Pattern Inductive Familie Need Not Store Their Indices, Ever

Or: Polymorphic Type Theory, with an Application to Term Compression

Andreas Abel[1]

[1]Department of Computer Science and Engineering
Chalmers and Gothenburg University, Sweden

Programming Logic Seminar
Chalmers, Gothenburg, Sweden
13 November 2013

Terminology: Datatype Parameters and Indices

- Type of vectors: parameter A, index n.

```
data Vec (A : Set) : (n : ℕ) → Set where
  []     :                                  Vec A 0
  _::_   : {n : ℕ} (x : A) (xs : Vec A n) → Vec A (suc n)
```

- Strict parameter: stays fixed, abstracted outside.
- Equivalent definition using a section:

```
module _ (A : Set) where
  data Vec  : (n : ℕ) → Set where
    []      :                                Vec 0
    _::_    : {n : ℕ} (x : A) (xs : Vec n) → Vec (suc n)
```

Relaxing Parameters

- Flexible parameters: can change in recursive occurrences, but not in target.

$$
\begin{array}{l}
\mathsf{data}\ \mathsf{Tm}\ (\Gamma : \mathsf{Cxt})\ (C : \mathsf{Ty}) : \mathsf{Set}\ \mathsf{where} \\
\quad \mathsf{var}\ : (x : \mathsf{Var}\ \Gamma\ C) \to \mathsf{Tm}\ \Gamma\ C \\
\quad \mathsf{app}\ : \forall\ \{A\}\ (r : \mathsf{Tm}\ \Gamma\ (A \Rightarrow C))\ (s : \mathsf{Tm}\ \Gamma\ A) \to \mathsf{Tm}\ \Gamma\ C \\
\quad \mathsf{abs}\ : \forall\ \{A\ B\}\ (t : \mathsf{Tm}\ (A :: \Gamma)\ B) \to \mathsf{Tm}\ \Gamma\ (A \Rightarrow B)
\end{array}
$$

- Γ is strict, C is flexible.
- Syntactic terminology: content of the parameter telescope.
- With abs, flexible parameter C must be turned into an index.

Pattern Inductive Families

- Target parameters maybe arbitrary patterns.

```
data Tm (_ : Cxt) (_ : Ty) : Set where
  var  : (x : Var Γ A)                          → Tm Γ A
  app  : ∀ {A} (r : Tm Γ (A ⇒ B)) (s : Tm Γ A)  → Tm Γ B
  abs  : (t : Tm (A :: Γ) B)                    → Tm Γ (A ⇒ B)
```

- A and B are bound by the target.
- Constructors: Function type notation misleading, use telescopes!

```
data Tm (_ : Cxt) (_ : Ty) : Set where
  var  (x : Var Γ A)                          : Tm Γ A
  app  {A} (r : Tm Γ (A ⇒ B)) (s : Tm Γ A)    : Tm Γ B
  abs  (t : Tm (A :: Γ) B)                    : Tm Γ (A ⇒ B)
```

Conclusions

Related Work