A Gentle Introduction to Equations

Or How to Match Regexps with Dependently-Typed Continuations

Matthieu Sozeau & Cyprien Mangin
Inria Paris & IRIF, Université Paris 7 Diderot

Saarland University
Saarbrücken
April 26th 2018

OUTLINE

1. Dependent Pattern-Matching 101
    a. History & Examples
    b. Unification
    c. Axiomatic vs Definitional Extensions
    d. Uniqueness of Identity Proofs
2. Equations Tutorial: Regexp Matching
    a. Tool presentation
    b. A simply-typed version
    c. Indexing to the rescue
    d. Boilerplate-free proofs on dependently-typed programs

DPM 101

"Pattern-Matching with Dependent Types", Coquand, 1992

The type-value dependency circle:

```
Inductive vector A : nat -> Type :=
| Vnil : vector A 0
| Vcons : forall (h:A) (n:nat), vector A n → vector A (S n).

Equations vtail {A n} (v : vector A (S n)) : vector A n :=
vtail (Vcons a n v') := v'.
```

DPM 101

"The view from the left", McBride, McKinna - JFP, 2004

The "with" rule:

```
Equations equal (n m : nat) : { n = m } + { n ≠ m } :=
equal O O := left eq_refl ;
equal (S n) (S m) with equal n m := {
  equal (S n) (S ?(n)) (left eq_refl) := left eq_refl ;
  equal (S n) (S m) (right p) := right _ } ;
equal x y := right _.
```

⇒ Later adopted in Agda 2/Epigram

DPM 101 - Splitting and Matching

When checking that a definition with clauses

```
fj pj1 … pjn := …
```

covers the typing:

```
f : ∀ (x1 : I u1 … un) (x2 : τ2) … (xn : τn), τ
```

We **match**`(p1 … pn, x1 … xn)`

```
match(p1..pn, t1..tn) ⇝ Unif σ   (σ substitution)
                    | Fail | Stuck x        (x variable)
```

- Standard first-match semantics
- Stuck `xi` tells which variable to split next, refining `xi` to constructors

DPM 101 - Unification

When checking that a clause

```
f (C_i p1 .. pn) q2 .. qm := …
```

matches the prototype:

```
f : ∀ (x1 : I u1 .. un) (x2 : τ2)…(xn : τn), τ
```

we **unify:** `I t1 .. tn` and `I u1 .. un`

```
t =? u ⇝ Unif σ        (σ substitution)
       | Fail | Stuck
```

Let's write the rules on the whiteboard!

DPM 101 - Axiomatic vs Definitional Extension

- **Axiomatic** extension: matching and unification part of the kernel (Alfa, Agda 2), trusted

- **Definitional** extension: compile pattern-matching to primitive eliminators (Epigram, Equations, Lean, GHC) E.g. in Coq:

```
Definition vtail (x : vector A (S n)) : vector A n :=
  match x in vec y return match y with 0 => unit | S n => vector A n end with
  | Vnil => tt
  | Vcons a n v => v
  end
```

This kind of "small" inversions is however limited (linearity, pure pattern structure, limited dependencies)

DPM 101 - Compilation

"Eliminating Dependent Pattern-Matching"
Goguen, McBride & McKinna, 2006

Main Idea: unification

```
t =? u ⇝ Unif σ
```

is **witnessed** by a proof of:

```
t = u → |σ|
```

where `|x:=t;σ| ≡ Σ p : x = t, |σ[p]|`

DPM 101 - Eliminating a variable

Start with: $\Gamma_0\ (x : \mathsf{I}\ \overline{t})\ \Gamma_1$

Generalize:

$$\Gamma_0\ (x : \mathsf{I}\ \overline{t})\ \Gamma_1\ (\overline{u} : \overline{\tau})\ (y : \mathsf{I}\ \overline{u})\ (e : (\overline{t}, x) = (\overline{u}, y))$$

Eliminate:

$$\Gamma_0\ (x : \mathsf{I}\ \overline{t})\ \Gamma_1\ (\overline{a} : \overline{T_i})\ (e : (\overline{t}, x) = (\overline{u_i}, \mathsf{C_i}\ \overline{a}))$$

Simplify equalities

DPM 101 - Compilation

Supporting definitions: J-rule, NoConfusion, NoCycle, K

Specialization by Unification (McBride, PhD'99, Cockx, ICFP'14). Gradually refine/simplify:

```
t =_(Σ δ : Δ. I δ) u → τ
```

Theorem (computational soundness):

Each clause `f c1 .. cn := t` turns into a **reduction** `f c1 .. cn → t`, even on open terms.

(When there are no overlapping patterns)

DPM 101 - Simplification

Sigma:

$$\forall (e : (x, p) = (y, q)),\ P\ e \Rightarrow \forall (e' : x = y)\ (e : \text{rew}\ e'\ p = q),\ P\ (\text{sigma\_eq}\ e'\ e)$$

Solution:

$$\forall \Gamma,\ \forall\ (e : x = t),\ P\ x\ e \Rightarrow \forall\ \Gamma',\ P\ t\ \text{eq\_refl}$$

Deletion:

$$\forall (e : t = t),\ P\ e \Rightarrow P\ \text{eq\_refl}$$

DPM 101 - Constructor Simplification

Pack:

$$\boxed{\forall (e : \mathsf{C}\ \bar{t} = \mathsf{D}\ \bar{u}),\ \mathsf{P}\ e} \Rightarrow \boxed{\forall (e : (\overline{idx_t},\ \mathsf{C}\ \bar{t}) = (\overline{idx_u},\ \mathsf{D}\ \bar{u})),\ \mathsf{ind\_pack\_inv}\ \mathsf{P}\ e}$$

NoConfusion:

$$\boxed{\forall (e : (\overline{idx_t},\ \mathsf{C}\ \bar{t}) = (\overline{idx_u},\ \mathsf{D}\ \bar{u})),\ \mathsf{P}\ e}$$

$$\Rightarrow \boxed{\begin{array}{l} - \ \forall\ (e : \bar{t} = \bar{u}),\ \mathsf{P}\ (\mathsf{noConf\_inv}\ e) \text{ if C and D are the same constructor} \\ - \ \text{solved if C and D are distinct constructors} \end{array}}$$

Unpack:

$$\boxed{\mathsf{ind\_pack\_inv}\ \mathsf{P}\ \mathsf{eq\_refl}} \Rightarrow \boxed{\mathsf{P}\ \mathsf{eq\_refl}}$$

True and False:

$$\boxed{\forall (e : \mathsf{True}),\ \mathsf{P}\ e} \Rightarrow \boxed{\mathsf{P}\ \mathsf{I}} \qquad \boxed{\forall (e : \mathsf{False}),\ \mathsf{P}\ e} \Rightarrow \boxed{\text{solved}}$$

DPM 101 - Uniqueness of Identity Proofs

- Supported by the unification
- Incompatible with Homotopy Type Theory models
- Can remove the “deletion” rule to disallow it.
- More refined solutions possible (Cockx et al, ICFP’14, CPP’17)
- Equations allow configuration of behavior

OUTLINE

1. Dependent Pattern-Matching 101
   a. History & Examples
   b. Unification
   c. Axiomatic vs Definitional Extensions
   d. Uniqueness of Identity Proofs
2. **Equations Tutorial: Regexp Matching**
   a. **Tool presentation**
   b. **A simply-typed version**
   c. **Indexing to the rescue**
   d. **Boilerplate-free proofs on dependently-typed programs**

Equations Reloaded

- Dependent Pattern-Matching à la Epigram, Agda
- Compiled-down to CIC using **telescope** simplification (à la Cockx circa 2016)
- Optional typeclass instances of K/decidable equality
- **Smart** case compilation: small proof terms, avoiding UIP
- Structural, nested and well-founded recursion (i.e. more than what Function/Program can handle)
- `Derive Signature NoConfusion Subterm EqDec for I`
- Generates graph, unfolding lemma, **elimination** principles

Playtime: Regexp matching

Implement regexp matching using continuations instead of derivatives or automata.

"Proof-directed debugging", Harper, JFP'99

Tutorial

More examples

- Hereditary substitution for Predicative System F (Mangin & Sozeau, LFMTP'15)
  Nested recursion, well-founded multiset ordering on types.
- Ordinal measures (Castéran)
- Reflexive ring-like tactic on polynomials. WF subterm order on indexed polynomials
- Prototyping without verifying termination using functional eliminator

mattam82.github.io/Coq-Equations/examples

Equations Summary

- Write **just what's needed** when programming with dependently-typed structures.
- **Definitional** extension of the Coq kernel
- Gives the **right** reasoning **principles** on your (mutual, nested, dependent) function.
- Good **target** for verification of total, purely functional **Haskell** programs (e.g. hs-to-coq, UPenn).

opam install coq-equations

http://mattam82.github.io/Coq-Equations/