Assistants de preuve

MPRI - 2007-2008

1- Du Calcul des Constructions au Calcul des Constructions Inductives

La structure du Calcul des Constructions Inductives

Calcul des Constructions Inductives (version imprédicative – Coq < 8.0)

=

Calcul des Constructions sur `Prop` et `Type`

pour la logique d'ordre supérieur

pour les types imprédicatifs (historiquement)

+

copie du Système F dans `Set`

pour l'extraction de programme

+

hiérarchie de type $\texttt{Type} = \texttt{Type}_1 : \texttt{Type}_2 : \texttt{Type}_3 \ldots$

pour l'expressivité logique

+

types inductifs

pour des formalisations et des types de données plus « naturels »

pour un surcroît d'expressivité calculatoire

pour un surcroît d'expressivité logique

La structure du Calcul des Constructions Inductives

Calcul des Constructions Inductives (version prédicative – Coq ≥ 8.0)

=

Calcul des Constructions sur `Prop` et `Type`

pour la logique d'ordre supérieur

pour les types imprédicatifs (historiquement)

+

hiérarchie de type `Set`: `Type`$=$ `Type`$_1$: `Type`$_2$: `Type`$_3$ ...

pour l'extraction de programme

pour l'expressivité logique

+

types inductifs

pour des formalisations et des types de données plus « naturels »

pour un surcroît d'expressivité calculatoire

pour un surcroît d'expressivité logique

Le Système F vu comme logique propositionnelle

système F "propositionnel"

$\Pi A : \texttt{Prop}.\, B$	(Coq: `forall A:Prop, B`)
$A \to B$	(Coq: `A -> B`)
Ex: $\Pi C : \texttt{Prop}.\, (A \to B \to C) \to C$	(la conjonction $A \wedge B$)

et l'abstraction, l'application, et les variables
implémentent les règles d'inférence de la logique

Le Système F vu comme calcul

système F "calculatoire"

ΠA : `Prop`. B	(Coq$_I$: `forall A:Set, B`)
$A \to B$	(Coq: `A -> B`)
Ex: ΠC : `Prop`. $(A \to B \to C) \to C$	(le produit $A \times B$)
λA : `Prop`. t	(Coq: `fun A:Set => t`)
λx : A. t	(Coq: `fun x:A => t`)
$t\, A$	(Coq: `t A`)
$t\, u$	(Coq: `t u`)
x	(Coq: `t u`)
Ex: λC : `Prop`. λf : $A \to B \to C$. $f\, a\, b$	(la paire (a, b))

Du Système F au Calcul des Constructions

Intérêt: pouvoir parler de la couche calculatoire du système F à l'intérieur de la couche logique du système F.

Ajout des produits de la forme

$$A \to \texttt{Prop}$$

Le Calcul des Constructions

Intérêt: pouvoir parler de la couche calculatoire du système F à l'intérieur de la couche logique du système F.

Ajout des produits de la forme

$$A \to \texttt{Prop}$$

... et ajout du polymorphisme d'ordre supérieur, produits de la forme

$$(\texttt{Prop} \to \texttt{Prop}) \to \texttt{Prop}$$

$\hookrightarrow$ Le Calcul des Constructions fait de la correspondance de Curry-Howard-de Bruijn une identité.

$\hookrightarrow$ Une logique originale capable de "parler" des preuves.

Le Calcul des Constructions: un système "complexe"

Les couches calculatoires et logiques sont superposées

$\hookrightarrow$ introduction de `Set` au côté de `Prop` (Paulin-Mohring): duplication de la couche système F avec cette intention:

- dans `Prop`, la nature des preuves n'importe pas; le principe d'indiscernabilité ($\forall P : Prop \forall pq : P.\, p = q$) est admissible.

- dans `Set`, les objets sont discriminables; par exemple, dans le type des booléens, `true` $\neq$ `false` sera admissible

Deux niveaux calculatoires dans le sous-système F_ω: système F en dessous, λ-calcul simplement typé au dessus.

Lequel préférer ? Par exemple, HOL (= la théorie des types simples de Church) place le calcul dans le niveau du haut (mais pas de conscience de Curry-Howard-de Bruijn à l'époque).

Logique qui reste malgré tout très faible

`Prop` peut être interprété par un type booléen : toute proposition prouvable est interprété par `true` et toute proposition prouvablement fausse par `false`.

$\hookrightarrow$ bien qu'on puisse coder les entiers, on ne peut pas prouver $0 \neq 1$

Les inconvénients du codage polymorphe des définitions inductives

Cas du codage imprédicatif

- $0 \neq 1$ n'est pas dérivable
- l'induction n'est pas « directement » dérivable (on n'a que le récurseur)

Cas du codage prédicatif dans le calcul avec univers

- OK au niveau expressivité (on a $0 \neq 1$ et une induction « indirecte »)

Mais dans tous les cas

- pas de prédécesseur en une étape
- pas "naturel"
- difficile d'écrire des outils automatiques capables de distinguer entre les constructeurs de types inductifs et termes arbitraires

$\hookrightarrow$ Mise en place de types inductifs primitifs à la Martin-Löf

Le Calcul des Constructions Inductives (Coq $\geq$ 5.6)

(Coquand – Paulin-Mohring 1989)

Un schéma général de constructions de types inductifs

- critère de positivité (pour garantir l'existence d'un plus petit ensemble contenant un ensemble donné de constructeurs)
- décomposition des récurseurs en un opérateur d'analyse de cas (`match-with`) et un combinateur pur de point fixe (`fix`)
- critères de terminaison des points fixes

Des conditions spécifiques d'élimination selon les sortes

- respecte le côté calculatoire de `Set` et `Type` et le côté purement logique de `Prop`
- évite les paradoxes liés à l'imprédicativité

Quelques conséquences

- $0 \neq 1$ dérivable
- principe d'induction dérivable
- axiome du choix intuitionniste dérivable

Le Calcul des Constructions Inductives prédicatif (Coq $\geq$ 8.0)

- La sorte `Set` rejoint la hiérarchie des types (`Set` $=$ `Type`$_0$)

- Plus de différence (sauf historique) entre mettre les types de donnée dans `Set` ou dans `Type`

- Une approche qui rejoint celle de HOL (mais avec types inductifs et hiérarchie d'univers)

- Compatible avec les axiomes mathématiques standard: logique classique, axiome du choix classique, extensionnalité (justifiés par plongement dans la théorie des ensembles)

Types inductifs : l'exemple des booléens

en Objective Caml

```
type bool =
| true
| false
```

dans la couche Système F de Coq

```
Definition bool := forall P:Prop, P -> P -> P.
Definition true :  bool := fun P:Prop => fun H1 H2 => H1.
Definition false :  bool := fun P:Prop => fun H1 H2 => H1.
```

comme type inductif primitif du Calcul des Constructions Inductives

```
Inductive bool :  Type
| true :  bool
| false :  bool.
```

Types inductifs : exemples

en Objective Caml

```
type bool =
| true
| false

let andb b1 b2 =
match b1 with
| true -> b2
| false -> false


 type nat =
| O
| S of nat

let rec fact n =
match n with
| O -> S(O)
| S(p) -> n * fact p
```

dans le CCI, syntaxe Coq

```
Inductive bool :  Type
| true :  bool
| false :  bool.

Definition andb b1 b2 :=
match b1 with
| true => b2
| false => false
end.

 Inductive nat =
| O : nat
| S : nat -> nat.

Fixpoint fact n :=
match n with
| O => S O
| S p -> n * fact p
end.
```

Typage des types inductifs (première étape) : l'exemple des booléens

```
Inductive bool : Type :=
| true : bool
| false : bool.
```

Une telle déclaration de types inductifs définit :

- un type

$$\frac{}{\Gamma \vdash bool : \texttt{Type}}$$

- un ensemble de règles d'introduction pour ce type : les constructeurs

$$\frac{}{\Gamma \vdash true : bool} \qquad \frac{}{\Gamma \vdash false : bool}$$

- une règle d'élimination, sous la forme d'un opérateur de filtrage

$$\frac{\Gamma \vdash t : bool \qquad \Gamma \vdash A : s \qquad \Gamma \vdash t_1 : A \qquad \Gamma \vdash t_2 : A}{\Gamma \vdash (\texttt{match}\ t\ \texttt{with}\ true\ \texttt{=>}\ t_1 \mid false\ \texttt{=>}\ t_2\ \texttt{end}) : A}$$

- des règles de réduction, dites règles de ι-réduction

$$(\texttt{match}\ true\ \texttt{with}\ true\ \texttt{=>}\ t_1 \mid false\ \texttt{=>}\ t_2\ \texttt{end}) \quad \rightarrow_\iota \quad t_1$$
$$(\texttt{match}\ false\ \texttt{with}\ true\ \texttt{=>}\ t_1 \mid false\ \texttt{=>}\ t_2\ \texttt{end}) \quad \rightarrow_\iota \quad t_2$$

Typage des types inductifs (première étape) : l'exemple des booléens

```
Inductive bool : Type :=
| true : bool
| false : bool.
```

Une telle déclaration de types inductifs définit :

- un type

$$\frac{}{\Gamma \vdash \mathit{bool} : \texttt{Type}}$$

- un ensemble de règles d'introduction pour ce type : les constructeurs

$$\frac{}{\Gamma \vdash \mathit{true} : \mathit{bool}} \qquad \frac{}{\Gamma \vdash \mathit{false} : \mathit{bool}}$$

- une règle d'élimination, sous la forme d'un opérateur de filtrage (avec explicitation du type des branches)

$$\frac{\Gamma \vdash t : \mathit{bool} \qquad \Gamma \vdash A : s \qquad \Gamma \vdash t_1 : A \qquad \Gamma \vdash t_2 : A}{\Gamma \vdash (\texttt{match}\ t\ \texttt{return}\ A\ \texttt{with}\ \mathit{true}\ \texttt{=>}\ t_1 \mid \mathit{false}\ \texttt{=>}\ t_2\ \texttt{end}) : A}$$

- des règles de ι-réduction

$$(\texttt{match}\ \mathit{true}\ \texttt{return}\ A\ \texttt{with}\ \mathit{true}\ \texttt{=>}\ t_1 \mid \mathit{false}\ \texttt{=>}\ t_2\ \texttt{end}) \quad \rightarrow_\iota \quad t_1$$
$$(\texttt{match}\ \mathit{false}\ \texttt{return}\ A\ \texttt{with}\ \mathit{true}\ \texttt{=>}\ t_1 \mid \mathit{false}\ \texttt{=>}\ t_2\ \texttt{end}) \quad \rightarrow_\iota \quad t_2$$

Types inductifs avec paramètres : l'exemple de la disjonction

en Objective Caml

```
type ('a,'b) or =
| or_introl of 'a
| or_intror of 'b
```

dans le CCI, syntaxe Coq

```
Inductive or (A:Prop) (B:Prop) :  Prop :=
| or_introl :  A -> or A B
| or_intror :  B -> or A B.
```

Types inductifs avec paramètres : l'exemple de la disjonction

```
Inductive or (A:Prop) (B:Prop) : Prop :=
| or_introl : A -> or A B
| or_intror : B -> or A B.
```

qui définit

- une famille de type

$$\frac{}{\Gamma \vdash or : \texttt{Prop} \to \texttt{Prop} \to \texttt{Prop}}$$

- un ensemble de règles d'introduction pour les types de cette famille

$$\frac{\Gamma \vdash A : \texttt{Prop} \qquad \Gamma \vdash B : \texttt{Prop} \qquad \Gamma \vdash p : A}{\Gamma \vdash or\_introl_{A,B}\ p : or\ A\ B} \qquad \frac{\Gamma \vdash A : \texttt{Prop} \qquad \Gamma \vdash B : \texttt{Prop} \qquad \Gamma \vdash q : B}{\Gamma \vdash or\_intror_{A,B}\ q : or\ A\ B}$$

- une règle d'élimination

$$\frac{\Gamma \vdash t : or\ A\ B \qquad \Gamma \vdash C : \texttt{Prop} \qquad \Gamma, p : A \vdash t_1 : C \qquad \Gamma, q : B \vdash t_2 : C}{\Gamma \vdash (\texttt{match}\ t\ \texttt{return}\ C\ \texttt{with}\ or\_introl_{A,B}\ p\ \texttt{=>}\ t_1 \mid or\_intror_{A,B}\ q\ \texttt{=>}\ t_2\ \texttt{end}) : C}$$

- des règles de ι-réduction

$$(\texttt{match}\ or\_introl_{A,B}\ t\ \texttt{return}\ C\ \texttt{with}\ or\_introl_{A,B}\ p\ \texttt{=>}\ t_1 \mid or\_intror_{A,B}\ q\ \texttt{=>}\ t_2\ \texttt{end}) \quad \to_\iota \quad t_1[t/p]$$
$$(\texttt{match}\ or\_intror_{A,B}\ u\ \texttt{return}\ C\ \texttt{with}\ or\_introl_{A,B}\ p\ \texttt{=>}\ t_1 \mid or\_intror_{A,B}\ q\ \texttt{=>}\ t_2\ \texttt{end}) \quad \to_\iota \quad t_2[u/q]$$

Remarque sur la syntaxe des types inductifs avec paramètres

En pratique, Coq définit les constructeurs sous leur forme curryfiée (contrairement à Objective Caml où les constructeurs sont nécessairement appliqués). Ainsi, les règles d'introduction pour la disjonction implantées par Coq sont, en fait:

$$\frac{}{\Gamma \vdash or\_introl : \forall A\, B : \texttt{Prop}, A \to or\ A\ B} \qquad \frac{}{\Gamma \vdash or\_intror : \forall A\, B : \texttt{Prop}, B \to or\ A\ B}$$

À l'inverse, les paramètres des constructeurs sont omis dans la syntaxe du `match` de Coq, ceci parce que c'est une information redondante déjà présente, implicitement, dans le type de l'argument filtré. Ainsi, la règle d'élimination s'écrit, *en Coq*:

$$\frac{\Gamma \vdash t : or\ A\ B \qquad \Gamma \vdash C : \texttt{Prop} \qquad \Gamma, p : A \vdash t_1 : C \qquad \Gamma, q : B \vdash t_2 : C}{\Gamma \vdash (\texttt{match}\ t\ \texttt{return}\ C\ \texttt{with}\ or\_introl\ p\ \texttt{=>}\ t_1 \mid or\_intror\ q\ \texttt{=>}\ t_2\ \texttt{end}) : C}$$

et les règles de ι-réduction s'écrivent, *en Coq*::

$$(\texttt{match}\ or\_introl\ A\ B\ t\ \texttt{return}\ C\ \texttt{with}\ or\_introl\ p\ \texttt{=>}\ t_1 \mid or\_intror\ q\ \texttt{=>}\ t_2\ \texttt{end}) \quad \to_\iota \quad t_1[t/p]$$
$$(\texttt{match}\ or\_intror\ A\ B\ u\ \texttt{return}\ C\ \texttt{with}\ or\_introl\ p\ \texttt{=>}\ t_1 \mid or\_intror\ q\ \texttt{=>}\ t_2\ \texttt{end}) \quad \to_\iota \quad t_2[u/q]$$

Dans la suite, on utilisera l'une ou l'autre notation...

Types inductifs (élimination dépendante) : l'exemple des booléens

```
Inductive bool : Type :=
| true : bool
| false : bool.
```

- la règle d'élimination dans toute sa généralité

$$\frac{\Gamma \vdash t : bool \qquad \Gamma, x : bool \vdash A(x) : s \qquad \Gamma \vdash t_1 : A(true) \qquad \Gamma \vdash t_2 : A(false)}{\Gamma \vdash (\texttt{match}\ t\ \texttt{as}\ x\ \texttt{return}\ A(x)\ \texttt{with}\ true\ \texttt{=>}\ t_1 \mid false\ \texttt{=>}\ t_2\ \texttt{end}) : A(t)}$$

- règles de réduction

$$\begin{array}{l}(\texttt{match}\ true\ \texttt{as}\ x\ \texttt{return}\ A(x)\ \texttt{with}\ true\ \texttt{=>}\ t_1 \mid false\ \texttt{=>}\ t_2\ \texttt{end}) \quad \rightarrow_\iota \quad t_1 \\ (\texttt{match}\ false\ \texttt{as}\ x\ \texttt{return}\ A(x)\ \texttt{with}\ true\ \texttt{=>}\ t_1 \mid false\ \texttt{=>}\ t_2\ \texttt{end}) \quad \rightarrow_\iota \quad t_2\end{array}$$

On vérifie en particulier la préservation du type par réduction.

Ce schéma permet de dériver l'analyse de cas sur les booléens

$$\begin{array}{l}\lambda P : bool \rightarrow \texttt{Prop}.\, \lambda H_{true} : P(true).\, \lambda H_{false} : P(false).\, \lambda x : bool. \\ \quad \texttt{match}\ x\ \texttt{as}\ y\ \texttt{return}\ P(y)\ \texttt{with}\ true\ \texttt{=>}\ H_{true} \mid false\ \texttt{=>}\ H_{false}\ \texttt{end}\end{array}$$

est une preuve de

$$\forall P : bool \rightarrow \texttt{Prop}.\, P(true) \rightarrow P(false) \rightarrow \forall x : bool.\, P(x)$$

Types inductifs (élimination dépendante) : l'exemple des booléens

La même en syntaxe Coq:

```
Definition bool_case
  : forall P : bool -> Prop, P true -> P false -> forall b : bool, P b
  : = fun (P : bool -> Prop) (Htrue : P true) (Hfalse : P false) (b : bool) =>
      match b as b0 return (P b0) with
      | true => Htrue
      | false => Hfalse
      end.
```

Types inductifs (élimination dépendante) : l'exemple des booléens

L'élimination dépendante permet aussi construire des fonctions dans des types produits

$$\lambda A : bool \to \texttt{Type}.\ \lambda H_{true} : A(true).\ \lambda H_{false} : A(false).\ \lambda x : bool.$$
$$\texttt{match}\ x\ \texttt{as}\ y\ \texttt{return}\ A(y)\ \texttt{with}\ true\ \texttt{=>}\ H_{true} \mid false\ \texttt{=>}\ H_{false}\ \texttt{end}$$

est un combinateur de type

$$\forall A : bool \to \texttt{Type}.\ A(true) \to A(false) \to \forall x : bool.\ A(x)$$

Il permet de construire des fonctions dans le type $\Pi x : bool.\ A(x)$.

Types inductifs avec preuves dépendantes : l'exemple de la disjonction

```
Inductive or (A:Prop) (B:Prop) : Prop :=
| or_introl : A -> or A B
| or_intror : B -> or A B.
```

- la règle d'élimination dans toute sa généralité

$$\frac{\Gamma \vdash t : or\ A\ B \qquad \Gamma, x : or\ A\ B \vdash C(x) : \texttt{Prop} \qquad \Gamma, p : A \vdash t_1 : C\,(or\_introl\ p) \qquad \Gamma, q : B \vdash t_2 : C\,(or\_intror\ q)}{\Gamma \vdash (\texttt{match}\ t\ \texttt{as}\ x\ \texttt{return}\ C(x)\ \texttt{with}\ or\_introl\ p\ \texttt{=>}\ t_1 \mid or\_intror\ q\ \texttt{=>}\ t_2\ \texttt{end}) : C(t)}$$

permet de raisonner par analyse de cas sur la forme d'une preuve. Ainsi,

$$\lambda P : or\ A\ B \to \texttt{Type}.\, \lambda H_l : (\forall p : A.\, P\,(or\_introl\ p)).\, \lambda H_r : (\forall q : B.\, P\,(or\_intror\ q)).\, \lambda x : or\ A\ B.$$
$$\texttt{match}\ x\ \texttt{as}\ y\ \texttt{return}\ P(y)\ \texttt{with}\ or\_introl\ p\ \texttt{=>}\ H_l\ p \mid or\_intror\ q\ \texttt{=>}\ H_r\ q\ \texttt{end}$$

est une preuve de

$$\forall P : (or\ A\ B) \to \texttt{Prop}.\, (\forall p : A,\, P(or\_introl\ p)) \to (\forall q : B,\, P(or\_intror\ q)) \to \forall x : (or\ A\ B).\, P(x)$$

Types inductifs récursifs : l'exemple des entiers

```
Inductive nat : Type :=
| O : nat
| S : nat -> nat.
```

qui définit

- un type

$$\frac{}{\Gamma \vdash nat : \texttt{Type}}$$

- un ensemble de règles d'introduction pour ce type : les constructeurs

$$\frac{}{\Gamma \vdash O : nat} \qquad \frac{\vdash n : nat}{\Gamma \vdash S\ n : nat}$$

- une règle d'élimination (opérateur de filtrage à résultat dépendant du filtre)

$$\frac{\Gamma \vdash t : nat \qquad \Gamma, x : nat \vdash A(x) : s \qquad \Gamma \vdash t_1 : A(O) \qquad \Gamma, n : nat \vdash t_2 : A(S\ n)}{\Gamma \vdash (\texttt{match}\ t\ \texttt{as}\ x\ \texttt{return}\ A(x)\ \texttt{with}\ O\ \texttt{=>}\ t_1 \mid S\ n\ \texttt{=>}\ t_2\ \texttt{end}) : A(t)}$$

- des règles de réduction préservant le typage (ι-réduction)

$$(\texttt{match}\ O\ \texttt{as}\ x\ \texttt{return}\ A(x)\ \texttt{with}\ O\ \texttt{=>}\ t_1 \mid S\ n\ \texttt{=>}\ t_2\ \texttt{end}) \quad \rightarrow_\iota \quad t_1$$
$$(\texttt{match}\ S\ n\ \texttt{as}\ x\ \texttt{return}\ A(x)\ \texttt{with}\ O\ \texttt{=>}\ t_1 \mid S\ n\ \texttt{=>}\ t_2\ \texttt{end}) \quad \rightarrow_\iota \quad t_2$$

Types inductifs récursifs : l'exemple des entiers

On a analyse de cas et construction par cas : le terme

$$
\begin{array}{l}
\lambda P : nat \to s. \\
\lambda H_O : P(O). \\
\lambda H_S : \forall m : nat.\, P(S\ m). \\
\lambda n : nat. \\
\ \texttt{match}\ n\ \texttt{as}\ y\ \texttt{return}\ P(y)\ \texttt{with} \\
\ \ |\ O\ \texttt{=>}\ H_O \\
\ \ |\ S\ m\ \texttt{=>}\ H_S\ m \\
\ \texttt{end}
\end{array}
$$

est une preuve de

$$\forall P : nat \to s.\, P(O) \to (\forall m : nat.\, P(S\ m)) \to \forall n : nat.\, P(n)$$

Types inductifs avec paramètres : l'exemple des listes

```
Inductive list (A:Type) : Type :=
| nil : list A
| cons : A -> list A -> list A.
```

qui définit

- une famille de type

$$\frac{}{\Gamma \vdash list : \texttt{Type} \rightarrow \texttt{Type}}$$

- un ensemble de règles d'introduction pour les types de cette famille

$$\frac{\Gamma \vdash A : \texttt{Type}}{\Gamma \vdash nil_A : list\ A} \qquad \frac{\Gamma \vdash A : \texttt{Type} \qquad \Gamma \vdash a : A \qquad \Gamma \vdash l : listA}{\Gamma \vdash cons_A\ a\ l : list\ A}$$

- une règle d'élimination (opérateur de filtrage à résultat dépendant du filtre)

$$\frac{\Gamma \vdash l : list\ A \qquad \Gamma, x : list\ A \vdash C(x) : s \qquad \Gamma \vdash t_1 : C(nil) \qquad \Gamma, a : A, l : list\ A \vdash t_2 : C(cons_A\ a\ l)}{\Gamma \vdash (\texttt{match}\ l\ \texttt{as}\ x\ \texttt{return}\ C(x)\ \texttt{with}\ nil\ \texttt{=>}\ t_1 \mid cons\ a\ l\ \texttt{=>}\ t_2\ \texttt{end}) : C(l)}$$

- des règles de réduction qui préservent le typage (ι-réduction)

$$(\texttt{match}\ nil_A\ \texttt{as}\ x\ \texttt{return}\ C(x)\ \texttt{with}\ nil\ \texttt{=>}\ t_1 \mid cons\ a\ l\ \texttt{=>}\ t_2\ \texttt{end}) \rightarrow_\iota t_1$$
$$(\texttt{match}\ cons_A\ a'\ l'\ \texttt{as}\ x\ \texttt{return}\ C(x)\ \texttt{with}\ nil\ p\ \texttt{=>}\ t_1 \mid cons\ a\ l\ \texttt{=>}\ t_2\ \texttt{end}) \rightarrow_\iota t_2[a'/a; l'/l]$$

Types inductifs avec paramètres de famille et paramètres de prédicat
l'exemple des vecteurs avec leur taille

```
Inductive vect (A:Type) : nat -> Type :=
| niln : vect A O
| consn : A -> forall n:nat, vect A n -> vect A (S n).
```

qui définit

- une famille de types-prédicats

$$\frac{}{\Gamma \vdash \mathit{vect} : \mathtt{Type} \to \mathit{nat} \to \mathtt{Type}}$$

- un ensemble de règles d'introduction pour les types de cette famille

$$\frac{\Gamma \vdash A : \mathtt{Type}}{\Gamma \vdash \mathit{niln}_A : \mathit{vect}\ A\ O} \qquad \frac{\Gamma \vdash A : \mathtt{Type} \quad \Gamma \vdash a : A \quad \Gamma \vdash n : \mathit{nat} \quad \Gamma \vdash l : \mathit{vect}\ A\ n}{\Gamma \vdash \mathit{consn}_A\ a\ n\ l : \mathit{list}\ A\ (S\ n)}$$

- une règle d'élimination (opérateur de filtrage à résultat dépendant du filtre)

$$\frac{\begin{array}{ll}\Gamma \vdash v : \mathit{vect}\ A\ n & \Gamma \vdash t_1 : C(O, \mathit{niln}_A) \\ \Gamma, m : \mathit{nat}, x : \mathit{vect}\ A\ m \vdash C(m,x) : s & \Gamma, a : A, n : \mathit{nat}, l : \mathit{vect}\ A\ n \vdash t_2 : C(S\ n, \mathit{consn}_A\ a\ n\ l)\end{array}}{\Gamma \vdash (\mathtt{match}\ v\ \mathtt{as}\ x\ \mathtt{in}\ \mathit{vect}\ \_\ p\ \mathtt{return}\ C(x)\ \mathtt{with}\ \mathit{niln}\ \mathtt{=>}\ t_1 \mid \mathit{consn}\ a\ n\ l\ \mathtt{=>}\ t_2\ \mathtt{end}) : C(n,v)}$$

- des règles de réduction qui préservent le typage (ι-réduction)

$$\begin{array}{ll}(\mathtt{match}\ \mathit{niln}_A\ \mathtt{as}\ x\ \mathtt{in}\ \mathit{vect}\ \_\ p\ \mathtt{return}\ C(x,p)\ \mathtt{with}\ \mathit{niln}\ \mathtt{=>}\ t_1 \mid \mathit{consn}\ a\ n\ l\ \mathtt{=>}\ t_2\ \mathtt{end}) & \to_\iota t_1 \\ (\mathtt{match}\ \mathit{consn}_A\ a'\ n'\ l'\ \mathtt{as}\ x\ \mathtt{in}\ \mathit{vect}\ \_\ p\ \mathtt{return}\ C(x,p)\ \mathtt{with}\ \mathit{niln}\ \mathtt{=>}\ t_1 \mid \mathit{consn}\ a\ n\ l\ \mathtt{=>}\ t_2\ \mathtt{end}) & \to_\iota t_2[a'/a; n'/n; l'/l]\end{array}$$