CHECKING A CLASS OF PROOF-CHECKERS IN COQ : METATHEORY AND PRACTICE

Bruno Barras

INRIA Rocquencourt

December 1996

$\boxed{\text{Plan}}$

1. General points about proof-checkers
2. Metatheory of a simple logical system (CC)
3. Building and using a stand-alone system
4. Modular proof-checker (PTS-$\subset$)

Motivations

- Improve reliability of the tool :

$$\text{logical system} \stackrel{?}{=} \text{implemented system}$$

- Extensibility of the system.

Scheme of work

- Formalizing metatheory of a logical system based on the Curry-Howard Isomorphism :

$$\text{Proof checking} = \text{Type checking}$$

- Verifying the algorithms (program extraction).

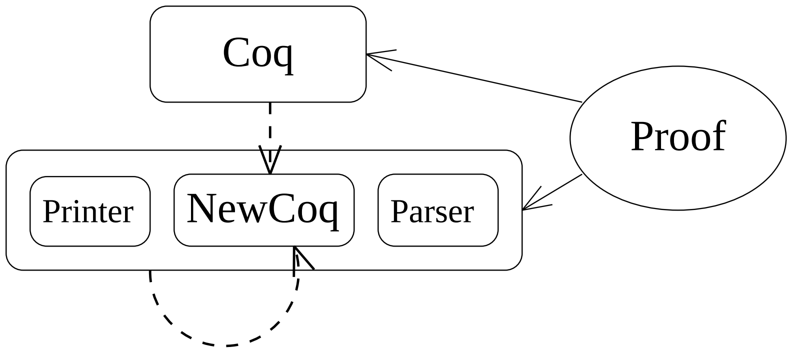
Using Coq

Study :

- Ability to check significant proofs.
- Practicability of program extraction.

Long term dream : bootstrap

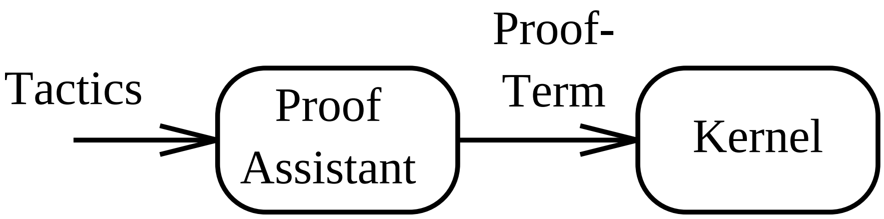
Similarity between Coq and the object logical system



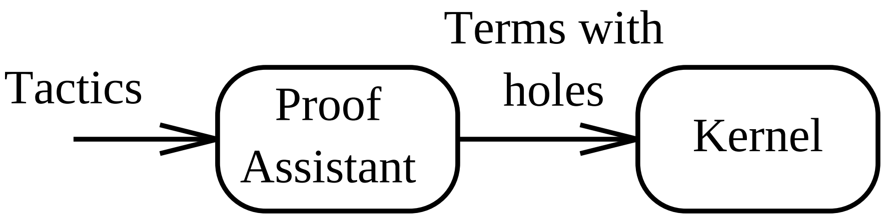
Bypassed restriction : Gdel's 2^{nd} incompleteness theorem.

Kernel

- Part of the system asserting safety.
- 2 kernel architectures :
  - A simple type-checker.



  - Construction of the proof checked.



Type system with metavariables, explicit substitutions.

2. Calculus of Constructions (Coquand)

- Main concern: being close to the usual formulation.
- De Bruijn notation :

$$\begin{aligned} \textsc{Sort} &:= \mathrm{Prop} \mid \mathrm{Kind} \\ \textsc{Term} &:= s \mid n \mid \lambda T.T \mid (T\ T) \mid \Pi T.T \end{aligned}$$

where $s \in \textsc{Sort}$ and $n \in I\!N$.

Typing rules

$$(\textsc{Prop})\ \frac{\Gamma \vdash}{\Gamma \vdash \mathrm{Prop} : \mathrm{Kind}}$$

$$(\textsc{Var})\ \frac{\Gamma \vdash \qquad \uparrow^{n+1}\Gamma(n) = T}{\Gamma \vdash n : T}$$

$$(\textsc{Abs})\ \frac{\Gamma \vdash T : s_1 \qquad \Gamma; T \vdash M : U \qquad \Gamma; T \vdash U : s_2}{\Gamma \vdash \lambda T.M : \Pi T.U}$$

$$(\textsc{App})\ \frac{\Gamma \vdash M : \Pi A.B \qquad \Gamma \vdash N : A}{\Gamma \vdash (M\ N) : B[0\backslash N]}$$

$$(\textsc{Prod})\ \frac{\Gamma \vdash T : s_1 \qquad \Gamma; T \vdash U : s_2}{\Gamma \vdash (\Pi T.U) : s_2}$$

$$(\textsc{Conv})\ \frac{\Gamma \vdash M : U \qquad \Gamma \vdash V : s \qquad U \overset{\beta}{\approx} V}{\Gamma \vdash M : V}$$

Context validity

$$(\text{WF-}[\,])\ \frac{}{[\,] \vdash} \qquad (\text{WF-}\textsc{var})\ \frac{\Gamma \vdash T : s}{\Gamma; T \vdash}$$

Metatheory

- Church-Rosser

$$A \stackrel{\beta}{\approx} B \Rightarrow \exists C.\ A \rhd_{\beta}^{*} C \land B \rhd_{\beta}^{*} C$$

- Type uniqueness

$$\Gamma \vdash M : T \land \Gamma \vdash M : U \Rightarrow T \stackrel{\beta}{\approx} U$$

- Substitution lemma

$$\frac{\Gamma; A \vdash M : T \qquad \Gamma \vdash N : A}{\Gamma \vdash M[0\backslash N] : T[0\backslash N]}$$

- Subject Reduction

$$\frac{\Gamma \vdash M : T \qquad M \rhd_{\beta}^{*} N}{\Gamma \vdash N : T}$$

- Strong Normalization

$$\Gamma \vdash M : T \Rightarrow M \in \mathcal{SN} \land T \in \mathcal{SN}$$

- Decidability of typing

$$\mathsf{Dec}\,(\Gamma \vdash M : T)$$

3. Building a proof-checker

- Hand-written code : command parser, printer
- State of the proof-checker : a well-formed context (and a list of distinct print names)
- Checked commands :

```
Axiom nat : Prop.
Axiom O : nat.
Axiom S : nat -> nat.

Check (S (S O)): nat.
Infer [P:nat->Prop][f0:(P O)]
    [fS:(n:nat)(P n)->(P (S n))](fS O f0).
```

Example : Proof of Newman's lemma

Statement : A locally confluent and nœtherian relation is confluent.

Checking procedure :

$$\begin{array}{c}\text{List of Coq tactics}\\ \Downarrow \text{Coq}\\ \text{Proof-term}\\ \Downarrow \mathsf{Cc}\\ \text{Re-checking}\end{array}$$

Efficiency :

	Checking time	Total time
Cc	$0.08s$	$0.15s$
Coq	$0.3s$	$3s$

Temporary conclusion

- Extracted program seems quite efficient.
- 2 layer architecture feasible (though awkward)
- Definition encodings (δ-reduction not considered)
- No abstraction of the logical system

4. A modular Proof-Checker

- Class of type systems (PTS-$\subset$) :
  - PTS
  - Subtyping (ex : cumulativity)
  - Definitions
- Parameters (Pollack 94) :
  - Sort, Ax, Rule
  - Subtyping relation : $\leq_\Gamma$
    Example : δ-reduction

$$\Gamma(n) = (\mathsf{Def}\ M : T) \Rightarrow n \rhd^{\delta}_{\Gamma} \uparrow^{n+1} M$$

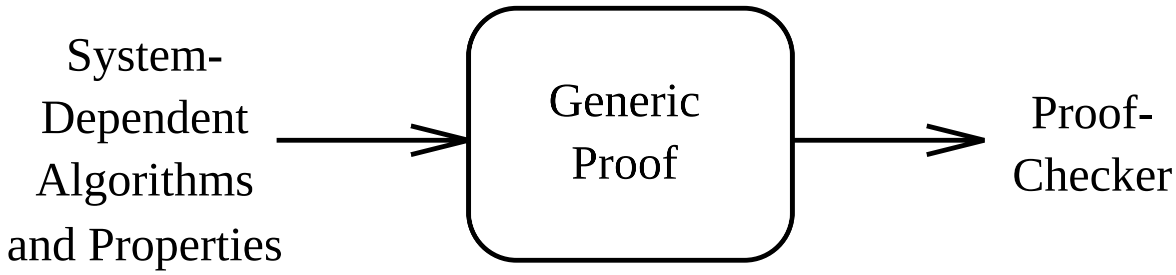
- New typing rules :

$$(\textsc{Sub})\ \frac{\Gamma \vdash M : T \qquad T \leq_\Gamma U \qquad \Gamma \vdash U : s}{\Gamma \vdash M : U}$$

$$(\textsc{WF-def})\ \frac{\Gamma \vdash M : T \qquad \Gamma \vdash T : s}{\Gamma; (\mathsf{Def}\ M : T) \vdash}$$

Proof-Checker

- Structure (4 operations) :
  - Inference of principal type
  - Type-checking
  - Context extension with an axiom
  - Context extension with a definition
- Functor-Proof :



Type-checking PTS-$\subset$

Input structure $\texttt{PTS\_dec}(\textsc{Sort}, \textsc{Ax}, \textsc{Rule}, \leq)$:

- Programs :
  - $\forall M \in \mathcal{WT}.\ \mathsf{Dec}\,(\exists^\star s \in \textsc{Sort}.\ M \equiv_\Gamma s)$
  - $\forall M \in \mathcal{WT}.\ \mathsf{Dec}\,(\exists^\star A, B \in \mathcal{WT}.\ M \equiv_\Gamma \Pi A.B)$
  - $\forall A, B \in \mathcal{WT}.\ \mathsf{Dec}\,(A \leq_\Gamma B)$
  - $\forall s_1.\ \exists^\star s_2.\ \begin{cases} (s_1, s_2) \in \textsc{Ax} \\ \forall s_2'.\ (s_1, s_2') \in \textsc{Ax} \Rightarrow s_2 \leq s_2' \end{cases}$
  - $\forall s_1, s_2.\ \exists^\star s_3.\ \begin{cases} (s_1, s_2, s_3) \in \textsc{Rule} \\ \forall (s_1', s_2', s_3') \in \textsc{Rule}. \\ \quad \left.\begin{array}{r} s_1 \leq s_1' \\ s_2 \leq s_2' \end{array}\right\} \Rightarrow s_3 \leq s_3' \end{cases}$
- Properties :
  - $A \leq_{\Gamma;T} B \Rightarrow \Pi T.A \leq_\Gamma \Pi T.B$
  - $\Pi A.B \leq_\Gamma \Pi C.D \Rightarrow C \leq_\Gamma A \wedge B \leq_{\Gamma;C} D$

$\mathcal{WT}$ = set of well-formed types

Example of subtyping : Cumulativity

- Parameters :
  - sort inclusion relation : $\prec$
  - reduction relation R (β, δ, etc.)
- Definition :

$$(\text{C-CONV})\ \frac{A \overset{R}{\approx}_\Gamma B}{A \leq_\Gamma^{Cm} B} \qquad (\text{C-UNIV})\ \frac{s_1 \prec s_2}{s_1 \leq_\Gamma^{Cm} s_2}$$

$$(\text{C-PROD})\ \frac{C \leq_\Gamma^{Cm} A \qquad B \leq_{\Gamma;C}^{Cm} D}{\Pi A.B \leq_\Gamma^{Cm} \Pi C.D}$$

- CTS-PTS functor :

$$(\textsc{Sort}, \textsc{Ax}, \textsc{Rule}, R, \prec) \qquad (\textsc{Sort}, \textsc{Ax}, \textsc{Rule}, \leq^{Cm})$$

$$\boxed{\text{CTS\_dec}} \xrightarrow{\text{Proof}} \boxed{\text{PTS\_dec}}$$

Type-checking cumulative PTS

Structure $\texttt{CTS\_dec}(\textsc{Sort}, \textsc{Ax}, \textsc{Rule}, \prec, R)$:

- Program affecting R : head normal form.

$$\forall M \in \mathcal{SN}^R.\ \exists^\star N.\ M \rhd_\Gamma^* N \wedge N \in \mathcal{HNF}_\Gamma^R$$

- Programs affecting $\textsc{Ax}, \textsc{Rule}, \prec$:
  - $\forall s_1.\ \exists^\star s_2.\ \begin{cases} (s_1, s_2) \in \textsc{Ax} \\ \forall s_2'.\ (s_1, s_2') \in \textsc{Ax} \Rightarrow s_2 \prec s_2' \end{cases}$
  - $\forall s_1, s_2.\ \exists^\star s_3.\ \begin{cases} (s_1, s_2, s_3) \in \textsc{Rule} \\ \forall (s_1', s_2', s_3') \in \textsc{Rule}. \\ \quad \left. \begin{array}{c} s_1 \prec s_1' \\ s_2 \prec s_2' \end{array} \right\} \Rightarrow s_3 \prec s_3' \end{cases}$

- Properties :
  - Sorts and products are head normal forms.
  - R is Church-Rosser.
  - R verifies Subject Reduction.
  - Strong Normalization : $\mathcal{WT} \subseteq \mathcal{SN}^R$

$\mathcal{HNF}^R$ = set of head normal forms of R

$\boxed{\beta \text{ and } \delta\text{-reduction}}$

- Subject Reduction δ always holds.
- Subject Reduction β holds if:

$$\Pi A.B \leq_\Gamma \Pi C.D \Rightarrow C \leq_\Gamma A \wedge B \leq_{\Gamma;C} D$$

- β and $\beta \cup \delta$ are confluent and head normal reduction algorithms exist.

Example : sorts of Coq

- V5.10 : $Prop, Set, Type(n), Typeset(n)$.

$$\begin{array}{rcl} Prop & : & Type(0) \\ Set & : & Typeset(0) \\ Type(n) & : & Type(n+1) \\ Typeset(n) & : & Typeset(n+1) \end{array}$$

  Cumulativity :

$$\begin{array}{rcl} Type(n) & \prec & Type(n+1) \\ Typeset(n) & \prec & Typeset(n+1) \end{array}$$

- V6.1 : $Prop, Set, Type(n)$.

$$\begin{array}{rcl} Prop & : & Type(0) \\ Set & : & Type(0) \\ Type(n) & : & Type(n+1) \end{array}$$

  Cumulativity :

$$\begin{array}{rcl} Type(n) & \prec & Type(n+1) \\ Prop & \prec & Type(0) \\ Set & \prec & Type(0) \end{array}$$

Conclusion

- Coq is well suited to this kind of applications.
- Structures should be extracted to modules.

Future work : towards Coq

- Gap between Coq and the formalized system :
  - Inductive Types
  - Program Extraction
  - Universe constraints
- New features :
  - Abstract context
  - Module system