Types Summer School
Gothenburg Sweden August 2005

Dependently Typed Programming

Benjamin Grégoire
INRIA Sophia Antipolis, France

Lecture 1:
Conversion Rule

How to do formal proofs:

- A nice theory (Type Theory)
- A nice implementation:
  - A proof checker (type checker)
  - A proof assistant
- A nice user

Subject: Conversion rule

1. User point of view:
   Why we need it and how we can use it?
2. Implementor point of view:
   - Type inference Algorithm with conversion rule
   - How to get an efficient conversion test

Why we need it?

Because we do not want to do large and boring proofs

$2 + 2 = 4$ in a deduction style:

$$\dfrac{\dfrac{\overline{\text{eqTrans}} \quad \overline{2+2 = S(1+2)}}{S(1+2) = 4 \Rightarrow 2+2 = 4} \qquad \dfrac{\overline{\text{eqS}} \qquad \dfrac{\dfrac{\overline{\text{eqTrans}} \quad \overline{1+2 = S(0+2)}}{S(0+2) = 3 \Rightarrow 1+2 = 3} \qquad \dfrac{\overline{\text{eqS}} \quad \overline{0+2 = 2}}{S(0+2) = 3}}{1+2 = 3}}{S(1+2) = 4}}{2+2 = 4}$$

eqS : $x = y \Rightarrow Sx = Sy$

eqTrans : $x = y \Rightarrow y = z \Rightarrow x = z$

How to prove $2+2=4$

Computational style: Replace the axioms on addition by rewriting rules:

$$\begin{array}{rcl} 0+m & \longrightarrow & m \\ Sn+m & \longrightarrow & S(n+m) \end{array}$$

$$2+2 \longrightarrow S(1+2) \longrightarrow SS(0+2) \longrightarrow SS(2)$$

Reason modulo rewriting rules:

$$\frac{4=4 \quad 2+2 \stackrel{*}{\longrightarrow} 4}{2+2=4}$$

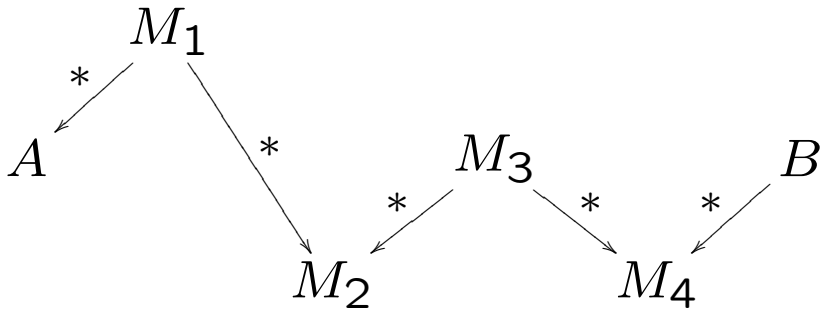
Conversion rule in PTS

A simple set of rewriting rules: reduction rules on terms (functional programs)

$$(\lambda x{:}T.\ M)\ N \xrightarrow{\beta} M[x := N]$$

With inductive definitions: reduction rule for pattern matching and fixpoint

$\approx$ ::= reflexive, symmetric and transitive closure of the reduction rules (β, ι, ...)



$$\frac{\Gamma \vdash M : A \qquad \Gamma \vdash B : s \qquad A \approx B}{\Gamma \vdash M : B}\,[\text{Conv}]$$

Encoding rewriting rule of addition

$$\begin{aligned} 0 + m &\longrightarrow m \\ Sn + m &\longrightarrow S(n + m) \end{aligned}$$

```
Inductive nat : Set :=
  | O : nat
  | S : nat -> nat.

Fixpoint plus (n m : nat) {struct n} : nat :=
  match n with
  | O => m
  | S n1 => S (plus n1 m)
  end.
```

$$\begin{aligned} \texttt{plus O m} &\xrightarrow{*} \texttt{m} \\ \texttt{plus (S n) m} &\xrightarrow{*} \texttt{S(plus n m)} \end{aligned}$$

$2 + 2 = 4$ **again!**

$$\dfrac{\dfrac{\dfrac{}{\vdash \texttt{refl\_eq} : \forall x{:}\texttt{nat}.\ x = x} \quad \dfrac{}{\vdash 4 : \texttt{nat}}}{\vdash \texttt{refl\_eq}\ 4 : 4 = 4} \qquad 4 = 4 \approx 2 + 2 = 4}{\vdash \texttt{refl\_eq}\ 4 : 2 + 2 = 4}\,[\text{Conv}]$$

- The reduction steps do not appear in the proof
  $\Rightarrow$ The proof is small : `refl_eq 4`

- Reduction steps appear in the type checking
  $\Rightarrow$ Cost remains in proof checking (in the conversion test)

If we reduce this cost, we get:

- Small proofs

- Quickly type checked

Reflexivity

- A predicate $P : T \to \texttt{Prop}$
- A decision procedure $f : T \to \texttt{bool}$
- A correctness lemma $C : \forall x{:}T.\ f\ x = \texttt{true} \to P\ x$

Assume $f\ a$ reduce to $\texttt{true}$, a proof of $P\ a$ is $C\ a\ (\texttt{refl\_eq true})$

$$\dfrac{\dfrac{\vdots}{\vdash C\ a : f\ a = \texttt{true} \to P\ a} \qquad \dfrac{\vdash \texttt{refl\_eq true} : \texttt{true} = \texttt{true} \qquad \texttt{true} = \texttt{true} \approx f\ a = \texttt{true}}{\vdash \texttt{refl\_eq true} : f\ a = \texttt{true}}}{\vdash C\ a\ (\texttt{refl\_eq true}) : P\ a}$$

Example: primality proof

Pocklington criteria:

Let n be a positive integer, if

- $n - 1 = q.p_1 \dots p_t$ where p_i are prime numbers
- there exits a such that $\begin{cases} a^{n-1} = 1 (\text{mod } n) \\ \gcd(a^{\frac{n-1}{p_i}} - 1, n) = 1 \text{ for } i = 1 \dots t \end{cases}$
- $p_1.p_2 \dots p_t \geq \sqrt{n}$

then n is prime

Formal proof by Martin Oostdijk and Olga Caprotti

Using deduction style the proof of

20988936657440586486151264256610222593863921

take 18509 lines.

Can we use reflexivity?

18th Mersenne number: $2^{3217} - 1$

2591170860132026277624676792244153094181887553125
4273039749231618740192665863620862012095168004834 06
5506952417331941774416895092388070174103777095975 12
0423130666240829163535179523111861548622656045476 91
1275958487756105687579311910177114088262521538490 35
8304011850721164247474618230314713983402292880745 45
6779079410372882358207058923510684338829868886166 58
6502809276920803396058693087905004095037098759021 19
0183719916209940025689351131365488297391126567973 03
2419865172501164127035097054277734779723498216764 43
4466683831193225400996489940517902416240565190544 83
6908096160616257430423617218633394158524264312087 37
2665919620617535357488928945996291951830826218608 53
4009379328394202618665861425032514507730962742353 76
8229386494071277008460771242118230808041392980870 57
5047138252645714483793711250320818261265664908425 1
6994539518877896136502484057393785945994443352311 88
2801236604062624686092121503499375847822922371443 39
6288584859382157388212323936870461606773629093150 71

A 969 digits number

The proof is 8461 chars!

Others examples

Proof automation (useful for the user):

- Ring or field equalities
- Presburger arithmetic
- Decide polynomial inequalities in $\mathbb{R}$ (CAD)

Exotic theorems:

- 4-colors theorem (Gonthier, Werner)
- Kepler conjecture (Hales)

Conversion rule is very useful and convenient

How to implement a type checker with the conversion rule?

Calculus of Constructions

$$\begin{array}{lrcl} \text{Terms:} & T & ::= & s \mid x \mid \forall x{:}T.\ T \mid \lambda x{:}T.\ T \mid T\ T \\ \text{Contexts:} & \Gamma & ::= & x_1{:}T_1, x_2{:}T_2, \ldots, x_n{:}T_n \end{array}$$

One reduction rule:

$$(\lambda x{:}T.\ M)\ N \xrightarrow{\beta} M[x := N]$$

Type judgment:

$$\Gamma \vdash M : T$$

Type inference: $\Gamma \vdash M \rightsquigarrow T$ such that $\Gamma \vdash M : T$

Typing rules

$$\frac{}{\mathsf{WF}(\emptyset)} \qquad \frac{\mathsf{WF}(\Gamma) \quad \Gamma \vdash T : s}{\mathsf{WF}(\Gamma, x:T)}$$

$$\frac{\mathsf{WF}(\Gamma)}{\Gamma \vdash \mathtt{Set} : \mathtt{Type}} \qquad \frac{\mathsf{WF}(\Gamma) \quad (x:T) \in \Gamma}{\Gamma \vdash x : T}$$

$$\frac{\Gamma \vdash A : s_1 \quad \Gamma, x:A \vdash B : s_2 \quad (s_1, s_2, s_3) \in \mathsf{Rules}}{\Gamma \vdash \forall x:A.\ B : s_3}$$

$$\frac{\Gamma \vdash \forall x:A.\ B : s \quad \Gamma, x:A \vdash M : B}{\Gamma \vdash \lambda x:A.\ M : \forall x:A.\ B}$$

$$\frac{\Gamma \vdash M : \forall x:A.\ B \quad \Gamma \vdash N : A}{\Gamma \vdash M\ N : B[x := N]}$$

$$\frac{\Gamma \vdash M : A \quad \Gamma \vdash B : s \quad A \approx B}{\Gamma \vdash M : B}$$

Definitions on reduction

Using only β-rule:

$$(\lambda x{:}T.\ M)\ N \xrightarrow{\beta} M[x := N]$$

The term $((\lambda x{:}T.\ M)\ N)\ P$ does not reduce

Context rule: $$\frac{t \xrightarrow{\beta} t'}{C(t) \xrightarrow{\beta} C(t')}$$

Weak reduction

$$\begin{array}{rcl} C & ::= & []\ N \mid M\ [] \\ & \mid & \lambda x{:}[].\ M \mid \forall x{:}[].\ B \end{array}$$

Strong reduction

$$\begin{array}{rcl} C & ::= & []\ N \mid M\ [] \\ & \mid & \lambda x{:}[].\ M \mid \forall x{:}[].\ B \\ & \mid & \lambda x{:}T.\ [] \mid \forall x{:}A.\ [] \end{array}$$

NF: Normal form using strong reduction

WNF: Normal form using weak reduction

WHNF: Normal form using $C ::= []\ N$

Meta-theory of CC

Subject reduction: $\Gamma \vdash M : T \Rightarrow M \xrightarrow{\beta} M' \Rightarrow \Gamma \vdash M' : T$

Strong Normalization: $\Gamma \vdash M : T \Rightarrow M \in \mathsf{SN}$

Uniqueness of typing: $\Gamma \vdash M : T_1 \Rightarrow \Gamma \vdash M : T_2 \Rightarrow T_1 \approx T_2$

Correctness of type:

$$\Gamma \vdash M : T \;\Rightarrow\; T = \mathtt{Type} \vee \Gamma \vdash T : s$$

Corollary: $\Gamma \vdash M : T \Rightarrow \Gamma \vdash M : \mathsf{WHNF}(T)$

$$\frac{\Gamma \vdash M : T \quad \Gamma \vdash \mathsf{WHNF}(T) : s \quad T \approx \mathsf{WHNF}(T)}{\Gamma \vdash M : \mathsf{WHNF}(T)}$$

Inference Algorithm

Type inference: $\Gamma \vdash M \rightsquigarrow T$ such that $\mathsf{WF}(\Gamma) \Rightarrow \Gamma \vdash M : T$

$$\frac{}{\Gamma \vdash \texttt{Set} \rightsquigarrow \texttt{Type}} \qquad \frac{(x:T) \in \Gamma}{\Gamma \vdash x \rightsquigarrow T}$$

$$\frac{\Gamma \vdash A \rightsquigarrow T_1 \quad \mathsf{WHNF}(T_1) = s_1 \quad \Gamma, x:A \vdash B \rightsquigarrow T_2 \quad \mathsf{WHNF}(T_2) = s_2}{\Gamma \vdash \forall x:A.\ B \rightsquigarrow s_3}$$

$$\frac{\Gamma \vdash A \rightsquigarrow T_1 \quad \mathsf{WHNF}(T_1) = s_1 \quad \Gamma, x:A \vdash M \rightsquigarrow B \quad B \neq \texttt{Type}}{\Gamma \vdash \lambda x:A.\ M \rightsquigarrow \forall x:A.\ B}$$

$$\frac{\Gamma \vdash M \rightsquigarrow T_1 \quad \mathsf{WHNF}(T_1) = \forall x:A.\ B \quad \Gamma \vdash N \rightsquigarrow T_2 \quad T_2 \approx A}{\Gamma \vdash M\ N \rightsquigarrow B[x := N]}$$

Soundness

Soundness: $\mathsf{WF}(\Gamma) \Rightarrow \Gamma \vdash M \rightsquigarrow T \Rightarrow \Gamma \vdash M : T$

Proof: by induction on M

$$\frac{\Gamma \vdash M \rightsquigarrow T_1 \quad \mathsf{WHNF}(T_1) = \forall x : A.\ B \quad \Gamma \vdash N \rightsquigarrow T_2 \quad T_2 \approx A}{\Gamma \vdash M\ N \rightsquigarrow B[x := N]}$$

$$\frac{\Gamma \vdash M : \forall x : A.\ B \quad \dfrac{\Gamma \vdash N : T_2 \quad \Gamma \vdash A : s \quad T_2 \approx A}{\Gamma \vdash N : A}}{\Gamma \vdash M\ N : B[x := N]}$$

Completeness

Completeness: $\Gamma \vdash M : T_1 \Rightarrow \Gamma \vdash M \rightsquigarrow T_2$

Proof: by induction on $\Gamma \vdash M : T_1$

$$\frac{\Gamma \vdash M : \forall x : A.\ B \quad \Gamma \vdash N : A}{\Gamma \vdash M\ N : B[x := N]}$$

$$\frac{\Gamma \vdash M \rightsquigarrow T_M \quad \mathsf{WHNF}(T_M) = \forall x : A'.\ B' \quad \Gamma \vdash N \rightsquigarrow T_N \quad T_N \approx A'}{\Gamma \vdash M\ N \rightsquigarrow B[x := N]}$$

(Geuvers): $T \approx \forall x : A.\ B \ \Rightarrow\ \exists A', B', \mathsf{WHNF}(T) = \forall x : A'.\ B'$

(Generation lemma):

$$\Gamma \vdash \forall x : A.\ B : T \ \Rightarrow\ \exists s_1, s_2, s_3, \begin{cases} \Gamma \vdash A : s_1 \wedge \Gamma, x : A \vdash B : s_2 \\ T \approx s_3 \end{cases}$$

So

We want a proof assistant

$\Rightarrow$ We develop a proof language

But it is also a nice functional programing language

- We have type checker
- We should now develop compiler