Performing Implicit Induction Reasoning with Certifying Proof Environments

Amira Henaien\textsuperscript{1,2} \hspace{1cm} Sorin Stratulat\textsuperscript{1}

\textsuperscript{1}LITA, University of Lorraine, Ile du Saulcy, 57000, Metz, France
\textsuperscript{2}Higher School of Communication of Tunis (Sup’Com), University of Carthage, Tunisia

Juin 17, 2013
Introduction

Automatic Software Certification

- Proof certification: an essential step in the development of critical software
- Ubiquitous unbounded data structures (naturals, lists, ...)
- Big proofs, hardly checkable by human beings

⇒

- Need of automatic certification process:
  - the induction reasoning on first-order specifications based on recursively defined data structures and functions can be mechanized
  - examples: validation of industrial-size applications as
    - the JavaCard platform [Barthe and Stratulat2003]
    - a conformity algorithm for the ABR telecommunications protocol [Rusinowitch and al.2003]
Performing Implicit Induction Reasoning with Certifying Proof Environments

Induction Proof Techniques

Calling the Spike Prover from the Coq Script

Implementation of the Spike Tactic

Conclusions and Future Works
Induction Proof Techniques

Induction Principles for First-Order Reasoning

$(M, <_M)$ a well-founded poset

Noetherian induction:

$$(\forall m \in M, (\forall k \in M, k <_M m \Rightarrow \phi(k)) \Rightarrow \phi(m)) \Rightarrow \forall m \in M, \phi(m)$$

First-order instances:

- term-based induction: $M$ is a set of term vectors
  
  $$(\forall m \in M, (\forall k \in M, \overline{k} <_M \overline{m} \Rightarrow \phi(\overline{k})) \Rightarrow \phi(\overline{m})) \Rightarrow \forall \overline{m} \in M, \phi(\overline{m})$$

- formula-based induction: $M$ is a set of a first-order formulas,
  
  $$(\forall \psi \in M, (\forall \rho \in M, \rho <_M \psi \Rightarrow \rho) \Rightarrow \psi) \Rightarrow \forall \gamma \in M, \gamma$$

\[ \phi(x) = x, \forall x \in M \]

\[ \text{soundness argument: the well-foundedness property of } <_M \]
Induction Proof Techniques

Conventional Induction

An example of term-based Noetherian induction

Definition of induction schemas

Advantages:

- local induction ordering, at schema level
- one-rule implementation, integration into current proof assistants

Disadvantages:

- eager induction
- no mutual induction with other formulas
Implicit Induction

An example of formula-based Noetherian induction

\[ M \] is the set of all ground instances of formulas from a proof

Advantages:

- lazy induction
- mutual induction

Disadvantages:

- global induction ordering, at proof level
- reductive reasoning: additional ordering constraints to inference rules
Induction Proof Techniques

General Motivation

Automatizing lazy and mutual induction reasoning by proof assistants

- combine conventional and implicit induction advantages

Previous works:

- lazy generation of explicit induction hypotheses [Protzen1994]
- reconstruction of implicit into explicit induction proofs [Courant1996, Kaliszyk2005]
- implementation of a term-based 'Descente Infinie' principle as a Coq tactic [Voicu and Li2009]

Our contribution: a Coq tactic that automatically performs implicit induction proofs by

1. calling the Spike implicit induction theorem prover on subgoals from a Coq script, then
2. translating and certifying [Stratulat2010, Stratulat and Demange2011], and finally integrating the Spike proof into the Coq script.
Calling the Spike Prover from the Coq Script

Spike

- first-order theorem prover [Bouhoula and al. 1995, Stratulat 2001]
- able to build implicit induction proofs of conjectures about conditional specifications consisting of axioms represented as conditional equalities
- the general form of a proof:

\[(E^0, \emptyset) \vdash (E^1, H^1) \vdash \ldots \vdash (E^{n-1}, H^{n-1}) \vdash (\emptyset, H^n),\]

where \(E^i (i \in [1..n])\) are multisets of conjectures and \(H^i (i \in [1..n])\) are multisets of previously treated conjectures
- the inference system is reductive: for each ground instance of a new conjecture in a proof step should exist a smaller logically equivalent instance of a formula from the current state
Calling the Spike Prover from the Coq Script

Axiomatic Specification in Coq

Parameter \textit{even oeven odd eodd} : \textit{nat \to bool}.

Axiom \textit{even1} : \textit{even 0 = true}.
Axiom \textit{even2} : \textit{even (S 0) = false}.
Axiom \textit{even3} : \forall x, \textit{even (S (S x))} = \textit{oeven (S (S x))}.
Axiom \textit{oeven1} : \textit{oeven 0 = true}.
Axiom \textit{oeven2} : \textit{oeven (S 0) = false}.
Axiom \textit{oeven3} : \forall x, \textit{odd x = true \to oeven (S (S x)) = false}.
Axiom \textit{oeven4} : \forall x, \textit{odd x = false \to oeven (S (S x)) = true}.

Axiom \textit{oeven (S (S x))} = \textit{even x}.

Axiom \textit{odd1} : \textit{odd 0 = false}.
Axiom \textit{odd2} : \textit{odd (S 0) = true}.
Axiom \textit{odd3} : \forall x, \textit{odd (S (S x))} = \textit{eodd (S (S x))}.
Axiom \textit{eodd1} : \textit{eodd 0 = false}.
Axiom \textit{eodd2} : \textit{eodd (S 0) = true}.
Axiom \textit{eodd3} : \forall x, \textit{even x = true \to eodd (S (S x)) = odd x}.
Axiom \textit{eodd4} : \forall x, \textit{even x = false \to eodd (S (S x)) = true}. 
Calling the Spike Prover from the Coq Script

Specifying and Proving Conjectures in Coq

Theorem `even_xx`: \( \forall x, \text{even} (\text{add} (x \ x)) = \text{true} \).

- explicit induction proving attempt

Explicit schema issued from the recursive definition of naturals

\[ \text{nat\_ind} = \text{fun} \ P : \text{nat} \rightarrow \text{Prop} \rightarrow \text{nat\_rect} \ P : \forall \ P : \text{nat} \rightarrow \text{Prop}, \]
\[ P \ 0 \rightarrow (\ \forall \ n : \text{nat}, \ P \ n \rightarrow P \ (S \ n) ) \rightarrow \forall \ n : \text{nat}, P \ n \]

- Coq proof: intro x. induction x ...

\[ \text{requires additional induction steps} \]

General difficulties when performing explicit induction proofs

- eager identification of the induction schema (induction hypotheses, induction variables and their instantiations)
- induction hypotheses may not be applied further in the proof or additional ones are required
Calling the Spike Prover from the Coq Script

Specifying and Proving Conjectures in Coq

proving automatically, using the Spike tactic

Theorem even_xx: \( \forall x, \text{even} (\text{add} (x \times x)) = \text{true} \).

\[
\text{Spike} \quad \text{equiv} \quad [[\text{even}, \text{oeven}, \text{odd}, \text{eodd}]]
\]
\[
\text{greater} \quad \left[ [[\text{even}, \text{true}, \text{false}, \text{S}, 0, \text{add}],
\quad [[\text{add}, \text{S}, 0]] \right].
\]

Qed.
Calling the Spike Prover from the Coq Script

The Spike Tactic

- implemented in OCaml

- exists in 4 variants:
  - **Spike**
    - the ordering constraints are inferred by Spike directly from the specification.
  - **Spike equiv** [ $S_1, \ldots, S_n$ ]
    - each $S_i$ ($i \in [1..n]$) is a set of equivalent symbols.
  - **Spike greater** [ $S_1, \ldots, S_n$ ]
    - each $S_i$ ($i \in [1..n]$) is of the form \{ $symb_1, symb_2, \ldots, symb_n$ \} such that $symb_i$ is greater than any of the symbols from the set \{ $symb_{i+1}, \ldots, symb_n$ \}.
  - **Spike equiv** [ $S_1, \ldots, S_n$ ] greater [ $S'_1, \ldots, S'_n$ ]
    - the combination of the two previous cases.
Calling the Spike Prover from the Coq Script

The General Schema of the Spike Tactic

Coq Environment
Declare ML Module "spike".
Section example.
Parameter: ...
Axioms: ...
Theorem even_xx:
...
Spike [precedes]

Start the Spike tactic

Coq –> Spike
Extraction of Spike specification and translation of the theorem even_xx into a Spike conjecture

Certification of even_xx

Spike: theorem prover
Building the implicit induction proof in Spike using an induction ordering based on precedences over the function symbols given as arguments to the Spike tactic.

Spike –> Coq
Translate the implicit proof of Spike into Coq script that explicitly represents the induction ordering and the comparisons between formulas.

Coq kernel: Qed
Check the Coq script by the Coq kernel. If successfully checked, the initial theorem even_xx is certified.

Performing Implicit Induction Reasoning with Certifying Proof Environments
Implementation of the Spike Tactic

Extraction of the Spike Specification from the Coq Script

The Coq script

Parameter even oeven odd eodd:

\[ \text{nat} \rightarrow \text{bool}. \]

The Spike specification

sorts:

\[ \text{bool} \ \text{nat}; \]

constructors:

\[ \text{true} : \rightarrow \text{bool}; \]
\[ \text{false} : \rightarrow \text{bool}; \]
\[ 0 : \rightarrow \text{nat}; \]
\[ \text{S}: \text{nat} \rightarrow \text{nat}; \]

defined functions:

\[ \text{even} : \text{nat} \rightarrow \text{bool}; \]
\[ \text{oeven} : \text{nat} \rightarrow \text{bool}; \]
\[ \text{odd} : \text{nat} \rightarrow \text{bool}; \]
\[ \text{eodd} : \text{nat} \rightarrow \text{bool}; \]
\[ \text{add} : \text{nat} \ \text{nat} \rightarrow \text{nat}; \]
Implementation of the Spike Tactic

Extraction of the Spike Specification from the Coq Script

The Coq script

\begin{verbatim}
Axiom even2 : even (S 0) = false.
Axiom even3 : \forall x, even (S (S x)) = oeven (S (S x)).

Axiom oeven2 : \forall x, odd x = true \rightarrow oeven (S (S x)) = false.
\end{verbatim}

The Spike specification

\begin{verbatim}
axioms:

even (S(0)) = false
even (S (S (x))) = oeven (S (S (x)))

odd x = true \Rightarrow oeven (S (S (x))) = false
\end{verbatim}
Implementation of the Spike Tactic
Translation of the Theorem even_xx into a Spike Conjecture

The Coq script

Theorem even_xx: 
\forall x, even (add (x x)) = true.

Spike equiv [[even, oeven, odd , eodd]]
greater [ [even, true ,false, S , 0, add], [ add, S, 0] ].

The Spike specification:

equiv:
even oeven odd eodd
greater:
even: true false S 0 add ;
add: S 0 ;

conjectures:
even (add (x , x)) = true
Implementation of the Spike Tactic

A Simplified Version of the Spike Inference System

- Generate (G): \((E \cup \{\phi\}, H) \vdash (E \cup \Phi, H \cup \{\phi\})\), where \(\Phi\) is a strict cover set of \(\phi\).

- Rewriting (R): \((E \cup \{\phi\}, H) \vdash (E \cup \{\phi'\}, H)\) if \(\phi \rightarrow_{A_x \cup HIs} \phi'\), where HIs are induction hypotheses.

- Simplify (S): \((E \cup \{\phi\}, H) \vdash (E \cup \{\phi'\}, H)\), if \((E \cup \{\phi'\} \cup H) <_{\phi} \vdash M \{\phi\}\).

- Tautology (T): \((E \cup \{\phi\}, H) \vdash (E, H)\) if \(\phi\) is a tautology.
Implementation of the Spike Tactic

The implicit proof of even_{xx} generated by Spike

\[
\begin{align*}
\{\text{even}(\text{add}(x, x))=\text{true}\}, \emptyset \vdash_G & \\
\{\text{even}(0)=\text{true}, \text{even}(\text{S}(\text{S}(\text{add}(x1,x1))))=\text{true}\}, \{\text{even}(\text{add}(x,x))=\text{true}\} \vdash_S & \\
\vdots & \\
\{\text{even}(\text{S}(\text{S}(\text{add}(x1,x1))))=\text{true}\}, \{\text{even}(\text{add}(x,x))=\text{true}\} \vdash_R & \\
\{\text{oeven}(\text{add}(x1,x1))=\text{true}\}, \{\text{even}(\text{add}(x,x))=\text{true}\} \vdash_G & \\
\{\text{oeven}(0)=\text{true}, \text{oeven}(\text{S}(\text{S}(\text{add}(x2,x2))))=\text{true}\}, \{\text{even}(\text{add}(x,x))=\text{true}, \text{oeven}(\text{add}(x1,x1))=\text{true}\} \vdash_S & \\
\vdots & \\
\{\text{oeven}(\text{S}(\text{S}(\text{add}(x2,x2))))=\text{true}\}, \{\text{even}(\text{add}(x,x))=\text{true}, \text{oeven}(\text{add}(x1,x1))=\text{true}\} \vdash_R & \\
\{\text{odd } (\text{add } (u3, u3)) = \text{false } \rightarrow \text{even } (\text{add } (u3, u3)) = \text{true}, \text{odd } (\text{add } (u3, u3)) = \text{true } \rightarrow \text{false } = \text{true}\}, \{\text{even}(\text{add}(x,x))=\text{true}, \text{oeven}(\text{add}(x1,x1))=\text{true}\} \vdash_R & \\
\{\text{odd } (\text{add } (u3, u3)) = \text{false } \rightarrow \text{true } = \text{true}, \text{odd } (\text{add } (u3, u3)) = \text{true } \rightarrow \text{false } = \text{true}\}, \{\text{even}(\text{add}(x,x))=\text{true}, \text{oeven}(\text{add}(x1,x1))=\text{true}\} \vdash & \\
\vdots & \\
\emptyset , \{\text{even}(\text{add}(x, x))=\text{true}\} & 
\end{align*}
\]
Implementation of the Spike Tactic
Translating the Spike Proof into Coq Script

1. Translation of induction orderings

- Induction orderings used by Spike are syntactic and exploit the tree representation of terms
- For each formula $F$, a comparison weight $W$ is associated:
  \[
  (\text{fun } \bar{x} \Rightarrow (F, W)),
  \]
  where $\bar{x}$ is the vector of shared variables between $F$ and $W$.
- Abstraction of the Coq formulas into multisets of special terms built from a term algebra provided by the COCCINELLE library [Contejean and al. 2007]
- Example: the functional associated to a conjecture

\[
F_{\_91} : \quad \text{type}_{\text{LF}_{\_91}} := (\text{fun } u1 \Rightarrow ((\text{even (add u1 u1)}) = \text{true}, (\text{Term id_even ((Term id_add ((\text{model_nat u1)::(model_nat u1)::nil))::nil))::(Term id_true nil)::nil})).
\]
Implementation of the Spike Tactic
Translating the Spike Proof into Coq Script

2 Translation of the inference rules
[Stratulat and Demange2011, Henaien and Stratulat2012]
- One-to-one translation, improved by the axiomatic representation of Coq functions
- Example: translating the rewriting operations
  - i) unconditional rewriting with an axiom: \textit{rewrite\_ax}.
    
    \begin{itemize}
    \item \texttt{Hint} \textit{Rewrite addS1 addx0 add1 add2 even1 even2 even3 oeven1 oeven2 odd1 odd2 odd3 eodd1 eodd2: rewrite\_axioms.}
    \item \texttt{Ltac rewrite\_ax:= autorewrite with rewrite\_axioms.}
    \end{itemize}
  
  - ii) conditional rewriting with an axiom:
    
    \begin{itemize}
    \item \textit{rewrite name\_of\_conditional\_axiom.}
    \end{itemize}
  
  - iii) rewriting with a conjecture:
    
    Axiom \textit{C axiom\_93}: \texttt{forall (u1: nat), (even (add u1 u1)) = true}.
    \textit{rewrite C axiom\_93.}
Implementation of the Spike Tactic
Certification of the Coq Script

The equivalent theorem to even_xx is

Theorem true_91: \( \forall u1, \ (\text{even } (\text{add } u1 \ u1)) = \text{true} \).

In the end, the Spike tactic generates:

Require Import “Coq script with the proof of true_91”.
apply true_91.
Implementation of the Spike Tactic

Other Examples

- Proofs of 33 conjectures involved in the validation of a conformity algorithm for the ABR telecommunications protocol
- 60% completely automatically (no arguments provided by the user to the tactic)
- The full Coq scripts can be found on Spike’s website
  
  http://code.google.com/p/spike-prover/
Conclusions and Future Works

Conclusions

- a new Coq tactic

Advantages:
- performs implicit induction reasoning
- highly automatized, using a black-box approach

Disadvantages:
- depends on external tools (Spike)
- application limited to Coq specifications transformable into conditional specifications with orientable equalities
Conclusions and Future Works

Future Works

- automatize the translation process of a fixpoint-based function definition into axioms
- define a set of Coq tactics that can build implicit inductive proofs directly into Coq either automatically or interactively
- certify in a similar way the lazy and mutual induction reasoning provided by cyclic proofs
A. Armando, M. Rusinowitch, and S. Stratulat.
Incorporating decision procedures in implicit induction.

G. Barthe and S. Stratulat.
Validation of the JavaCard platform with implicit induction techniques.

A. Bouhoula, E. Kounalis, and M. Rusinowitch.
Automated mathematical induction.

E. Contejean, P. Courtieu, J. Forest, O. Pons, and X. Urbain.
Certification of automated termination proofs.

J. Courant.
Proof reconstruction.
Preliminary version.

A. Henaien and A. Stratulat.
Performing Implicit Induction Reasoning with Certifying Proof Environment.

C. Kaliszyk.
Validation des preuves par récurrence implicite avec des outils basés sur le calcul des constructions inductives.

M. Protzen.
Lazy generation of induction hypotheses.

M. Rusinowitch, S. Stratulat, and F. Klay.
Mechanical verification of an ideal incremental ABR conformance algorithm.
S. Stratulat and V. Demange.
Automated certification of implicit induction proofs.

S. Stratulat.
A general framework to build contextual cover set induction provers.

S. Stratulat.
Integrating implicit induction proofs into certified proof environments.

The Coq Development Team.

R. Voicu and M. Li.
Descente Infinie proofs in Coq.