Conjecture Synthesis for Inductive Theory Formation

Moa Johansson

Dipartimento di Informatica
Università degli Studi di Verona

Workshop on Formal and Automated Theorem Proving and Applications
Belgrade 29-30 January 2010

My Background

Automated inductive theorem proving in HOL. PhD at the University of Edinburgh (2009), now at Università degli Studi di Verona.

- **Case-Analysis for Rippling and Inductive Proof**.
  M. Johansson, L. Dixon and A. Bundy. Submitted to ITP 2010.
- **Lemma Discovery Techniques and Middle-Out Reasoning for Automated Inductive Proofs**.
  M. Johansson, L. Dixon and A. Bundy. Submitted to ITP 2010.
- **Conjecture Synthesis for Inductive Theories**.
  M. Johansson, L. Dixon and A. Bundy. Under revision for JAR, 2010.

My Background

Automated inductive theorem proving in HOL. PhD at the University of Edinburgh (2009), now at Università degli Studi di Verona.

- **Case-Analysis for Rippling and Inductive Proof**.
  M. Johansson, L. Dixon and A. Bundy. Submitted to ITP 2010.
- **Lemma Discovery Techniques and Middle-Out Reasoning for Automated Inductive Proofs**.
  M. Johansson, L. Dixon and A. Bundy. Submitted to ITP 2010.
- **Conjecture Synthesis for Inductive Theories**.
  M. Johansson, L. Dixon and A. Bundy. Under revision for JAR, 2010.

Introduction and Motivation

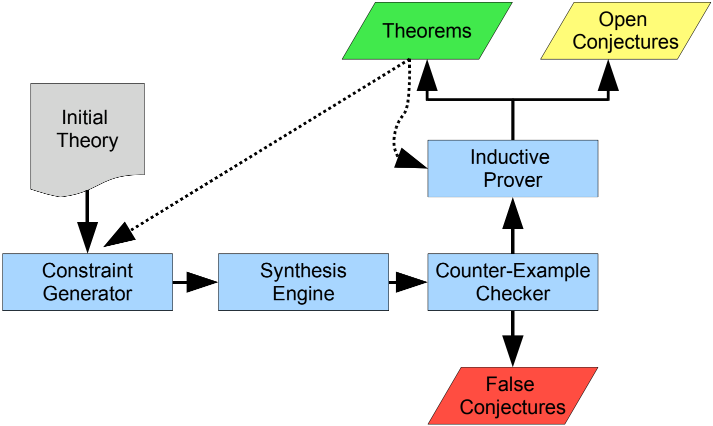
Induction: Reasoning about repetition, e.g. recursive datatypes and functions.
Challenge: Automate lemma discovery for (rewrite based) inductive proofs.

- Lemma typically need a separate inductive proof, not just an intermediate result.
- Generally assumed to require user intervention.
- Large libraries of previously proved theorems/lemmas e.g. Isabelle.
- Libraries insufficient for new theory developments.

IsaCoSy: Inductive Conjecture Synthesis

- Build conjectures from available functions, datatypes and variables.
- General: Can be applied to any recursive datatype defined in Isabelle without modification.
- Key idea for tractability: Turn rewriting upside-down.
  - Only generate irreducible terms.
- Enforced by constraints on term-synthesis. Avoid naive generate-and-test.
- Counter-example checking (Isabelle) + automatic inductive prover (IsaPlanner)
- New theorems provide more constraints.

Overview of IsaCoSy



Motivating Example: Definitions of List Reversal

Definition of *rev*:

$$rev([\,]) = [\,]$$

$$rev(h\#t) = rev(t)@[h]$$

Constraints on synthesis:

- Disallow [ ] to occur as argument of *rev*.
- Disallow # (cons) to occur as argument of *rev*.

Motivating Example: Distinctness for Lists

From definition of lists, Isabelle automatically derives:

$$[\,] \neq (h \# t)$$

Constraint on synthesis:

- Disallow $[\,]$ and $\#$ as simultaneous top-level arguments to opposite sides of an equality.

Motivating Example: Reflexivity

Reflexivity as a rewrite rule:

$$(x = x) = True$$

Constraint on synthesis:

- Disallow both sides of equality to be instantiated to the same term.

Motivating Example: Commutativity

Suppose we know that max is commutative:

$$max\ x\ y = max\ y\ x$$

Not a rewrite rule. Derive constraint on argument order:

- Measure of 1st argument $\geq$ measure of 2nd argument.
- Cuts out many symmetries.

Constraint Generation

- Initial constraints automatically derived from rules in input theory.
- Expressed in IsaCoSy's constraint language.
- Constraint from rule stored for its principal function symbol.

Constraint Generation

- Initial constraints automatically derived from rules in input theory.
- Expressed in IsaCoSy's constraint language.
- Constraint from rule stored for its principal function symbol.

Reflexivity: $(x = x) = True$

$UnEqual(arg_1, arg_2)$

List Distinctness: $[\,] \neq (h \# t)$

$NotAllowed(arg_1, [\,])$

$NotAllowed(arg_2, \#)$

Additional Heuristics

Can be configured by the user:

- Number of different variables. Default: $1 +$ max arity of functions.
- Where variables occur e.g. $Vars(RHS) \subseteq Vars(LHS)$
- Eagerly check for associativity and commutativity prior to synthesis.

The Synthesis Process

- Input: Initial constraints, max size of terms, user controlled heuristics.
- Start small: $\underbrace{?h_1}_{\textit{size } 1} = \underbrace{?h_2}_{\textit{size } 1}$
- Insert allowed constants and variables.
- After each size-iteration, counter-example check and prove.
- Generate new constraints from any new theorems.
- Increase term-size.

Evaluation

- Evaluated on theories about natural numbers, lists and binary trees.
- **Quality:** How does the set of theorems produced by IsaCoSy's compare to Isabelle's libraries?
- **Efficiency:** How much does IsaCoSy's heuristics improve over naive generate and test?

Natural Numbers

10/16 synthesised theorems are also in Isabelle's library:

$$a + 0 = a \qquad a + Suc\ b = Suc(a + b)$$
$$a * 0 = 0 \qquad a * Suc\ b = a + (a * b)$$
$$a + b = b + a \qquad a * b = b * a$$
$$(a + b) + c = a + (b + c) \qquad (a * b) * c = a * (b * c)$$
$$\left.\begin{array}{r}(a * b) + (c * b) = (a + c) * b \\ (a * b) + (a * c) = (b + c) * a\end{array}\right\} + \text{six variants not in library}$$

Isabelle's library contains another 2 theorems:

$$(Suc\ m) + n = m + (Suc\ n) \qquad x + (y + z) = y + (x + z)$$

Recall: 83%

Precision: 63%

Lists

9/24 synthesised theorems are also in Isabelle's library (with @ denoting append):

$$a \,@\, [\,] = a$$
$$rev(rev\ a) = a$$
$$rev(map\ a\ b) = map\ a(rev\ b)$$
$$(map\ a\ b) \,@\, (map\ a\ c) = map\ a\ (b \,@\, c)$$
$$foldl\ a\ (foldl\ a\ b\ c)\ d = foldl\ a\ b\ (c \,@\, d)$$
$$foldr\ a\ b\ (foldr\ a\ c\ d) = foldr\ a\ (b \,@\, c)\ d$$

$$(a \,@\, b) \,@\, c = a \,@\, (b \,@\, c)$$
$$(rev\ a) \,@\, (rev\ b) = rev\ (b \,@\, a)$$
$$len(rev\ a) = len\ a$$

$$\Big\} + 13 \text{ theorems}$$

- Isabelle's library contains only the 9 theorems above.
- Extra 13 theorems mostly about *rev* and *append*.

Recall: 100%

Precision: 38%

Binary Trees

Small theory about binary trees, involving functions *mirror*, *nodes* and *height*. No Isabelle library to compare.

$$mirror(mirror\ t) = t \qquad size(mirror\ t) = size\ t$$
$$height(mirror\ t) = height\ t \qquad max\ (size\ t)\ (height\ t) = size\ t$$

Run-times

- Compared to naive version: exponential cut in search space size.
- Synthesis generally takes a couple of hours, depending on maximum term size.
- Can cut run-times by restricting instantiation of type-variables for polymorphic datatypes (e.g. lists).
- Largest portion of time spent counter-example checking.

Future Directions and Applications

- Theory Library Formation:
  - Novel theory developments, generating routine library lemmas.
  - Generate benchmarks for inductive provers.
  - Generate libraries in Rich Model Language that can be shared between systems?
- Synthesis for generating/refining loop invariants.
  - *Refinement and Term Synthesis in Loop Invariant Generation. Maclean, Ireland, Atkey, Dixon. WING 2009*.

Conclusions & Summary

- IsaCoSy: Inductive theory formation by synthesis.
- Only generates irreducible terms, which keeps search space tractable.
- High recall, many interesting theorems synthesised.
- Lower precision. Too many variants of theorems generated.
- Using synthesised background theory increase power of prover. Mange to prove harder theorems automatically.
- `dream.inf.ed.ac.uk/projects/lemmadiscovery/`

Current Work in Verona

With Maria Paola Bonacina. Just starting:

- Extending SMT solver with F.O. reasoning: DPLL($\Gamma + T$).
  - DPLL is good at large conjunctions.
  - Rewrite based F.O. proves can handle quantifiers better.
- Contrast efficiency/expressiveness of logics. Typed/untyped settings.
- Extending combination of theories for F.O. provers.
- Combining non-stably infinite theories.