Category theory to the rescue of type checking

Guilhem Moulin

February 15, 2010

Simply typed λ-calculus: types and terms

```
1  data Ty : Set where
2    ⋆     : Ty
3    _⇒_ : Ty → Ty → Ty

5  Cxt : Set
6  Cxt = List Ty

8  data Tm : Set where
9    Var : Nat → Tm
10   _@_ : Tm → Tm → Tm
11   λ     : Tm → Tm
```

Simply typed λ-calculus: typing rules

$$\frac{}{\Gamma_1, \ldots, \Gamma_n \vdash \Gamma_i} \qquad \frac{\Gamma \vdash a \Rightarrow b \qquad \Gamma \vdash a}{\Gamma \vdash b}$$

$$\frac{\Gamma, a \vdash b}{\Gamma \vdash a \Rightarrow b}$$

Simply typed λ-calculus: typing rules

$$\frac{}{\texttt{Var}\ i : \Gamma_1, \ldots, \Gamma_n \vdash \Gamma_i} \qquad \frac{f : \Gamma \vdash a \Rightarrow b \qquad x : \Gamma \vdash a}{f\ @\ x : \Gamma \vdash b}$$

$$\frac{e : \Gamma, a \vdash b}{\lambda e : \Gamma \vdash a \Rightarrow b}$$

Simply typed λ-calculus: typing rules

$$\frac{}{\texttt{Var}\ i : \Gamma_1, \ldots, \Gamma_n \vdash \Gamma_i} \qquad \frac{f : \Gamma \vdash a \Rightarrow b \qquad x : \Gamma \vdash a}{f\ @\ x : \Gamma \vdash b}$$

$$\frac{e : \Gamma, a \vdash b}{\lambda e : \Gamma \vdash a \Rightarrow b}$$

➥ `"_:  _ ⊢ _":  Cxt → Tm → Ty → Prop`

Simply typed λ-calculus: type-checking algorithm

Decidability of typing:

```
decTS : (Γ : Cxt) → (a : Ty) → (t : Tm)
                  → (t : Γ ⊢ a)  ∨  Not (t : Γ ⊢ a)
```

Simply typed λ-calculus: type-checking algorithm

Decidability of typing:

```
decTS : (Γ : Cxt) → (a : Ty) → (t : Tm)
                  → (t : Γ ⊢ a)  ∨  Not (t : Γ ⊢ a)
```

Perform *program extraction*:

```
isTm : Cxt → Ty → Tm → Bool
```

Simply typed λ-calculus: type-checking algorithm

Decidability of typing:

```
decTS : (Γ : Cxt) → (a : Ty) → (t : Tm)
                  → (t : Γ ⊢ a)  ∨  Not (t : Γ ⊢ a)
```

Perform *program extraction*:

```
isTm : Cxt → Ty → Tm → Bool
```

Correctness of type-checking:

```
corrTC : (Γ : Cxt) → (a : Ty) → (t : Tm)
                   → t : Γ ⊢ a  ↔  isTm Γ a t = true
```

What is dependent typed λ-calculus?

- *dependent* function types: $(x : A) \Rightarrow B(x)$ rather than $A \Rightarrow B$
- universe `Set` of "sets" (small types)

What is dependent typed λ-calculus?

- *dependent* function types: $(x : A) \Rightarrow B(x)$ rather than $A \Rightarrow B$
- universe `Set` of "sets" (small types)

Here is a Coq example:

```
1 Inductive Vect (A:Set): ∀ n:nat, Set :=
2   | nil  : Vect A 0
3   | cons : ∀ n, A → Vect A n → Vect A (n+1).
```

What is dependent typed λ-calculus?

- *dependent* function types: $(x : A) \Rightarrow B(x)$ rather than $A \Rightarrow B$
- universe `Set` of "sets" (small types)

Here is a Coq example:

```
1 Inductive Vect (A:Set): ∀ n:nat, Set :=
2   | nil  : Vect A 0
3   | cons : ∀ n, A → Vect A n → Vect A (n+1).
```

- computation of types: may get a loop. *We restrict to normal types*

Decidability of the typing relation for dependent types

Very important:

Proof assistants type-checking algorithm = core of proof assistants like Coq or Agda

Constructive foundations the decidability of $a : A$ is fundamental in constructivism based on Curry-Howard

Decidability of the typing relation for dependent types

Very important:

Proof assistants type-checking algorithm = core of proof assistants like Coq or Agda

Constructive foundations the decidability of $a : A$ is fundamental in constructivism based on Curry-Howard

But it has been difficult to obtain clear correctness proofs.
Success approaches: *normalisation by evaluation* and *category-theoretic models* of dependent types.

Dependent typed λ-calculus: syntax

Normal and neutral *raw* terms:

$$\mathtt{no}_n \coloneqq \mathtt{ne}_n \mid \lambda\ \mathtt{no}_{n+1} \mid \Pi\ \mathtt{no}_n\ \mathtt{no}_{n+1} \mid U$$
$$\mathtt{ne}_n \coloneqq \mathtt{Var}\ i\ (i < n) \mid \mathtt{ne}_n\ @\ \mathtt{no}_n$$

Dependent typed λ-calculus: syntax

Normal and neutral *raw* terms:

$$\mathtt{no}_n \coloneqq \mathtt{ne}_n \mid \lambda\ \mathtt{no}_{n+1} \mid \Pi\ \mathtt{no}_n\ \mathtt{no}_{n+1} \mid U$$
$$\mathtt{ne}_n \coloneqq \mathtt{Var}\ i\ (i < n) \mid \mathtt{ne}_n\ @\ \mathtt{no}_n$$

Contexts and substitutions:

$$\mathtt{Cxt}_0 \coloneqq [\,] \qquad \mathtt{Subst}_{m,0} \coloneqq \langle\rangle_m$$
$$\mathtt{Cxt}_{n+1} \coloneqq (\mathtt{Cxt}_n; \mathtt{no}_{n+1}) \qquad \mathtt{Subst}_{m,n+1} \coloneqq \langle \mathtt{Subst}_{m,n}, \mathtt{no}_m \rangle$$

Dependent typed λ-calculus: some rules

$$\frac{\Gamma : \mathtt{Cxt}_n \qquad A : \mathrm{Type}_\Gamma}{(\Gamma; A) : \mathtt{Cxt}_{n+1}}$$

$$\frac{\Gamma, \Delta : \mathtt{Cxt} \qquad A : \mathrm{Type}_\Gamma \qquad \gamma : \Gamma \to \Delta \qquad a : \Delta \vdash A[\gamma]}{\langle \gamma, a \rangle : \Delta \to (\Gamma; A)}$$

$$\frac{\Gamma : \mathtt{Cxt} \qquad A : \mathrm{Type}_\Gamma \qquad B : \mathrm{Type}_{(\Gamma;A)} \qquad b : (\Gamma; A) \vdash B}{\lambda b : \Gamma \vdash \Pi\ A\ B}$$

Type-checking functions

We need to check contexts and types:

1. `isCo`: $\forall\ m,\ \mathsf{Cxt}\ m \to \mathsf{Bool}$
2. `isSu`: $\forall\ m\ n,\ \mathsf{Cxt}\ m \to \mathsf{Cxt}\ n \to \mathsf{Subst}\ m\ n \to \mathsf{Bool}$
3. `isTy`: $\forall\ n,\ \mathsf{Cxt}\ n \to \mathit{No}\ n \to \mathsf{Bool}$
4. `isTm`: $\forall\ n,\ \mathsf{Cxt}\ n \to \mathit{No}\ n \to \mathit{No}\ n \to \mathsf{Bool}$

A category with families (cwf) consists of:

- a category $\mathcal{C}$ which has a terminal object: the category of contexts
- a family-valued functor $T : \mathcal{C}^{\mathrm{op}} \to \mathbf{Fam}$, to model types, terms and substitutions
- extra structure to model context comprehension

A category with families (cwf) consists of:

- a category $\mathcal{C}$ which has a terminal object: the category of contexts
- a family-valued functor $T : \mathcal{C}^{\mathrm{op}} \to \mathbf{Fam}$, to model types, terms and substitutions
- extra structure to model context comprehension

A ΠU-cwf has in addition to this:

- extra structure to model Π
- extra structure to model U

A category with families (cwf) consists of:

- a category $\mathcal{C}$ which has a terminal object: the category of contexts
- a family-valued functor $T : \mathcal{C}^{\mathrm{op}} \to \mathbf{Fam}$, to model types, terms and substitutions
- extra structure to model context comprehension

A ΠU-cwf has in addition to this:

- extra structure to model Π
- extra structure to model U

↪ abstract data to ignore syntax problems

In our case

- objects of $\mathcal{C}$: contexts
- arrow of $\mathcal{C}$: substitutions.
- $T\ \Gamma = (\mathrm{Type}_\Gamma, (\{t \mid t : \Gamma \vdash A\})_{A \in \mathrm{Type}_\Gamma})$
- $T\ \gamma = (A \mapsto A[\gamma], (t \mapsto t[\gamma])_{A \in \mathrm{Type}_\Gamma})$

In our case

- objects of $\mathcal{C}$: contexts
- arrow of $\mathcal{C}$: substitutions.
- $T\ \Gamma = (\mathrm{Type}_\Gamma, (\{t \mid t : \Gamma \vdash A\})_{A \in \mathrm{Type}_\Gamma})$
- $T\ \gamma = (A \mapsto A[\gamma], (t \mapsto t[\gamma])_{A \in \mathrm{Type}_\Gamma})$

- terminal object: $[\,]$
- terminal arrows: empty substitutions

The ΠU-cwf $\mathcal{N}$ of type-checked normal forms

Correctness of the algorithm iff $\mathcal{N}$ is an initial ΠU-cwf.

Difficult part: prove that

$$\gamma : \Delta \to \Gamma \Rightarrow t : \Gamma \vdash A \Rightarrow t[\gamma] : \Delta \vdash A[\gamma].$$

The ΠU-cwf $\mathcal{N}$ of type-checked normal forms

Correctness of the algorithm iff $\mathcal{N}$ is an initial ΠU-cwf.

Difficult part: prove that

$$\gamma : \Delta \to \Gamma \Rightarrow t : \Gamma \vdash A \Rightarrow t[\gamma] : \Delta \vdash A[\gamma].$$

Solution: reducibility predicates

$$\mathtt{RedTm}_{\Gamma\ (\Pi\ a\ b)}\ t \quad \text{iff} \quad \forall \Delta, \sigma : \Delta \to \Gamma,\ \forall t',\ \mathtt{RedTm}_{\Delta\ a[\sigma]}\ t' \Rightarrow \mathtt{RedTm}_{\Delta\ b[\langle \sigma, t' \rangle]}\ (\mathtt{app}\ t[\sigma]\ t')$$