 3)} \mathcal{L}_2}{\text{le}(2, 4)} \mathcal{L}_2$$

Exemple 8. On souhaite définir une relation *triée* sur nlist . Pour cela, on pose les trois règles ci-dessous :

$$\frac{}{\text{triée Nil}} \mathcal{T}_1 \quad \frac{}{\text{triée Cons}(x, \text{Nil})} \mathcal{T}_2 ,$$

$$\frac{\text{le}(x, y) \quad \text{triée Cons}(x, \text{Nil})}{\text{triée Cons}(x, \text{Cons}(y, \ell))} \mathcal{T}_3 .$$

Ceci permet de dériver, modulo quelques abus de notations, que

la liste $[1;3;4]$ est triée :

$$\begin{array}{c}
 \frac{1 \leq 1}{1 \leq 2} \mathcal{L}_1 \quad \frac{3 \leq 3}{3 \leq 4} \mathcal{L}_1 \quad \frac{}{\text{triée } [4]} \mathcal{R}_2 \\
 \frac{1 \leq 2}{1 \leq 3} \mathcal{L}_2 \quad \frac{3 \leq 4}{\text{triée } [3;4]} \mathcal{R}_2 \quad \frac{}{\text{triée } [1;3;4]} \mathcal{R}_3
 \end{array}
 .$$

Les parties en bleu de l'arbre ne concernent pas la relation *triée*, mais la relation *le*.

Exemple 9. On définit la relation *mem* d'appartenance à une liste. Pour cela, on définit $\text{mem} \subseteq \text{nat} * \text{nlist}$ par les règles d'inférences :

$$\frac{}{\text{mem}(k, \text{Cons}(k, \ell))} \mathcal{M}_1 \quad \frac{\text{mem}(k, \ell)}{\text{mem}(k, \text{Cons}(x, \ell))} \mathcal{M}_2 .$$

On peut constater qu'il y a plusieurs manières de démontrer

$$\text{mem}(0, [0;1;0]).$$

Ceci est notamment dû au fait qu'il y a deux '0' dans la liste.

Remarque 3. Attention ! Dans les prémisses d'une règle, on ne peut pas avoir « $\neg r(\dots)$ ». Les règles ne peuvent qu'être « constructive », donc pas de négation.

Exemple 10. On définit la relation $\text{ne} \subseteq \text{nat} * \text{nat}$ de non égalité entre deux entiers.

On pourrait imaginer créer une relation d'égalité et de définir *ne* comme sa négation. Mais non, c'est ce que nous dit la remarque 3.

On peut cependant définir la relation **ne** par :

$$\frac{}{\text{ne}(\mathbf{Z}, \mathbf{S} \ k)} \mathcal{N}_1 \quad \frac{}{\text{ne}(\mathbf{S} \ n, \mathbf{Z})} \mathcal{N}_2 \quad \frac{\text{ne}(n, k)}{\text{ne}(\mathbf{S} \ n, \mathbf{S} \ k)} \mathcal{N}_3.$$

Il est également possible de définir **ne** à partir de la relation **le**.

Exemple 11. En utilisant la relation **ne** (définie dans l'exemple 10), on peut revenir sur la relation d'appartenance et définir une relation alternative à celle de l'exemple 9. En effet, soit la relation **mem'** définie par les règles d'inférences ci-dessous :

$$\frac{}{\text{mem}'(n, \text{Cons}(n, \ell))} \mathcal{M}'_1 \quad \frac{\text{mem}'(n, \ell) \quad \text{ne}(k, n)}{\text{mem}'(n, \text{Cons}(k, \ell))} \mathcal{M}'_2.$$

Il est (*sans doute ?*) possible de montrer que :

$$\forall(n, \ell) \in \text{nat} * \text{nlist}, \text{mem}(n, \ell) \iff \text{mem}'(n, \ell).$$

Remarque 4. Dans le cas général, une définition inductive d'une relation **Rel**, c'est k règles d'inférences de la forme :

$$\frac{H_1 \quad \dots \quad H_n}{\text{Rel}(x_1, \dots, x_m)} \mathcal{R}_i,$$

où chaque H_j est :

- ▷ soit $\text{Rel}(\dots)$;
- ▷ soit une autre relation pré-existante (*c.f.* la définition de triée dans l'exemple 8).

On appelle les H_j les *prémisses*, et $\text{Rel}(x_1, \dots, x_m)$ la *conclusion*. Elles peuvent faire intervenir des *métavariabes*.

4 Preuves par induction sur une relation inductive.

On souhaite établir une propriété de la forme

$$\forall (x_1, \dots, x_m), \text{Rel}(x_1, \dots, x_m) \implies \mathcal{P}(x_1, \dots, x_m).$$

Pour cela, on établit autant de propriétés qu'il y a de règles d'inférences sur la relation `Rel`. Pour chacune de ces propriétés, on a une hypothèse d'induction pour chaque prémisses de la forme `Rel(...)`.

Exemple 12 (Induction sur la relation `le`). Pour prouver une propriété

$$\forall (n, k) \in \text{nat} * \text{nat}, \text{le}(n, k) \implies \mathcal{P}(n, k),$$

il suffit d'établir *deux* propriétés :

1. $\forall n, \mathcal{P}(n, n)$;
2. pour tout (n, k) , montrer $\mathcal{P}(n, S\ k)$ en supposant $\mathcal{P}(n, k)$.

Exemple 13. Supposons que l'on ait une fonction ayant pour signature `sort : nlist → nlist` qui trie une `nlist`. On souhaite démontrer la propriété :

$$\forall \ell \in \text{nlist}, \text{triée}(\ell) \implies \text{sort}(\ell) = \ell.$$

On considère deux approches pour la démonstration : par induction sur ℓ et par induction sur la relation `triée`.

1. par induction sur la liste ℓ , il y a *deux* cas à traiter :
 - ▷ montrer que $\text{triée}(\text{Nil}) \implies \text{sort}(\text{Nil}) = \text{Nil}$,
 - ▷ montrer que :

$$\text{triée}(\text{Cons}(n, \ell)) \implies \text{sort}(\text{Cons}(n, \ell)) = \text{Cons}(n, \ell);$$

2. par induction sur la relation `triée`(ℓ), il y a *trois* cas à traiter :

- ▷ montrer $\text{sort}(\text{Nil}) = \text{Nil}$,
- ▷ montrer $\text{sort}(\text{Cons}(n, \text{Nil})) = \text{Cons}(n, \text{Nil})$,
- ▷ montrer $\text{sort}(\text{Cons}(x, \text{Cons}(y, \ell))) = \text{Cons}(x, \text{Cons}(y, \ell))$,
en supposant :
 - $\text{triée}(\text{Cons}(y, \ell))$ et $\mathcal{P}(\text{Cons}(y, \ell))$, pour la première prémisse ;
 - $\text{le}(x, y)$, pour la seconde prémisse.