A New Elimination Rule for Coq

B. Barras P. Corbineau B. Grégoire H. Herbelin
J.L. Sacchini

October 28, 2008

Example: head and tail functions in Haskell

```
data List a = nil | cons a (List a)

head :: List a -> a
head (cons x _) = x

tail :: List a -> List a
tail (cons _ xs) = xs
```

- Applying these functions to nil raises an exception

Example: head and tail functions in Agda

```
data Vec ( A : Set ) : Nat -> Set where
  vnil : Vec A 0
  vcons : forall {n} -> A -> Vec A n -> Vec A (S n)

head :: {n : Nat} -> Vec A (S n) -> A
head (cons x _) = x

tail :: {n : Nat} -> Vec A (S n) -> Vec A n
tail (cons _ xs) = xs
```

- No need to consider the `nil` case. Typechecking ensures that these functions cannot be applied to an empty list
- Slogan of programming with dependent types: more precise types

Defining the tail function in Coq: Inversion

```
Definition tail A (n : nat) (v : vector A (S n)) :=
  match v in (vector _ n0) return
        (n0 = S n) -> vector A n with
  | Vnil => fun (H : 0 = S n) =>
      False_rect (vector A n)
       (eq_ind 0 (fun e : nat =>
          match e with 0 => True | S _ => False end)
       I (S n) H)
  | Vcons _ n1 tl => fun (Heq : S n1 = S n) =>
      eq_rect n1 (fun n2 : nat => vector A n2) tl n
        (f_equal
          (fun e : nat => match e with
           | 0 => n1
           | S n2 => n2
           end) Heq)
  end (refl_equal (S n)).
```

Defining the tail function in Coq: a nice solution

```
Definition tail (A : Type) (n : nat) (v : vector A (S n)) :
 match v in (vector _ n0) return
     match n0 with 0 => ID | S n1 => vector A n1 end
 with
 | Vnil => id
 | Vcons _ _ v0 => v0
 end.
```

Problematic

- The elimination rule *loses* information
- Hard to write directly (without tactics)
- Pollution of the reduction (using inversion)
- Hard to reason about such definition

Defining the `tail` function in Coq: future

```
Definition tail A n (v : vec A (S n)) : A :=
  match v with
  | Vnil => _
  | Vcons a (n0:=n) tl => tl
  end.
```

Since constructors are disjoint, $0 \neq \mathtt{S}\, n$. Therefore, v can never reduce to `Vnil`.
Furthermore `tl` has type `vector A n0` (i.e convertible with `vector A n`)

Formal presentation

Inductive types

$\texttt{Inductive}\ I\ \vec{p} : \Delta_I \to s :=$

$\mid C_i : \Pi\Delta_I^i . I\ \vec{p}\ \vec{u^i}$

$\mid \ldots$

$\vec{p}$: parameters

Δ_I: arguments

C_i: constructors

Example

```
Inductive vec (A : Type) : nat -> Type :=
| vnil : vec A 0
| vcons : forall n, A -> vec A n -> vec A (S n)
```

Current elimination rule

$$\frac{\Gamma \vdash v : I\,\vec{q}\,\vec{u} \qquad \Gamma(\vec{y} : \Delta_I[\vec{p} := \vec{q}])(v_0 : I\,\vec{q}\,\vec{y}) \vdash P : s \\ \Gamma(z_i : \Delta_I^i[\vec{p} := \vec{q}]) \vdash t_i : P[\vec{y} := u_I^i[\vec{p} := \vec{q}]][v_0 := C_i\,\vec{z_i}]}{\Gamma \vdash \texttt{match}\ v\ \texttt{as}\ v_0\ \texttt{in}\ I\ \_\ \vec{y}\ \texttt{return}\ P\ \texttt{with} \\ \ldots C_i\ \vec{z_i} \Rightarrow t_i \ldots : P[\vec{y} := \vec{u}][v_0 := x]}$$

Example

```
match v as v0 in vec _ n0 return P with
| vnil => t1
| vcons n' x xs => t2
```

$P[n0 := 0][v0 := vnil]$

$P[n0 := S\,n'][v0 := vcons\,n'\,x\,xs]$

Our proposal

$$v : I\,\vec{p}\,\vec{u}$$

$$C_i\,\vec{z_i} : I\,\vec{p}\,\vec{u_I^i}$$

$$\texttt{match}\ v\ \texttt{as}\ v_0\ \texttt{in}\ I\ \_\ \vec{y}\ \texttt{return}\ P\ \texttt{with}$$
$$\mid C_i\,\vec{z_i} \Rightarrow t_i \ldots$$

- We only need to consider constructors for which $\vec{u}$ can be unified with $\vec{u_I^i}$
- By unification, we mean to find a substitution σ from variables to terms, such that $\vec{u}\sigma = \vec{u_I^i}\sigma$

Unification

Definition

Given two sequences of terms $\vec{u}$ and $\vec{v}$ and a set of variables ζ, a unification problem is to find a substitution σ whose domain is a subset of ζ, such that, $\vec{u}\sigma = \vec{v}\sigma$. We denote this by

$$\zeta \vdash [\vec{u} = \vec{v}] : \sigma$$

- This problem is undecidable
- So, our algorithm can have three possible outcomes
  - Positive success: a σ is found such that $\zeta \vdash [\vec{u} = \vec{v}] : \sigma$
  - Negative success: the terms are not unifiable, denoted by $\zeta \vdash [\vec{u} = \vec{v}] : \perp$
  - Failure: the problem is too difficult
- We use properties of constructors: injectivity and disjointness

Unification rules

$$\frac{x \in \zeta \qquad x \notin FV(v)}{\zeta \vdash [x = v] : \{x \mapsto v\}}\ [VarL] \qquad \frac{x \in \zeta \qquad x \notin FV(v)}{\zeta \vdash [v = x] : \{x \mapsto v\}}\ [VarR]$$

$$\frac{\zeta \vdash [\vec{u} = \vec{v}] : \sigma}{\zeta \vdash [C\,\vec{u} = C\,\vec{v}] : \sigma}\ [Inj]$$

$$\frac{C_1 \neq C_2}{\zeta \vdash [C_1\,\vec{u} = C_2\,\vec{v}] : \bot}\ [Discr]$$

$$\frac{u \approx v}{\zeta \vdash [u = v] : id}\ [Conv]$$

$$\frac{\zeta \vdash [u = v] : \sigma_1 \qquad \zeta \vdash [\vec{u}\sigma_1 = \vec{v}\sigma_1] : \sigma_2}{\zeta \vdash [u\,\vec{u} = v\,\vec{v}] : \sigma_1\sigma_2}\ [Tel]$$

Which variables are open to unification ?

$$v : I\, \vec{p}\, \vec{u}$$

$$C_i\, \vec{z_i} : I\, \vec{p}\, \vec{u_I^i}$$

$$\texttt{match}\ v\ \texttt{as}\ v_0\ \texttt{in}\ I\ \_\ \vec{y}\ \texttt{return}\ P\ \texttt{with}$$
$$\mid C_i\, \vec{z_i} \Rightarrow t_i \ldots$$

- Variable of the constructor : z_i
- Free variables of $\vec{u}$: not stable by reduction

Extending the syntax: Inversion pattern

We extend again the syntax

$$t ::= \ldots \mid \begin{array}{l} \texttt{match}\ M\ \texttt{as}\ x\ \texttt{in}\ [\Delta] I\ \_\ \vec{t}\ \texttt{where}\ \Delta := \vec{q} \\ \texttt{return}\ P\ \texttt{with} C\, \vec{x} \Rightarrow B \ldots \end{array}$$

$$B ::= \bot \mid t\ \texttt{where}\ \sigma$$

Example

$$\frac{\Gamma \vdash v : \texttt{vec}\, A\, (\texttt{S}\, n) \qquad \Gamma(n_0 : \texttt{nat})(v_0 : \texttt{vec}\, A\, (S\, n_0)) \vdash P : s \ldots}{\Gamma \vdash \texttt{match}\ v\ \texttt{in}\ [n_0 : \texttt{nat}]\texttt{vec}\ \_\ (S\, n_0)\ \texttt{where}\ n_0 := n\ \texttt{return}\ \ldots}$$

- The assignment of Δ should make the pattern convertible with the arguments of the term analysed
- The problem now is how to obtain the values of Δ for each branch
  Short answer: Unification

Example

```
Definition tail A n (v : vec A (S n)) : vec A n :=
match v return vec A n with
| Vnil => ⊥
| Vcons x (n':=n) xs => xs
```

The unification problem for the second branch is

$$\texttt{n'} \vdash [\texttt{S n'} = \texttt{S n}] : \{\texttt{n'} \mapsto \texttt{n}\}$$

The second branch satisfies the following type judgment

$$\ldots \texttt{(x : A)(n' := n : nat)(xs : vec A n')} \vdash \texttt{xs} : \texttt{vec A n}$$

Examples

```
Inductive leq : nat -> nat -> Prop :=
  | leqZero : forall n, leq 0 n
  | leqSuc : forall m n, leq m n -> leq (S m) (S n).

Definition leq_10 (n : nat) (H : leq (S 0) 0)
                                          : False :=
  match H in [ ] leq (S 0) 0 return False with
  end.
```

leqZero $\{x\} \vdash [0 = \mathsf{S}\,0, x = 0] : \bot$

leqSuc $\{x, y\} \vdash [\mathsf{S}\,x = \mathsf{S}\,0, \mathsf{S}\,y = 0] : \bot$

Examples

```
Fixpoint leq_nn (n : nat) (H : leq (S n) n)
                              { struct H } : False :=
  match H in [ n0 : nat ] leq (S n0) n0 where n0 := n
                            return False with
  | leqSuc x y H  => leq_nn y H
                       where (x := S y) (n0 := S y)
  end.
```

leqZero $\{n_0, x\} \vdash [0 = \mathsf{S}\, n_0, x = n_0] : \bot$

leqSuc $\{n_0, x, y\} \vdash [\mathsf{S}\, x = \mathsf{S}\, n_0, \mathsf{S}\, y = n_0] :$
$\{n0 \mapsto S\, y, x \mapsto \mathsf{S}\, y\}$

The new typing rule

$$\dfrac{\begin{array}{c}\mathrm{Ind}(I[\Delta_p] : \Pi\Delta_a.s := \{C_i : \Pi\Delta_i.I\,\mathcal{D}om\,(\Delta_p)\ \vec{u_i}\}_i) \in \Sigma \\ \Gamma \vdash M : I\,\vec{p}\,\vec{u} \qquad \Gamma\Delta(x : I\,\vec{p}\,\vec{t}) \vdash P : s \\ \Gamma \vdash \vec{q} : \Delta \qquad \Gamma \vdash \vec{u} \approx \vec{t}[\Delta := \vec{q}] \\ \Gamma; (\vec{z_i} : \Delta_i^*); \Delta; [\vec{u_i^*} = \vec{t} : \Delta_a^*] \vdash b_i : P[x := C_i\,\vec{p}\,\vec{z_i}]\end{array}}{\Gamma \vdash \left(\begin{array}{l}\texttt{match}\ M\ \texttt{as}\ x\ \texttt{in} \\ [\Delta]\,I\ \vec{p}\ \vec{t}\ \texttt{where}\ \Delta := \vec{q} \\ \texttt{return}\ P\ \texttt{with}\ \{C_i\ \vec{z_i} \Rightarrow b_i\}_i\end{array}\right) : P[\Delta := \vec{q}][x := M]}$$

The new typing rule: branches

$$(\text{B-}\bot)\quad \frac{\Gamma; \Delta_i\Delta, \mathcal{D}om\,(\Delta_i) \cup \mathcal{D}om\,(\Delta) \vdash [\vec{u} = \vec{v} : \Theta] \mapsto \bot}{\Gamma; \Delta_i; \Delta; [\vec{u} = \vec{v} : \Theta] \vdash \bot : P}$$

$$(\text{B-S{\scriptsize UB}})\quad \frac{\begin{array}{c}\Gamma; \Delta_i\Delta, \mathcal{D}om\,(\rho) \cup \mathcal{D}om\,(\Delta) \vdash [\vec{u} = \vec{v} : \Theta] \mapsto \Delta', \emptyset \vdash \sigma \\ \mathrm{FV}(t) \cap \mathcal{D}om\,(\Delta) = \emptyset \qquad \Gamma\Delta' \vdash t : P \qquad \Gamma\Delta' \vdash \sigma \approx \rho\end{array}}{\Gamma; \Delta_i; \Delta; [\vec{u} = \vec{v} : \Theta] \vdash t \ \texttt{where}\ \rho : P}$$

Very brief history of pattern matching
in dependent type theory

- Coquand (1992) shows how to define pattern matching in dependent type theory, showing that axiom K is valid

$$K : \forall(A : \mathtt{Type})(x : A)(P : x = x \to \mathtt{Prop}) \\ (H : P(\mathtt{refl\_equal}\ x))(p : x = x), P\ p$$

- Streicher and Hofmann (1993) show that axiom K is not derivable in CC
- McBride, McKinna and Goguen (2004) show that axiom K is all that is needed to have pattern matching as introduced by Coquand

Axiom K is derivable

```
Definition K (x : A) (P : x=x -> Prop)
             (H : P (refl_equal x)) (p : x=x) : P p :=

  match p as p0 in [ ] _ = x return P p0
  | refl_equal => H
  end.
```

Consequence

Heterogeneous equality, injectivity of dependent equality, uniqueness of identity proofs are all provable

Old rule vs. New rule

The old elimination rule can be expressed with the new rule.

Given $v : I\, \vec{p}\, \vec{u}$

$C_i\, \vec{z_i} : I\, \vec{p}\, \vec{u_I^i}$

Old rule

$$\texttt{match } v \texttt{ as } v_0 \texttt{ in } I\ \_\ \vec{y} \texttt{ return } P \texttt{ with}$$

$$\mid C_i\, \vec{z_i} \Rightarrow t_i \ldots$$

New rule

$$\texttt{match } v \texttt{ as } v_0 \texttt{ in } [\vec{y}]\, I\ \_\ \vec{y} \texttt{ where } \vec{y} := \vec{u} \texttt{ return } P \texttt{ with}$$

$$\mid C_i\, \vec{z_i} \Rightarrow t_i \texttt{ where } \vec{y} := \vec{u_I^i} \ \ldots$$

Remark: The unification succeeds positively for all branches

Metatheory

We have proved the following results:

- Substitution Lemma
- Subject Reduction
- Consistency (by a type-preserving translation to CIC+K)

Note: The translation does not preserve reductions. Therefore, it cannot be used to prove Strong Normalization

Conclusions

- We propose a rule for elimination that simplifies writing functions by case analysis
- As a consequence, axiom K is derivable
- This means that we can have pattern matching with dependent types as introduced by Coquand, and implemented in Agda