Compilation of dependent pattern matching using small inversion

W. Ait-Moussa[1] Pierre Boutillier[2] Hugo Herbelin[1]
Meven Lennon-Bertrand[1] Thierry Martinez[3] Gabriel Scherer[1]

[1]Université Paris Cité - INRIA - CNRS - IRIF

[2]ENS Paris

[3]INRIA Paris

May 4th 2026

Inductive datatypes, recursors and pattern matching

```
type nat    =   | Zero : nat
                | Succ : nat -> nat
type α list =   | Nil  : α list
                | Cons : α -> α list -> α list
```

Inductive datatypes, recursors and pattern matching

```
type nat    =   | Zero : nat
                | Succ : nat -> nat
type α list =   | Nil  : α list
                | Cons : α -> α list -> α list
```

- Meaning and computational behavior given by recursors

```
rec_nat  : β -> (nat -> β -> β) -> nat -> β
rec_list : β -> (α -> α list -> β -> β) -> α list -> β
```

Inductive datatypes, recursors and pattern matching

```
type nat     =   | Zero : nat
                 | Succ : nat -> nat
type α list  =   | Nil  : α list
                 | Cons : α -> α list -> α list
```

- Meaning and computational behavior given by recursors

```
rec_nat  : β -> (nat -> β -> β) -> nat -> β
rec_list : β -> (α -> α list -> β -> β) -> α list -> β
```

- Better manipulated with pattern matching and recursion

```
let rec list_lte (t : nat list) (t' : nat list) =
  match t, t' with
  | Nil, _ -> true
  | Cons _ _, Nil -> false
  | Cons n l, Cons n' l' ->
      nat_lte n n' && list_equal l l'
```

Dependent inductive datatypes

```
ℕ : □ᵢ := | O : ℕ | S : ℕ -> ℕ
𝕍(X:□ᵢ, n:ℕ) : □ᵢ :=
     | Nil : 𝕍(X,O)
     | Cons(hd:X, n:ℕ, tl:𝕍(X, n)) : 𝕍(X, S(n))
```

Dependent inductive datatypes

```
ℕ : □ᵢ := | O : ℕ | S : ℕ -> ℕ
𝕍(X:□ᵢ, n:ℕ) : □ᵢ :=
     | Nil : 𝕍(X,O)
     | Cons(hd:X, n:ℕ, tl:𝕍(X, n)) : 𝕍(X, S(n))
```

- With their dependent recursors

```
rec_nat : (A : ℕ -> □ᵢ) -> (a : A O)
       -> ((n : ℕ) -> (a : A n) -> A S(n))
       -> (n : ℕ) -> A n
rec_vect : (A : (n : ℕ) -> (v : 𝕍(X, n)) -> □ᵢ)
       -> ...
```

Dependent inductive datatypes

```
ℕ : □ᵢ := | O : ℕ | S : ℕ -> ℕ
𝕍(X:□ᵢ, n:ℕ) : □ᵢ :=
     | Nil : 𝕍(X,O)
     | Cons(hd:X, n:ℕ, tl:𝕍(X, n)) : 𝕍(X, S(n))
```

- With their dependent recursors

```
rec_nat : (A : ℕ -> □ᵢ) -> (a : A O)
      -> ((n : ℕ) -> (a : A n) -> A S(n))
      -> (n : ℕ) -> A n
rec_vect : (A : (n : ℕ) -> (v : 𝕍(X, n)) -> □ᵢ)
      -> ...
```

- Can be split into primitive pattern matching and guarded recursion

```
let fix rec_vect A c1 c2 n v {struct v} :=
   match v in 𝕍(X, n) return A n v with
   | Nil -> c1
   | Cons(_, n, a, v') -> c2 n hd tl (rec_vect A c1 c2 n v')
```

Dependent inductive datatypes

```
ℕ : □ᵢ := | O : ℕ | S : ℕ -> ℕ
𝕍(X:□ᵢ, n:ℕ) : □ᵢ :=
     | Nil : 𝕍(X,O)
     | Cons(hd:X, n:ℕ, tl:𝕍(X, n)) : 𝕍(X, S(n))
```

- With their dependent recursors

```
rec_nat : (A : ℕ -> □ᵢ) -> (a : A O)
       -> ((n : ℕ) -> (a : A n) -> A S(n))
       -> (n : ℕ) -> A n
rec_vect : (A : (n : ℕ) -> (v : 𝕍(X, n)) -> □ᵢ)
       -> ...
```

- Can be split into primitive pattern matching and guarded recursion

```
let fix rec_vect A c1 c2 n v {struct v} :=
   match v in 𝕍(X, n) return A n v with
   | Nil -> c1
   | Cons(_, n, a, v') -> c2 n hd tl (rec_vect A c1 c2 n v')
```

Dependent inductive datatypes

```
ℕ : □ᵢ := | O : ℕ | S : ℕ -> ℕ
𝕍(X:□ᵢ, n:ℕ) : □ᵢ :=
     | Nil : 𝕍(X,O)
     | Cons(hd:X, n:ℕ, tl:𝕍(X, n)) : 𝕍(X, S(n))
```

- With their dependent recursors

```
rec_nat : (A : ℕ -> □ᵢ) -> (a : A O)
      -> ((n : ℕ) -> (a : A n) -> A S(n))
      -> (n : ℕ) -> A n
rec_vect : (A : (n : ℕ) -> (v : 𝕍(X, n)) -> □ᵢ)
      -> ...
```

- Can be split into primitive pattern matching and guarded recursion

```
let fix rec_vect A c1 c2 n v {struct v} :=
   match v in 𝕍(X, n) return A n v with
   | Nil -> c1
   | Cons(_, n, a, v') -> c2 n hd tl (rec_vect A c1 c2 n v')
```

Generalized dependent pattern matching

A high-level dependent matching construct [1]

[1] Coquand (1992), "Pattern Matching with Dependent Types"

Generalized dependent pattern matching

A high-level dependent matching construct [1]

```
let tail (n : ℕ) (v : 𝕍(X, S(n))) =
  match v with
  | Cons _ _ tl -> tl
```

- Discard impossible cases

[1] Coquand (1992), "Pattern Matching with Dependent Types"

Generalized dependent pattern matching

A high-level dependent matching construct [1]

```
let tail (n : ℕ) (v : 𝕍(X, S(n))) =
   match v with
   | Cons _ _ tl -> tl
let rec map2 (n : ℕ) (v : 𝕍(X, n)) (v' : 𝕍(Y, n)) f: 𝕍(Z, n) =
   match v, v' with
   | Nil, Nil -> Nil
   | Cons n0 hd tl, Cons .(n0) hd' tl' ->
            Cons (f hd hd') (map2 n0 tl tl' f)
```

- Discard impossible cases
- Relate dependent inductive indices

[1] Coquand (1992), "Pattern Matching with Dependent Types"

Generalized dependent pattern matching

A high-level dependent matching construct [1]

```
let tail (n : ℕ) (v : 𝕍(X, S(n))) =
   match v with
   | Cons _ _ tl -> tl
let rec map2 (n : ℕ) (v : 𝕍(X, n)) (v' : 𝕍(Y, n)) f: 𝕍(Z, n) =
   match v, v' with
   | Nil, Nil -> Nil
   | Cons n0 hd tl, Cons .(n0) hd' tl' ->
         Cons (f hd hd') (map2 n0 tl tl' f)
```

- Discard impossible cases
- Relate dependent inductive indices

A core construct in Agda, available in Rocq-Equations and in Lean's elaborator.

[1] Coquand (1992), "Pattern Matching with Dependent Types"

Generalized dependent pattern matching

A high-level dependent matching construct [1]

```
let tail (n : ℕ) (v : 𝕍(X, S(n))) =
   match v with
   | Cons _ _ tl -> tl
let rec map2 (n : ℕ) (v : 𝕍(X, n)) (v' : 𝕍(Y, n)) f: 𝕍(Z, n) =
   match v, v' with
   | Nil, Nil -> Nil
   | Cons n0 hd tl, Cons .(n0) hd' tl' ->
            Cons (f hd hd') (map2 n0 tl tl' f)
```

- Discard impossible cases
- Relate dependent inductive indices

A core construct in Agda, available in Rocq-Equations and in Lean's elaborator.
Primary concern: checking and compiling such a construct to primitive recursors.

[1] Coquand (1992), "Pattern Matching with Dependent Types"

Recipe for eliminating dependent pattern matching (1/2)

Step 1. Elaboration into *splitting trees*[2]

```
map2 {A B C} f {n} (v:𝕍 A n) (v':𝕍 B n) : 𝕍 C n
map2 f nil nil = nil
map2 f (cons n a v) (cons .n b v') = cons n (f a b) (map2 f v v')
```

[2] Cockx and Abel (2018), "Elaborating dependent (co)pattern matching"

Recipe for eliminating dependent pattern matching (1/2)

Step 1. Elaboration into *splitting trees*[2]

```
map2 {A B C} f {n} (v:𝕍 A n) (v':𝕍 B n) : 𝕍 C n
map2 f nil nil = nil
map2 f (cons n a v) (cons .n b v') = cons n (f a b) (map2 f v v')
```

$$\Longrightarrow$$

$$\text{map2}\, f\, v\, v' \begin{cases} \text{map2}\, f\, \text{nil}\, v' \begin{cases} \text{map2}\, f\, \text{nil}\, \text{nil} = \text{nil} \\ \cancel{\text{map2}\, f\, \text{nil}\, (\text{cons} \ldots)} \end{cases} \\ \text{map2}\, f\, (\text{cons}\, n\, a\, v)\, v' \begin{cases} \cancel{\text{map2}\, f\, (\text{cons}\, n\, a\, v)\, \mathit{nil}} \\ \text{map2}\, f\, (\text{cons}\, n\, a\, v)(\text{cons}\, .n\, b\, v') = \ldots \end{cases} \end{cases}$$

[2] Cockx and Abel (2018), "Elaborating dependent (co)pattern matching"

Recipe for eliminating dependent pattern matching (1/2)

Step 1. Elaboration into *splitting trees*[2]

```
map2 {A B C} f {n} (v:𝕍 A n) (v':𝕍 B n) : 𝕍 C n
map2 f nil nil = nil
map2 f (cons n a v) (cons .n b v') = cons n (f a b) (map2 f v v')
```

$$\Longrightarrow$$

$$\text{map2}\ f\ v\ v' \begin{cases} \text{map2}\ f\ \text{nil}\ v' \begin{cases} \text{map2}\ f\ \text{nil}\ \text{nil} = \text{nil} \\ \sout{\text{map2}\ f\ \text{nil}\ (\text{cons} \ldots)} \end{cases} \\ \text{map2}\ f\ (\text{cons}\ n\ a\ v)\ v' \begin{cases} \sout{\text{map2}\ f\ (\text{cons}\ n\ a\ v)\ nil} \\ \text{map2}\ f\ (\text{cons}\ n\ a\ v)(\text{cons}\ .n\ b\ v') = \ldots \end{cases} \end{cases}$$

- Each node splits exactly one scrutinee
- Refines each child pattern by unifying with the scrutinee indices
- Leaves correspond to high-level clauses

[2] Cockx and Abel (2018), "Elaborating dependent (co)pattern matching"

Recipe for eliminating dependent pattern matching (2/2)

Step 1. Elaboration into splitting trees
In Agda exhaustive splitting trees are the final compilation target.

Step 2. Translation to recursors[3] [4]

A. Proving inaccessible branches are impossible to reach,
  - using injectivity, discrimination and acyclicity of constructors,
  - mechanized through equational reasoning

B. Encoding every recursive call appropriately into the recursors

Long-term goal: making Rocq built-in pattern matching more convenient.
In this work, we focus on (A) using small inversion.

[3] Cockx, Devriese, and Piessens (2016), "Eliminating dependent pattern matching without K"
[4] Goguen, McBride, and McKinna (2006), "Eliminating Dependent Pattern Matching"

Small inversion

Impossible cases using equational reasoning

```
Definition head_1 (A:□) (n:ℕ) (v:𝕍(A, S(n))) : A :=
   match v in 𝕍(_, n') return (n' = S(n)) -> A with
   | Cons(_, m, hd, tl) => fun e => hd
   | Nil => fun (e : O = S(n)) => efq (zero_neq_succ e)
   end eq_refl
```

Small inversion

Impossible cases using equational reasoning

```
Definition head_1 (A:□) (n:ℕ) (v:𝕍(A, S(n))) : A :=
   match v in 𝕍(_, n') return (n' = S(n)) -> A with
   | Cons(_, m, hd, tl) => fun e => hd
   | Nil => fun (e : O = S(n)) => efq (zero_neq_succ e)
   end eq_refl
```

Using small inversion

```
Definition head_2 (A:□) (n:ℕ) (v:𝕍(A, S(n))) : A :=
   match v in 𝕍(_, n')
   return match n' with
          | O => True
          | S(_) => A
   with
   | Nil => I
   | Cons(_, m, hd, tl) => hd
```

The name *small inversion* was coined by Monin (2010) "Proof Trick: Small Inversions": smaller terms with more natural structure.

This work

- An alternative translation (to Cockx, Devriese, and Piessens (2016) and Goguen, McBride, and McKinna (2006)) of splitting trees to primitive pattern matching using small inversion
- Presented in a pure language of dependent inductive types and dependent pattern matching
- Based on a 2022 implementation for Rocq by Thierry Martinez
  - Still trying to merge our paper specification with the implementation
- We sideline the recursion/guard condition questions
  - Deserves a thorough treatment but we haven't looked into it yet
  - The primitive match/guard condition as separate constructs view gives a good justification for separating the two

Plan

Introduction

A pure language for dependent pattern matching

Compiling $\texttt{match}_\star$

Compiling $\texttt{match}_1$ using small inversion

A pure language for dependent pattern matching: target

A signature of inductive types

$$
\begin{aligned}
\mathbb{N} &:= \mid \texttt{O} : \mathbb{N} \mid \texttt{S}(n : \mathbb{N}) : \mathbb{N} \\
\mathbb{V}(A : \square, n : \mathbb{N}) &:= \mid \texttt{Nil}(A : \square) : \mathbb{V}(\texttt{O}) \\
&\quad \mid \texttt{Cons}(A : \square, n : \mathbb{N}, hd : A, tl : \mathbb{V}(A, n)) : \mathbb{V}(A, \texttt{S}(n)) \\
\texttt{eq}(A : \square, x : A, y : A) &:= \mid \texttt{refl}(A : \square, x : A) : \texttt{eq}(A, x, x) \\
\texttt{True} &:= \mid \texttt{I} : \texttt{True}
\end{aligned}
$$

A pure language for dependent pattern matching: target

A signature of inductive types

$$
\begin{aligned}
\mathbb{N} &:= \mid \mathtt{O} : \mathbb{N} \mid \mathtt{S}(n : \mathbb{N}) : \mathbb{N} \\
\mathbb{V}(A : \Box, n : \mathbb{N}) &:= \mid \mathtt{Nil}(A : \Box) : \mathbb{V}(\mathtt{O}) \\
&\quad \mid \mathtt{Cons}(A : \Box, n : \mathbb{N}, hd : A, tl : \mathbb{V}(A, n)) : \mathbb{V}(A, \mathtt{S}(n)) \\
\mathtt{eq}(A : \Box, x : A, y : A) &:= \mid \mathtt{refl}(A : \Box, x : A) : \mathtt{eq}(A, x, x) \\
\mathtt{True} &:= \mid \mathtt{I} : \mathtt{True}
\end{aligned}
$$

Terms made of

- Variables $x, y, z, \ldots$ and ambiguous universes $\Box$

A pure language for dependent pattern matching: target

A signature of inductive types

$$\begin{aligned}
\mathbb{N} &:= \mid \mathtt{O} : \mathbb{N} \mid \mathtt{S}(n : \mathbb{N}) : \mathbb{N} \\
\mathbb{V}(A : \Box, n : \mathbb{N}) &:= \mid \mathtt{Nil}(A : \Box) : \mathbb{V}(\mathtt{O}) \\
&\quad \mid \mathtt{Cons}(A : \Box, n : \mathbb{N}, hd : A, tl : \mathbb{V}(A, n)) : \mathbb{V}(A, \mathtt{S}(n)) \\
\mathtt{eq}(A : \Box, x : A, y : A) &:= \mid \mathtt{refl}(A : \Box, x : A) : \mathtt{eq}(A, x, x) \\
\mathtt{True} &:= \mid \mathtt{I} : \mathtt{True}
\end{aligned}$$

Terms made of

- Variables $x, y, z, \ldots$ and ambiguous universes $\Box$
- Fully applied inductive types $\mathbb{V}(X, S(m))$, $\mathbb{N}$, ...
- Fully applied constructors $\mathtt{Cons}(X, m, t, \mathtt{Nil}(X))$, $\mathtt{O}$, ...

A pure language for dependent pattern matching: target

A signature of inductive types

$$
\begin{aligned}
\mathbb{N} &:= \mid \texttt{O} : \mathbb{N} \mid \texttt{S}(n : \mathbb{N}) : \mathbb{N} \\
\mathbb{V}(A : \Box, n : \mathbb{N}) &:= \mid \texttt{Nil}(A : \Box) : \mathbb{V}(\texttt{O}) \\
&\quad \mid \texttt{Cons}(A : \Box, n : \mathbb{N}, hd : A, tl : \mathbb{V}(A, n)) : \mathbb{V}(A, \texttt{S}(n)) \\
\texttt{eq}(A : \Box, x : A, y : A) &:= \mid \texttt{refl}(A : \Box, x : A) : \texttt{eq}(A, x, x) \\
\texttt{True} &:= \mid \texttt{I} : \texttt{True}
\end{aligned}
$$

Terms made of

- Variables $x, y, z, \ldots$ and ambiguous universes $\Box$
- Fully applied inductive types $\mathbb{V}(X, S(m))$, $\mathbb{N}$, ...
- Fully applied constructors $\texttt{Cons}(X, m, t, \texttt{Nil}(X))$, $\texttt{O}$, ...
- Primitive pattern matching with generalization (new!)

$$
\begin{aligned}
&\texttt{match}_0\ e \\
&\quad \texttt{as}\ w : \texttt{eq}_A(x, y) \\
&\quad \texttt{over}\ (z := h : P\,x\,w) \\
&\quad \texttt{return}\ (\texttt{eq}_A(x, y)) \\
&\texttt{with}\ \mid \texttt{refl}_A(x') \mapsto z
\end{aligned}
$$

A pure language for dependent pattern matching: source

$\texttt{match}_\star$ for high-level dependent pattern matching

The map_2 example:

$$
\begin{array}{l}
\texttt{match}_\star\ (t, u) \\
\quad \texttt{as}\ (v_1 : \mathbb{V}(A, n), v_2 : \mathbb{V}(B, n)) \\
\quad \texttt{return}\ \mathbb{V}(C, n) \\
\texttt{with} \\
\mid \texttt{Nil}(A'), \texttt{Nil}(B') \mapsto \texttt{Nil}(C) \\
\mid \texttt{Cons}(A', n', a, v_1'), \texttt{Cons}(B', n', b, v_2') \mapsto \texttt{Cons}(C, n', \mathsf{f}\ a\ b, \mathsf{map}_2\ \mathsf{f}\ n'\ v_1'\ v_2')
\end{array}
$$

The tail example:

$$
\begin{array}{l}
\texttt{match}_\star\ (t,)\ \mathsf{as}\ l \\
\quad \texttt{as}\ (l : \mathbb{V}(A, S(n)),) \\
\quad \texttt{return}\ A \\
\texttt{with} \\
\mid \texttt{Cons}(A', n', a, l') \mapsto a
\end{array}
$$

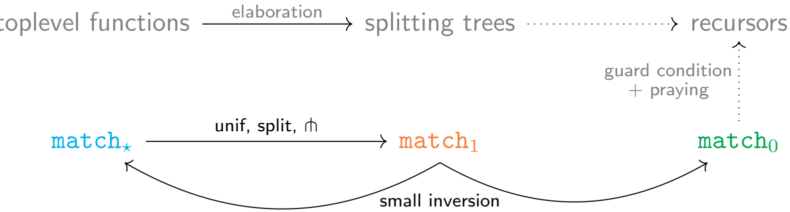
Compilation scheme

Where we have the intermediary construct, **constrained case analysis**

$$\texttt{match}_1\ t\ \texttt{as}\ x : I\,\varphi\ \texttt{with}\quad |_i\ C_i\,\psi_i\ \mapsto\ r_i$$

Where

- r_i is either a term or a refutation $\pitchfork$
- The $(C_i\,\psi_i)^i$ are the *most refined* patterns with respect to $I\,\varphi$



Plan

Introduction

A pure language for dependent pattern matching

Compiling $\texttt{match}_\star$

Compiling $\texttt{match}_1$ using small inversion

Compiling $\texttt{match}_\star$: generalization

Each step of $\texttt{match}_\star$'s compilation splits on a scrutinee, producing a $\texttt{match}_1$.

$$
\begin{array}{l}
\texttt{match}_\star\ (t, u)\ \texttt{as}\ (v_1 : \mathbb{V}(A, n), v_2 : \mathbb{V}(B, n)) \\
\quad \texttt{return}\ \mathbb{V}(C, n) \\
\texttt{with} \\
\mid \texttt{Nil}(A'), \texttt{Nil}(B') \mapsto \texttt{Nil}(C) \\
\mid \texttt{Cons}(A', n', a, v_1'), \texttt{Cons}(B', n', b, v_2') \mapsto \texttt{Cons}(C, n', \mathsf{f}\ a\ b, \mathsf{map}_2\ \mathsf{f}\ n'\ v_1'\ v_2')
\end{array}
$$

Compiling $\texttt{match}_\star$: generalization

Each step of $\texttt{match}_\star$'s compilation splits on a scrutinee, producing a $\texttt{match}_1$.

$$
\begin{array}{l}
\texttt{match}_\star\ (t, u)\ \texttt{as}\ (v_1 : \mathbb{V}(A, n), v_2 : \mathbb{V}(B, n)) \\
\quad \texttt{return}\ \mathbb{V}(C, n) \\
\texttt{with} \\
\mid \texttt{Nil}(A'), \texttt{Nil}(B') \mapsto \texttt{Nil}(C) \\
\mid \texttt{Cons}(A', n', a, v_1'), \texttt{Cons}(B', n', b, v_2') \mapsto \texttt{Cons}(C, n', \mathsf{f}\ a\ b, \mathsf{map}_2\ \mathsf{f}\ n'\ v_1'\ v_2')
\end{array}
$$

compiles to

$$
\begin{array}{l}
\texttt{match}_1\ t\ \texttt{as}\ v_1 : \mathbb{V}(A, n) \\
\quad \texttt{return}\ \mathbb{V}(C, n) \\
\texttt{with} \\
\mid \texttt{Nil}(A') \mapsto \\
\mid \texttt{Cons}(A', n', a, v_1') \mapsto
\end{array}
$$

Compiling $\texttt{match}_\star$: generalization

Each step of $\texttt{match}_\star$'s compilation splits on a scrutinee, producing a $\texttt{match}_1$.

$$
\begin{aligned}
&\texttt{match}_\star\ (t, u)\ \texttt{as}\ (v_1 : \mathbb{V}(A, n), v_2 : \mathbb{V}(B, n))\\
&\quad \texttt{return}\ \mathbb{V}(C, n)\\
&\texttt{with}\\
&\mid \texttt{Nil}(A'), \texttt{Nil}(B') \mapsto \texttt{Nil}(C)\\
&\mid \texttt{Cons}(A', n', a, v_1'), \texttt{Cons}(B', n', b, v_2') \mapsto \texttt{Cons}(C, n', \mathsf{f}\ a\ b, \mathsf{map}_2\ \mathsf{f}\ n'\ v_1'\ v_2')
\end{aligned}
$$

compiles to

$$
\begin{aligned}
&\texttt{match}_1\ t\ \texttt{as}\ v_1 : \mathbb{V}(A, n)\\
&\quad \boxed{\texttt{over}\ \ (B := B : \square, v_2 := u : \mathbb{V}(B, n))}\\
&\quad \texttt{return}\ \mathbb{V}(C, n)\\
&\texttt{with}\\
&\mid \texttt{Nil}(A') \mapsto \left\{ \boxed{\texttt{match}_\star\ (v_2,)\ \texttt{as}\ (v_2 : \mathbb{V}(B, \texttt{O}),)} \right.\\
&\mid \texttt{Cons}(A', n', a, v_1') \mapsto \left\{ \boxed{\texttt{match}_\star\ (v_2,)\ \texttt{as}\ (v_2 : \mathbb{V}(B, S(n')),)} \right.
\end{aligned}
$$

Compiling $\mathtt{match}_\star$: generalization

Each step of $\mathtt{match}_\star$'s compilation splits on a scrutinee, producing a $\mathtt{match}_1$.

$$
\begin{array}{l}
\mathtt{match}_\star\ (t, u)\ \mathtt{as}\ (v_1 : \mathbb{V}(A, n), v_2 : \mathbb{V}(B, n)) \\
\quad \mathtt{return}\ \mathbb{V}(C, n) \\
\mathtt{with} \\
\mid \mathtt{Nil}(A'), \mathtt{Nil}(B') \mapsto \mathtt{Nil}(C) \\
\mid \mathtt{Cons}(A', n', a, v_1'), \mathtt{Cons}(B', n', b, v_2') \mapsto \mathtt{Cons}(C, n', \mathsf{f}\ a\ b, \mathsf{map}_2\ \mathsf{f}\ n'\ v_1'\ v_2')
\end{array}
$$

compiles to

$$
\begin{array}{l}
\mathtt{match}_1\ t\ \mathtt{as}\ v_1 : \mathbb{V}(A, n) \\
\quad \mathtt{over}\ \ (B := B : \square, v_2 := u : \mathbb{V}(B, n)) \\
\quad \mathtt{return}\ \mathbb{V}(C, n) \\
\mathtt{with} \\
\mid \mathtt{Nil}(A') \mapsto \begin{cases} \mathtt{match}_\star\ (v_2,)\ \mathtt{as}\ (v_2 : \mathbb{V}(B, \mathtt{0}),)\ \mathtt{return}\ \mathbb{V}(C, \mathtt{0})\ \mathtt{with} \\ \mid \mathtt{Nil}(B') \mapsto \mathtt{Nil}(C) \end{cases} \\
\mid \mathtt{Cons}(A', n', a, v_1') \mapsto \begin{cases} \mathtt{match}_\star\ (v_2,)\ \mathtt{as}\ (v_2 : \mathbb{V}(B, S(n')),) \\ \quad \mathtt{over}\ (v_1' := v_1' : \mathbb{V}(A', n')) \\ \quad \mathtt{return}\ \mathbb{V}(C, S(n')) \\ \mathtt{with} \\ \mid \mathtt{Cons}(B', n'', b, v_2') \mapsto \mathtt{Cons}(C, n'', \mathsf{f}(a, b), \mathsf{map}_2(\mathsf{f}, n'', v_1', v_2')) \end{cases}
\end{array}
$$

Compiling $\texttt{match}_\star$: refutations

$$\texttt{match}_\star\ (t,)\ \texttt{as}\ (l : \mathbb{V}(A, \texttt{S}(\texttt{S}(n))),)\ \texttt{return}\ \mathbb{V}(A, \texttt{S}(n))\ \texttt{with}\ \mid \texttt{Cons}(A', n', a, l') \mapsto l'$$

compiles to

$$\texttt{match}_1\ t\ \texttt{as}\ l : \mathbb{V}(A, \texttt{S}(\texttt{S}(n)))\ \texttt{return}\ \mathbb{V}(A, \texttt{S}(n))\ \texttt{with}\ \mid \texttt{Nil}(A') \mapsto \pitchfork\ \mid \texttt{Cons}(A', \texttt{S}(n'), a, l') \mapsto l'$$

Plan

Introduction

A pure language for dependent pattern matching

Compiling `match`$_\star$

Compiling `match`$_1$ using small inversion

Small inversion

For instance,

$$\begin{array}{l}\texttt{match}_1\ t\ \texttt{as}\ l : \mathbb{V}(A, \texttt{S}(\texttt{S}(n)))\\ \quad \texttt{return}\ \mathbb{V}(A, \texttt{S}(n))\\ \texttt{with}\\ \mid \texttt{Nil}(A') \mapsto \pitchfork\\ \mid \texttt{Cons}(A', \texttt{S}(n'), a, l') \mapsto l'\end{array}$$

is compiled to

$$\begin{array}{l}\texttt{match}_0\ t\ \texttt{as}\ l : \mathbb{V}(A_0, n_0)\\ \quad \texttt{return}\ \begin{pmatrix}\texttt{match}_\star\ (A_0, n_0)\ \texttt{with}\\ \mid (A, \texttt{S}(\texttt{S}(n))) \mapsto \mathbb{V}(A, \texttt{S}(n))\\ \mid x \mapsto \texttt{True}\end{pmatrix}\\ \texttt{with}\\ \mid \texttt{Nil}(A') \mapsto \texttt{I}\\ \mid \texttt{Cons}(A', n', a, l') \mapsto \begin{pmatrix}\texttt{match}_\star\ (A', n', a, l')\ \texttt{with}\\ \mid (A', \texttt{S}(n'), a, l') \mapsto l'\\ \mid x \mapsto \texttt{I}\end{pmatrix}\end{array}$$

Small inversion - general case

In general, if we have the inductive family $I\,\Omega := |_i\; C_i\,\Delta_i : I\,\omega_i$, then the term

$$
\begin{aligned}
&\texttt{match}_1\; t\; \texttt{as}\; l : I\,\varphi\; \texttt{return}\; T\; \texttt{with}\\
&|_i\; C_i\;\; \psi_i \;\;\mapsto\;\; r_i\\
&|_j\; C_j\,\Delta_j \;\;\mapsto\;\; \pitchfork
\end{aligned}
$$

is compiled to

$$
\begin{aligned}
&\texttt{match}_0\; t\; \texttt{as}\; l : I\,\Omega\\
&\quad\texttt{return}\;\begin{pmatrix}\texttt{match}_\star\; id_\Omega\; \texttt{with}\\ |\;\varphi \mapsto T\\ |\;x \mapsto \texttt{True}\end{pmatrix}\\
&\texttt{with}\\
&|_i\; C_i\,\Delta_i \mapsto \begin{pmatrix}\texttt{match}_\star\; id_{\Delta_i}\; \texttt{with}\\ |\;\psi_i \mapsto l'\\ |\;x \mapsto \texttt{I}\end{pmatrix}\\
&|_j\; C_j\,\Delta_j \mapsto \texttt{I}
\end{aligned}
$$

Remaining work

- Complete the prototype implementation
- Remaining `match`$_\star$ design questions:
  - understand the differences with the usual elaboration of toplevel functions
  - propose a precise relationship between linearity in in-clauses and K
  - Add support for forced arbitrary terms in patterns: $\mathbb{V}\ .(\mathbb{V}\ A\ S(n))\ n$
  - add a syntax for arbitrary equalities controlled by the user
- interaction with the guard condition[5]

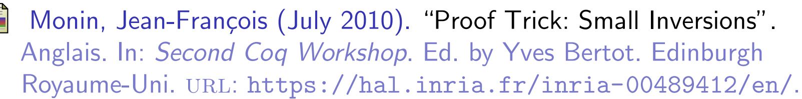
[5] Hugo Herbelin (n.d.). "How much do System T recursors lift to dependent types?" In: *TYPES 2024*.

Bibliography I

- Cockx, Jesper and Andreas Abel (July 2018). "Elaborating dependent (co)pattern matching". In: *Proc. ACM Program. Lang.* 2.ICFP. DOI: 10.1145/3236770. URL: https://doi.org/10.1145/3236770.
- Cockx, Jesper, Dominique Devriese, and Frank Piessens (2016). "Eliminating dependent pattern matching without K". In: *J. Funct. Program.* 26, e16.
- Coquand, Thierry (1992). "Pattern Matching with Dependent Types". In: *Proceedings of the 1992 Workshop on Types for Proofs and Programs*. Ed. by B. Nordström, K. Petersson, and G. Plotkin.
- Goguen, Healfdene, Conor McBride, and James McKinna (2006). "Eliminating Dependent Pattern Matching". In: *Algebra, Meaning, and Computation, Essays Dedicated to Joseph A. Goguen on the Occasion of His 65th Birthday*. Ed. by Kokichi Futatsugi, Jean-Pierre Jouannaud, and José Meseguer. Vol. 4060. Lecture Notes in Computer Science. Springer, pp. 521–540. ISBN: 3-540-35462-X. DOI: 10.1007/11780274_27.

Bibliography II

Herbelin, Hugo (n.d.). “How much do System T recursors lift to dependent types?” In: *TYPES 2024*.


Monin, Jean-François (July 2010). “Proof Trick: Small Inversions”. Anglais. In: *Second Coq Workshop*. Ed. by Yves Bertot. Edinburgh Royaume-Uni. URL: https://hal.inria.fr/inria-00489412/en/.