IT UNIVERSITY OF COPENHAGEN

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

$$\frac{\{P\}\ c\ \{Q\}}{\{P * R\}\ c\ \{Q * R\}}$$

R does not mention c

Specifications

$$o.m(args) \mapsto \{P\}\ \_\ \{Q\}$$

The List-predicate

```
Node_list p [] => p = null
Node_list p (x::xs) =>
   ∃v:.  p.value ↦ x *
         p.next ↦ v *
         Node_list v xs
```

List p xs =>
  ∃h, p.head ↦ h * Node_list h xs

Program correctness

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

$$\begin{aligned}\vdash\ & \mathtt{C.m_1(a,\ b)} \mapsto \{P_1\}\ \_\ \{Q_1\}\ \wedge \\ & \mathtt{C.m_2(a,\ b)} \mapsto \{P_2\}\ \_\ \{Q_2\}\ \wedge \\ & \mathtt{C.m_3(a,\ b)} \mapsto \{P_3\}\ \_\ \{Q_3\}\end{aligned}$$

Program correctness

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

$$\vdash \begin{array}{l} \texttt{C.m}_1\texttt{(a, b)} \mapsto \{P_1\}\ \_\ \{Q_1\} \land \\ \texttt{C.m}_2\texttt{(a, b)} \mapsto \{P_2\}\ \_\ \{Q_2\} \land \\ \texttt{C.m}_3\texttt{(a, b)} \mapsto \{P_3\}\ \_\ \{Q_3\} \end{array}$$

Specification logic formula

Program correctness

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

$$
\begin{aligned}
\vdash\ & \texttt{C.m}_1\texttt{(a, b)} \mapsto \{\boxed{P_1}\}\ \_\ \{\boxed{Q_1}\}\ \wedge \\
& \texttt{C.m}_2\texttt{(a, b)} \mapsto \{\boxed{P_2}\}\ \_\ \{\boxed{Q_2}\}\ \wedge \\
& \texttt{C.m}_3\texttt{(a, b)} \mapsto \{\boxed{P_3}\}\ \_\ \{\boxed{Q_3}\}
\end{aligned}
$$

Separation logic formulas

Program correctness

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

$$
\begin{aligned}
\vdash\ & \mathtt{C.m_1(a,\ b)} \mapsto \{P_1\}\ \_\ \{Q_1\}\ \wedge \\
& \mathtt{C.m_2(a,\ b)} \mapsto \{P_2\}\ \_\ \{Q_2\}\ \wedge \\
& \mathtt{C.m_3(a,\ b)} \mapsto \{P_3\}\ \_\ \{Q_3\}
\end{aligned}
$$

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

$$\frac{}{\mathtt{P} \vdash \triangleright\mathtt{P}}\ \text{Weaken} \qquad \frac{\triangleright\mathtt{P} \vdash \mathtt{P}}{\vdash \mathtt{P}}\ \text{Löb}$$

This is modelled using step indexes

Program correctness

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

$$
\begin{aligned}
\vdash\ & \mathtt{C.m_1(a,\ b)} \mapsto \{P_1\}\ \_\ \{Q_1\}\ \wedge \\
& \mathtt{C.m_2(a,\ b)} \mapsto \{P_2\}\ \_\ \{Q_2\}\ \wedge \\
& \mathtt{C.m_3(a,\ b)} \mapsto \{P_3\}\ \_\ \{Q_3\}
\end{aligned}
$$

Program correctness

$$
\begin{aligned}
\vdash\ & \texttt{C.m}_1\texttt{(a, b)} \mapsto \{P_1\}\ \_\ \{Q_1\} \land \\
& \texttt{C.m}_2\texttt{(a, b)} \mapsto \{P_2\}\ \_\ \{Q_2\} \land \\
& \texttt{C.m}_3\texttt{(a, b)} \mapsto \{P_3\}\ \_\ \{Q_3\}
\end{aligned}
$$

Program correctness

$$
\begin{aligned}
\vdash\ & \mathtt{C.m}_1\mathtt{(a,\ b)} \mapsto \{P_1\}\ \_\ \{Q_1\} \land \\
& \mathtt{C.m}_2\mathtt{(a,\ b)} \mapsto \{P_2\}\ \_\ \{Q_2\} \land \\
& \mathtt{C.m}_3\mathtt{(a,\ b)} \mapsto \{P_3\}\ \_\ \{Q_3\}
\end{aligned}
$$

Program correctness

$$\triangleright\left(\begin{array}{l}\mathtt{C.m_1(a,\ b)} \mapsto \{P_1\}\ \_\ \{Q_1\} \wedge \\ \mathtt{C.m_2(a,\ b)} \mapsto \{P_2\}\ \_\ \{Q_2\} \wedge \\ \mathtt{C.m_3(a,\ b)} \mapsto \{P_3\}\ \_\ \{Q_3\}\end{array}\right)$$

$$\begin{array}{ll}\vdash & \mathtt{C.m_1(a,\ b)} \mapsto \{P_1\}\ \_\ \{Q_1\} \wedge \\ & \mathtt{C.m_2(a,\ b)} \mapsto \{P_2\}\ \_\ \{Q_2\} \wedge \\ & \mathtt{C.m_3(a,\ b)} \mapsto \{P_3\}\ \_\ \{Q_3\}\end{array}$$

Apply Löb rule

Program correctness

$$\triangleright \begin{pmatrix} \texttt{C.m}_1\texttt{(a, b)} \mapsto \{P_1\} \_ \{Q_1\} \land \\ \texttt{C.m}_2\texttt{(a, b)} \mapsto \{P_2\} \_ \{Q_2\} \land \\ \texttt{C.m}_3\texttt{(a, b)} \mapsto \{P_3\} \_ \{Q_3\} \end{pmatrix}$$

$$\begin{array}{l} \vdash \texttt{C.m}_1\texttt{(a, b)} \mapsto \{P_1\} \_ \{Q_1\} \land \\ \texttt{C.m}_2\texttt{(a, b)} \mapsto \{P_2\} \_ \{Q_2\} \land \\ \texttt{C.m}_3\texttt{(a, b)} \mapsto \{P_3\} \_ \{Q_3\} \end{array}$$

Later distributes over conjunction

IT UNIVERSITY OF COPENHAGEN

Program correctness

$$
\begin{aligned}
&\triangleright \mathtt{C.m_1(a,\ b)} \mapsto \{\mathrm{P}_1\} \ \_\ \{\mathrm{Q}_1\} \wedge \\
&\triangleright \mathtt{C.m_2(a,\ b)} \mapsto \{\mathrm{P}_2\} \ \_\ \{\mathrm{Q}_2\} \wedge \\
&\triangleright \mathtt{C.m_3(a,\ b)} \mapsto \{\mathrm{P}_3\} \ \_\ \{\mathrm{Q}_3\}
\end{aligned}
$$

$$
\begin{aligned}
\vdash\ &\mathtt{C.m_1(a,\ b)} \mapsto \{\mathrm{P}_1\} \ \_\ \{\mathrm{Q}_1\} \wedge \\
&\mathtt{C.m_2(a,\ b)} \mapsto \{\mathrm{P}_2\} \ \_\ \{\mathrm{Q}_2\} \wedge \\
&\mathtt{C.m_3(a,\ b)} \mapsto \{\mathrm{P}_3\} \ \_\ \{\mathrm{Q}_3\}
\end{aligned}
$$

Later distributes over conjunction

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

LTac is a tactic language for Coq

pros

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

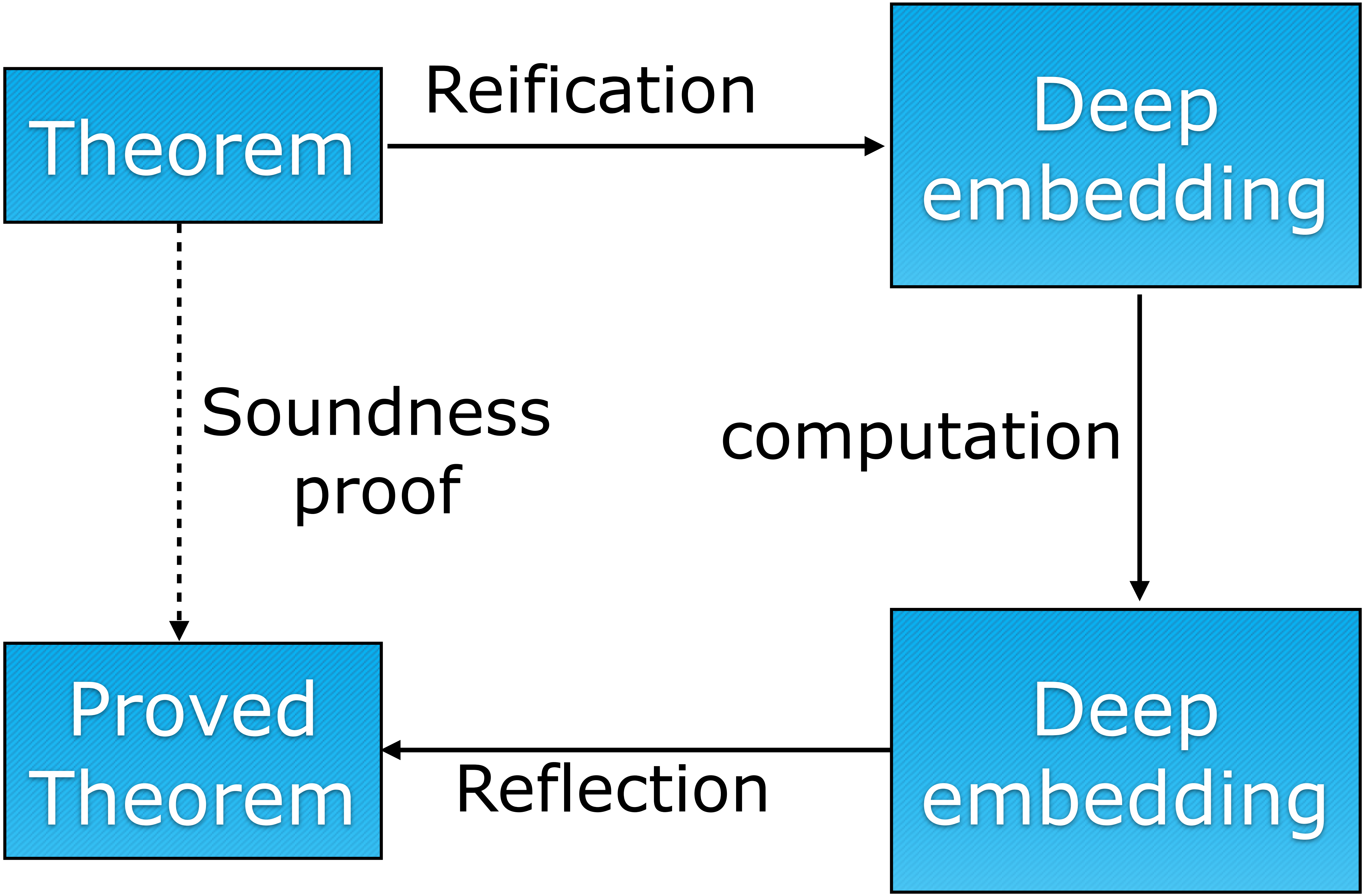
cons

- 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



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

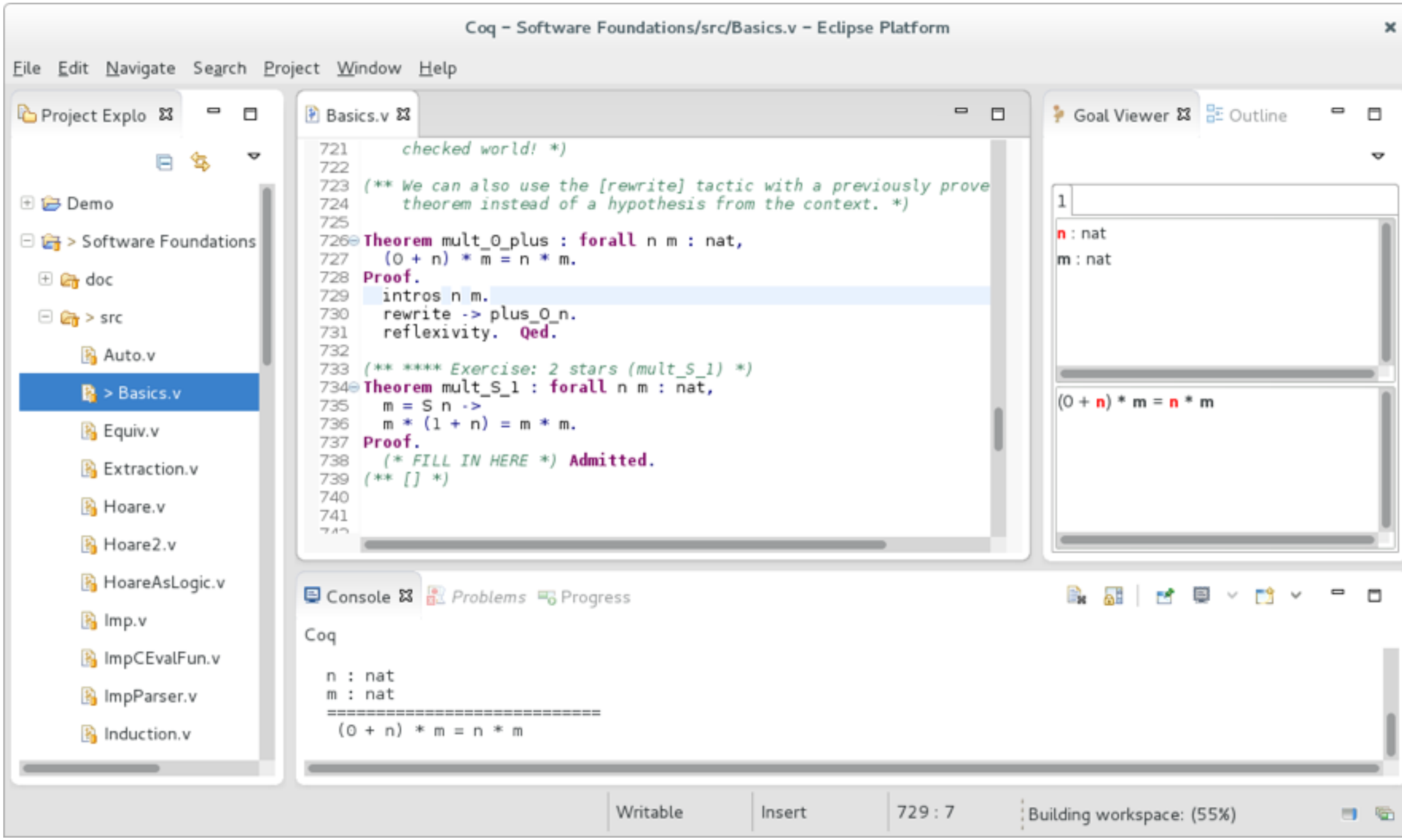
- Requires soundness proofs
- Requires reification
- Can be very difficult to debug

Interactive Development Environments (IDEs)

Coqoon

Eclipse plugin for Coq theory development

- Structured project
- PIDE support
- Coq model in Eclipse


Coq - Software Foundations/src/Basics.v - Eclipse Platform
File Edit Navigate Search Project Window Help
Project Explo
Demo
> Software Foundations
doc
> src
Auto.v
> Basics.v
Equiv.v
Extraction.v
Hoare.v
Hoare2.v
HoareAsLogic.v
Imp.v
ImpCEvalFun.v
ImpParser.v
Induction.v
Basics.v
checked world! *)
(** We can also use the [rewrite] tactic with a previously prove
theorem instead of a hypothesis from the context. *)
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.
(** [] *)
Goal Viewer
Outline
1
n : nat
m : nat
(0 + n) * m = n * m
Console
Problems
Progress
Coq
n : nat
m : nat
============================
(0 + n) * m = n * m
Writable
Insert
729 : 7
Building workspace: (55%)

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