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 :

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

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 $\mathtt{t}$ 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* $\mathtt{t}$ est la donnée de k constructeurs $\mathtt{C}_1, \ldots, \mathtt{C}_k$, où chaque $\mathtt{C}_i$ a pour argument un n_i-uplet dont le type est $\mathsf{u}_1^i * \mathsf{u}_2^i * \cdots * \mathsf{u}_{n_i}^i$. L'opération « $*$ » représente le produit cartésien, avec une notation « à la OCaml ». De plus, chaque u_j^i est, soit $\mathtt{t}$, 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 $X = 1 + \mathsf{nat} * X$.
- ▷ Le type t option est alors défini par $1 + \mathtt{t}$.

Ces ensembles définis inductivement nous intéresse 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 $\forall x \in \mathsf{nat}, \mathscr{P}(x)$, il suffit de prouver *deux* choses (parce que l'on a deux constructeurs à l'ensemble nat) :

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

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

> On montre $\mathscr{P}(\mathtt{C}_i(x_1, \ldots, x_n))$ avec les hypothèses d'inductions $\mathscr{P}(x_j)$ lorsque $\mathtt{u}^i_j = \mathtt{t}$.

Exemple 5. Avec le type $\mathtt{t}_2$ 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 $\mathscr{P}(x_1)$ et $\mathscr{P}(x_3)$ comme hypothèses d'induction, et on montre $\mathscr{P}(\mathtt{N2}(x_1, k, x_3))$, où l'on se donne $k \in \mathsf{nat}$.

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

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

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

1. Cas $\mathtt{Nil}$: $\forall k \in \mathsf{nat}, \mathtt{size}(\mathtt{red}\ k\ \mathtt{Nil}) \le \mathtt{size}\ \mathtt{Nil}$;
2. Cas $\mathtt{Cons}(x, \ell')$: on suppose

$$\forall k \in \mathsf{nat}, \mathtt{size}(\mathtt{red}\ k\ \ell) \le \mathtt{size}\ \ell,$$

et on veut montrer que

$$\forall k \in \mathsf{nat}, \mathtt{size}(\mathtt{red}\ k\ \mathtt{Cons}(x, \ell')) \le \mathtt{size}\ \mathtt{Cons}(x, \ell'),$$

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

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 $\mathsf{le} \subseteq \mathsf{nat} * \mathsf{nat}$ est une relation binaire. Cette relation représente $\le$, « *lesser than or equal to* » en anglais.

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

Pour définir cette relation, on peut écrire :

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

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

mais, on écrira plutôt :

> Soit $\mathsf{le} \subseteq \mathsf{nat} * \mathsf{nat}$ la relation définie (inductivement) à partir des règles d'inférence suivantes :
>
> $$\dfrac{}{\mathsf{le}(n, n)}\ \mathscr{L}_1 \qquad \dfrac{\mathsf{le}(n, k)}{\mathsf{le}(n, \mathtt{S}\ k)}\ \mathscr{L}_2.$$

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

▷ Les *métavariables* 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 métavariables*) avec des axiomes aux feuilles.

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

$$\dfrac{\dfrac{\dfrac{}{\mathsf{le}(2, 2)}\ \mathscr{L}_1}{\mathsf{le}(2, 3)}\ \mathscr{L}_2}{\mathsf{le}(2, 4)}\ \mathscr{L}_2$$

Exemple 8. On souhaite définir une relation $\mathsf{triée}$ sur nlist. Pour cela, on pose les trois règles ci-dessous :

$$\dfrac{}{\mathsf{triée}\ \mathtt{Nil}}\ \mathscr{T}_1 \qquad \dfrac{}{\mathsf{triée}\ \mathtt{Cons}(x, \mathtt{Nil})}\ \mathscr{T}_2,$$

$$\dfrac{\mathsf{le}(x, y) \qquad \mathsf{triée}\ \mathtt{Cons}(x, \mathtt{Nil})}{\mathsf{triée}\ \mathtt{Cons}(x, \mathtt{Cons}(y, \ell))}\ \mathscr{T}_3.$$

Ceci permet de dériver, modulo quelques abus de notations, que la liste [1;3;4] est triée :

$$\dfrac{\dfrac{\dfrac{\dfrac{}{1 \le 1}\,\mathscr{L}_1}{1 \le 2}\,\mathscr{L}_2}{1 \le 3}\,\mathscr{L}_2 \quad \dfrac{\dfrac{\dfrac{}{3 \le 3}\,\mathscr{L}_1}{3 \le 4}\,\mathscr{L}_2 \quad \dfrac{}{\mathsf{triée}\ \texttt{[4]}}\,\mathscr{R}_2}{\mathsf{triée}\ \texttt{[3;4]}}\,\mathscr{R}_3}{\mathsf{triée}\ \texttt{[1;3;4]}}\,\mathscr{R}_3 .$$

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 $\mathsf{mem} \subseteq \mathsf{nat} * \mathsf{nlist}$ par les règles d'inférences :

$$\dfrac{}{\mathsf{mem}(k, \mathtt{Cons}(k, \ell))}\,\mathscr{M}_1 \qquad \dfrac{\mathsf{mem}(k, \ell)}{\mathsf{mem}(k, \mathtt{Cons}(x, \ell))}\,\mathscr{M}_2 .$$

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

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

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 « $\neg\mathsf{r}(\dots)$ ». Les règles ne peuvent qu'être « constructive », donc pas de négation.

Exemple 10. On définit la relation $\mathsf{ne} \subseteq \mathsf{nat} * \mathsf{nat}$ 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 :

$$\frac{}{\mathsf{ne}(\mathtt{Z}, \mathtt{S}\ k)}\ \mathcal{N}_1 \qquad \frac{}{\mathsf{ne}(\mathtt{S}\ n, \mathtt{Z})}\ \mathcal{N}_2 \qquad \frac{\mathsf{ne}(n, k)}{\mathsf{ne}(\mathtt{S}\ n, \mathtt{S}\ k)}\ \mathcal{N}_3.$$

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 :

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

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

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

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

$$\frac{H_1 \qquad \cdots \qquad H_n}{\mathsf{Rel}(x_1, \ldots, x_m)}\ \mathcal{R}_i,$$

où chaque H_j est :

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

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

4 Preuves par induction sur une relation inductive.

On souhaite établir une propriété de la forme

$$\forall(x_1,\ldots,x_m),\ \mathsf{Rel}(x_1,\ldots,x_m) \implies \mathscr{P}(x_1,\ldots,x_m).$$

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 prémisse de la forme $\mathsf{Rel}(\ldots)$.

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

$$\forall(n,k) \in \mathsf{nat} * \mathsf{nat},\quad \mathsf{le}(n,k) \implies \mathscr{P}(n,k),$$

il suffit d'établir *deux* propriétés :

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

Exemple 13. Supposons que l'on ait une fonction ayant pour signature $\mathtt{sort} : \mathsf{nlist} \to \mathsf{nlist}$ qui trie une nlist. On souhaite démontrer la propriété :

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

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 $\mathsf{triée}(\mathtt{Nil}) \implies \mathtt{sort}(\mathtt{Nil}) = \mathtt{Nil}$,
   - ▷ montrer que :

     $$\mathsf{triée}(\mathtt{Cons}(n,\ell)) \implies \mathtt{sort}(\mathtt{Cons}(n,\ell)) = \mathtt{Cons}(n,\ell);$$
2. par induction sur la relation $\mathsf{triée}(\ell)$, il y a *trois* cas à traiter :

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