TALLINN UNIVERSITY OF TECHNOLOGY

Verifying simple imperative programs with the Coq proof assistant

Andres Toom

Institute of Cybernetics at TUT

IRIT/ACADIE Université de Toulouse

Tallinn 06.05.2010

Outline

- About Coq
  - Curry-Howard isomorphism
  - Sorts
  - Propositions
  - Logic
- Inductive datatypes and programming in Coq
- The While language
  - Formalisation of While
  - Verifying While programs
- Examples of Coq in use
- Future work

References

- B. C. Pierce, C. Casinghino, M. Greenber. Software Foundations

  (http://www.cis.upenn.edu/~bcpierce/sf/)
- Coq Reference Manual

  (www.lix.polytechnique.fr/coq/doc)
- H. R. Nielson, F. Nielson: Semantics with Applications: A Formal Introduction. Wiley, 1992

  (http://www.daimi.au.dk/~bra8130/Wiley_book/wiley.html)
- Y. Bertot, P. Castéran. Interactive Theorem Proving and Program Development. Coq'Art: The Calculus of Inductive Constructions. An EATCS Series, 2004

Coq

- Coq is an interactive theorem prover developed at INRIA and some other research institutions in France within the TypiCal (ex-LogiCal) project.
- Provides a formal language for writing mathematical definitions, executable algorithms and theorems together with an environment for semi-interactive development of machine-checked proofs.
- Based on the theory of the calculus of inductive constructions a derivative of calculus of constructions initially developed by Thierry Coquand (1988).
- Implemented in Objective Caml.

Curry-Howard isomorphism

- Also known as
  - Curry–Howard correspondence
  - proofs-as-programs correspondence or
  - formulae-as-types correspondence
- Establishes a direct relationship between computer programs and proofs in constructive mathematics
- Observations by Curry (1934 and 1958) and Howard (1969) that
  - Proof systems and models of computation have same structure
- Examples of formal systems based on this
  - Per Martin-Löf's intuitionistic type theory
  - Thierry Coquand's Calculus of Constructions

Sorts in Coq

- In Coq all objects belong into following sorts:
  - Set, Prop, Type
- Set – the universe of program types or specifications. Specifications are typing the programs.
- Prop – the universe of logical propositions. Logical propositions are typing the proofs.
- Type – the type of Set and Prop

Sorts in Coq (2)

- Sorts can be manipulated as ordinary terms
- Sorts need to have a type too ..
- Assuming Set has type Set leads to an inconsistent type theory
- Hence, there are in addition to Set and Prop a hierarchy of universes Type(i) for any integer i
- The set of sorts:

$$\mathcal{S} \equiv \{\mathsf{Prop}, \mathsf{Set}, \mathsf{Type}(i) | i \in \mathbb{N}\}$$

Propositions in Coq

- Coq demo …

Logic in Coq

- Coq demo …

Inductive datatypes and programming in Coq

- Coq demo …

The While language

While syntax

- While expressions and statements

$$
\begin{array}{lcl}
a & ::= & n \mid x \mid a_1 + a_2 \mid a_1 \star a_2 \mid a_1 - a_2 \\
b & ::= & \texttt{true} \mid \texttt{false} \mid a_1 = a_2 \mid a_1 \leq a_2 \mid \neg b \mid b_1 \wedge b_2 \\
S & ::= & x := a \mid \texttt{skip} \mid S_1 \; ; \; S_2 \mid \texttt{if } b \texttt{ then } S_1 \texttt{ else } S_2 \\
  & \mid & \texttt{while } b \texttt{ do } S
\end{array}
$$

Nielsen & Nielsen. Semantics with Applications

Semantics of arithmetic expressions

$$
\begin{aligned}
\mathcal{A}[\![n]\!]s &= \mathcal{N}[\![n]\!] \\
\mathcal{A}[\![x]\!]s &= s\ x \\
\mathcal{A}[\![a_1 + a_2]\!]s &= \mathcal{A}[\![a_1]\!]s + \mathcal{A}[\![a_2]\!]s \\
\mathcal{A}[\![a_1 \star a_2]\!]s &= \mathcal{A}[\![a_1]\!]s \star \mathcal{A}[\![a_2]\!]s \\
\mathcal{A}[\![a_1 - a_2]\!]s &= \mathcal{A}[\![a_1]\!]s - \mathcal{A}[\![a_2]\!]s
\end{aligned}
$$

Nielsen & Nielsen. Semantics with Applications

Semantics of boolean expressions

$$
\begin{array}{lcl}
\mathcal{B}[\![\texttt{true}]\!]s & = & \mathbf{tt} \\
\mathcal{B}[\![\texttt{false}]\!]s & = & \mathbf{ff} \\
\mathcal{B}[\![a_1 = a_2]\!]s & = & \begin{cases} \mathbf{tt} & \text{if } \mathcal{A}[\![a_1]\!]s = \mathcal{A}[\![a_2]\!]s \\ \mathbf{ff} & \text{if } \mathcal{A}[\![a_1]\!]s \neq \mathcal{A}[\![a_2]\!]s \end{cases} \\
\mathcal{B}[\![a_1 \leq a_2]\!]s & = & \begin{cases} \mathbf{tt} & \text{if } \mathcal{A}[\![a_1]\!]s \leq \mathcal{A}[\![a_2]\!]s \\ \mathbf{ff} & \text{if } \mathcal{A}[\![a_1]\!]s > \mathcal{A}[\![a_2]\!]s \end{cases} \\
\mathcal{B}[\![\neg\, b]\!]s & = & \begin{cases} \mathbf{tt} & \text{if } \mathcal{B}[\![b]\!]s = \mathbf{ff} \\ \mathbf{ff} & \text{if } \mathcal{B}[\![b]\!]s = \mathbf{tt} \end{cases} \\
\mathcal{B}[\![b_1 \wedge b_2]\!]s & = & \begin{cases} \mathbf{tt} & \text{if } \mathcal{B}[\![b_1]\!]s = \mathbf{tt} \text{ and } \mathcal{B}[\![b_2]\!]s = \mathbf{tt} \\ \mathbf{ff} & \text{if } \mathcal{B}[\![b_1]\!]s = \mathbf{ff} \text{ or } \mathcal{B}[\![b_2]\!]s = \mathbf{ff} \end{cases}
\end{array}
$$

Nielsen & Nielsen. Semantics with Applications

Semantics of statements

$$[\text{ass}_{\text{ns}}] \quad \langle x := a,\ s\rangle \to s[x \mapsto \mathcal{A}[\![a]\!]s]$$

$$[\text{skip}_{\text{ns}}] \quad \langle \texttt{skip},\ s\rangle \to s$$

$$[\text{comp}_{\text{ns}}] \quad \frac{\langle S_1,\ s\rangle \to s',\ \langle S_2,\ s'\rangle \to s''}{\langle S_1;S_2,\ s\rangle \to s''}$$

$$[\text{if}^{\text{tt}}_{\text{ns}}] \quad \frac{\langle S_1,\ s\rangle \to s'}{\langle \texttt{if}\ b\ \texttt{then}\ S_1\ \texttt{else}\ S_2,\ s\rangle \to s'} \ \text{if}\ \mathcal{B}[\![b]\!]s = \mathbf{tt}$$

$$[\text{if}^{\text{ff}}_{\text{ns}}] \quad \frac{\langle S_2,\ s\rangle \to s'}{\langle \texttt{if}\ b\ \texttt{then}\ S_1\ \texttt{else}\ S_2,\ s\rangle \to s'} \ \text{if}\ \mathcal{B}[\![b]\!]s = \mathbf{ff}$$

$$[\text{while}^{\text{tt}}_{\text{ns}}] \quad \frac{\langle S,\ s\rangle \to s',\ \langle \texttt{while}\ b\ \texttt{do}\ S,\ s'\rangle \to s''}{\langle \texttt{while}\ b\ \texttt{do}\ S,\ s\rangle \to s''} \ \text{if}\ \mathcal{B}[\![b]\!]s = \mathbf{tt}$$

$$[\text{while}^{\text{ff}}_{\text{ns}}] \quad \langle \texttt{while}\ b\ \texttt{do}\ S,\ s\rangle \to s\ \text{if}\ \mathcal{B}[\![b]\!]s = \mathbf{ff}$$

Nielsen & Nielsen. Semantics with Applications

Verifying While programs

- Coq demo …

Examples of Coq in use

- CompCert – Specifying and verifying a compiler from a significant subset of C language to PowerPC assembly (X. Leroy)
- Verifying the correctness of the Esterel Lustre compiler.
- Verifying the security properties of JavaCard.
- Formalisation of the Coq kernel (B. Barras, B. Bernardo) and Coq extractor (P. Letouzey, S. Glondu)

Future work

- Different semantic approaches
  - Denotational semantics (direct and continuation passing style), …
- Using Coq within a specific pragmatic approach to specify and verify correctness of program transformations from MBD oriented domain specific languages.

Thank you!