TIP: Tons of Inductive Problems

Koen Claessen, Moa Johansson, Dan Rosén, and Nicholas Smallbone

Department of Computer Science and Engineering, Chalmers University of Technology
{koen,jomoa,danr,nicsma}@chalmers.se

Abstract. This paper describes our collection of benchmarks for inductive theorem provers. The recent spur of interest in automated inductive theorem proving has increased the demands for evaluation and comparison between systems. We expect the benchmark suite to continually grow as more problems are submitted by the community. New challenge problems will promote further development of provers which will greatly benefit both developers and users of inductive theorem provers.

1 Introduction

We have recently seen increased interest in inductive theorem proving, both with specialised provers such as IsaPlanner, Zeno and HipSpec [3,5,13], SMT-solvers such as CVC4 [11], the auto-active prover Dafny [10], recent work on the first-order SPASS prover [14], as well as some support in proof assistants [8,9].

To ease evaluation and development, and compare the relative strengths of the different systems, it is important to have good standard benchmarks. The contribution of this paper is an accessible standard benchmark suite for inductive theorem provers which can be extended by users and developers. The benchmarks are publicly available at:

`https://github.com/tip-org/benchmarks`

We have so far collected 340 problems in our benchmark suite, which we have called "TIP", for Tons of Inductive Problems—so named in the hope of attracting many more problems! We invite the community to submit additional problems and challenges and expect the collection to continuously grow and provide new challenges for developers.

2 The Benchmark Format

The benchmarks are expressed in a variant of SMT-LIB [1], extended with support for algebraic datatypes and higher-order functions. The format is described in detail in [4]. Starting from the basic SMT-LIB format, we import the following features from existing systems:

- Algebraic datatypes, which are declared using the `declare-datatypes` syntax as supported in Z3 and CVC4.

- Recursive function definitions, which use the `define-funs-rec` syntax implemented in CVC4 and proposed for SMT-LIB 2.5 [1].
- Polymorphic functions, using the proposed `par` syntax [2], which is implemented in a version of CVC4.

We then add more features of our own, which are specific to TIP:

- In the standard `declare-datatypes` syntax, functions over algebraic datatypes are defined using projection functions like `head` and `tail`. We add a pattern-matching syntax, which is more convenient for many provers.
- Many of our benchmark problems use higher-order functions, such as `map`. We add syntax for lambda functions and higher-order functions.
- Many inductive provers treat the goal specially, as opposed to SMT-LIB which expresses the goal as a negated axiom. We add a construct `(assert-not p)` which declares `p` as the goal; it is semantically equivalent to `(assert (not p))`.

We do not necessarily expect every prover to support TIP natively. Instead, we have made a tool which can translate TIP problems to and from a variety of other formats. Currently our tool can translate TIP problems to a CVC4-compatible version of SMT-LIB or to WhyML, and can compile Haskell programs into TIP properties. It can also perform a number of transformations for tools which do not support the full TIP language, such as removing higher-order functions by defunctionalisation [12]. Our aim is to support many different source and target formats in this tool.

Example

As an example of what the benchmarks look like, we show property 12 from the IsaPlanner benchmark set (see Section 3.1 below), which states that the functions `drop` and `map` distribute over one another:

`drop n (map f xs) = map f (drop n xs)`.

We declare two simple algebraic datatypes representing natural numbers and polymorphic lists.

```
(declare-datatypes (a)
  ((list (nil) (cons (head a) (tail (list a))))))
(declare-datatypes () ((Nat (Z) (S (p Nat)))))
```

Next, we declare two recursive functions: `map`, which is a higher-order function applying a unary function `f` to each element of a list, and `drop`, which recursively drops a given number of elements from the front of the list.

```
(define-funs-rec
  ((par (a b) (map ((f (=> a b)) (xs (list a))) (list b))))
  ((match xs
     (case nil (as nil (list b)))
     (case (cons y ys) (cons (@ f y) (map f ys))))))
```

```
(define-funs-rec
  ((par (a) (drop ((n Nat) (xs (list a))) (list a))))
  ((match n
     (case Z xs)
     (case (S m)
       (match xs
         (case nil xs)
         (case (cons y ys) (drop m ys)))))))
```

These definitions illustrate several features of TIP:

- Polymorphism: `par` introduces type variables.
- Higher-order functions: `(=> a b)` is the type of functions from `a` to `b`, and `@` applies first-class functions to their arguments.
- Pattern-matching using `match` and `case`, which binds new variables.

Finally, the benchmark problem itself is declared with the keyword `assert-not`:

```
(assert-not
  (par (a b)
    (forall ((n Nat) (f (=> a b)) (xs (list a)))
      (= (drop n (map f xs)) (map f (drop n xs))))))
(check-sat)
```

Each benchmark file is stand-alone and only contains one property.

3 Sample Benchmarks

In this section we give a short overview of some the benchmark problems currently available in the repository. At the moment, there are three different main sources of problems. We expect more to be added.

3.1 IsaPlanner's Rippling and Case-Analysis Benchmarks

This set of 85 problems comes from the evaluation of IsaPlanner's rippling-heuristic for guiding rewriting in inductive proofs in the context of functions with case- and if-statements [7]. It has been used in the evaluation of many of the recent inductive theorem provers and includes theorems about lists, natural numbers and binary trees. The problems are relatively easy, most of the theorems can be proved by structural induction using only the function definitions and only 15 require auxiliary lemmas to be discovered.

3.2 Productive Use of Failure Benchmarks

This is another benchmark suite which has been used to evaluate several recent provers. It consists of 50 theorems about lists and natural numbers and originates from evaluation of techniques for discovering auxiliary lemmas in the CLAM prover [6]. The original paper did not provide definitions for the functions used in the benchmarks, so the definitions provided here come from the evaluation of the HipSpec system [3]. These proofs are generally a bit harder, and may require additional lemmas to be found and proved (by another induction) or generalisation of the conjecture in order to strengthen the inductive hypothesis.

3.3 New TIP Benchmarks

This set contains 205 new benchmarks including, amongst others, properties of the Agda standard library[1] implementation of integers on top of natural numbers, problems about natural numbers in binary representation, various sorting functions with correctness properties expressed in alternative ways, problems about regular expressions, binary search trees, grammars and skew heaps. The problems about sorting, regular expressions, grammars and heaps have to our knowledge not all been fully automated yet and are offered as challenges!

4 Contribute to TIP

We invite the theorem proving community to contribute additional inductive benchmarks and challenge problems to TIP. Instructions for how to submit problems can be found in the README file for the repository (`https://github.com/tip-org/benchmarks`.)

We are developing a toolchain for translating to and from our format. The development is in its own repository (`https://github.com/tip-org/tools`). The tool can currently read in problems in TIP format or Haskell, and output TIP, SMT-LIB and WhyML, with some caveats:

– The generated SMT-LIB uses `declare-datatypes`, `define-funs-rec` and polymorphism.
– The generated WhyML makes no special effort to pass Why3's termination checker.

We encourage the community to request and contribute additional input and output formats to our tool chain.

5 Conclusion and Further Work

TIP is intended to be a standard benchmark suite for developers and users of inductive theorem provers. We hope that this initiative will ease comparison and evaluation of systems and spur further collaboration and development by attracting submissions of additional challenge problems from the community.

In addition to serving as a standard benchmark suite for inductive provers, the TIP benchmarks may also be useful for developers of theory exploration systems. Theory exploration is a technique for automatically discovering interesting conjectures about a given set of functions and datatypes, and is used in for example the HipSpec prover to discover lemmas. The TIP benchmarks can be compared to the output from the theory explorer in precision/recall analysis to assess the quality and interestingness of the conjectures generated. A good theory exploration system may also be used to generate new benchmarks for TIP.

[1] `https://github.com/agda/agda-stdlib`

Another aim is to also support non-theorems, for evaluation of tools for counter-example finding. This requires no extension to the format at all, but it requires a standardization on how to annotate problems with their expected answer (theorem or non-theorem), as well as a common solution format.

It is important to have good tool support if TIP is to be used by the community. Our tool is currently in active development, in order to support more input and output formats, as well as various strategies for encoding features of TIP for provers that do not support the full language.

In the future, we may want to extend the language to support a richer variety of problems. For example, we may want to include problems about lazy functions and co-datatypes.

References

1. Clark Barrett, Aaron Stump, and Cesare Tinelli. The SMT-LIB standard – version 2.0. In *Proceedings of the 8th International Workshop on Satisfiability Modulo Theories (SMT '10)*, July 2010. Edinburgh, Scotland.
2. François Bobot. [RFC] Add adhoc polymorphism. `https://github.com/CVC4/CVC4/pull/51`.
3. Koen Claessen, Moa Johansson, Dan Rosén, and Nicholas Smallbone. Automating inductive proofs using theory exploration. In *Proceedings of the Conference on Automated Deduction (CADE)*, volume 7898 of *LNCS*, pages 392–406. Springer, 2013.
4. Koen Claessen, Moa Johansson, Dan Rosén, and Nick Smallbone. The TIP language. `http://tip-org.github.io/format.html`.
5. Lucas Dixon and Moa Johansson. IsaPlanner 2: A proof planner in Isabelle. Technical report, University of Edinburgh, 2007.
6. Andrew Ireland and Alan Bundy. Productive use of failure in inductive proof. *Journal of Automated Reasoning*, 16:79–111, 1996.
7. Moa Johansson, Lucas Dixon, and Alan Bundy. Case-analysis for rippling and inductive proof. In *Proceedings of ITP*, pages 291–306, 2010.
8. Moa Johansson, Dan Rosén, Nicholas Smallbone, and Koen Claessen. Hipster: Integrating theory exploration in a proof assistant. In *Conference on Intelligent Computer Mathematics*, 2014.
9. Matt Kaufmann, Manolios Panagiotis, and J Strother Moore. *Computer-Aided Reasoning: An Approach.* Kluwer Academic Publishers, 2000.
10. K. Rustan Leino. Automating induction with an SMT solver. In *Proceedings of VMCAI*. Springer, 2012.
11. Andrew Reynolds and Viktor Kuncak. Induction for SMT solvers. In *Proceedings of VMCAI*, 2015.
12. John C. Reynolds. Definitional interpreters for higher-order programming languages. In *Proceedings of the ACM Annual Conference - Volume 2*, ACM '72, pages 717–740, New York, NY, USA, 1972. ACM.
13. Willam Sonnex, Sophia Drossopoulou, and Susan Eisenbach. Zeno: An automated prover for properties of recursive datatypes. In *Proceedings of TACAS*, pages 407–421. Springer, 2012.
14. Daniel Wand and Christoph Weidenbach. Automatic induction inside superposition. `https://people.mpi-inf.mpg.de/~dwand/datasup/draft.pdf`.