Pattern matching without K

Jesper Cockx
Dominique Devriese
Frank Piessens

DistriNet – KU Leuven

13 May 2014

How can we recognize definitions by pattern matching that do not depend on K?

By taking identity proofs into account during unification of the indices!

How can we recognize definitions by pattern matching that do not depend on K?

By taking identity proofs into account during unification of the indices!

Pattern matching without K

1. Dependent pattern matching
2. The K axiom
3. Translation to eliminators
4. Proof-relevant unification

Pattern matching without K

1. Dependent pattern matching
2. The K axiom
3. Translation to eliminators
4. Proof-relevant unification

Simple pattern matching

data $\mathbb{N}$: `Set` **where**
  `z` : $\mathbb{N}$
  `s` : $\mathbb{N} \to \mathbb{N}$

$$\begin{array}{llllll}\texttt{min} : & \mathbb{N} & \to & \mathbb{N} & \to & \mathbb{N} \\ \texttt{min} & x & & y & = & ?\end{array}$$

Simple pattern matching

data $\mathbb{N}$: `Set` **where**
  `z` : $\mathbb{N}$
  `s` : $\mathbb{N} \to \mathbb{N}$

`min` : $\mathbb{N} \to \mathbb{N} \to \mathbb{N}$
`min` `z` y = `z`
`min` (`s` x) y = ?

Simple pattern matching

$$
\begin{array}{l}
\textbf{data}\ \mathbb{N} : \texttt{Set}\ \textbf{where} \\
\quad \texttt{z} : \mathbb{N} \\
\quad \texttt{s} : \mathbb{N} \to \mathbb{N}
\end{array}
$$

$$
\begin{array}{lllll}
\texttt{min} : & \mathbb{N} \to & \mathbb{N} & \to & \mathbb{N} \\
\texttt{min} & \texttt{z} & y & = & \texttt{z} \\
\texttt{min} & (\texttt{s}\ x) & \texttt{z} & = & \texttt{z} \\
\texttt{min} & (\texttt{s}\ x) & (\texttt{s}\ y) & = & \texttt{s}\ (\texttt{min}\ x\ y)
\end{array}
$$

Dependent pattern matching

data $\_ \leq \_ : \mathbb{N} \to \mathbb{N} \to \texttt{Set}$ **where**

$\texttt{lz} : (n : \mathbb{N}) \to \texttt{z} \leq n$

$\texttt{ls} : (m\ n : \mathbb{N}) \to m \leq n \to \texttt{s}\ m \leq \texttt{s}\ n$

$\texttt{antisym} : (x\ y : \mathbb{N}) \to x \leq y \to y \leq x \to x \equiv y$

$\texttt{antisym}\ \ x\ \ y\ \ p\ \ q\ = \ ?$

Dependent pattern matching

$$\textbf{data}\ \_ \leq \_ : \mathbb{N} \to \mathbb{N} \to \texttt{Set}\ \textbf{where}$$
$$\quad \texttt{lz} : (n : \mathbb{N}) \to \texttt{z} \leq n$$
$$\quad \texttt{ls} : (m\ n : \mathbb{N}) \to m \leq n \to \texttt{s}\ m \leq \texttt{s}\ n$$

$$\begin{array}{lllllll}
\texttt{antisym} : (x\ y : \mathbb{N}) \to & x \leq y \to & & y \leq x & \to & x \equiv y \\
\texttt{antisym}\ \lfloor \texttt{z} \rfloor & \lfloor y \rfloor & (\texttt{lz}\ y) & q & = & ? \\
\texttt{antisym}\ \lfloor \texttt{s}\ x \rfloor & \lfloor \texttt{s}\ y \rfloor & (\texttt{ls}\ x\ y\ p) & q & = & ?
\end{array}$$

Dependent pattern matching

```
data _≤_ : ℕ → ℕ → Set where
  lz : (n : ℕ) → z ≤ n
  ls : (m n : ℕ) → m ≤ n → s m ≤ s n

antisym : (x y : ℕ) → x ≤ y →    y ≤ x      → x ≡ y
antisym   ⌊z⌋   ⌊z⌋   (lz ⌊z⌋)   (lz ⌊z⌋) =  refl
antisym   ⌊s x⌋ ⌊s y⌋ (ls x y p) q        =   ?
```

Dependent pattern matching

```
data _≤_ : ℕ → ℕ → Set where
  lz : (n : ℕ) → z ≤ n
  ls : (m n : ℕ) → m ≤ n → s m ≤ s n

antisym : (x y : ℕ) → x ≤ y →    y ≤ x → x ≡ y
antisym   ⌊z⌋   ⌊z⌋   (lz ⌊z⌋)   (lz ⌊z⌋) = refl
antisym   ⌊s x⌋ ⌊s y⌋ (ls x y p) (ls ⌊y⌋ ⌊x⌋ q)
    = cong s (antisym x y p q)
```

Pattern matching without K

1. Dependent pattern matching
2. The K axiom
3. Translation to eliminators
4. Proof-relevant unification

The identity type as an inductive family

```
data _≡_ (x : A) : A → Set where
  refl : x ≡ x

trans : (x y z : A) → x ≡ y → y ≡ z → x ≡ z
trans x ⌊x⌋ ⌊x⌋ refl refl = refl
```

The identity type as an inductive family

```
data _≡_ (x : A) : A → Set where
  refl : x ≡ x

trans : (x y z : A) → x ≡ y → y ≡ z → x ≡ z
trans x ⌊x⌋ ⌊x⌋ refl refl = refl
```

K follows from pattern matching

$$
\begin{aligned}
&\mathtt{K} : (P : a \equiv a \to \mathtt{Set}) \to \\
&\qquad (p : P\ \mathtt{refl}) \to \\
&\qquad (e : a \equiv a) \to P\ e \\
&\mathtt{K} \quad P\ p\ \mathtt{refl} = p
\end{aligned}
$$

We don't always want to assume K

K is incompatible with univalence:

- K implies that `subst` e `true` $=$ `true` for all e : `Bool` $\equiv$ `Bool`
- Univalence gives `swap` : `Bool` $\equiv$ `Bool` such that `subst swap true` $=$ `false`

hence `true` $=$ `false`!

The –without-K flag in Agda

- When making a case split, the indices must be applications of constructors to distinct variables (constructor parameters are treated as other arguments).
- These distinct variables must not be free in the parameters.

New specification of –without-K

- It is not allowed to delete reflexive equations.
- When applying injectivity on an equation $\mathtt{c}\ \bar{s} = \mathtt{c}\ \bar{t}$ of type $\mathtt{D}\ \bar{u}$, the indices $\bar{u}$ should be *self-unifiable*.

Pattern matching without K

Eliminating dependent pattern matching

1. Basic case analysis:
   Translate each case split to an eliminator.
2. Specialization by unification:
   Solve the equations on the indices.
3. Structural recursion:
   Fill in the recursive calls.

Specialization by unification

$$x \simeq x, \Delta \Rightarrow \Delta \qquad \text{(Deletion)}$$

$$t \simeq x, \Delta \Rightarrow \Delta[x \mapsto t] \qquad \text{(Solution)}$$

$$\mathsf{c}\ \bar{s} \simeq \mathsf{c}\ \bar{t}, \Delta \Rightarrow \bar{s} \simeq \bar{t}, \Delta \qquad \text{(Injectivity)}$$

$$\mathsf{c}_1\ \bar{s} \simeq \mathsf{c}_2\ \bar{t}, \Delta \Rightarrow \bot \qquad \text{(Conflict)}$$

$$x \simeq \mathsf{c}\ \bar{p}[x], \Delta \Rightarrow \bot \qquad \text{(Cycle)}$$

```
antisym : (m n : ℕ) → m ≤ n → n ≤ m → m ≡ n
antisym = elim≤ (λm; n; _ . n ≤ m → m ≡ n)
  (λn; e. elim≤ (λn; m; _ . m ≡ z → m ≡ n)
    (λn; e. e)
    (λk; l; _; _; e. elim⊥(λ_ . s l ≡ s k)
      (noConfℕ (s l) z e))
    n z e refl)
  (λm; n; _; H; q. cong s
    (H
      (elim≤ (λk; l; _ . k ≡ s n → l ≡ s m → n ≤ m)
        (λ_; e; _. elim⊥ (λ_ . n ≤ m)
          (noConfℕ z (s n) e))
        (λk; l; e; _; p; q. subst (λn. n ≤ m)
          (noConfℕ (s k) (s n) p)
          (subst (λm. k ≤ m)
            (noConfℕ (s l) (s m) q) e))
        (s n) (s m) q refl refl)))
```

Pattern matching without K

1. Dependent pattern matching
2. The K axiom
3. Translation to eliminators
4. Proof-relevant unification

Heterogeneous equality

$$\frac{a : A \qquad b : B}{a \simeq b : \texttt{Set}} \qquad\qquad \frac{a : A}{\texttt{refl} : a \simeq a}$$

$$\texttt{eqElim} : (x\ y : A) \to (e : x \simeq y) \to$$
$$D\ x\ \texttt{refl} \to D\ y\ e$$

This elimination rule is equivalent with K ...

Homogeneous telescopic equality

We can use the first equality proof
to fix the types of the following equations.

$$a_1, a_2 \equiv b_1, b_2$$
$$\Downarrow$$
$$(e_1 : a_1 \equiv b_1)(e_2 : \texttt{subst}\ e_1\ a_2 \equiv b_2)$$

Deletion

$$x \simeq x, \Delta \Rightarrow \Delta$$

$$\Downarrow$$

$$\mathtt{e} : x \equiv x, \Delta \Rightarrow \Delta[\mathtt{e} \mapsto \mathtt{refl}]$$

Solution

$$t \simeq x, \Delta \Rightarrow \Delta[x \mapsto t]$$

$$\Downarrow$$

$$\texttt{e} : t \equiv x, \Delta \Rightarrow \Delta[x \mapsto t, \texttt{e} \mapsto \texttt{refl}]$$

Injectivity

$$\mathsf{c}\ \bar{s} \simeq \mathsf{c}\ \bar{t}, \Delta \Rightarrow \quad \bar{s} \simeq \bar{t}, \Delta$$

$$\Downarrow$$

$$\mathsf{e} : \mathsf{c}\ \bar{s} \equiv \mathsf{c}\ \bar{t}, \Delta \Rightarrow \bar{\mathsf{e}} : \bar{s} \equiv \bar{t}, \Delta[\mathsf{e} \mapsto \mathtt{conf}\ \bar{\mathsf{e}}]$$

Conflict

$$\mathsf{c}_1\ \bar{u} \simeq \mathsf{c}_2\ \bar{v}, \Delta \Rightarrow \bot$$

$$\Downarrow$$

$$\mathsf{e} : \mathsf{c}_1\ \bar{s} \equiv \mathsf{c}_2\ \bar{t}, \Delta \Rightarrow \bot$$

Cycle

$$x \simeq \mathsf{c}\ \bar{p}[x], \Delta \Rightarrow \bot$$

$$\Downarrow$$

$$\mathsf{e} : x \equiv \mathsf{c}\ \bar{p}[x], \Delta \Rightarrow \bot$$

Future work

- Detecting types that satisfy K (i.e. sets)
- Implementing the translation to eliminators
- Extending pattern matching
  to higher inductive types

Future work

- Detecting types that satisfy K (i.e. sets)
- Implementing the translation to eliminators
- Extending pattern matching
  to higher inductive types

Future work

- Detecting types that satisfy K (i.e. sets)
- Implementing the translation to eliminators
- Extending pattern matching
  to higher inductive types

Conclusion

By restricting the unification algorithm,
we can make sure that K is never used.

You no longer have to worry
when using pattern matching for HoTT!

http://people.cs.kuleuven.be/
∼ jesper.cockx/Without-K/

Standard library without K
Fixable errors: 16

Module	Functions
Algebra.RingSolver	$\stackrel{?}{=}$H, $\stackrel{?}{=}$N
Data.Fin.Properties	drop-suc
Data.Vec.Equality	trans, $\stackrel{?}{=}$
Data.Vec.Properties	::-injective, ...
Relation.Binary.Vec.Pointwise	head, tail
Data.Fin.Subset.Properties	drop-there, $\notin\bot$, ...
Data.Fin.Dec	$\in$?
Data.List.Countdown	drop-suc

Unfixable/unknown errors: 20

Module	Functions
Relation.Binary.	
HeterogeneousEquality	$\cong$-to-$\equiv$, subst, cong, ...
PropositionalEquality	proof-irrelevance
Sigma.Pointwise	Rel$\leftrightarrow\equiv$, inverse
Data.	
Colist	Any-cong, $\sqsubseteq$-Poset
Covec	setoid
Container.Indexed	setoid, natural, $\circ$-correct
List.Any.BagAndSetEquality	drop-cons
Star.Decoration	gmapAll, $\triangleleft\triangleleft\triangleleft$
Star.Pointer	lookup
Vec.Properties	proof-irrelevance-[]=

Why deletion has to be disabled

$\mathtt{UIP} : (e : a \equiv a) \to e \equiv \mathtt{refl}$

$\mathtt{UIP}\ \ \underline{\mathtt{refl}} = \mathtt{refl}$

Couldn't solve reflexive equation $a = a$ of type A because K has been disabled.

Why injectivity has to be restricted

$\mathtt{UIP'} : (e : \mathtt{refl} \equiv_{a \equiv a} \mathtt{refl}) \to e \equiv \mathtt{refl}$

$\mathtt{UIP'}\ \ \underline{\mathtt{refl}} = \mathtt{refl}$

Couldn't solve reflexive equation $a = a$ of type A because K has been disabled.