Higher-Order Dynamic Pattern Unification for Dependent Types and Records

Andreas Abel
(joint work with Brigitte Pientka)

Department of Computer Science
Ludwig-Maximilians-University Munich

Typed Lambda Calculi and Applications
Novi Sad, Serbia
1 June 2011

A little anniversary...

My 5th consecutive TLCA paper!

- 2003 Valencia
- 2005 Nara
  with Thierry Coquand
- 2007 Paris
- 2009 Brasilia
  with Thierry Coquand
  and Miguel Pagano
- 2011 Novi Sad
  with Brigitte Pientka


FIVE YEARS

Miller pattern unification

- Hidden arguments and placeholders _ create unification problems.

```
data Sigma (A : Set)(B : A → Set) : Set where
  _,_ : (a : A) → B a → Sigma A B

pair : (X : Set)(Y : X → Set)(x : X)(y : Y x) → Sigma X Y
pair X Y x y = x , y
```

- Look behind the curtain with `{-# OPTIONS -v tc.meta:20 #-}`
- Making hidden arguments explicit:

```
_,_ : {A : Set}{B : A → Set}(a : A) → B a → Sigma A B
pair X Y x y = _,_ {_} {_} x y
_A : (X : Set)(Y : X → Set)(x : X)(y : Y x) → Set
_B : (X : Set)(Y : X → Set)(x : X)(y : Y x) → _A X Y x y
                                            → Set
pair X Y x y = _,_ {_A X Y x y} {_B X Y x y} x y
```

Generating and solving constraints

```
pair X Y x y = _,_ {_A X Y x y} {_B X Y x y} x y

check  _,_ {_A X Y x y} {_B X Y x y} x y : Sigma X Y

check  x : _A X Y x y
infer  x : X    problem  _A X Y x y   = X    solved (linear)

check  y : _B X Y x y x
infer  y : Y x  problem  _B X Y x y x = Y x  postponed (¬lin.)

infer  _,_ X {_B ...} x y : Sigma X (_B ...)
problem  _B X Y x y   = Y   : X -> Set
problem  _B X Y x y z = Y z : Set           solved
revisit  _B X Y x y x = Y x                 solved
```

Extending Miller patterns: Inessential non-linearity

- Agda issue 323 (`test/succeed/Issue323.agda`)

```
_,_ : {A : Set}{B : A → Set}(a : A) → B a → Sigma A B

data Sing {A : Set}(a : A) : Set where
  sing : Sing a

singPair : (C : Set)(x y : C) → Sing (_,_ {_A} {_B} x y)
singPair C x y = sing
```

- Check $y : (\_B\ C\ x\ y)\ x$.
- $\_B\ C\ x\ y\ x = C$ solved by $\_B = \lambda C\ x\ y\ x' \to C$.

Pruning

- In Agda; called killing.
- Example inspired by `test/succeed/FlexRemoval.agda`

```
data Wrap : Set  where  wrap : A → Wrap

data D : Set where
  c1 : D
  c2 : (x : Wrap) → (A → T x) → D

T : Wrap → Set
T (wrap y) = D

foo : D
foo = c2 _ (λ a → c1)
```

Pruning

```
check   c2 _x (λ a → c1) : D
infer   c2 _x            : (A → T _x) → D
check   λ a → c1         :  A → T _x
check     a : A ⊢ c1 : T _x
infer     a : A ⊢ c1 : D
problem   a : A ⊢ T _x = D
problem   a : A ⊢ _x = wrap (_y a)
occurs check fails
prune     _y = λ a → _z
solve     _x = wrap _z
```

Eta-contract towards Miller pattern

- Agda's definitional equality is $\beta\eta$.
- Goal: unification modulo $\beta\eta$.
- `test/succeed/EtaContractToMillerPattern.agda`

```
test : let X : (A × B → C) → (A × B → C)
           X = _
       in (x : A × B → C) →
            X (λ z → x (fst z , snd z)) ≡ x
test x = refl
```

- Problem turned into Miller pattern by η-reduction

$$\begin{array}{rll} & \_X\ (\lambda z \to x\ (\mathsf{fst}\, z\, , \mathsf{snd}\, z)) & = \quad x \\ \longrightarrow_\eta & \_X\ (\lambda z \to x\, z) & = \quad x \\ \longrightarrow_\eta & \_X\ x & = \quad x \end{array}$$

Expanding record meta variables

- $\Gamma \to (A \times B) \cong (\Gamma \to A) \times (\Gamma \to B)$ [Norell's thesis, p.71]
- See `test/succeed/ProjectingRecordMeta.agda`

```
test : (y z : A × B) →
  let X : A × B
      X = _
  in (fst X ≡ fst y) × (snd X ≡ snd z)
test y z = (refl , refl)

problem  fst (_X y z) = fst y
         snd (_X y z) = snd z
expand   _X = λ y z → (_Y y z , _Z y z)
solve    _Y y z = fst y
         _Z y z = snd z
```

- Strategy: projection applied to meta variable triggers expansion.

New: Eliminating Projections

- Agda cannot solve this:

```
test : let X : A → B → A × B
           X = _
       in (z : A × B) →
            X (fst z) (snd z) ≡ z
test z = refl
```

- Replace bound variable z by (x, y) for new bound variables x, y.

$$\begin{array}{rlll} & X\ (\mathsf{fst}\ z)\ (\mathsf{snd}\ z) & = & z \\ \rightsquigarrow & X\ x\ y & = & (x, y) \end{array}$$

New: Flattening Record Types

- Agda cannot solve this:

```
test : let X : A × B → A
           X = _
       in (x : A) → (y : B) →
            X (x , y) ≡ x
test x y = refl
```

- Use isomorphism $A \times B \to A \cong A \to B \to A$.
- Set $X\ z = X'\ (\mathsf{fst}\ z)\ (\mathsf{snd}\ z)$.

$$\begin{array}{rrcl} & X\ (x, y) & = & x \\ \rightsquigarrow & X'\ x\ y & = & x \end{array}$$

Code sprint proposal

- Overhaul unification:
  - Freeze unsolved meta-variables at end of declaration.
  - Separate pruning from solving.
  - Restrict η-contraction to Miller patterns.
- Extend pattern unification to records:
  - Implement missing isomorphism
    $(z : \Sigma x{:}A.\, B\, x) \to C\, z \cong (x : A) \to (y : B\, x) \to C\, (x, y)$.
  - Eliminating projections.
  - Flattening record types in type of meta.

References

- Dale Miller, *Unification Under a Mixed Prefix*, HOSC 1992
- Ulf Norell, *Towards a Practical Programming Language Based on Dependent Type Theory*, 2007
- Jason Reed, *Higher-Order Constraint Simplification in Dependent Type Theory*, LFMTP 2009