Automated Theory Exploration for Interactive Theorem Proving:
An introduction to the Hipster system

Moa Johansson

Department of Computer Science and Engineering, Chalmers University of Technology
`moa.johansson@chalmers.se`

Abstract. Theory exploration is a technique for automatically discovering new interesting lemmas in a mathematical theory development using testing. In this paper I will present the theory exploration system Hipster, which automatically discovers and proves lemmas about a given set of datatypes and functions in Isabelle/HOL. The development of Hipster was originally motivated by attempts to provide a higher level of automation for proofs by induction. Automating inductive proofs is tricky, not least because they often need auxiliary lemmas which themselves need to be proved by induction. We found that many such basic lemmas can be discovered automatically by theory exploration, and importantly, quickly enough for use in conjunction with an interactive theorem prover without boring the user.

1 Introduction

Theory exploration is a technique for discovering and proving new and interesting basic lemmas about given functions and datatypes. The concept of theory exploration was first introduced by Buchberger [3], to describe the workflow of a human mathematician: instead of proving theorems in isolation, like automated theorem provers do, mathematical software should support an exploratory workflow where basic lemmas relating new concepts to old ones are proved first, before proceeding to complex propositions. This is arguably the mode of usage supported in many modern proof assistants, including Buchberger's Theorema system [4] as well as Isabelle [13]. However, the discovery of new conjectures has mainly been the task for the human user. *Automated theory exploration systems*, [11,9,12,6], aims at addressing this by automatically both discover and prove basic lemmas. In the HipSpec system [6], automated theory exploration has been shown a successful technique for lemma discovery in inductive theorem proving solving several challenge problems where auxiliary lemmas were required. In this paper, we describe HipSpec's sister system Hipster, which is integrated with Isabelle/HOL and in addition produce certified proofs of lemmas and offer the user more flexibility and control over proof strategies.

Hipster consists of two main components: the *exploration component*, called QuickSpec [16], is implemented in Haskell and efficiently generates candidate conjectures using random testing and heuristics. The conjectures are then passed on to the *prover component* which is implemented in Isabelle. Hipster discards any conjectures with trivial proofs, and outputs snippets of proof scripts for each interesting lemma it discovers. The user can then easily paste the discovered lemmas and their proofs into the Isabelle theory file by a mouse-click, thus assisting and speeding up the development of new theories.

Example 1. As a first simple example consider the following small theory about binary trees with two functions, `mirror`, which recursively swaps the left and right subtrees and `tmap`, which applies a function to each element in the tree[1].

```
datatype 'a Tree =
  Leaf 'a
  | Node "'a Tree" 'a "'a Tree"

fun mirror :: "'a Tree => 'a Tree"
where
  "mirror (Leaf x) = Leaf x"
| "mirror (Node l x r) = Node (mirror r) x (mirror l)"

fun tmap :: "('a => 'b) => 'a Tree => 'b Tree"
where
  "tmap f (Leaf x) = Leaf (f x)"
| "tmap f (Node l x r) = Node (tmap f l) (f x) (tmap f r)"
```

We can ask Hipster to discover some properties about these two functions by issuing a command in the Isabelle theory file telling Hipster which functions it should explore:

```
hipster tmap mirror
```

Almost immediately, Hipster outputs the following two lemmas (and nothing else), which it has proved by structural induction followed by simplification, using the function definitions above:

```
lemma lemma_a [thy_expl]: "mirror (mirror y) = y"
  apply (induct y)
  apply simp
  apply simp
  done

lemma lemma_aa [thy_expl]: "mirror (tmap y z) = tmap y (mirror z)"
  apply (induct z)
  apply simp
  apply simp
  done
```

Here, Hipster was configured in such a way to consider lemmas requiring inductive proofs interesting, and other conjectures requiring only simplification trivial.

[1] This example can be found online: `https://github.com/moajohansson/IsaHipster/blob/master/Examples/ITP2017/Tree.thy`.

We believe our work on automated theory exploration can complement systems like Sledgehammer [14]. Sledgehammer is a popular tool allowing Isabelle users to call various external automated first order provers and SMT solvers. A key feature of Sledgehammer is its relevance filter which selects facts likely to be useful in proving a given conjecture from Isabelle's huge library, which otherwise would swap the external prover. However, if a crucial lemma is missing, Sledgehammer will fail, as might well be the case in a new theory development.

Current state of the project

The first version of Hipster has been described in [10]. The Hipster project is ongoing and the system is under active development. The version described in this paper is a snapshot of forthcoming second version. It includes several improvements:

- Hipster now uses the recent QuickSpec 2 [16] as backend for conjecture generation, which is much more efficient than the previously used first version. QuickSpec 2 also has a generic interface via the TIP-language and tools [7,15] avoiding ad-hoc translation from Haskell to Isabelle. Figure 1 shows the new architecture of HipSpec.
- Hipster can use any Isabelle tactic as specified by the user, now also including Sledgehammer, which allows it to exploit knowledge from Isabelle's existing libraries more efficiently. The aim is to make it easy for the user to customise Hipster's proof strategies according to his/her needs.
- The proof output from Hipster has been improved, referring only to standard Isabelle tactics. Unlike the first version, which produced single-line proofs using a Hipster-specific tactic, the proofs now displays the variable on which Hipster did induction, as well as the tactics and lemmas it used to prove the base- and step cases. This also saves Isabelle re-doing a lot of search when the proof is replayed.

Hipster is open source with code available from GitHub: `https://github.com/moajohansson/IsaHipster`. We happily invite those interested in Hipster to try it out and welcome contributions to further development.

2 Architecture of a Theory Exploration System

A theory exploration system has two main tasks: First of all, it needs to generate candidate conjectures (of which at least the majority should be theorems) and secondly, it needs access to a sufficiently powerful automated theorem prover to (at least) prove most of the interesting conjectures, and dismiss uninteresting ones. Hipster's conjecture generation is outsourced to QuickSpec 2 [16] and proofs are performed by the tactics of Isabelle/HOL. In this section we describe both these parts.

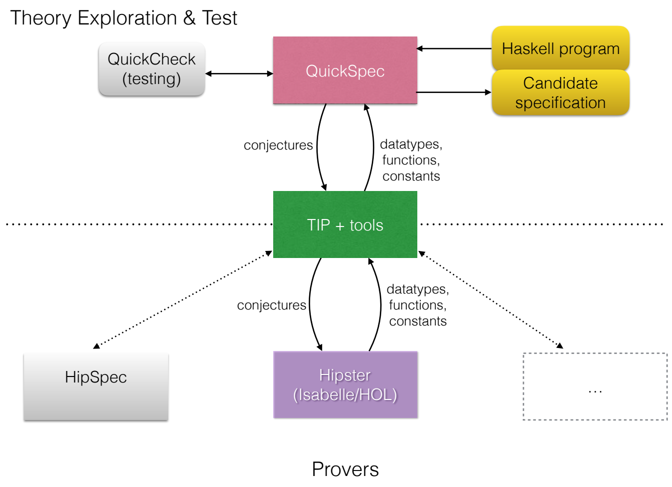

Fig. 1. Theory exploration architecture. Hipster and its sister-system HipSpec.

2.1 Conjecture Generation

A trivial approach to conjecture generation would be to exhaustively generate all possible terms that could be constructed from the input functions and datatypes, but this would quickly become intractable, so some heuristics are necessary. Furthermore, we do not want to waste time trying to prove conjectures that are obviously false, so the conjecture generation should filter those out using testing, or if possible, avoid generating them in the first place.

Earlier theory exploration systems for Isabelle/HOL, IsaCoSy [9], and IsaScheme [12], took different approaches. IsaCoSy was restricted to generate only irreducible terms, starting from small term size, and interleaved inductive proofs with exploration before increasing the term size, so discovered equations could be used to further restrict the search space. IsaScheme generated conjectures by instantiating user provided term schemas (templates) and combined this rewriting and completion. To avoid false conjecture, both IsaCoSy and IsaScheme filtered the resulting conjectures through Isabelle's counter-example checker.

Hipster is considerably faster than both IsaCoSy and IsaScheme, much thanks to QuickSpec's clever conjecture generation. The key idea is that term generation is interleaved with testing and evaluation of terms, using Haskell's QuickCheck tool [5], which enables many terms to be tested at once, instead of one at the time (see Example 2 below). Put simply, the conjecture generation algorithm proceeds by iterating the following steps:

1. Generate new terms of the current term size and add them the the current universe of terms. The algorithm start from term size 1 and iterates up to user-specified max size.

2. Test and evaluate the terms generated so far using QuickCheck. Divide them into equivalence classes.
3. Extract equations from the equivalence classes. Using these equations, prune the search space for the next iteration of term genration when term size is increased.

Example 2 (Conjecture generation in QuickSpec). As a small example, suppose the universe of terms generated so far include the terms in the first column of Table 1 below. QuickSpec will generate many (by default 1000) random test cases and evaluate all terms on these. Initially, all terms are in one equivalence class, but as testing proceeds, terms are split according to which ones evaluate to the same value. Table 1 shows how our small set of terms are split into three equivalence classes using two random tests. Testing would then proceed on many more random values, but no more splits would occur. When the equivalence

Test-case:	$xs \to [b, a]$, $ys \to [\,]$	
Term	**Instance**	**Evaluation**
xs	[b,a]	[b,a]
rev(rev xs)	rev(rev [b,a])	[b,a]
sort xs	sort [b,a]	[a,b]
sort (rev xs)	sort (rev [b,a])	[a,b]
sort (xs @ ys)	sort([b,a] @ [ ])	[a,b]

Test-case:	$xs \to [b, a, c]$, $ys \to [c]$	
Term	**Instance**	**Evaluation**
xs	[b,a,c]	[b,a,c]
rev(rev xs)	rev(rev [b,a,c])	[b,a,c]
sort xs	sort [b,a,c]	[a,b,c]
sort (rev xs)	sort (rev [b,a,c])	[a,b,c]
sort (xs @ ys)	sort([b,a,c] @ [c])	[a,b,c,c]

Table 1. How QuickSpec divides terms into equivalence classes based on their evaluation two random test cases. The first test case (top) splits the terms into two equivalence classes. The second test case (bottom) splits off a third equivalence class.

classes are stable, QuickSpec extracts two equations:

$$\textit{rev(rev xs)} = \textit{xs} \text{ and } \textit{sort(rev xs)} = \textit{sort xs}.$$

Note that these conjectures have been tested many times, so they are likely to be true, but they have not yet been proved. QuickSpec's pruner will now use these two equations to restrict its search space. It will prune all terms of the shapes *rev(rev _)* and *sort(rev _)* on account of such terms being reducible by the two equations QuickSpec found. This stops generation of arguably less interesting equations, for example *rev(rev(xs @ ys)) = xs @ ys*, *rev(rev(xs @ ys @ zs)) = xs @ ys @ zs* and so on.

The new version of Hipster described here use QuickSpec 2 where the conjecture generation algorithm has been further refined compared to the simplified version described above, incorporating ideas from both IsaCoSy (avoiding generation of reducible terms) and IsaScheme (generation of schematic terms first). We refer to [16] for details of all heuristics in QuickSpec 2.

QuickSpec was originally designed to generate candidate specifications for Haskell programmes and can also be used as a stand alone light-weight verification tool for this purpose, producing a candidate specification consisting of equations that has been thoroughly tested, but not proved. With QuickSpec 2, an interface using the TIP-language [7], was added to facilitate communication with external systems such as Hipster and its sister system HipSpec [6]. Hipster translates its given input functions and datatypes into TIP before sending them to QuickSpec. Similarly, QuickSpec outputs the resulting conjectures in TIP format, and Hipster translates them back into Isabelle/HOL (see Figure 1). The TIP-language is based on SMT-LIB [2], with extensions to accommodate recursive datatypes and functions. It was originally designed for creation of a shared benchmark repository for inductive theorem provers. TIP comes with a number of tools for translating between it and various other formats, such as standard SMT-LIB, TPTP [17] and Isabelle/HOL, as well as libraries for facilitating writing additional pretty printers and parsers for other prover languages [15]. QuickSpec should therefore be relatively easy to integrate with additional provers.

2.2 Proving Discovered Conjectures

When Hipster gets the set of candidate conjectures from QuickSpec, it enters its proof loop, where it tries to prove each conjecture in turn. The proof loop is shown in Figure 2. Hipster is parametrised by two tactics, one for *easy reasoning* and one for *hard reasoning*, with the idea being that conjectures proved by the easy reasoning tactic are trivial, and not interesting enough to be presented to the user. The hard reasoning tactic is more powerful, and the conjectures requiring this tactic are considered interesting and are output to the user. Should the proof fail the first time around, Hipster retries the conjecture at the next iteration if any additional lemmas have been proved in between. Otherwise, the unproved conjectures are also presented to the user. As QuickSpec has tested each conjecture thoroughly it is likely to either be interesting as it is a theorem with a difficult proof, or, have a very subtle counter-example. So far, the combinations of hard and easy reasoning we have experimented with has been various combinations of simplification and/or first order reasoning for the easy reasoning tactic, and some form of induction for hard reasoning. We plan to do a more thorough experimentation with different tactic combinations to extract suitable heuristics.

As mentioned in the previous section, QuickSpec has its own heuristics for reducing the search space and removing seemingly trivial conjectures. However, QuickSpec does not know anything about Isabelle's libraries, nor does it assume that it necessarily has access to the function definitions (when used as a standalone tool, it is designed to be able to explore properties also about Haskell

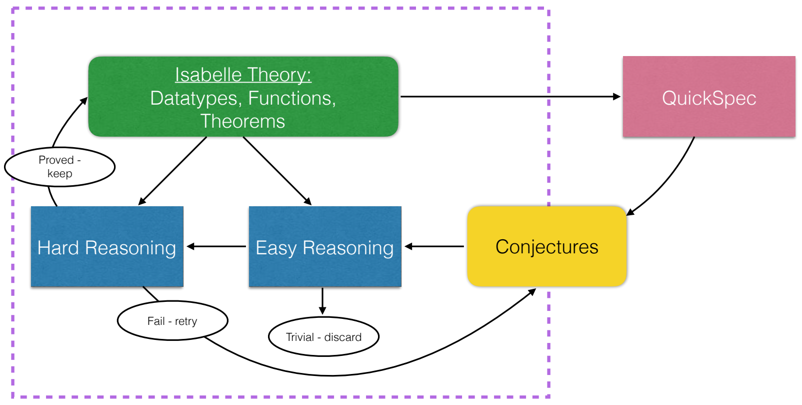

Fig. 2. Hipster's proof loop. Conjectures from QuickSpec are discarded if they can be proved by the *easy reasoning tactic* and presented to the user if they are proved by the *hard reasoning tactic*. Conjectures not proved at the first attempt might be retried once additional lemmas have been proved.

programs which it can test, but not have the source code for). Hence, there will usually always be at least a few trivial conjectures from the Isabelle point of view. Priming QuickSpec with existing facts from Isabelle is future work.

Example 3 (Using Hipster for the maths homework). As an undergraduate at Chalmers, you might be faced with something like the following exercise in your discrete maths class: *Prove that the rev function and the tail-recursive qrev function produce the same result: rev xs = qrev xs [ ].* An Isabelle-savvy student formalises this as follows and tries to prove the exercise using induction and Isabelle's Sledgehammer tool[2]:

```
fun qrev :: "'a list => 'a list => 'a list"
where
  "qrev [] acc  = acc"
| "qrev (x#xs) acc = qrev xs (x#acc)"

theorem hardExercise: "rev xs = qrev xs []"
apply (induct xs)
apply auto
sledgehammer
```

Unfortunately, not even Sledgehammer succeeds in proving the step-case, here using the external provers CVC4 [1] and Z3 [8]:

```
Sledgehammering...
"cvc4": Timed out
"z3": Timed out
```

[2] The source code is at: `https://github.com/moajohansson/IsaHipster/blob/master/Examples/ITP2017/Qrev.thy`.

Something must be missing, so let us use Hipster to see what it can discover about these functions:

```
hipster rev qrev

lemma lemma_a [thy_expl]: "qrev (qrev z y) [] = qrev y z"
apply (induct z arbitrary: y)
apply simp
apply simp
done

lemma lemma_aa [thy_expl]: "rev y @ z = qrev y z"
apply (induct y arbitrary: z)
apply simp
apply simp
apply (metis append_eq_append_conv2 rev.simps(2) rev_append
             rev_singleton_conv rev_swap)
done
```

Hipster returns two interesting lemmas which it needed its *hard reasoning* tactic to prove. In this example, hard reasoning was set to structural induction followed by simplification and/or Sledgehammer[3], while the *easy reasoning* tactic was set to just simplification and/or Sledgehammer. Note that the second discovered lemma, `lemma_aa`, is a generalisation of our theorem. This is exactly what we need, as is confirmed by Sledgehammer:

```
theorem hardExercise: "rev xs = qrev xs []"
apply (induct xs)
apply auto
sledgehammer
by (metis lemma_aa) (*** This line is now found by Sledgehammer ***)
```

As a matter of fact, we could even prove the exercise without induction now, as it is a special case of `lemma_aa`.

Example 4 (Configuring Hipster's proof methods). If we were to study the intermediate output from Hipster while it is running on Example 3, we would notice that there are in fact 17 lemmas discovered by QuickSpec, most of which got discarded by Hipster. These include re-discovery of the function definitions (remember, QuickSpec does not assume it has direct access to the source code, only that it can test functions), a couple of lemmas about `rev` already present in Isabelle's library, and also theorem `hardExercise` from Example 3. Why did it get discarded?

The anser is simple: The conjectures returned from QuickSpec happens to come in an order so that Hipster tries to prove `hardExercise` before it has tried the essential `lemma_aa`. The first proof attempt therefore fails, and it is returned to the queue of open conjectures (see Figure 2). In the next iteration of the

[3] The proof command `metis` (followed by a list of required library facts) in the proof of `lemma_aa` is produced by Sledgehammer. Metis is Isabelle's built in first order prover used to reconstruct proofs from external provers.

proof-loop, Hipster has already proved `lemma_aa` and can prove `hardExercise` using just its easy reasoning tactic (here Sledgehammer). Suppose we consider Hipster a bit overzealous in its pruning, and want to see also proofs found by Sledgehammer. We can easily reconfigure it to use a different combination of tactics, for example an easy reasoning tactic which only use simplification with existing Isabelle facts, and a hard reasoning tactic which use Sledgehammer or induction[4]:

```
setup Tactic_Data.set_sledge_induct_sledge

hipster rev qrev
...

lemma lemma_ab [thy_expl]: "qrev (qrev (qrev x2 z) y) x3 =
                                        qrev y (qrev (qrev z x2) x3)"
apply (metis Qrev.lemma_aa append.assoc append.right_neutral lemma_a)
done

lemma lemma_ac [thy_expl]: "qrev y [] = rev y"
apply (metis Qrev.lemma_aa append.right_neutral)
done
```

Now, Hipster keeps two additional lemmas, which both follows from the previously discovered lemmas by first-order reasoning. `lemma_ac` is theorem `hardExercise` with the left- and right-hand sides flipped, while `lemma_ab` is a slightly exotic formulation of associativity for `qrev` and arguably not something a human would come up with.

3 Ongoing and Future Work

We plan to do a more comprehensive evaluation of various tactics in Hipster. As we saw in Example 4, the results of theory exploration are different depending on how we configure the hard- and easy reasoning tactics. Furthermore, there is a trade-off in run-time depending on how powerful we make the respective tactics. Experimental evaluation is needed to decide on some suitable heuristics and default combinations. In the examples shown here, we only used structural induction, but we would also like to compare it in detail to, for instance, recursion induction based on function definitions as default [18]. An extension to co-recursion and co-datatypes is also being developed as part of the MSc project of Sólrún Halla Einarsdottir at Chalmers.

The version of Hipster described here is under active development, and not all features has yet been ported to the new version which uses QuickSpec 2. The first version of Hipster had some very basic support for discovery of conditional equations [18], where the user specified a predicate for the condition, which

[4] The interested reader may consult the file `Tactic_Data.ML` in the Hipster source code repository for details of several pre-defined combinations of easy/hard reasoning tactics, as well as how to define additional ones.

was passed to QuickSpec 1. Testing conditional equations is tricky, one need to generate test-cases where the condition holds which is a non-trivial task. In QuickSpec 1, the test-cases not satisfying the condition were just discarded, meaning that many extra test-cases had to be evaluated and testing become much slower and false conjectures are more likely to slip through. This has been improved in QuickSpec 2 [16], but at the time of writing not fully integrated in the new version of Hipster.

4 Summary

Hipster is a theory exploration system for Isabelle/HOL. It automatically conjectures and proves basic lemmas about given functions and datatypes, which can be particularly useful as part of an automated inductive theorem prover. Hipster is parametrised by two proof strategies which can be set by the user, one for *easy reasoning* and one for *hard reasoning.* Conjectures solved by easy reasoning (e.g. simplification) are considered trivial and uninteresting, while those requiring hard reasoning (e.g. induction) are considered worth presenting to the user.

Hipster use an external conjecture generation engine called QuickSpec. The systems are connected via an interface language called TIP, which is an extension of SMT-LIB, and related tools for parsing and pretty printing. We believe this interface has potential to be very useful for connecting additional provers wishing to benefit from theory exploration.

Hipster, QuickSpec and TIP are all under active development by our group at Chalmers. We invite anyone interested to test the tools and contribute to their development.

References

1. Clark Barrett, Christopher L. Conway, Morgan Deters, Liana Hadarean, Dejan Jovanović, Tim King, Andrew Reynolds, and Cesare Tinelli. CVC4. In Ganesh Gopalakrishnan and Shaz Qadeer, editors, *Computer Aided Verification: 23rd International Conference, CAV 2011*, pages 171–177, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.
2. Clark Barrett, Pascal Fontaine, and Cesare Tinelli. The SMT-LIB standard. http://smtlib.cs.uiowa.edu/standard.shtm.
3. Bruno Buchberger. Theory exploration with Theorema. *Analele Universitatii Din Timisoara, ser. Matematica-Informatica*, 38(2):9–32, 2000.
4. Bruno Buchberger, Adrian Creciun, Tudor Jebelean, Laura Kovacs, Temur Kutsia, Koji Nakagawa, Florina Piroi, Nikolaj Popov, Judit Robu, Markus Rosenkranz, and Wolfgang Windsteiger. Theorema: Towards computer-aided mathematical theory exploration. *Journal of Applied Logic*, 4(4):470 – 504, 2006. Towards Computer Aided Mathematics.
5. Koen Claessen and John Hughes. QuickCheck: a lightweight tool for random testing of Haskell programs. In *Proceedings of ICFP*, pages 268–279, 2000.
6. Koen Claessen, Moa Johansson, Dan Rosén, and Nicholas Smallbone. Automating inductive proofs using theory exploration. In *Proceedings of the Conference on*

Auomtated Deduction (CADE), volume 7898 of *LNCS*, pages 392–406. Springer, 2013.

7. Koen Claessen, Moa Johansson, Dan Rosén, and Nicholas Smallbone. TIP: Tons of inductive problems. In *Conference on Intelligent Computer Mathematics*, volume 9150 of *LNCS*, pages 333–337. Springer, 2015.
8. Leonardo De Moura and Nikolaj Bjørner. Z3: an efficient SMT solver. In *Proceedings of TACAS*, TACAS'08/ETAPS'08, pages 337–340. Springer-Verlag, 2008.
9. Moa Johansson, Lucas Dixon, and Alan Bundy. Conjecture synthesis for inductive theories. *Journal of Automated Reasoning*, 47(3):251–289, 2011.
10. Moa Johansson, Dan Rosén, Nicholas Smallbone, and Koen Claessen. Hipster: Integrating theory exploration in a proof assistant. In *Conference on Intelligent Computer Mathematics*, volume 8543 of *LNCS*, pages 108–122. Springer, 2014.
11. Roy L. McCasland, Alan Bundy, and Patrick F. Smith. Ascertaining mathematical theorems. *Electronic Notes in Theoretical Computer Science*, 151(1):21 – 38, 2006.
12. Omar Montano-Rivas, Roy McCasland, Lucas Dixon, and Alan Bundy. Scheme-based theorem discovery and concept invention. *Expert Systems with Applications*, 39(2):1637–1646, 2012.
13. Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.
14. Lawrence C Paulson and Jasmin Christian Blanchette. Three years of experience with Sledgehammer, a practical link between automatic and interactive theorem provers. *IWIL-2010*, 2010.
15. Dan Rosén and Nicholas Smallbone. TIP: Tools for inductive provers. In *Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR)*, volume 9450 of *LNCS*, pages 219–232. Springer, 2015.
16. Nicholas Smallbone, Moa Johansson, Claessen Koen, and Maximilian Algehed. Quick specifications for the busy programmer. Journal of Functional Programming, to appear., 2017.
17. G. Sutcliffe. The TPTP Problem Library and Associated Infrastructure: The FOF and CNF Parts, v3.5.0. *Journal of Automated Reasoning*, 43(4):337–362, 2009.
18. Irene Lobo Valbuena and Moa Johansson. Conditional lemma discovery and recursion induction in Hipster. *ECEASST*, 72, 2015.