The Theory Behind TheoryMine

The Theory behind TheoryMine

Alan Bundy
School of Informatics, University of Edinburgh,
Edinburgh EH8 9AB, Edinburgh, Scotland.
a.bundy@ed.ac.uk

Flaminia Cavallo
TheoryMine, Appleton Tower
Edinburgh, EH8 9LE
fl.cavallo@googlemail.com

Lucas Dixon
School of Informatics, University of Edinburgh,
Edinburgh EH8 9AB, Edinburgh, Scotland.
lucas.dixon@ed.ac.uk

Moa Johansson
Dipartimento di Informatica
Universita degli Studi di Verona
371 34 Verona, Italy
moakristin.johansson@univr.it

Roy McCasland
School of Informatics, University of Edinburgh,
Edinburgh EH8 9AB, Edinburgh, Scotland.
rmccasla@staffmail.ed.ac.uk *

Abstract

We describe the technology behind the TheoryMine novelty gift company. A tower of four computer systems is used to generate recursive theories, then to speculate conjectures in those theories and then to prove these conjectures. All stages of the process are entirely automatic. The process guarantees large numbers of sound, novel theorems of some intrinsic merit.

1 Introduction

TheoryMine[1] is a spin-out company in the novelty gift market. It generates and proves novel inductive theorems for customers and gives them the opportunity to name these theorems, e.g., after themselves, a friend, a relative or a pet. Customers are provided with a certificate containing a statement of the theorem, a summary of its proof and the definitions of the functions and types occurring in it. An example certificate is given in Figure 1.

The purchase of theorems is not new to mathematics. In 1694, the Marquis de l'Hospital paid Johann Bernoulli 300 Francs a year to use his theorems in any way he wished [Truesdell, 1958][59-62]. l'Hospital described these theorems in his book *l'Analyse des Infiniment Petits pour l'Intelligence des Lignes Courbes*. As a result of this, one of Bernoulli's theorems, l'Hospital's Rule, was ascribed to l'Hospital.

The theory and technology underpinning TheoryMine has been developed over several decades, mostly by members of the Mathematical Reasoning Group at the University of Edinburgh. In this extended abstract we outline this theory and technology.

The TheoryMine technology consists of a tower of automated reasoning systems, which we list below.

IsaWannaThm [Cavallo, 2009][2] generates novel recursive types and functions to form new recursive theories, then uses IsaCoSy to generate new theorems in those theories.

*This work was supported by EPSRC grants EP/E005713/1 and EP/F033559/1, and an EPSRC studentship to Dr Johansson. We would like to thank two anonymous AUTOMATHEO reviewers for their constructive feedback on an earlier draft.

[1] http://theorymine.co.uk/

[2] This is Flaminia Cavello's dissertation, which is currently embargoed for commercial reasons. We hope it will be possible to make it available in the medium term. Meanwhile, the current paper goes some way to fill the gap left by its non-publication.

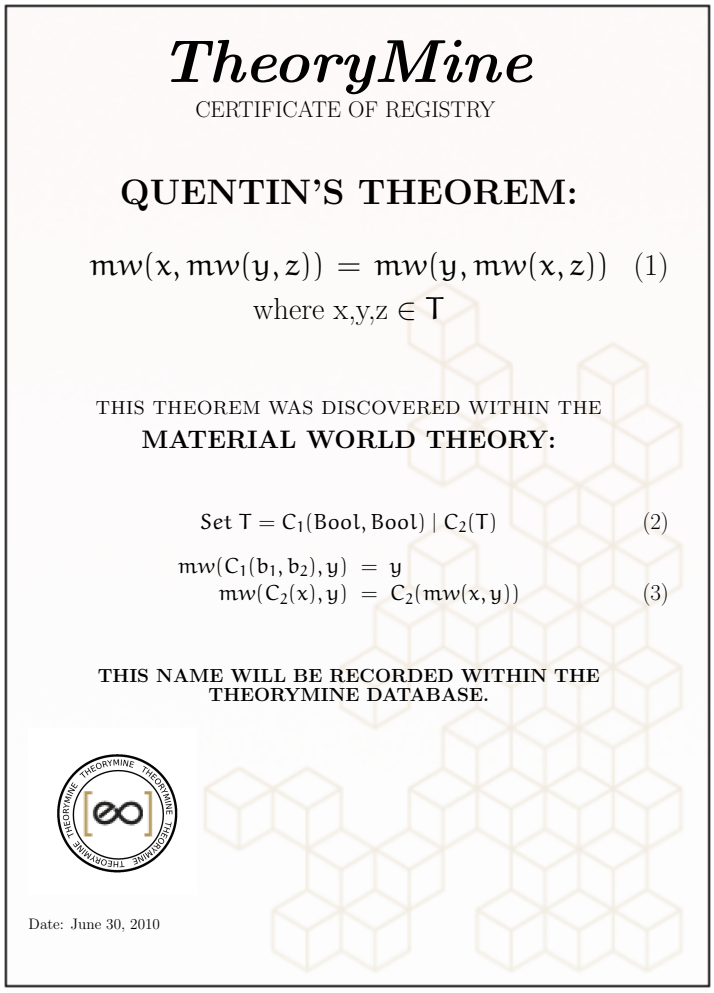

Figure 1: **Example Customer's Certificate** showing: the theorem statement (1); the theorem's name; the recursive type definition (2), which is here described as a set; and the recursive function definition (3). The type is a four-flavoured version of the natural numbers, in which $C_1(Bool, Bool)$ provides four different 'zero's and $C_2(T)$ provides a successor function successively to generate the next 'number' in the sequence. *mw* is a kind of addition on these 'numbers'. Note that *mw* is associative, but not commutative. Quentin's Theorem describes a very restricted variant of commutativity. This particular theorem was created in honour of Quentin Cooper, who interviewed us for the BBC Radio 4 science magazine programme Material World on 15th April 2010. The associativity of *mw* is called "The Herdman Theorem", in honour of Karen Herdman, who won this theorem as a prize in a Scottish Enterprise competition as part of its SECC All Staff Event on 2 June 2010.

IsaCoSy [Johansson *et al*, 2010a], given a recursive theory, generates interesting inductive conjectures in that theory, using IsaPlanner to prove them.

IsaPlanner [Dixon & Fleuriot, 2004], given an inductive conjecture, tries to prove it using an inductive proof plan to guide Isabelle in the search for a proof.

Isabelle [Paulson, 1994] is an open-source, generic, interactive proof assistant system built in Cambridge and Munich.

In this next few sections we briefly describe each of these systems.

2 IsaWannaThm

IsaWannaThm was developed by Flaminia Cavallo during her final-year undergraduate project at the University of Edinburgh. It creates novel recursive theories by incremental, exhaustive generation from a series of grammars. Firstly, it generates novel recursive types, then it defines recursive functions over these types, then it defines a recursive theory as a set of these definitions.

2.1 Generating Recursive Types

Consider the following two BNF grammars defining two recursive types: a unary representation of the natural numbers, $\mathbb{N}$, and then lists of $\mathbb{N}$s.

$$\begin{aligned} \mathbb{N} &::= 0 \mid s(\mathbb{N}) \\ natlist &::= nil \mid cons(\mathbb{N}, natlist) \end{aligned}$$

Note that such recursive types are uniquely defined by a collection of constructor functions: 0 and s in the case of $\mathbb{N}$ and *nil* and *cons* in the case of $natlist$. To generate a recursive type, we need to fix the following parameters:

- *The number of constructor functions*, e.g., $\mathbb{N}$ has two, 0 and s, and $natlist$ also has two *nil* and *cons*.
- *For each constructor function, its arity and the types of its arguments.* In particular, whether these arguments are recursive, such as the single argument of s and the second argument of *cons*, or whether they refer to previously defined types, such as the first argument of *cons*. At least one of these constructors must have only non-recursive arguments, or there will be no finite members of the type. These are called *base* constructors and those with recursive arguments are called *step* constructors. Note that 0 and *nil* are nullary, i.e., have no arguments, so are trivially base types.

By systematically exploring the space defined by these parameters, we can generate infinitely many recursive types. This can be viewed as exhaustive generation from a meta-grammar of recursive type grammars. Upper limits are set on the parameters to prevent the generated types becoming too complex for successful theorem proving. The example given in Figure 1 is:

$$T ::= C_1(Bool, Bool) \mid C_2(T) \qquad (2)$$

Following standard mathematical terminology, types are presented to customers as *sets* and ::= as =. To name its new types and constructor functions, IsaWannaThm generates short words, consisting of letters from a mixture of different alphabets. Type names start with an initial T and constructors with an initial upper-case letter that is not T.

To ensure that TheoryMine's theorems are novel, IsaWannaThm avoids generating types isomorphic to well-known recursive types. It does this by filtering out any types that already appear in Isabelle libraries. Unfortunately, we cannot entirely rule out duplication of more obscure ones. However, it does start with two well-known base types: $\mathbb{N}$ and *bool*, so that recursive functions can use these types, as long as at least one of their inputs has novel type.

IsaWannaThm is currently restricted to free types, i.e., ones in which syntactically different constructor terms are unequal. An example of a non-free type would be the integers, defined as

$$\mathbb{Z} \quad ::= \quad 0 \mid s(\mathbb{Z}) \mid p(\mathbb{Z})$$

where p is the predecessor function, since $s(p(x)) = p(s(x)$. It also avoids mutually recursive types, such as:

$$\begin{array}{lcl} \tau_1 & ::= & \mathit{null} \mid c_1(\tau_1, \tau_2) \\ \tau_2 & ::= & \mathit{null} \mid c_2(\tau_1, \tau_2) \end{array}$$

Lifting these restrictions is a topic for future work.

2.2 Generating Recursive Functions

Now assume that IsaWannaThm has defined a recursive type, using the methods of §2.1, and that, without loss of generality, it has the form:

$$\tau \quad ::= \quad \ldots \mid c(\vec{\tau'}, \tau, \ldots, \tau) \mid \ldots$$

where c is a typical constructor, $\vec{\tau'}$ is a vector of (possibly distinct) non-recursive arguments and the last n arguments of c are all of type τ.

IsaWannaThm will now define novel recursive functions on this type using a simple primitive recursive function schema. Each definition will have one case for each constructor function, taking the form:

$$\begin{array}{lcl} & \vdots & \vdots \quad \vdots \\ f(\vec{x}, c(\vec{y}, z_1, \ldots, z_n)) & ::= & t(\vec{x}, \vec{y}, f(\vec{x}, z_1), \ldots, f(\vec{x}, z_n)) \\ & \vdots & \vdots \quad \vdots \end{array}$$

where f is the recursive function being defined, with a vector $\vec{x}$ of non-recursive arguments and one final recursive argument and t is a (possibly compound) term constructed from previously defined functions. Note that these previously defined functions can include standard functions, such as $+$ and $\wedge$, defined on the base types $\mathbb{N}$ and *bool*. When $n = 0$ the case will be a *base case*; otherwise, it will be a *step case*. To name its new functions, IsaWannaThm generates short words, consisting of letters from a mixture of different alphabets, starting with an initial lower-case letter. The variables used in the definitions are restricted to x, y, z, u, v, w.

By systematically exploring the space of possible values of $\vec{x}$, n and t, IsaWannaThm generates a potentially infinite set of recursive functions for each recursive type. The example functions given in Figure 1 are given in Figure 2. Again, upper limits are set on the possible values to prevent the generated functions becoming too complex for successful theorem proving.

Note that IsaWannaThm is restricted to very simple structural recursive function definitions. We currently ignore the opportunities to allow: non-recursive arguments of f to take different values on left- and right-hand sides; cases to have compound recursive patterns; two or more functions to be defined mutually; etc. Such extensions are topics for future work.

$$
\begin{array}{rcll}
mw : T \times T \mapsto T & & & \\
mw(C_1(x,y),z) & ::= & z & \\
mw(C_2(x),y) & ::= & C_2(mw(x,y)) & (3)
\end{array}
$$

Figure 2: **Example Recursive Function Definition** consisting of a type declaration followed by one base case and one step case.

2.3 Generating Recursive Theories

Recursive theories are created by IsaWannaThm by systematically generating recursive types, then recursive functions on these types, whose definitions become the axioms of the theories, and finally by generating conjectures and trying to prove them to be theorems. An example recursive theory, the *Material World Theory*, is given in Figure 1. In §3, we describe how theorems of these theories are automatically generated by the IsaCoSy system.

Note that the theories generated in this way are *purely definitional*, i.e., all their axioms are recursive definitions. Purely definitional theories are guaranteed to be consistent. This was a major consideration in the design of IsaWannaThm. Had it merely generated random formulae as axioms, there would be no guarantee that the resulting theories would be consistent, so that customers' theorems would run the risk of being trivially true, since all formulae are provable in an inconsistent theory.

To ensure that TheoryMine's theorems are always novel, we ensure that each theory's particular combination of types and functions is unique to it. We have also added the additional restriction to IsaCoSy that each conjecture generated for a theory must use *all* of the functions in a theory. The motivation for this is that a conjecture that does *not* use all the functions would already have been generated as a conjecture of a smaller theory. This restriction avoids duplication of conjectures and is why IsaWannaThm generates *all* subsets of its set of recursive functions, and not just the maximal ones. The alternative strategy of generating only theories maximal up to some complexity threshold would have run the risk that the resulting theories would prove too complex to be successfully processed by one of the constituent systems.

3 IsaCoSy

IsaCoSy was developed by Moa Johansson during her PhD at the University of Edinburgh. It creates inductive conjectures in a recursive theory by exhaustively generating terms in irreducible form, forming equations between them, then filtering out most non-theorems using the counter-example finders QuickCheck [Berghofer & Nipkow, 2004]. Upper limits are set on the complexity of the conjectures to prevent them from becoming too complex to be synthesised or proved. Conjectures that survive these filters are sent to IsaPlanner to be proved. Those that are successfully proved become potential products of TheoryMine. A description of IsaCoSy is included below to make this paper self-contained, but more details can be found in [Johansson, 2009, Johansson *et al*, 2010a, Johansson *et al*, 2010b], the last of which is a paper in this workshop.

The example given in Figure 1 is:

$$mw(x, mw(y,z)) \quad = \quad mw(y, mw(x,z)) \qquad (1)$$

Note that x and y are commuted, but only in the context of z. To see that mw is not commutative in

general, consider for instance:

$$mw(C_1(t,f),C_1(f,t)) = C_1(f,t) \neq C_1(t,f) = mw(C_1(f,t),C_1(t,f))$$

where $Bool = \{t,f\}$.

All terms generated by IsaCoSy are guaranteed to be irreducible both by the recursive definitions of the theory's functions and by all previously generated theorems[3] considered as rewrite rules. Rather than first generate potentially reducible terms and then rewriting them into normal form, IsaCoSy uses a constraint language to ensure they are not generated in the first place. For instance, suppose $f(c(x))$ was known to the left-hand side of a rewrite rule arising from a definition or previously proved theorem, a constraint will be generated to ban the generation of any term containing an occurrence of $f(c(\ldots))$. As new theorems are proved by IsaPlanner new constraints are generated. Typically, thousands of equations are generated, but only a handful pass the counter-example check, leaving on the order of tens to be proved.

The heuristic of requiring all terms in a conjecture to be irreducible is intended to filter out trivial theorems, leaving only those of some intrinsic interest. This simple heuristic has proven to be surprisingly successful. It was evaluated by precision/recall comparisons with manually generated sets of theorems from independent sources, such as Isabelle's libraries [Johansson *et al*, 2010a]. Such libraries contain simple theorems, such as associativity, commutativity, distributivity, idempotency, etc. Typical IsaCoSy theorems were:

$$\begin{aligned} a \times b &= b \times a \\ (a+b)+c &= a+(b+c) \\ (a \times b)+(c \times b) &= (a+c) \times b \\ rev(map\ a\ b) &= map\ a(rev\ b) \\ foldl\ a\ (foldl\ a\ b\ c)\ d &= foldl\ a\ b\ (c@d) \end{aligned}$$

IsaCoSy is restricted to generating only variable-free[4] theorems, so cannot generate, for instance, theorems containing existential quantifiers, e.g., $m < n \implies (\exists k.\, n = Suc(m+k))$. The evaluation of IsaCoSy demonstrated that it tended to generate all and only the theorems considered interesting by human experts. Where it differed, it was usually possible to argue that this was down to legitimate variation in judgement, i.e., additional theorems were similar in structure to those manually produced, and the missing ones were typically trivially derivable form ones that *were* generated. Of course, this evaluation could only be conducted for well-known recursive theories, not the novel ones generated by IsaWannaThm, but was still indicative of general effectiveness. This confirmation is important to TheoryMine, as we want customers' theorems to have some intrinsic merit[5].

4 IsaPlanner

IsaPlanner was initially created by Lucas Dixon during his PhD at the University of Edinburgh, then further developed as part of an EPSRC project. It uses proof planning [Bundy, 1991] to guide Isabelle in an inductive proof of input conjectures. In particular, it uses rippling [Bundy *et al*, 2005] to guide the step cases of inductive proofs, by manipulating the induction conclusion so that it matches the induction

[3] We have also experimented with using unproven but unfalsified conjectures, since in practice these have always turned out to be theorems, and failure to reduce with respect to them tends to lead to an over-production of conjectures. If they *aren't* theorems then no harm is caused other than an under-production of conjectures.

[4] I.e., implicitly universally quantified.

[5] Although, none of them is likely to earn anyone a Field's Medal.

hypothesis. It also uses some proof critic techniques, such as lemma calculation, to recover from an initially failed proof attempt.

These proof planning techniques enable IsaPlanner to prove many inductive conjectures entirely automatically. Such automation is essential to TheoryMine, as it enables theorems and their proofs to be generated without the need for human intervention and, therefore, to scale the service to a large number of customers at very low cost. Of course, IsaPlanner cannot automatically prove all inductive theorems — since recursive theories are undecidable in general. This is not a problem for TheoryMine, provided that a large number of theorems *can* be proved, which is the case. We estimate that, within the current complexity thresholds, IsaWannaThm is capable of generating of the order of 10^{16} theorems. That's more than a million for each person on the planet.

5 Isabelle

Isabelle is being developed by Larry Paulson's group (University of Cambridge) and Tobias Nipkow's group (Technische Universität München). It is a generic, interactive proof assistant. Mathematical theories can be expressed in a variety of logics, although classical higher-order logic is the most popular. The user can then guide an attempt to prove a conjecture written in the chosen logic and within the chosen theory. Isabelle is an LCF-style prover. This means that it has a small trusted core of logical rules, and that every proof must ultimately consist of a combination of operations within that core. This architecture provides a very high level to assurance of the correctness of any theorems produced by Isabelle. This is important to TheoryMine, as we need our customers to be sure that what they buy are indeed *theorems*.

Proofs can be partially automated by the use of tactics. These combine the basic rules of inference and axioms, structuring the proof at a higher-level of granularity, so that the user has fewer choice points to navigate. Tactics can range in power from the composition of a few rules to sub-routine calls to entire third-party theorem provers. Although this enables simple theorems to be proved entirely automatically, most non-trivial theorems do require human intervention. IsaPlanner uses Isabelle's tactics to instruct it as to which steps to make.

6 Conclusion

We have described the underlying theory and technology behind the TheoryMine novelty gift company, that produces theorems to be named by customers. This technology ensures the following desirable properties of TheoryMine's products.

- Restricting TheoryMine's scope to purely definitional theories ensures that they are always consistent, so that not all formulae are trivially provable.
- Isabelle's LCF-style architecture ensures that the theorems are correctly proved.
- IsaPlanner's proof planning ensures that these proofs are produced entirely automatically.
- IsaCoSy's irreducibility heuristic ensures that the theorems have some intrinsic interest.
- IsaWannaThm's meta-grammars for types and functions generate a huge number of novel theories and theorems.

These properties ensure that TheoryMine can automatically serve a large number of customers at minimal cost with theorems that are correctly proved, non-trivial and intrinsically interesting.

References

[Berghofer & Nipkow, 2004] Berghofer, S. and Nipkow, T. (2004). Random testing in Isabelle/HOL. In *SEFM '04: Proceedings of the Software Engineering and Formal Methods, Second International Conference*, pages 230–239, Washington, DC, USA. IEEE Computer Society.

[Bundy, 1991] Bundy, A. (1991). A science of reasoning. In Lassez, J.-L. and Plotkin, G., (eds.), *Computational Logic: Essays in Honor of Alan Robinson*, pages 178–198. MIT Press.

[Bundy *et al*, 2005] Bundy, A., Basin, D., Hutter, D. and Ireland, A. (2005). *Rippling: Meta-level Guidance for Mathematical Reasoning*, volume 56 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press.

[Cavallo, 2009] Cavallo, F., (2009). Vanity theorem proving. Undergraduate Dissertation, University of Edinburgh.

[Dixon & Fleuriot, 2004] Dixon, L. and Fleuriot, J. D. (2004). Higher order rippling in IsaPlanner. In *Theorem Proving in Higher Order Logics'04*, volume 3223 of *LNCS*, pages 83–98. Springer.

[Johansson, 2009] Johansson, M. (2009). *Automated Discovery of Inductive Lemmas*. Unpublished Ph.D. thesis, University of Edinburgh.

[Johansson *et al*, 2010a] Johansson, M., Dixon, L. and & Bundy, A. (2010a). Conjecture synthesis for inductive theories. *Journal of Automated Reasoning*. Forthcoming.

[Johansson *et al*, 2010b] Johansson, M., Dixon, L. and Bundy, A. (2010b). Formalising term synthesis for IsaCoSy. In *Automatheo 2010*. An IJCAR 2010 workshop.

[Paulson, 1994] Paulson, L. C. (1994). *Isabelle: A generic theorem prover*. Springer-Verlag.

[Truesdell, 1958] Truesdell, C. (Mar 1958). The new Bernoulli edition. *Isis*, 49(1). Truesdell discusses the strange agreement between Bernoulli and l'Hôpital on pages 59-62.