Dependent types, pattern matching, elimination

Nicolas Oury

18/01/2007

Pattern matching

- In functional programming languages:

```
data Nat = O | S Nat
data List = Empty | Cons Nat List
plus O     m = m
plus (S n) m = S (plus n m)

tail (Cons _ l) = l
tail Empty = error "empty list"
```

Pattern matching

- In functional programming languages:

```
data Nat = O | S Nat
data List = Empty | Cons Nat List
plus O     m = m
plus (S n) m = S (plus n m)

tail (Cons _ l) = l
tail Empty = error "empty list"
```

Pattern matching

- In functional programming languages:

```
data Nat = O | S Nat
data List = Empty | Cons Nat List
plus O     m = m
plus (S n) m = S (plus n m)
```

```
tail (Cons _ l) = l
tail Empty = error "empty list"
```

Pattern matching

- In functional programming languages:

```
data Nat = O | S Nat
data List = Empty | Cons Nat List
plus O     m = m
plus (S n) m = S (plus n m)

tail (Cons _ l) = l
tail Empty = error "empty list"
```

Dependent types

- More precise data types:

$$\text{data}\,\frac{n : \mathrm{Nat}}{\mathrm{List}\ n : \star}$$

- More precise types for constructors:

$$\frac{}{\mathrm{Empty} : \mathrm{List}\ O} \quad \frac{e : \mathrm{Nat};\ l : \mathrm{List}\ n}{\mathrm{Cons}\ e\ l : \mathrm{List}\ (S\ n)}$$

Dependent types

- More precise data types:

$$\text{data}\,\frac{n : \mathsf{Nat}}{\mathsf{List}\ n : \star}$$

- More precise types for constructors:

$$\frac{}{\mathsf{Empty} : \mathsf{List}\ O} \quad \frac{e : \mathsf{Nat};\ l : \mathsf{List}\ n}{\mathsf{Cons}\ e\ l : \mathsf{List}\ (S\ n)}$$

Dependent types

- More precise data types:

$$\text{data}\ \frac{n : \mathrm{Nat}}{\mathrm{List}\ n : \star}$$

- More precise types for constructors:

$$\frac{}{\mathrm{Empty} : \mathrm{List}\ O} \quad \frac{e : \mathrm{Nat};\ l : \mathrm{List}\ n}{\mathrm{Cons}\ e\ l : \mathrm{List}\ (S\ n)}$$

Dependent types

- More precise data types:

$$\text{data}\ \frac{n : \mathrm{Nat}}{\mathrm{List}\ n : \star}$$

- More precise types for constructors:

$$\frac{}{\mathrm{Empty} : \mathrm{List}\ O} \quad \frac{e : \mathrm{Nat};\ l : \mathrm{List}\ n}{\mathrm{Cons}\ e\ l : \mathrm{List}\ (S\ n)}$$

Dependent types

- More precise data types:

$$\text{data}\,\frac{n : \text{Nat}}{\text{List } n : \star}$$

- More precise types for constructors:

$$\frac{}{\text{Empty} : \text{List } O} \quad \frac{e : \text{Nat};\, l : \text{List } n}{\text{Cons } e\; l : \text{List } (S\; n)}$$

Dependent types

- Represent predicates:

$$\text{data}\,\frac{n, m : \mathrm{Nat}}{n \leq m : \star}$$

$$\frac{n : \mathrm{Nat}}{\mathrm{Le\_eq}\ n : n \leq n} \quad \frac{n, m : \mathrm{Nat};\, p : n \leq m}{\mathrm{Le\_s}\ n\ m\ p : n \leq (S\ m)}$$

Dependent types

- Represent predicates:

$$\text{data}\ \frac{n, m : \text{Nat}}{n \leq m : \star}$$

$$\frac{n : \text{Nat}}{\text{Le\_eq}\ n : n \leq n} \quad \frac{n, m : \text{Nat};\ p : n \leq m}{\text{Le\_s}\ n\ m\ p : n \leq (S\ m)}$$

Dependent types

- Represent predicates:

$$\text{data}\,\frac{n, m : \text{Nat}}{n \leq m : \star}$$

$$\frac{n : \text{Nat}}{\text{Le\_eq}\ n : n \leq n} \quad \frac{n, m : \text{Nat};\ p : n \leq m}{\text{Le\_s}\ n\ m\ p : n \leq (S\ m)}$$

Dependent types

- Represent predicates:

$$\text{data}\,\frac{n, m : \text{Nat}}{n \leq m : \star}$$

$$\frac{n : \text{Nat}}{\text{Le\_eq } n : n \leq n} \quad \frac{n, m : \text{Nat};\, p : n \leq m}{\text{Le\_s } n\ m\ p : n \leq (S\ m)}$$

Pattern matching with dependent types

- More precise types for functions:

```
tail :: list (S n) → list n
tail (Cons _ l) = l
tail Empty      = ???
```

- What do we want to write for ????
  - A default case?
  - A proof the case is useless?
- We want to automatically eliminate such a case
- Undecidable problem

Pattern matching with dependent types

- More precise types for functions:

```
tail :: list (S n) → list n
tail (Cons _ l) = l
tail Empty            = ???
```

- What do we want to write for ????
  - A default case?
  - A proof the case is useless?
- We want to automatically eliminate such a case
- Undecidable problem

Pattern matching with dependent types

- More precise types for functions:

```
tail :: list (S n) → list n
tail (Cons _ l) = l
tail Empty        = ???
```

- What do we want to write for ????
  - A default case?
  - A proof the case is useless?
- We want to automatically eliminate such a case
- Undecidable problem

Pattern matching with dependent types

- More precise types for functions:

```
tail :: list (S n) → list n
tail (Cons _ l) = l
tail Empty         = ???
```

- What do we want to write for ????
  - A default case?
  - A proof the case is useless?
- We want to automatically eliminate such a case
- Undecidable problem

Pattern matching with dependent types

- More precise types for functions:

```
tail :: list (S n) → list n
tail (Cons _ l) = l
tail Empty      = ???
```

- What do we want to write for ????
  - A default case?
  - A proof the case is useless?
- We want to automatically eliminate such a case
- Undecidable problem

Pattern matching with dependent types

- More precise types for functions:

```
tail :: list (S n) → list n
tail (Cons _ l) = l
tail Empty          = ???
```

- What do we want to write for ????
  - A default case?
  - A proof the case is useless?
- We want to automatically eliminate such a case
- Undecidable problem

Pattern matching with dependent types

- More precise types for functions:

```
tail :: list (S n) → list n
tail (Cons _ l) = l
tail Empty          = ???
```

- What do we want to write for ????
  - A default case?
  - A proof the case is useless?
- We want to automatically eliminate such a case
- Undecidable problem

Pattern matching with dependent types

- More precise types for functions:

```
tail :: list (S n) → list n
tail (Cons _ l) = l
tail Empty      = ???
```

- What do we want to write for ????
  - A default case?
  - A proof the case is useless?
- We want to automatically eliminate such a case
- Undecidable problem

Undecidability

- Post problem
  - $(u_1, v_1) \ldots (u_n, v_n)$ words on $\{a; b\}$
  - $u_{i_1} \ldots u_{i_k} = v_{i_1} \ldots v_{i_k}$ for a non empty $(i_j)_{1 \le j \le k}$?
  - This is an undecidable problem
- Translates words into an inductive data type :

$$\begin{array}{l} \texttt{Word} = \\ \quad \epsilon : \texttt{Word} \\ \quad \texttt{A} : \texttt{Word} \longrightarrow \texttt{Word} \\ \quad \texttt{B} : \texttt{Word} \longrightarrow \texttt{Word} \end{array}$$

- Notation to add a prefix to a word :

$$\overline{abb}(w) = \texttt{A}\,(\texttt{B}\,(\texttt{B}\,(\texttt{w})))$$

Undecidability

- Post problem
  - $(u_1, v_1) \ldots (u_n, v_n)$ words on $\{a; b\}$
  - $u_{i_1} \ldots u_{i_k} = v_{i_1} \ldots v_{i_k}$ for a non empty $(i_j)_{1 \leq j \leq k}$?
  - This is an undecidable problem
- Translates words into an inductive data type :

$$\begin{array}{l} \mathtt{Word} = \\ \quad \epsilon : \mathtt{Word} \\ \quad \mathtt{A} \;\; : \mathtt{Word} \longrightarrow \mathtt{Word} \\ \quad \mathtt{B} \;\; : \mathtt{Word} \longrightarrow \mathtt{Word} \end{array}$$

- Notation to add a prefix to a word :

$$\overline{abb}(w) = \mathtt{A(B(B(w)))}$$

Undecidability

- Post problem
  - $(u_1, v_1) \ldots (u_n, v_n)$ words on $\{a; b\}$
  - $u_{i_1} \ldots u_{i_k} = v_{i_1} \ldots v_{i_k}$ for a non empty $(i_j)_{1 \leq j \leq k}$?
- We translates a post problem into an inductive family of data types :

$$
\begin{array}{l}
\mathtt{I}\ \_\ \_\ = \\
\quad \mathtt{init} : \mathtt{I}\ \epsilon\ \ \epsilon \\
\quad \mathtt{u1v1} : \mathtt{I}\ \mathtt{u}\ \mathtt{v} \to \mathtt{I}\ \overline{u1}[\mathtt{u}]\ \ \overline{v1}[\mathtt{v}] \\
\ldots \\
\quad \mathtt{unvn} : \mathtt{I}\ \mathtt{u}\ \mathtt{v} \to \mathtt{I}\ \overline{un}[\mathtt{u}]\ \ \overline{vn}[\mathtt{v}]
\end{array}
$$

- Is this function total?

```
f :: I w w → nat
f init = O
```

Undecidability

- Post problem
  - $(u_1, v_1) \ldots (u_n, v_n)$ words on $\{a; b\}$
  - $u_{i_1} \ldots u_{i_k} = v_{i_1} \ldots v_{i_k}$ for a non empty $(i_j)_{1 \leq j \leq k}$?
- We translates a post problem into an inductive family of data types :

$$
\begin{array}{l}
\mathtt{I\ \_\ \_\ =} \\
\quad \mathtt{init} : \mathtt{I}\ \epsilon\ \ \epsilon \\
\quad \mathtt{u1v1} : \mathtt{I\ u\ v} \rightarrow \mathtt{I}\ \overline{u1}\mathtt{[u]}\ \ \overline{v1}\mathtt{[v]} \\
\ldots \\
\quad \mathtt{unvn} : \mathtt{I\ u\ v} \rightarrow \mathtt{I}\ \overline{un}\mathtt{[u]}\ \ \overline{vn}\mathtt{[v]}
\end{array}
$$

- Is this function total?

```
f :: I w w → nat
f init = O
```

Pattern matching by elimination

- An eliminator :

$$
\begin{array}{l}
\forall\ \mathtt{P},\\
\quad \forall\ \Delta_1\ \mathtt{P}\ \vec{s}_1\ \longrightarrow\\
\ldots\\
\quad \forall\ \Delta_m\ \mathtt{P}\ \vec{s}_m\ \longrightarrow\\
\forall\ \Delta\ \ \mathtt{P}\ \vec{t}
\end{array}
$$

- For example, for l : List (S n),
  List–elim (S n) l :

```
∀ P,
  P O Empty →
  ∀ m,e : Nat, ∀ l' : List  m, P (S m) (Cons e l')
 P (S n) l
```

- What P to use?

Pattern matching by elimination

- An eliminator :

  $\forall$ `P,`
  $\quad \forall\ \Delta_1$ `P` $\vec{s_1} \longrightarrow$
  $\ldots$
  $\quad \forall\ \Delta_m$ `P` $\vec{s_m} \longrightarrow$
  $\forall\ \Delta$ `P` $\vec{t}$

- For example, for l : List (S n),
  List–elim (S n) l :

$\forall$ `P,`
`P O Empty` $\rightarrow$
$\forall$ `m,e : Nat,` $\forall$ `l' : List m, P (S m) (Cons e l')`
`P (S n) l`

- What P to use?

Pattern matching by elimination

- An eliminator :

$$\begin{array}{l} \forall\ \mathtt{P},\\ \quad \forall\ \Delta_1\ \mathtt{P}\ \vec{s_1}\ \rightarrow\\ \ldots\\ \quad \forall\ \Delta_m\ \mathtt{P}\ \vec{s_m}\ \rightarrow\\ \forall\ \Delta\ \ \mathtt{P}\ \vec{t} \end{array}$$

- For example, for l : List (S n),
  List–elim (S n) l :

```
∀ P,
  P O Empty →
  ∀ m,e : Nat, ∀ l′ : List  m, P (S m) (Cons e l′)
 P (S n) l
```

- What P to use?

Pattern matching by elimination

- An eliminator :

$$\begin{array}{l} \forall\ \mathtt{P}, \\ \quad \forall\ \Delta_1\ \mathtt{P}\ \vec{s_1}\ \rightarrow \\ \ldots \\ \quad \forall\ \Delta_m\ \mathtt{P}\ \vec{s_m}\ \rightarrow \\ \forall\ \Delta\ \ \mathtt{P}\ \vec{t} \end{array}$$

- For example, for l : List (S n), List–elim (S n) l :

$$\begin{array}{l} \forall\ \mathtt{P}, \\ \quad \mathtt{P}\ \mathtt{O}\ \mathtt{Empty}\ \rightarrow \\ \quad \forall\ \mathtt{m,e}\ :\ \mathtt{Nat},\ \forall\ \mathtt{l'}\ :\ \mathtt{List}\ \ \mathtt{m},\ \mathtt{P}\ (\mathtt{S}\ \mathtt{m})\ (\mathtt{Cons}\ \mathtt{e}\ \mathtt{l'}) \\ \ \mathtt{P}\ (\mathtt{S}\ \mathtt{n})\ \mathtt{l} \end{array}$$

- What P to use?

Pattern matching by elimination

$$\forall\ \mathtt{P},$$
$$\forall\ \Delta_1\ \mathtt{P}\ \vec{s_1}\ \rightarrow\ \ldots\ \ \forall\ \Delta_m\ \mathtt{P}\ \vec{s_m}\ \rightarrow$$
$$\forall\ \Delta\ \ \mathtt{P}\ \vec{t}$$

- We apply the elimination to a goal $\forall \Gamma, T$
- We have an eliminator targeting $P\ \vec{t}$
- We choose $P \equiv \lambda \vec{x}, \forall \Gamma, \vec{x} = \vec{t} \rightarrow T$
- So $P\ \vec{t} \equiv \forall \Gamma, \vec{t} = \vec{t} \rightarrow T$
- New goals: $\forall \Delta_i, \forall \Gamma, \vec{s_i} = \vec{t} \rightarrow T$

Pattern matching by elimination

$$\forall\ \mathtt{P},$$
$$\forall\ \Delta_1\ \mathtt{P}\ \vec{s_1}\ \rightarrow\ \ldots\ \ \forall\ \Delta_m\ \mathtt{P}\ \vec{s_m}\ \rightarrow$$
$$\forall\ \Delta\ \ \mathtt{P}\ \vec{t}$$

- We apply the elimination to a goal $\forall \Gamma, T$
- We have an eliminator targeting $P\ \vec{t}$
- We choose $P \equiv \lambda \vec{x}, \forall \Gamma, \vec{x} = \vec{t} \rightarrow T$
- So $P\ \vec{t} \equiv \forall \Gamma, \vec{t} = \vec{t} \rightarrow T$
- New goals: $\forall \Delta_i, \forall \Gamma, \vec{s_i} = \vec{t} \rightarrow T$

Pattern matching by elimination

$$\forall\ \mathtt{P},$$
$$\forall\ \Delta_1\ \mathtt{P}\ \vec{s_1}\ \rightarrow\ \ldots\quad \forall\ \Delta_m\ \mathtt{P}\ \vec{s_m}\ \rightarrow$$
$$\forall\ \Delta\quad \mathtt{P}\ \vec{t}$$

- We apply the elimination to a goal $\forall \Gamma, T$
- We have an eliminator targeting $P\ \vec{t}$
- We choose $P \equiv \lambda \vec{x}, \forall \Gamma, \vec{x} = \vec{t} \rightarrow T$
- So $P\ \vec{t} \equiv \forall \Gamma, \vec{t} = \vec{t} \rightarrow T$
- New goals: $\forall \Delta_i, \forall \Gamma, \vec{s_i} = \vec{t} \rightarrow T$

Pattern matching by elimination

$$\forall\ \mathtt{P},$$
$$\forall\ \Delta_1\ \mathtt{P}\ \vec{s_1}\ \rightarrow\ \ldots\ \ \forall\ \Delta_m\ \mathtt{P}\ \vec{s_m}\ \rightarrow$$
$$\forall\ \Delta\ \ \mathtt{P}\ \vec{t}$$

- We apply the elimination to a goal $\forall \Gamma, T$
- We have an eliminator targeting $P\ \vec{t}$
- We choose $P \equiv \lambda \vec{x}, \forall \Gamma, \vec{x} = \vec{t} \rightarrow T$
- So $P\ \vec{t} \equiv \forall \Gamma, \vec{t} = \vec{t} \rightarrow T$
- New goals: $\forall \Delta_i, \forall \Gamma, \vec{s_i} = \vec{t} \rightarrow T$

Pattern matching by elimination

$$\forall\ \mathtt{P},$$
$$\forall\ \Delta_1\ \mathtt{P}\ \vec{s_1}\ \rightarrow\ \ldots\ \ \forall\ \Delta_m\ \mathtt{P}\ \vec{s_m}\ \rightarrow$$
$$\forall\ \Delta\ \ \mathtt{P}\ \vec{t}$$

- We apply the elimination to a goal $\forall \Gamma, T$
- We have an eliminator targeting $P\ \vec{t}$
- We choose $P \equiv \lambda \vec{x}, \forall \Gamma, \vec{x} = \vec{t} \rightarrow T$
- So $P\ \vec{t} \equiv \forall \Gamma, \vec{t} = \vec{t} \rightarrow T$
- New goals: $\forall \Delta_i, \forall \Gamma, \vec{s_i} = \vec{t} \rightarrow T$

Pattern matching by elimination

$$\forall\ \mathtt{P},$$
$$\forall\ \Delta_1\ \mathtt{P}\ \vec{s_1}\ \rightarrow\ \ldots\ \ \forall\ \Delta_m\ \mathtt{P}\ \vec{s_m}\ \rightarrow$$
$$\forall\ \Delta\ \ \mathtt{P}\ \vec{t}$$

- We apply the elimination to a goal $\forall\Gamma, T$
- We have an eliminator targeting $P\ \vec{t}$
- We choose $P \equiv \lambda\vec{x}, \forall\Gamma, \vec{x} = \vec{t} \rightarrow T$
- So $P\ \vec{t} \equiv \forall\Gamma, \vec{t} = \vec{t} \rightarrow T$
- New goals: $\forall\Delta_i, \forall\Gamma, \vec{s_i} = \vec{t} \rightarrow T$

Example

```
∀ P,
  P O Empty →
∀ m, e : Nat, ∀ l′ : List m, P (S m) (Cons e l′) →
  P (S n) l
```

- tail : $\forall$n : Nat, $\forall$l : List (S n), List n

```
P ≡ λ p. λ l0 : List p,
  ∀ n, l : List (S n), p=S n → l0 = l → List n
```

- New goals:
  - $\forall n, l$: List (S n), $O = S\ n \rightarrow Empty = l \rightarrow$ List n
  - $\forall m, l'$: List m, $\forall n, l$: List (S n), $S\ m = S\ n \rightarrow$ Cons $e\ l' = l \rightarrow$ List n

```
tail n l ⇐ case l
    tail (S m) (Cons e l′) ⇒ l′
```

Example

```
∀ P,
  P O Empty →
∀ m, e : Nat, ∀ l' : List m, P (S m) (Cons e l') →
  P (S n) l
```

- tail : ∀n : Nat, ∀l : List (S n), List n

```
P ≡ λ p. λ l0 : List p,
  ∀ n, l : List (S n), p=S n → l0 = l → List n
```

- New goals:
  - $\forall n, l$: List (S n), $O = S\ n \rightarrow \mathit{Empty} = l \rightarrow$ List n
  - $\forall m, l'$: List m, $\forall n, l$: List (S n), $S\ m = S\ n \rightarrow$ Cons $e\ l' = l \rightarrow$ List n

```
tail n l ⇐ case l
    tail (S m) (Cons e l') ⇒ l'
```

Example

```
∀ P,
  P O Empty →
∀ m, e : Nat, ∀ l′ : List m, P (S m) (Cons e l′) →
  P (S n) l
```

- tail : $\forall$n : Nat, $\forall$l : List (S n), List n

```
P ≡ λ p. λ l0 : List p,
  ∀ n, l : List (S n), p=S n → l0 = l → List n
```

- New goals:
  - $\forall n, l$: List (S n), $O = S\ n \to Empty = l \to$ List n
  - $\forall m, l'$: List $m, \forall n, l$: List (S n),
    $S\ m = S\ n \to Cons\ e\ l' = l \to$ List n

```
tail n l ⇐ case l
    tail (S m) (Cons e l′) ⇒ l′
```

Example

```
∀ P,
  P O Empty →
∀ m, e : Nat, ∀ l′ : List m, P (S m) (Cons e l′) →
  P (S n) l
```

- tail : ∀n : Nat, ∀l : List (S n), List n

```
P ≡ λ p. λ l0 : List p,
  ∀ n, l : List (S n), p=S n → l0 = l → List n
```

- New goals:
  - $\forall n, l$: List (S n), $O = S\ n \to Empty = l \to$ List n
  - $\forall m, l'$: List $m, \forall n, l$: List (S n), $S\ m = S\ n \to$ Cons $e\ l' = l \to$ List n

```
tail n l ⇐ case l
    tail (S m) (Cons e l′) ⇒ l′
```

Example

```
∀ P,
  P O Empty →
∀ m, e : Nat, ∀ l′ : List m, P (S m) (Cons e l′) →
  P (S n) l
```

- tail : ∀n : Nat, ∀l : List (S n), List n

```
P ≡ λ p. λ l0 : List p,
  ∀ n, l : List (S n), p=S n → l0 = l → List n
```

- New goals:
  - $\forall n, l : \mathrm{List}\ (\mathrm{S}\ n), O = S\ n \rightarrow Empty = l \rightarrow \mathrm{List}\ n$
  - $\forall m, l' : \mathrm{List}\ m, \forall n, l : \mathrm{List}\ (\mathrm{S}\ n),$
    $S\ m = S\ n \rightarrow \mathrm{Cons}\ e\ l' = l \rightarrow \mathrm{List}\ n$

```
tail n l ⇐ case l
    tail (S m) (Cons e l′) ⇒ l′
```

Example

```
∀ P,
  P O Empty →
∀ m, e : Nat, ∀ l′ : List m, P (S m) (Cons e l′) →
  P (S n) l
```

- tail : ∀n : Nat, ∀l : List (S n), List n

```
P ≡ λ p. λ l0 : List p,
  ∀ n, l : List (S n), p=S n → l0 = l → List n
```

- New goals:
  - $\forall n, l : \text{List } (\text{S } n), O = S\ n \rightarrow Empty = l \rightarrow \text{List } n$
  - $\forall m, l' : \text{List } m, \forall n, l : \text{List } (\text{S } n),$
    $S\ m = S\ n \rightarrow \text{Cons } e\ l' = l \rightarrow \text{List } n$

```
  tail n l ⇐ case l
      tail (S m) (Cons e l′) ⇒ l′
```

Example

```
∀ P,
  P O Empty →
∀ m, e : Nat, ∀ l′ : List m, P (S m) (Cons e l′) →
  P (S n) l
```

- tail : ∀n : Nat, ∀l : List (S n), List n

```
P ≡ λ p. λ l0 : List p,
  ∀ n, l : List (S n), p=S n → l0 = l → List n
```

- New goals:
  - $\forall n, l$: List (S n), $O = S\ n \rightarrow Empty = l \rightarrow$ List n
  - $\forall m, l'$: List $m, \forall n, l$: List (S n),
    $S\ m = S\ n \rightarrow$ Cons $e\ l' = l \rightarrow$ List n

```
  tail n l ⇐ case l
    tail (S m) (Cons e l′) ⇒ l′
```

Some ideas to remove more useless cases

- Some useless cases are not removed:

```
useless :: ∀ n : Nat, (S n) ≤ n → False
 useless n p ⇐ case p
   useless (S m) p' ⇒ ???
```

- ??? :: ∀n, p, q : Nat, S p = S n → S q = n → p ≤ q → False
- Approximations of sets of inductive terms

$$p : n \le m \Rightarrow \{|n|_S \le |m|_S; |p|_{le\_S} = |n|_S - |m|_S\}$$

- Apply these approximations to new goals

Some ideas to remove more useless cases

- Some useless cases are not removed:

```
useless :: ∀ n : Nat, (S n) ≤ n → False
 useless n p ⇐ case p
   useless (S m) p' ⇒ ???
```

- ??? :: $\forall$n, p, q : Nat, S p = S n → S q = n → p ≤ q → False
- Approximations of sets of inductive terms

$$p : n \leq m \Rightarrow \{|n|_S \leq |m|_S ; |p|_{le\_S} = |n|_S - |m|_S\}$$

- Apply these approximations to new goals

Some ideas to remove more useless cases

- Some useless cases are not removed:

```
useless :: ∀ n : Nat, (S n) ≤ n → False
 useless n p ⇐ case p
   useless (S m) p′ ⇒ ???
```

- `???` :: ∀n, p, q : Nat, S p = S n →S q = n →p ≤ q →False
- Approximations of sets of inductive terms

$$p : n \leq m \Rightarrow \{|n|_S \leq |m|_S; |p|_{le\_S} = |n|_S - |m|_S\}$$

- Apply these approximations to new goals

Some ideas to remove more useless cases

- Some useless cases are not removed:

```
useless :: ∀ n : Nat, (S n) ≤ n → False
 useless n p ⇐ case p
   useless (S m) p′ ⇒ ???
```

- `???` :: ∀n, p, q : Nat, S p = S n →S q = n →p ≤ q →False
- Approximations of sets of inductive terms

$$p : n \leq m \Rightarrow \{|n|_S \leq |m|_S; |p|_{le\_S} = |n|_S - |m|_S\}$$

- Apply these approximations to new goals

Some ideas to remove more useless cases

- Some useless cases are not removed:

```
useless :: ∀ n : Nat, (S n) ≤ n → False
 useless n p ⇐ case p
   useless (S m) p′ ⇒ ???
```

- `???` :: ∀n, p, q : Nat, S p = S n → S q = n → p ≤ q → False
- Approximations of sets of inductive terms

$$p : n \leq m \Rightarrow \{|n|_S \leq |m|_S; |p|_{le\_S} = |n|_S - |m|_S\}$$

- Apply these approximations to new goals

Some ideas to remove more useless cases

- Some useless cases are not removed:

```
useless :: ∀ n : Nat, (S n) ≤ n → False
 useless n p ⇐ case p
   useless (S m) p′ ⇒ ???
```

- `???` :: ∀n, p, q : Nat, S p = S n → S q = n → p ≤ q → False
- Approximations of sets of inductive terms

$$p : n \leq m \Rightarrow \{|n|_S \leq |m|_S; |p|_{le\_S} = |n|_S - |m|_S\}$$

- Apply these approximations to new goals