Où sont passés les types inductifs ?

- Ils sont partout !

```
Inductive nat : Set :=
  | O : nat
  | S : nat → nat
```

- Une définition inductive c'est :
  - Un type
  - Des constructeurs
- Les constructeurs :
  - ne se recouvrent pas → `discriminate`
  - sont suffisants pour construire toutes valeurs de la définition inductive → `destruct` / `inversion`
  - sont injectifs → `injection`
- De forts liens avec les théorèmes de plus petit point fixe !
  - Coq nous génère donc aussi un principe d'induction ! → `induction`

Les types inductifs sont vraiment partout !

- `Print and.`

```
Inductive and (A B : Prop) : Prop :=
   conj : A → B → A ∧ B
```

- `Print or.` Essayez de trouver puis vérifiez !
- `Print not.`

```
not = fun A : Prop => A → False
  : Prop → Prop
```

- `Print iff.`

```
iff =
  fun A B : Prop => (A → B) ∧ (B → A)
    : Prop → Prop → Prop
```

Tactiques pour manipuler les types inductifs

- `destruct x` pour une analyse par cas de `x`
- `discriminate` pour utiliser la disjonction des constructeurs
- `injection H` pour utiliser l'injection des constructeurs dans `H`
- `inversion H` $\simeq$ `destruct H`, mais plus pour les prédicats
- `induction H` $\simeq$ `destruct H` avec hypothèse d'induction
- `constructor` pour appliquer un constructeur du type de la conclusion

A vos tactiques !

Interprétation de Brower-Heyting-Kolmoroff

On associe à chaque formule l'espace éventuellement vide de ses preuves.

Sémantique de Heyting (cadre général du calcul des prédicats) :

- Une preuve de $A \rightarrow B$ est une fonction qui à toute preuve de A associe une preuve de B.
- Une preuve de $A \wedge B$ est un couple composé d'une preuve de A et d'une preuve de B.
- L'ensemble des preuves de la proposition absurde *False* est vide.
- Une preuve de $A \vee B$ est un couple de la forme (i, p) où si i vaut 1, alors p est une preuve de A, et si i vaut 2, alors p est une preuve de B.
- Une preuve de $\forall x.A$ est une fonction qui à tout objet x associe une preuve de A.
- Une preuve de $\exists x.A$ est un couple (t, p) où t est un objet (appelé **témoin**) et p est une preuve de la formule $A[x \leftarrow t]$.
  Montrer l'existence en logique constructive c'est exhiber un témoin.

En conclusion

- **Ce qu'on a vu**
  - Un langage fonctionnel pur, Gallina
  - Un mécanisme pour définir des types (et des prédicats) inductifs très puissant
  - Un langage de tactiques pour écrire des preuves
- **Ce qu'on va voir**
  - Notions avancées de COQ
  - Prouveur automatique
  - Preuve de sémantique
  - et d'autres outils encore !
- **Modalités de validation**
  - Cette UE = 22h30 de présence
  - Cette UE = 6 ECTS
  - "La valeur d'un crédit représente environ 25 à 30 heures de travail"
  - 1 DM + 1 Projet + 1 CC

Plus aller plus loin

- **Idéal pour commencer (progressif, très reconnu et très utilisé)**
  Software Foundations, Benjamin Pierce et al.
  `https://softwarefoundations.cis.upenn.edu/`
- **Plus poussé**
  Certified Programming with Dependent Types, Adam Chlipala
  `http://adam.chlipala.net/cpdt/`
- **Coq' Art**, Yves Bertot et Pierre Castéran
  `https://www.labri.fr/perso/casteran/CoqArt/`
- **La documentation de Coq**
  `https://coq.inria.fr/refman/`
  `https://coq.inria.fr/refman/command-index.html`
  `https://coq.inria.fr/refman/tactic-index.html`