Software verification using proof assistants

Jesper Bengtson
IT University of Copenhagen
My background

- Ph.D. from University of Uppsala
  - Formalising Process Calculi, Supervisor: Joachim Parrow
- PostDoc IT University of Copenhagen
  - Tools and Methods for Scalable Software Verification, Supervisor (Boss): Lars Birkedal
- Associate professor at ITU since August 2013
Main goal

Constructs tools that allow software developers to prove full functional correctness of systems

- Expressive specification language
- Interactive proof development
- IDE support
Main goal

Constructs tools that allow software developers to prove full functional correctness of systems

- Expressive specification language
- Interactive proof development
- IDE support

Higher-order separation logic
Main goal

Constructs tools that allow software developers to prove full functional correctness of systems

- Expressive specification language
- Interactive proof development
- IDE support

The Coq interactive proof assistant
Main goal

Constructs tools that allow software developers to prove full functional correctness of systems

- Expressive specification language
- Interactive proof development
- IDE support

Eclipse
Separation Logic
Triples and specifications

Hoare Triples

\{P\} \ c \ \{Q\}

Frame rule

\frac{\{P\} \ c \ \{Q\}}{\{P * R\} \ c \ \{Q * R\}} \quad \text{R does not mention } c

Specifications

\text{o.m(args)} \rightarrow \{P\} \_ \{Q\}
The List-predicate

\[
\begin{align*}
\text{Node\_list}\ p\ [\ ] & \Rightarrow p = \text{null} \\
\text{Node\_list}\ p\ (x::xs) & \Rightarrow \\
& \exists v:: p.\text{value} \mapsto x * \\
& p.\text{next} \mapsto v * \\
& \text{Node\_list}\ v\ xs
\end{align*}
\]

\[
\begin{align*}
\text{List}\ p\ xs & \Rightarrow \\
& \exists h, p.\text{head} \mapsto h * \text{Node\_list}\ h\ xs
\end{align*}
\]
Proving a program correct is done by proving one predicate in a specification logic.

\[ \vdash \text{C.m}_1(a, b) \leftrightarrow \{P_1\} \land \{Q_1\} \land \]
\[ \text{C.m}_2(a, b) \leftrightarrow \{P_2\} \land \{Q_2\} \land \]
\[ \text{C.m}_3(a, b) \leftrightarrow \{P_3\} \land \{Q_3\} \]
Proving a program correct is done by proving one predicate in a specification logic

$$\forall a, b \in \mathbb{R} \quad C.m_1(a, b) \iff \{P_1\} \quad \overset{\wedge}{\implies} \quad \{Q_1\}$$

$$\forall a, b \in \mathbb{R} \quad C.m_2(a, b) \iff \{P_2\} \quad \overset{\wedge}{\implies} \quad \{Q_2\}$$

$$\forall a, b \in \mathbb{R} \quad C.m_3(a, b) \iff \{P_3\} \quad \overset{\wedge}{\implies} \quad \{Q_3\}$$

Specification logic formula
Proving a program correct is done by proving one predicate in a specification logic.

\[ \vdash C.m_1(a, b) \leftrightarrow \{ P_1 \} \enspace \land \enspace \{ Q_1 \} \land \\
C.m_2(a, b) \leftrightarrow \{ P_2 \} \enspace \land \enspace \{ Q_2 \} \land \\
C.m_3(a, b) \leftrightarrow \{ P_3 \} \enspace \land \enspace \{ Q_3 \} \land \]
Program correctness

Proving a program correct is done by proving one predicate in a specification logic

\[ \vdash C.m_1(a, b) \leftrightarrow \{P_1\} \_ \_ \{Q_1\} \land \\
C.m_2(a, b) \leftrightarrow \{P_2\} \_ \_ \{Q_2\} \land \\
C.m_3(a, b) \leftrightarrow \{P_3\} \_ \_ \{Q_3\} \]

What do we do about function calls?
The later operator

The predicate $\triangleright P$ states that $P$ is not necessarily true now, but it will be true later.

\[
\begin{align*}
P & \vdash \triangleright P \\
\triangleright P & \vdash P
\end{align*}
\]

Weaken

\[
\begin{align*}
\triangleright P & \vdash \top \\
\top & \vdash P
\end{align*}
\]

Löb

This is modelled using step indexes

Session types meet separation logic

Jesper Bengtson
Program correctness

Proving a program correct is done by proving one predicate in a specification logic

\[ \vdash C.m_1(a, b) \iff \{P_1\} \land \{Q_1\} \land \\
C.m_2(a, b) \iff \{P_2\} \land \{Q_2\} \land \\
C.m_3(a, b) \iff \{P_3\} \land \{Q_3\} \]
Program correctness

\[
\vdash C.m_1(a, b) \Rightarrow \{P_1\} _= \{Q_1\} \land \\
C.m_2(a, b) \Rightarrow \{P_2\} _= \{Q_2\} \land \\
C.m_3(a, b) \Rightarrow \{P_3\} _= \{Q_3\}
\]
Program correctness

\[ \vdash C.m_1(a, b) \mapsto \{P_1\}_1 \land \{Q_1\}_1 \land \\
C.m_2(a, b) \mapsto \{P_2\}_2 \land \{Q_2\}_2 \land \\
C.m_3(a, b) \mapsto \{P_3\}_3 \land \{Q_3\}_3 \]
Program correctness

\[
\begin{align*}
\triangleright & \left( C.m_1(a, b) \mapsto \{ P_1 \} \land \{ Q_1 \} \right) \\
& (C.m_2(a, b) \mapsto \{ P_2 \} \land \{ Q_2 \}) \\
& (C.m_3(a, b) \mapsto \{ P_3 \} \land \{ Q_3 \}) \\
\vdash & C.m_1(a, b) \mapsto \{ P_1 \} \land \{ Q_1 \} \\
& C.m_2(a, b) \mapsto \{ P_2 \} \land \{ Q_2 \} \\
& C.m_3(a, b) \mapsto \{ P_3 \} \land \{ Q_3 \}
\end{align*}
\]

Apply Löb rule
Later distributes over conjunction
Later distributes over conjunction

\[\vdash C.m_1(a, b) \mapsto \{P_1\} _\sqcap \{Q_1\} \land \]
\[\vdash C.m_2(a, b) \mapsto \{P_2\} _\sqcap \{Q_2\} \land \]
\[\vdash C.m_3(a, b) \mapsto \{P_3\} _\sqcap \{Q_3\} \land \]
Program verification in Proof Assistants (PAs)
Program verification in PAs

• Code extraction
• Semantic embedding
Semantic embedding

- Program logic
  - Language model
  - Logic model
  - Operational semantics
  - Derived axiomatic semantics
- Heuristics for symbolic execution
- Entailment checkers
Program verification in PAs

- Program logic
  - Language model
  - Memory model
  - Operational semantics
  - Derived axiomatic semantics

Proof assistants excel at this
Program verification in PAs

Proof assistants are very bad at this

- Heuristics for symbolic execution
- Entailment checkers
Automation in Isabelle

- Sledgehammer
- ML-level
Automation in Coq

- LTac
- MTac (Ziliani 2013)
- OCaml-level
Key insight

Proof assistants are very good at mechanising logics and proving meta-theoretic properties about these logics.

Proof assistants are very bad at proving theorems using a logic other than their own.
Charge!

- Charge! is a framework for program verification using higher-order separation logic in Coq (*Bengtson et al. 2011*)
  - Symbolic execution of Java programs
  - Separation logic entailment checking
- All automation is handled using LTac tactics
LTac is a tactic language for Coq

pros

• Rapid prototyping
• Easy to automate frequently occurring proof patterns
• Soundness proofs not required

cons

• Can be very difficult to debug
• Can be very slow
• Can produce big proof objects
Computational reflection

• Translate your theorem to a deep embedding in your proof assistant (Reification)

• Solve your theorem using a program that you have proven sound within your proof assistant

• Apply a denotation function to the result to return to the logic of the proof assistant (Reflection)
Proof by reflection

Theorem

Deep embedding

Reification

Soundness proof

Deep embedding

Soundness proof

Reflection

computation

Deep embedding

Reflection
RTac

Tactic language for reflective tactics *(Malecha et al. 2015)*

- Includes a plugin for reification
- Extensible
- Contains tactic connectives such as INTRO, REPEAT, REC and APPLY
- Soundness proofs are automatic as long as you stick to the connectives
- Tactics are Gallina programs making it possible to construct your own
Proof by reflection

pros

• Fast
• Small proof objects

cons

• Requires soundness proofs
• Requires reification
• Can be very difficult to debug
Interactive Development Environments (IDEs)
Coqoon

Eclipse plugin for Coq theory development

- Structured project
- PIDE support
- Coq model in Eclipse
Theorem mult_0_plus : forall n m : nat, (0 + n) * m = n * m.

Proof.
intros n m.
rewrite -> plus_O_n.
reflexivity. Qed.

(* **** Exercise: 2 stars (mult_S_1) *)

Theorem mult_S_1 : forall n m : nat,
  m = S n ->
  m * (1 + n) = m * m.

Proof.
(*) (* FILL IN HERE *) Admitted.
(*) (*)

n : nat
m : nat

(0 + n) * m = n * m
Java verification in Coqoon

- Allow users to write Coq pre- and postcondition to their methods, and loop invariants for their loops
- Execute the program symbolically and allow users to interleave the code with Coq proofs when it gets stuck
- PIDE model will allow for a familiar work flow (no stepping through code like a debugger).
Status

• I am working on a new version of Charge! that uses RTac (ETA 2 weeks)

• Coqoon is stable. PIDE support only works on the Coq 8.5 beta

• We have just started work on the Java development environment
Thank you