Mechanically Certifying Formula-based Noetherian Induction Reasoning

WAIT workshop, Amsterdam – June 28, 2018

Sorin Stratulat

Université de Lorraine

Motivation

Automatisation of induction reasoning:

- large proofs, hard to be checked by humans
- difficulty to certify the underlying code (inference system, orderings,. . .)

☞ (automatized) certification of proof traces by formal certifying tools

Formula-based Noetherian Induction

Noetherian induction principles

Noetherian induction: let $(\mathcal{E}, <)$ be a *well-founded* poset

$$\frac{\forall m \in \mathcal{E}, (\forall k \in \mathcal{E}, k < m \Rightarrow \phi(k)) \Rightarrow \phi(m)}{\forall p \in \mathcal{E}, \phi(p)}$$

☞ $\phi(k)$ are induction hypotheses (IHs)

In a first-order setting, $\mathcal{E}$ can be a set of

- (vector of) terms

$$\frac{\forall \overline{m} \in \mathcal{E}, (\forall \overline{k} \in \mathcal{E}, \overline{k} <_t \overline{m} \Rightarrow \phi(\overline{k})) \Rightarrow \phi(\overline{m})}{\forall \overline{p} \in \mathcal{E}, \phi(\overline{p})}$$

- (first-order) formulas

$$\frac{\forall \gamma \in \mathcal{E}, (\forall \delta \in \mathcal{E}, \delta <_f \gamma \Rightarrow \ ) \Rightarrow}{\forall \rho \in \mathcal{E},}$$

☞ $\phi(\gamma) = \gamma$, $\forall \gamma \in \mathcal{E}$

Noetherian induction principles

Noetherian induction: let $(\mathcal{E}, <)$ be a *well-founded* poset

$$\frac{\forall m \in \mathcal{E}, (\forall k \in \mathcal{E}, k < m \Rightarrow \phi(k)) \Rightarrow \phi(m)}{\forall p \in \mathcal{E}, \phi(p)}$$

☞ $\phi(k)$ are induction hypotheses (IHs)

In a first-order setting, $\mathcal{E}$ can be a set of

- (vector of) terms

$$\frac{\forall \overline{m} \in \mathcal{E}, (\forall \overline{k} \in \mathcal{E}, \overline{k} <_t \overline{m} \Rightarrow \phi(\overline{k})) \Rightarrow \phi(\overline{m})}{\forall \overline{p} \in \mathcal{E}, \phi(\overline{p})}$$

- (first-order) formulas

$$\frac{\forall \gamma \in \mathcal{E}, (\forall \delta \in \mathcal{E}, \delta <_f \gamma \Rightarrow \phi(\delta)) \Rightarrow \phi(\gamma)}{\forall \rho \in \mathcal{E}, \phi(\rho)}$$

☞ $\phi(\gamma) = \gamma, \forall \gamma \in \mathcal{E}$

Noetherian induction principles

Noetherian induction: let $(\mathcal{E}, <)$ be a *well-founded* poset

$$\frac{\forall m \in \mathcal{E}, (\forall k \in \mathcal{E}, k < m \Rightarrow \phi(k)) \Rightarrow \phi(m)}{\forall p \in \mathcal{E}, \phi(p)}$$

☞ $\phi(k)$ are induction hypotheses (IHs)

In a first-order setting, $\mathcal{E}$ can be a set of

- (vector of) terms

$$\frac{\forall \overline{m} \in \mathcal{E}, (\forall \overline{k} \in \mathcal{E}, \overline{k} <_t \overline{m} \Rightarrow \phi(\overline{k})) \Rightarrow \phi(\overline{m})}{\forall \overline{p} \in \mathcal{E}, \phi(\overline{p})}$$

- (first-order) formulas

$$\frac{\forall \gamma \in \mathcal{E}, (\forall \delta \in \mathcal{E}, \delta <_f \gamma \Rightarrow \delta) \Rightarrow \gamma}{\forall \rho \in \mathcal{E}, \rho}$$

☞ $\phi(\gamma) = \gamma$, $\forall \gamma \in \mathcal{E}$

Formula-based induction proof techniques

(to recall, $$\frac{\forall \gamma \in \mathcal{E}, (\forall \delta \in \mathcal{E}, \delta <_f \gamma \Rightarrow \delta) \Rightarrow \gamma}{\forall \rho \in \mathcal{E}, \rho}$$)

- inductionless induction ($\mathcal{E}$ has equalities from the proof)
- term-rewriting induction [Reddy, 1990]
- implicit induction [Bronsard *et al.*, 1994], [Bouhoula *et al.*, 1995]

  ☞ generalization of [Reddy, 1990] and of the inductive procedures for conditional equalities from [Kounalis and Rusinowitch, 1990; Bronsard and Reddy, 1991]
- cyclic induction [Stratulat, 2012a]

  ☞ induction performed along *cycles* of formulas

Advantages: lazy induction, mutual induction

Disadvantages: global ordering (at proof or cycle level), cannot be captured by some specific inference rule

Direct relations between term- and formula-based induction principles

Theorem (customizing term- to formula-based proofs)
The term-based induction principle can be represented as a formula-based induction principle.
Proof. If $\mathcal{E}'$ is the set of term vectors for proving $\phi(\overline{x})$, take $\mathcal{E} = \{\phi(\overline{u}) \mid \overline{u} \in \mathcal{E}'\}$ and define $<_f$ as:

$$\phi(\overline{u}) <_f \phi(\overline{v}) \text{ if } \overline{u} <_t \overline{v}$$

Theorem (customizing formula- to term-based proofs)
The formula-based induction principle can be represented as a term-based induction principle when $\mathcal{E}$ is of the form $\{\phi(\overline{t_1}), \ldots, \phi(\overline{t_n})\}$.
Proof. Define $\overline{u} <_t \overline{v}$ if $\phi(\overline{u}) <_f \phi(\overline{v})$.

☞ the general case is conjectured. Translating implicit into explicit induction proofs is *not* satisfactory [Courant, 1996; Kaliszyk, 2005; Nahon *et al.*, 2009]

What about the 'Descente Infinie' ?

☞ contrapositive version of Noetherian induction

(to recall, $\dfrac{\forall m \in \mathcal{E}, (\forall k \in \mathcal{E}, k < m \Rightarrow \phi(k)) \Rightarrow \phi(m)}{\forall p \in \mathcal{E}, \phi(p)}$)

Definition ('Descente Infinie' induction)

$$\frac{\forall m \in \mathcal{E}, \neg\phi(m) \Rightarrow (\exists k \in \mathcal{E}, k < m \wedge \neg\phi(k))}{\forall p \in \mathcal{E}, \phi(p)}$$

What about the 'Descente Infinie' ?

☞ contrapositive version of Noetherian induction

(to recall, $\dfrac{\forall m \in \mathcal{E}, (\forall k \in \mathcal{E}, k < m \Rightarrow \phi(k)) \Rightarrow \phi(m)}{\forall p \in \mathcal{E}, \phi(p)}$)

Definition ('Descente Infinie' induction)

$$\frac{\forall m \in \mathcal{E}, \neg\phi(m) \Rightarrow (\exists k \in \mathcal{E}, k < m \wedge \neg\phi(k))}{\forall p \in \mathcal{E}, \phi(p)}$$

☞ the formula-based version:

$$\frac{\forall \gamma \in \mathcal{E}, \neg\gamma \Rightarrow (\exists \delta \in \mathcal{E}, \delta < \gamma \wedge \neg\delta)}{\forall p \in \mathcal{E}, p}$$

Proof by formula-based induction

$$0 + y = y$$
$$s(u) + v = s(u + v)$$

$\mathcal{E}$:

$$\{z + 0 = z, 0 + 0 = 0, s(x) + 0 = s(x), s(x + 0) = s(x), s(x) = s(x)\}$$

Induction ordering such that

- $s(x + 0) = s(x) <_f s(x) + 0 = s(x)$, $\forall x \in \mathbb{N}$, and
- $x + 0 = x <_f s(x + 0) = s(x)$, $\forall x \in \mathbb{N}$

☞ multiset extension of syntactic orderings (rpo, mpo,. . .)

Proof (à la Descente Infinie).

By contradiction, we assume that $\mathcal{E}$ has a minimal counterexample.

After case analysis, there is no minimal counterexample. □

Proof by formula-based induction

$$0 + y = y$$

$$s(u) + v = s(u + v)$$

$\mathcal{E}$:

$$\{z + 0 = z, 0 + 0 = 0, s(x) + 0 = s(x), s(x + 0) = s(x), s(x) = s(x)\}$$

Induction ordering such that

- $s(x + 0) = s(x) <_f s(x) + 0 = s(x)$, $\forall x \in \mathbb{N}$, and
- $x + 0 = x <_f s(x + 0) = s(x)$, $\forall x \in \mathbb{N}$

☞ multiset extension of syntactic orderings (rpo, mpo,. . .)

Proof (à la Descente Infinie).

By contradiction, we assume that $\mathcal{E}$ has a minimal counterexample.
After case analysis, there is no minimal counterexample. □

Proof by formula-based induction

$$0 + y = y$$
$$s(u) + v = s(u + v)$$

$\mathcal{E}$:

$\{z + 0 = z, 0 + 0 = 0, s(x) + 0 = s(x), s(x + 0) = s(x), s(x) = s(x)\}$

Induction ordering such that

- $s(x + 0) = s(x) <_f s(x) + 0 = s(x)$, $\forall x \in \mathbb{N}$, and
- $x + 0 = x <_f s(x + 0) = s(x)$, $\forall x \in \mathbb{N}$

☞ multiset extension of syntactic orderings (rpo, mpo,. . .)

Proof (à la Descente Infinie).

By contradiction, we assume that $\mathcal{E}$ has a minimal counterexample.

After case analysis, there is no minimal counterexample. □

Proof by formula-based induction

$$0 + y = y$$

$$s(u) + v = s(u + v)$$

$\mathcal{E}$:

$$\{z + 0 = z, 0 + 0 = 0, s(x) + 0 = s(x), s(x + 0) = s(x), s(x) = s(x)\}$$

Induction ordering such that

- $s(x + 0) = s(x) <_f s(x) + 0 = s(x)$, $\forall x \in \mathbb{N}$, and
- $x + 0 = x <_f s(x + 0) = s(x)$, $\forall x \in \mathbb{N}$

☞ multiset extension of syntactic orderings (rpo, mpo,. . .)

Proof (à la Descente Infinie).

By contradiction, we assume that $\mathcal{E}$ has a minimal counterexample.

After case analysis, there is no minimal counterexample. □

Proof by formula-based induction

$$0 + y = y$$
$$s(u) + v = s(u + v)$$

$\mathcal{E}$:

$$\{z + 0 = z, 0 + 0 = 0, s(x) + 0 = s(x), s(x + 0) = s(x), s(x) = s(x)\}$$

Induction ordering such that

- $s(x + 0) = s(x) <_f s(x) + 0 = s(x)$, $\forall x \in \mathbb{N}$, and
- $x + 0 = x <_f s(x + 0) = s(x)$, $\forall x \in \mathbb{N}$

☞ multiset extension of syntactic orderings (rpo, mpo,. . .)

Proof (à la Descente Infinie).

By contradiction, we assume that $\mathcal{E}$ has a minimal counterexample.

After case analysis, there is no minimal counterexample. □

Mechanical Proof Certification Methodology

The Coq certification environment

- Coq: proof assistant based on the Calculus of Inductive Constructions (http://coq.inria.fr)
  ☞ integrates Noetherian induction
- proof certification
  ☞ Curry-Howard correspondence:
    - proofs as programs, written in the Gallina language
    - formulas as types

  ☞ proof terms are checked by the kernel

Methodology for certifying formula-based induction reasoning

Idea: explicitly formalize

(1) the induction ordering and the formula weights by means of a syntactic representation of formulas
(2) the formula-based induction principle
(3) the inference steps from the formula-based proof

Advantage: no proof reconstruction techniques are required

Weights for formulas

☞ abstract term algebra: COCCINELLE [Contejean *et al.*, 2007]

- syntactic representation of terms in Coq

  Inductive **term** : Set :=
  | Var : variable → **term**
  | Term : symbol → list **term** → **term**

Defining induction orderings in COCCINELLE

```
Inductive rpo (bb : nat) : term → term → Prop :=
    | Subterm : ∀ f l t s, mem equiv s l → rpo_eq bb t s → rpo bb t (Term f l)
    | Top_gt :
       ∀ f g l l', prec P g f → (∀ s', mem equiv s' l' → rpo bb s' (Term f l)) →
          rpo bb (Term g l') (Term f l)
    | Top_eq_lex :
       ∀ f g l l', status P f = Lex → status P g = Lex → prec_eq P f g → (length
l = length l' ∨ (length l' ≤ bb ∧ length l ≤ bb)) → rpo_lex bb l' l →
          (∀ s', mem equiv s' l' → rpo bb s' (Term g l)) →
          rpo bb (Term f l') (Term g l)
    | Top_eq_mul :
       ∀ f g l l', status P f = Mul → status P g = Mul → prec_eq P f g →
rpo_mul bb l' l →
          rpo bb (Term f l') (Term g l)

    with rpo_mul ( bb : nat) : list term → list term → Prop :=
    | List_mul : ∀ a lg ls lc l l',
       permut0 equiv l' (ls ++ lc) → permut0 equiv l (a :: lg ++ lc) →
       (∀ b, mem equiv b ls → ∃ a', mem equiv a' (a :: lg) ∧ rpo bb b a') →
       rpo_mul bb l' l.
```

Notation less := (**rpo_mul** (bb)).

Defining Coq specification and translation functions

```
Fixpoint plus (x y:nat): nat :=
match x with
| O ⇒ y
| (S x') ⇒ S (plus x' y)
end.
```

- COCCINELLE symbols: id_0, id_S, id_plus
  ☞ precedence and status
- translation function for any natural into a COCCINELLE term

```
Fixpoint model_nat (v: nat): term :=
match v with
| O ⇒ (Term id_0 nil)
| (S x) ⇒ let r := model_nat x in (Term id_S (r::nil))
end.
```

Defining Coq specification and translation functions

```
Fixpoint plus (x y:nat): nat :=
match x with
| O ⇒ y
| (S x') ⇒ S (plus x' y)
end.
```

- COCCINELLE symbols: id_0, id_S, id_plus
  ☞ precedence and status
- translation function for any natural into a COCCINELLE term

```
Fixpoint model_nat (v: nat): term :=
match v with
| O ⇒ (Term id_0 nil)
| (S x) ⇒ let r := model_nat x in (Term id_S (r::nil))
 end.
```

Defining the set $\mathcal{E}$ and formula weights from a Spike proof

- syntactically represent each conjecture ϕ as a weight w_ϕ
- the variables are shared using anonymous functions

$$\texttt{fun}\ \overline{x} \Rightarrow (\phi, w_\phi)$$

- $\mathcal{E}'$ will consist of anonymous functions

Example

$\mathcal{E}'$: {(fun *u1* ⇒ ((plus *u1* 0) = *u1* , w_1::w_2::nil),...}, where

- w_1 is (Term id_plus ((model_nat *u1*):: (Term id_0 nil):: nil))
- w_2 is model_nat *u1*

- $\mathcal{E}$ is computed from $\mathcal{E}'$

Formalizing the formula-based induction principle

☞ COCCINELLE extended with dual computable function for 'less '

Adding lemmas showing

- its equivalence with 'less '
- properties (well-foundedness, stability)

Specifying the formula-based induction principle

(to recall, $$\frac{\forall \gamma \in \mathcal{E}, (\forall \delta \in \mathcal{E}, \delta <_f \gamma \Rightarrow \delta) \Rightarrow \gamma}{\forall \rho \in \mathcal{E}, \rho}$$)

(1) (main lemma)
∀ *F*, In *F* $\mathcal{E}'$ → ∀ *u1*, (∀ *F'*, In *F'* $\mathcal{E}'$ → ∀ *e1*, less (snd (*F' e1*)) (snd (*F u1*)) → fst (*F' e1*)) → fst (*F u1*).

(2) (all_true lemma)
∀ *F*, In *F* $\mathcal{E}'$ → ∀ *u1*: **nat**, fst (*F u1*).

☞ (2) is derived from (1) using Coq's Noetherian induction

Proving the main lemma

☞ the anonymous functions from $\mathcal{E}'$ are treated independently, one-by-one.

the conjecture of each anonymous function may be proved using (instances of) other conjectures that are

- logically equivalent (deductive reasoning)
- smaller

Proving logical equivalences

- variable instantiations are controlled by Coq functional schemas [Barthe and Courtieu, 2002]

Example (x is replaced by 0 and $(S\ z)$ using f)

```
Fixpoint f (x: nat) {struct x} : nat :=
 match x with
| 0 ⇒ 0
| (S z) ⇒ 0
end.

Functional Scheme f_ind := Induction for f Sort Prop.
```

The instances are generated by the Coq script

```
pattern x, (f x).  apply f_ind.
```

One-to-one translations

- Equality reasoning using rewriting
  - rewriting $C[f(t)]$ with $f(x) = \ldots$ yields
    `pattern` t`.  simpl` f`. cbv  beta.`
    - `pattern` t isolates t from C,
    - `simpl` f rewrites $f(t)$,
    - `cbv beta` puts back the resulted term in C.
- Tautologies (of the form $t = t$) are proved using `auto.`

Weight comparisons

User-defined tacticals for automatization:

- rewrite with model functions
- compute the ordering

(1) terms of the form $(model\_sort\ (f\ x_1 \cdots x_n))$ will be replaced by $(Id\_f\ (model\_sort\ x_1) \cdots (model\_sort\ x_n))$

(2) the replacement of terms of the form (model_*sort* t) with COCCINELLE abstraction variables of the form (*Var* i), $i \in \mathbb{N}$;

(3) computing by reflection the comparison result of weights with abstraction variables;

(4) the use of stability property of 'less ' to compare with abstraction variables instead of original weights.

Examples

Implicit induction inference systems

- inference rules: transitions between states

$$(conjectures, premises)$$

☞ premises are 'previous' conjectures with no minimal counterexamples (w.r.t. $<_f$).

- derivation of E^0 with an inference system I:
$(E^0,\ \emptyset) \vdash_I (E^1,\ H^1) \vdash_I \ldots$

- proof: finite derivation whose last state has no conjectures:
$(E^0,\ \emptyset) \vdash_I (E^1,\ H^1) \vdash_I \ldots \vdash_I (\emptyset, H^n)$

The concrete inference system I_{imp}

☞ Ax are axioms oriented into rewrite rules

GenNat (G): $(E \cup \{\phi\langle x \rangle\},\ H) \vdash_{I_{imp}} (E \cup \{\phi_1, \phi_2\},\ H \cup \{\phi\})$,
where $\phi\{x \mapsto 0\} \to_{Ax} \phi_1$, $\phi\{x \mapsto s(x')\} \to_{Ax} \phi_2$

SimpEq (S): $(E \cup \{\phi\},\ H) \vdash_{I_{imp}} (E \cup \{\psi\},\ H)$,
if $\phi \to_{Ax \cup (E \cup H)_{\leq_f \phi}} \psi$

ElimTaut (E): $(E \cup \{\phi\}, H) \vdash_{I_{imp}} (E,\ H)$,
if ϕ is a tautology

An I_{imp}-proof of $x + 0 = x$

Rewrite rules

$$0 + y \to y$$

$$s(u) + v \to s(u + v)$$

I_{imp}-proof of $x + 0 = x$:

$(\{x + 0 = x\}, \emptyset)$

$\vdash^{G}_{I_{imp}} (\{0 = 0, s(x' + 0) = s(x')\}, \{x + 0 = x\})$

$\vdash^{S}_{I_{imp}} (\{0 = 0, s(x') = s(x')\}, \{x + 0 = x\})$

$\vdash^{E(2)}_{I_{imp}} (\emptyset, \{x + 0 = x\})$

An I_{imp}-proof of $x + 0 = x$

Rewrite rules

$$0 + y \to y$$

$$s(u) + v \to s(u + v)$$

I_{imp}-proof of $x + 0 = x$:

$(\{x + 0 = x\}, \emptyset)$

$\vdash^{G}_{I_{imp}} (\{0 = 0, s(x' + 0) = s(x')\}, \{x + 0 = x\})$

$\vdash^{S}_{I_{imp}} (\{0 = 0, s(x') = s(x')\}, \{x + 0 = x\})$

$\vdash^{E(2)}_{I_{imp}} (\emptyset, \{x + 0 = x\})$

GenNat (G): $(E \cup \{\phi\langle x \rangle\},\ H) \vdash_{I_{imp}} (E \cup \{\phi_1, \phi_2\},\ H \cup \{\phi\})$,
where $\phi\{x \mapsto 0\} \to_{Ax} \phi_1$, $\phi\{x \mapsto s(x')\} \to_{Ax} \phi_2$

An I_{imp}-proof of $x + 0 = x$

Rewrite rules

$$0 + y \to y$$

$$s(u) + v \to s(u + v)$$

I_{imp}-proof of $x + 0 = x$:

$(\{x + 0 = x\}, \emptyset)$

$\vdash^{G}_{I_{imp}} (\{0 = 0, s(x' + 0) = s(x')\}, \{x + 0 = x\})$

$\vdash^{S}_{I_{imp}} (\{0 = 0, s(x') = s(x')\}, \{x + 0 = x\})$

$\vdash^{E(2)}_{I_{imp}} (\emptyset, \{x + 0 = x\})$

SimpEq (S): $(E \cup \{\phi\},\ H) \vdash_{I_{imp}} (E \cup \{\psi\},\ H)$,
if $\phi \to_{Ax \cup (E \cup \Phi \cup H)_{\leq_f \phi}} \psi$

An I_{imp}-proof of $x + 0 = x$

Rewrite rules

$$0 + y \to y$$

$$s(u) + v \to s(u + v)$$

I_{imp}-proof of $x + 0 = x$:

$(\{x + 0 = x\}, \emptyset)$

$\vdash^{G}_{I_{imp}} (\{0 = 0, s(x' + 0) = s(x')\}, \{x + 0 = x\})$

$\vdash^{S}_{I_{imp}} (\{0 = 0, s(x') = s(x')\}, \{x + 0 = x\})$

$\vdash^{E(2)}_{I_{imp}} (\emptyset, \{x + 0 = x\})$

ElimTaut (*E*): $(E \cup \{\phi\}, H) \vdash_{I_{imp}} (E,\ H)$,
if ϕ is a tautology

Certifying the I_{imp}-proof of $x + 0 = x$

- ordering

```
Definition index (f:symb) :=
  match f with
  | id_0 ⇒ 2
  | id_S ⇒ 3
  | id_plus ⇒ 7
  end.
```

```
Definition status (f:symb) :=
  match f with
  | id_0 ⇒ rpo.Mul
  | id_S ⇒ rpo.Mul
  | id_plus ⇒ rpo.Mul

  end.
```

- list of anonymous functions

```
Definition type_LF := nat → Prop × List.list term.
Definition ℰ' := [F_1, F_2, F_3, F_4].
(* for all equalities from the proof *)
```

```
Definition F_1 : type_LF:= (fun u1 ⇒ ((plus u1 0) = u1 ,
(Term id_plus ((model_nat u1):: (Term id_0
nil)::nil))::(model_nat u1)::nil)).

Definition F_2 : type_LF:= (fun _ ⇒ (0 = 0, (Term id_0
nil)::(Term id_0 nil)::nil)).

Definition F_3 : type_LF:= (fun u2 ⇒ ((S (plus u2 0)) = (S
u2), (Term id_S ((Term id_plus ((model_nat u2):: (Term id_0
nil)::nil))::nil))::(Term id_S ((model_nat u2)::nil))::nil)).

Definition F_4 : type_LF:= (fun u2 ⇒ ((S u2) = (S u2),
(Term id_S ((model_nat u2)::nil) )::(Term id_S ((model_nat
u2)::nil))::nil)).
```

Proof of the main lemma

$\forall$ *F*, In *F* $\mathcal{E}'$ $\rightarrow$ $\forall$ *u1*, ($\forall$ *F'*, In *F'* $\mathcal{E}'$ $\rightarrow$ $\forall$ *e1*, less (snd (*F' e1*)) (snd (*F u1*)) $\rightarrow$ fst (*F' e1*)) $\rightarrow$ fst (*F u1*).

Proof.

By case analysis.

- F_1 (recall, (plus *u1* 0) = *u1*): instantiate *u1* by
  `pattern u1, (f u1).`
  - case *u1* is 0: by `auto.`
  - case *u1* is S *u2*: choose as IH
    F_3 (recall, S (plus *u2* 0) = (S *u2*)), then simplify
- F_2 (recall, 0=0): by `auto.`
- F_3: choose as IH F_1, then simplify
- F_4 (recall, (S *u2*) = (S *u2*)): by `auto.`

□

Discussions

Implicit induction reasoning:

- easily automatized (Spike, RRL)
- generate large Spike proofs
  - validation of the JavaCard platform [Barthe and Stratulat, 2003]

instruction	proved	lemmas	Generate	U. R.	C. R.	Subsumption	Taut.	time
ACONST_NULL	yes	0	0	4	1	0	1	0.5s
ALOAD	n.y.	0	0	0	0	0	0	0.0
ARITH	yes	33	100	8771	2893	979	2178	8m

  - validation of telecommunication protocols[Rusinowitch *et al.*, 2003] ☞ 40% of the lemmas are automatically certified

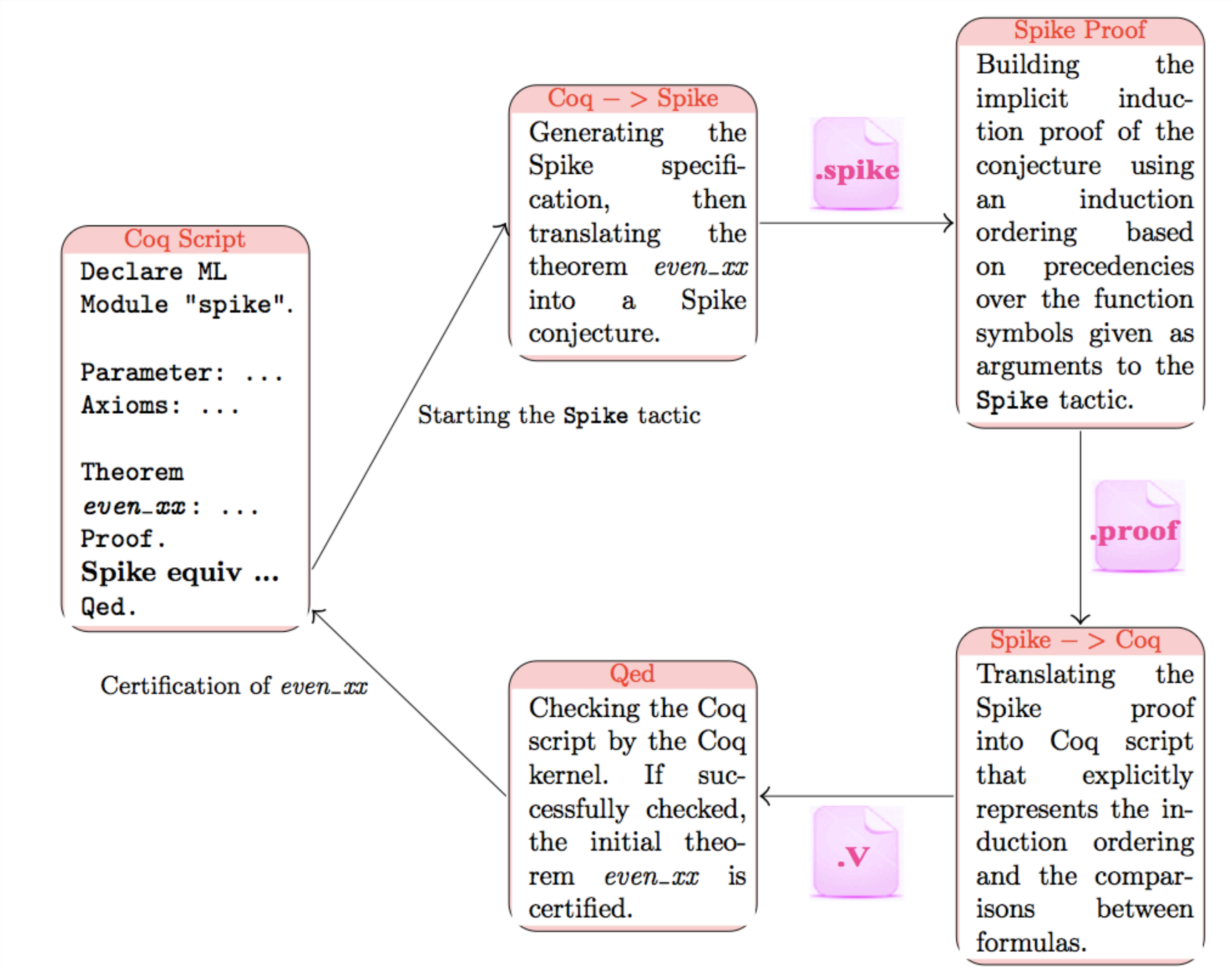
The certification process may be less effective

- check every reductive ordering constraint
  ☞ multiple calls to COCCINELLE functions
- check every formula from the proof
  ☞ large $\mathcal{E}'$ sets.

The Coq tactic Spike

☞ solves the translation problems at specification level

```
Theorem even_xx: ∀ x, even (add (x  x)) = true.
 Proof.
 Spike    equiv [[even, odd]]
          greater [ [even, true ,false, S , 0, add],
                             [ add, S, 0] ].
 Qed.
```



Cyclic reasoning on one slide

☞ non-reductive reasoning

$$0 + y = y$$
$$s(u) + v = s(u + v)$$

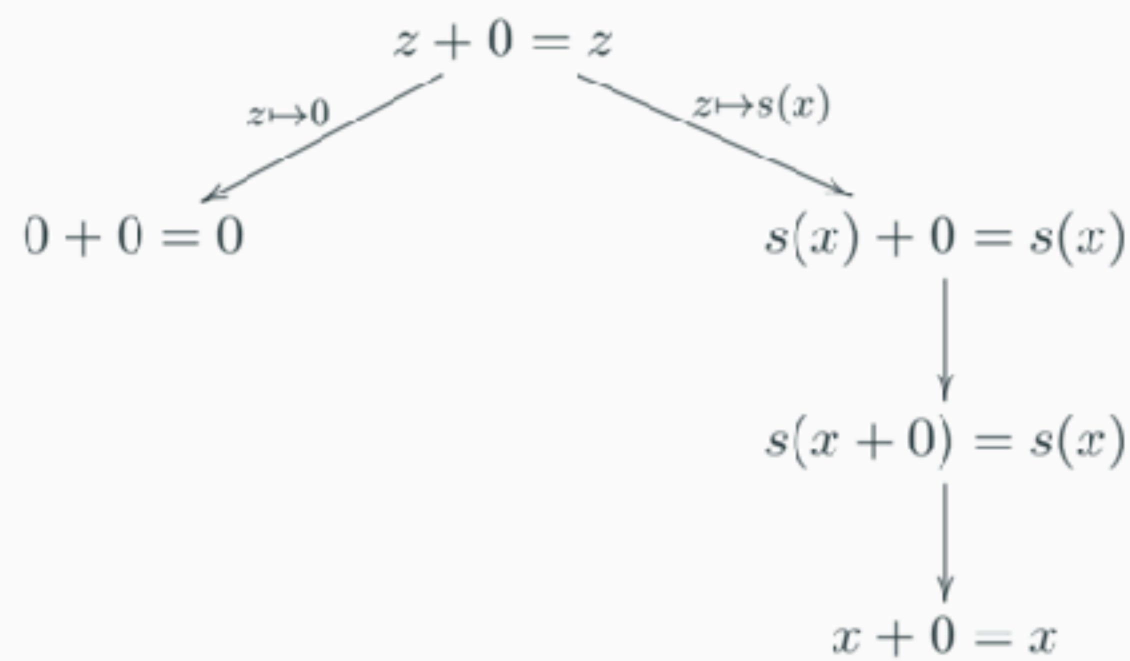
$\mathcal{E}$: $\{z + 0 = z, 0 + 0 = 0, s(x) + 0 = s(x)\}$

☞ less elements in $\mathcal{E}$

Cyclic reasoning on one slide

☞ non-reductive reasoning

$$0 + y = y$$
$$s(u) + v = s(u + v)$$



$\mathcal{E}$: $\{z + 0 = z, 0 + 0 = 0, s(x) + 0 = s(x)\}$

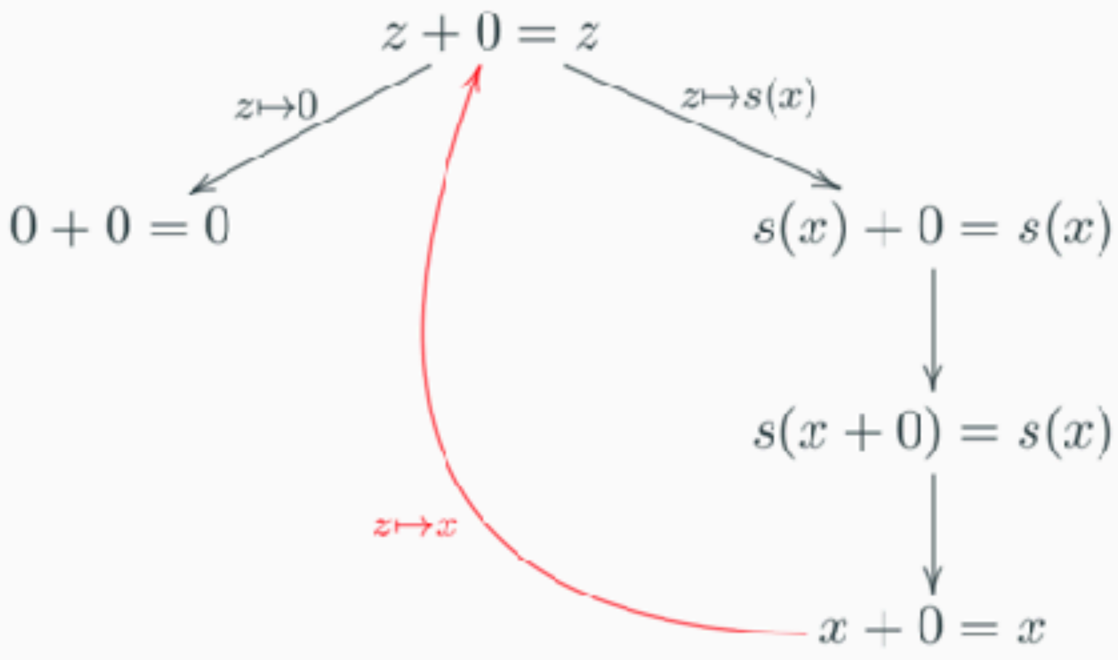
☞ less elements in $\mathcal{E}$

Cyclic reasoning on one slide

☞ non-reductive reasoning

$$0 + y = y$$

$$s(u) + v = s(u + v)$$



$$x + 0 = x <_f s(x) + 0 = s(x)$$

$\mathcal{E}$: $\{z + 0 = z, 0 + 0 = 0, s(x) + 0 = s(x)\}$

☞ less elements in $\mathcal{E}$

Conclusions and Future Work

Conclusions

- methodology for automatically certifying any formula-based induction proof
  ☞ implicit induction, cyclic induction

- automatic Coq certification of Spike's implicit induction proofs
  ☞ Coq checkpoints for Spike specifications and proofs:
  (1) (ground) convergence and completeness properties: acceptance of the translated functions by Coq
  (2) variable instantiation schemas: functional schemes
  (3) certifying the induction principle: the main lemma

  ☞ limited Spike specifications + control in the automatic translation of the proofs

Future Work

- Spike proof certification : allow more general specifications and inference rules
  ☞ certifying reductive-free cyclic proofs
- building a formula-based induction proof environment directly in Coq
  - for lazy reasoning and cyclic induction
  - for automatically performing implicit induction
    ☞ direct use of Coq tactics and no translation
- dissemination and implementation for other proof environments (Isabelle/HOL, PVS, . . .)

More information at

- recent article (2017)

  `S. Stratulat. Mechanically certifying formula-based Noetherian induction reasoning. Journal of Symbolic Computation, 41 pages.`

- http://code.google.com/p/spike-prover/

More information at

- recent article (2017)

  S. Stratulat. Mechanically certifying formula-based Noetherian induction reasoning. Journal of Symbolic Computation, 41 pages.

- http://code.google.com/p/spike-prover/

Thank you !

References

G. Barthe and P. Courtieu. Efficient reasoning about executable specifications in Coq. In *Theorem Proving in Higher Order Logics*, volume 2410 of *LNCS*, pages 31–46. Springer Berlin, 2002.

G. Barthe and S. Stratulat. Validation of the JavaCard platform with implicit induction techniques. In R. Nieuwenhuis, editor, *RTA (Rewriting Techniques and Applications)*, volume 2706 of *LNCS*, pages 337–351. Springer, 2003.

A. Bouhoula, E. Kounalis, and M. Rusinowitch. Automated mathematical induction. *Journal of Logic and Computation*, 5(5):631–668, 1995.

F. Bronsard and U. S. Reddy. Conditional rewriting in Focus. In *Conditional and Typed Rewriting Systems*, pages 1–13, 1991.

F. Bronsard, U.S. Reddy, and R. Hasker. Induction using term orderings. In *CADE (Conf. on Automated Deduction)*, volume 814 of *LNCS*, pages 102–117. Springer, 1994.

R. M. Burstall. Proving properties of programs by structural induction. *The Computer Journal*, 12:41–48, 1969.

E. Contejean, P. Courtieu, J. Forest, O. Pons, and X. Urbain. Certification of automated termination proofs. *Frontiers of Combining Systems*, pages 148–162, 2007.

J. Courant. Proof reconstruction. Research Report RR96-26, LIP, 1996. Preliminary version.

G. Gonthier, A. Asperti, J. Avigad, Y. Bertot, C. Cohen, F. Garillot, S. Le Roux, A. Mahboubi, R. O'Connor, S. Ould Biha, I. Pasca, L. Rideau, A. Solovyev, E. Tassi, and L. Théry. A machine-checked proof of the Odd Order Theorem. In S. Blazy, C. Paulin-Mohring, and D. Pichardie, editors, *Interactive Theorem Proving - 4th International Conference, ITP 2013,* *Rennes, France, July 22-26, 2013. Proceedings*, volume 7998 of *Lecture Notes Computer Science*, pages 163–179. Springer, 2013.

A. Henaien and S. Stratulat. Performing implicit induction reasoning with certifying proof environments. In A. Bouhoula, T. Ida, and F. Kamareddine, editors, *Proceedings Fourth International Symposium on Symbolic Computation in Software Science, Gammarth, Tunisia, 15-17 December 2012*, volume 122 of *Electronic Proceedings in Theoretical Computer Science*, pages 97–108. Open Publishing Association, 2013.

C. Kaliszyk. Validation des preuves par récurrence implicite avec des outils basés sur le calcul des constructions inductives. Master's thesis, Université Paul Verlaine - Metz, 2005.

D.E. Knuth and P.B. Bendix. Simple word problems in universal algebras. In *Computational Problems in Abstract Algebra*, pages 263–297, 1970.

E. Kounalis and M. Rusinowitch. Mechanizing inductive reasoning. In *Proceedings of the eighth National conference on Artificial intelligence - Volume 1*, AAAI'90, pages 240–245. AAAI Press, 1990.

J. McCarthy. A basis for a mathematical theory of computation. In *Computer Programming and Formal Systems*, pages 33–70. North-Holland, 1963.

D. R. Musser. On proving inductive properties of abstract data types. In *POPL*, pages 154–162, 1980.

F. Nahon, C. Kirchner, H. Kirchner, and P. Brauner. Inductive proof search modulo. *Annals of Mathematics and Artificial Intelligence*, 55(1–2):123–154, 2009.

H. Poincaré. *La Science et l'Hypothèse*. Flammarion, 1902.

U.S. Reddy. Term Rewriting Induction. *Proceedings of the 10th*

International Conference on Automated Deduction, pages 162–177, 1990.

M. Rusinowitch, S. Stratulat, and F. Klay. Mechanical verification of an ideal incremental ABR conformance algorithm. *Journal of Automated Reasoning*, 30(2):53–177, 2003.

S. Stratulat and V. Demange. Automated certification of implicit induction proofs. In *CPP'2011 (First International Conference on Certified Programs and Proofs)*, volume 7086 of *Lecture Notes Computer Science*, pages 37–53. Springer Verlag, 2011.

S. Stratulat. A general framework to build contextual cover set induction provers. *J. Symb. Comput.*, 32(4):403–445, 2001.

S. Stratulat. Integrating implicit induction proofs into certified proof environments. In *IFM'2010 (8th International Conference on Integrated Formal Methods)*, volume 6396 of *Lecture Notes in Computer Science*, pages 320–335, 2010.

S. Stratulat. A unified view of induction reasoning for first-order logic. Séance poster de la conférence *Turing-100*, Juin 2012.

The CompCert project, 2014.

C.-P. Wirth. Descente infinie + Deduction. *Logic Journal of the IGPL*, 12(1):1–96, 2004.

H. Zhang, D. Kapur, and M. S. Krishnamoorthy. A mechanizable induction principle for equational specifications. In *Proceedings of the 9th International Conference on Automated Deduction*, pages 162–181, London, UK, 1988. Springer-Verlag.