IsaCoSy: Synthesis of Inductive Theorems

Moa Johansson, Lucas Dixon, and Alan Bundy

University of Edinburgh
Informatics Forum, 10 Crichton Street, Edinburgh EH8 9AB, UK.
{moa.johansson, l.dixon, a.bundy}@ed.ac.uk

Abstract. We have implemented a program for inductive theory formation, called IsaCoSy, which synthesises conjectures about recursively defined datatypes and functions. Only irreducible terms are generated, which keeps the search space tractably small. The synthesised terms are filtered through counter-example checking and then passed on to the automatic inductive prover IsaPlanner. Experiments have given promising results, with high recall of 83% for natural numbers and 100% for lists when compared to libraries for the Isabelle theorem prover. However, precision is somewhat lower, 38-63%.

1 Introduction

Discovering unknown theorems and lemmas is a major challenge for automated inductive theorem proving. It has generally been assumed that such discovery requires user intervention. Consequently, most theorem provers rely on the user to supply any additional lemmas that might be needed for a proof. Interactive theorem provers, such as Isabelle [?], often come with large theory libraries of previously proved theorems and lemmas. Automating the formation of these theory libraries is an important challenge. Given a set of initial definitions of recursive datatypes and functions, we aim to automatically produce a useful set of theorems, that will be useful as lemmas in further proofs, by either a human or an automated theorem prover. The set of synthesised theorems can also provide a 'sanity check', ensuring that the theory has been appropriately axiomatised by ensuring no unintended theorems are included.

Although a number of theory formation systems exists, [?,?,?], only the MATHsAiD system has previously been applied to inductive theories [?,?]. The main difference between IsaCoSy and MATHsAiD is how new theorems are produced. While IsaCoSy follows a *generative approach*, where conjectures are synthesised and then counter-example checked and proved, MATHSAiD follows a *deductive approach*, attempting to produce new theorems by reasoning forward from known facts.

2 Overview of the IsaCoSy system

The IsaCoSy system (**Isa**belle **Co**njecture **Sy**nthesis) is built on top of the proof-planner IsaPlanner [?,?] and Isabelle. IsaCoSy synthesises conjectures from available constants and variables in a 'bottom-up' fashion. It incrementally builds larger terms using the set of available constants and function symbols in a given theory. The key idea for making this tractable is to turn rewriting upside down: only irreducible terms (those not matching any rewrite rule) are synthesised. In terms of the implementation, these restrictions turn into constraints on the term-synthesis process, thus avoiding a naive and inefficient generate-and-test style procedure. Counter-example checking is still used to prune out obviously false conjectures, but as this can be rather slow, we want to use it as little as possible. The remaining conjectures are given to IsaPlanner to prove automatically by induction using the *rippling* heuristic [?]. Any theorems found can then be used to generate further constraints as synthesis is attempted on larger terms.

The aim of the IsaCoSy system is to automatically generate inductive theorems and lemmas that are interesting or will be useful in further proofs. IsaCoSy does not attempt to invent new concepts or definitions, it will only synthesise theorems about given datatypes and function definitions.

The implementation of IsaCoSy consists of three main parts:

- A language for expressing constraints on synthesis.
- A constraint generator, which produces constraints from available theorems.
- The synthesis engine itself, including procedures for updating and propagating constraints.

In addition, IsaCoSy also has a set of additional heuristics which can be configured by the user. These include:

- The number of different variables allowed in a term. From studying theorems in Isabelle's library, the default value for this is 1 + maximum arity of any function involved.
- Where variables are allowed to occur. When synthesising equations, a common heuristic from rewriting is to only allow rules where the variables in the right-hand side are a subset of those on the left.
- Eager checks for associativity and commutativity. Knowing whether functions that are associative and/or commutative provides IsaCoSy with useful constraints on the synthesis search space. By checking for these properties prior to synthesis, the initial search space is much smaller.

3 Motivating Examples

Unless we employ heuristics and constraints, the search space for conjecture synthesis is very large. We want to avoid generating conjectures that are considered to be more complicated versions of known theorems. IsaCoSy's approach to achieve this is to only produce irreducible terms, that cannot be rewritten by any existing rule. To illustrate a few useful types of constraints to restrict term synthesis, we shall in this section consider a few examples about natural numbers.

Example 1: Definition of Addition. Addition is defined by the two equations $0 + y = y$ and $Suc(x) + y = Suc(x + y)$. The definitions can be used as rewrite rules. The first applies to any term that has 0 in the first argument position of +, while the second applies to any term that has a *Suc* in the first position (regardless of what the *Suc* is applied to). Any such terms are excluded by IsaCoSy.

Example 2: Injectivity of ***Suc.*** Isabelle automatically derives some theorems about user-defined datatypes. This include injectivity. The injectivity of *Suc* is expressed in Isabelle as the rewrite rule $(Suc\ n = Suc\ m) = n = m$. To avoid synthesising terms to which this rewrite is applicable, IsaCoSy generate a constraint that forbids the two arguments of = to both be instantiated to *Suc* at the same time.

Example 3: Reflexivity. Reflexivity can be expressed as the rewrite rule $(x = x) = True$. The constraint we derive from this theorem is that the two arguments of = never should be equal in a term we have synthesised.

Example 4: Commutativity. Suppose we know that addition is commutative: $a\ +\ b = b\ +\ a$. Commutativity is not typically allowed as a rewrite rule, as it is non-terminating. However, IsaCoSy can identify commutativity theorems, and will derive constraints on the order of arguments of the commutative function. Currently, we introduce a constraint specifying that the first argument has to be of larger or equal size to the second, which cuts out many symmetric theorems. As the commutativity of equality is available as a library theorem, IsaCoSy automatically introduces this type of constraint for equality from the start.

4 Results and Conclusions

Automation of inductive theorem proving can be improved by providing richer background theories. IsaCoSy has been evaluated on several inductive theories about natural numbers, lists and binary trees (see [**?**], chapter 8). We verified that IsaCoSy is more efficient than a naive version of synthesis, which explores the whole search space, and that it produces good quality theorems, of the kind that are found in Isabelle's libraries. In particular, we compared IsaCoSy to a naive version of synthesis on several different inductive theories, showing an exponential reduction in search space size. IsaCoSy is thus not only faster, but also able to explore larger term-sizes before running out of memory.

To evaluate the quality of theorems found by IsaCoSy, we compared them with those in the Isabelle's libraries (when available). IsaCoSy produces many good theorems, resulting in high recall of 83% for natural numbers and 100% for lists. It does however produce a number of other, perhaps less interesting theorems too, so precision is lower, 63% for natural numbers and 38% for lists. Tables 1 and 2 show some examples of synthesised theorems, that also appear in Isabelle's libraries. A full list of synthesised theorems, also including run-times, can be found on-line[1]. With positive experimental results, theory formation by conjecture synthesis seems a promising area for further research.

[1] `dream.inf.ed.ac.uk/projects/lemmadiscovery/synth_results.php`

$a + 0 = a$	$a + Suc\ b = Suc(a + b)$
$a * 0 = 0$	$a * Suc\ b = a + (a * b)$
$a + b = b + a$	$a * b = b * a$
$(a + b) + c = a + (b + c)$	$(a * b) * c = a * (b * c)$
$(a * b) + (c * b) = (a + c) * b$	$(a * b) + (a * c) = (b + c) * a$

Table 1. Some synthesised theorems about addition and multiplication. These all occur in Isabelle's library.

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

Table 2. Some synthesised theorems about lists, also occurring in Isabelle's library. Note that @ denotes append.