The Theory behind TheoryMine

Alan Bundy, Flaminia Cavallo, Lucas Dixon, Moa Johansson, and Roy McCasland

***Abstract*—We describe the technology behind the TheoryMine novelty gift company, which sells the rights to name novel mathematical theorems. A pipeline 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 theorem discovery and proof processes are entirely automatic. The process guarantees large numbers of sound, novel theorems of some intrinsic merit.**

***Index Terms*—Novelty Gifts, Conjecture Generation, Automated Theorem Proving.**

I. Introduction

TheoryMine (theorymine.co.uk) is a spin-out company in the novelty gift market, started in 2010. 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 proof hint and the definitions of the functions and types occurring in it. An example certificate is given in Figure 1. The name of the theorem is registered in TheoryMine's database for future reference.

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 [13][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. The purpose of TheoryMine is to provide a fun, tongue-in-cheek application of these automated reasoning technologies. It also serves as a popular-science introduction to more serious applications of this research. In particular, automated reasoning is an important component in verification tools to make software more reliable and safe to use, as well as tools to ease the exploration of new mathematical concepts. TheoryMine has been featured in mainstream media several times, for example in The New Scientist [4], British newspapers The Guardian [2] and The Herald [3], as well as on BBC Radio 4 [1]. In this article, we outline the theory and technology behind TheoryMine. The theorem discovery and proof process is entirely automatic, with human intervention limited to administering orders.

The TheoryMine system consists of a pipeline of four automated reasoning systems, which we list below.

- **IsaWannaThm** [8], generates novel recursive types and functions to form new recursive theories, then uses IsaCoSy to generate new theorems in those theories.
- **IsaCoSy** [10], given a recursive theory, generates inductive conjectures in that theory, using IsaPlanner to prove them.
- **IsaPlanner** [9], given an inductive conjecture, tries to prove it using an inductive proof plan to guide Isabelle in the search for a proof.
- **Isabelle** [12], 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.

II. IsaWannaThm

IsaWannaThm was initially developed by Flaminia Cavallo during her final-year undergraduate project at the University of Edinburgh [8], but has subsequently been largely revised and rewritten to improve its range and performance, as described below. It creates novel recursive theories by incremental, exhaustive generation from grammars describing the spaces of possible recursive types and possible recursive functions from and to those types.

- Given an initial set of types, it incrementally defines new recursive types. The default initial set consists of the booleans $bool$ and the natural numbers $\mathbb{N}$;
- It then uses the initial types and the newly defined types to construct candidate types for recursive functions.
- Finally, it uses IsaCoSy, parameterised by a set of constructor and defined functions, to define novel recursive functions.

A. 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 $natlist$. The booleans $bool$

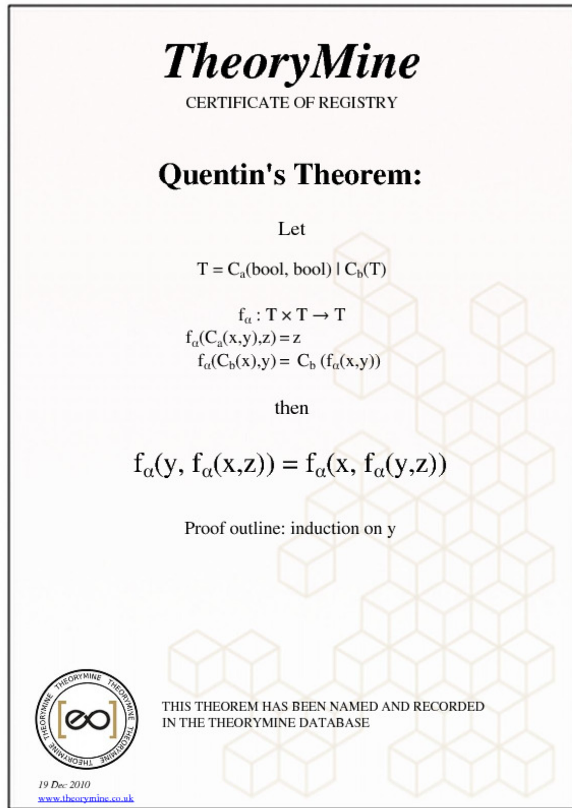
This work was supported by EPSRC grants EP/E005713/1 and EP/F033559/1, and an EPSRC studentship to Dr Johansson. An earlier version of this work was presented at the AUTOMATHEO 2010 workshop. It has not been published as the workshop did not have proceedings.

A. Bundy and R. McCasland are with the School of Informatics, University of Edinburgh, UK (e-mail: a.bundy@ed.ac.uk, rmccasla@staffmail.ed.ac.uk).

F. Cavallo is the CEO of TheoryMine Ltd, UK (e-mail: f.cavallo@theorymine.co.uk).

L. Dixon is with Google, New York, USA (e-mail: lucas.dixon@gmail.com).

M. Johansson is with the Department of Computer Science and Engineering, Chalmers University of Technology, Gothenburg, Sweden (e-mail: moa.johansson@chalmers.se).



The certificate shows in turn:

- *the theorem's name "Quentin's Theorem";*
- *the recursive definition of the type T;*
- *the type declaration and recursive definition of the function f_α;*
- *the theorem statement and a proof hint. All proofs are by induction on one variable, here the y.*

The type T can be understood as a four-coloured version of the natural numbers, in which $C_a(bool, bool)$ provides four different 'zero's and $C_b(T)$ successively generates the next 'number' in the sequence of each colour. f_α is a kind of addition on these 'numbers'. Note that f_α 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 f_α 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.*

Fig. 1. **Example Customer's Certificate**

can be considered as a degenerate recursive type, with two base cases and no step cases.

$$bool \quad = \quad True \mid False$$

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 of base type.

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 that describes all possible ways of generating recursive type definitions.

We consider two types to be isomorphic if a permutation of constructor names and argument order makes them syntactically identical. IsaWannaThm generates datatypes uniquely by constructing them in an ordered fashion, with constructor names and arguments sorted by a total ordering. Thus it avoids isomorphic variants of the same constructor function for different types, e.g., it does not generate both $C_c(T, \mathbb{N})$ and $C_c(\mathbb{N}, T)$. Upper limits are set on the parameters to prevent the generated types becoming too complex for successful theorem proving.

The example of a recursive type given in Figure 1 is:

$$T \quad = \quad C_a(bool, bool) \mid C_b(T)$$

IsaWannaThm's naming convention for types is a T possibly subscripted with a natural number. Its naming convention for constructor functions is a C usually subscripted with a Roman lower case letter.

To ensure that TheoryMine's theorems are novel, IsaWannaThm avoids generating types isomorphic to well-known recursive types. Any types that already appear in Isabelle libraries are thus filtered out. 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 thus generates recursive types of the form

$$\tau = C_1(t_{1_1}, \ldots, t_{1_m}) | \ldots | C_n(t_{n_1}, \ldots t_{n_k})$$

where each constructor C_i has zero or more arguments. Each argument type t_j is either τ itself, $bool$, $\mathbb{N}$ or a previously generated type. This means that IsaWannaThm currently only produces first-order types. IsaWannaThm is also restricted to freely generated 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:

$$\begin{aligned} \tau_1 &= null \mid c_1(\tau_1, \tau_2) \\ \tau_2 &= null \mid c_2(\tau_1, \tau_2) \end{aligned}$$

Lifting these restrictions is a topic for future work.

B. Generating Recursive Functions

Now assume that IsaWannaThm has defined a recursive type, using the methods of §II-A, 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 now generates the types for recursive functions on τ. The example of a function type given in Figure 1 is $T \times T \rightarrow T$, where T is the new recursive type. This function type could also have used *bool* and $\mathbb{N}$. Type variables are not currently used, i.e., IsaWannaThm cannot generate polymorphic functions.
- Without loss of generality and to avoid redundancy, functions are assumed to be recursively defined on their first argument. To ensure that the function is novel, the type of this first argument must be one of those that IsaWannaThm has generated, e.g., T in our example, but not *bool* or $\mathbb{N}$.
- In generating these function types, IsaWannaThm avoids associative or commutative variants in the arguments of the function. For instance, it does not generate both $T_1 \times T_2 \times \mathbb{N} \rightarrow T_3$ and $T_1 \times \mathbb{N} \times T_2 \rightarrow T_3$. It does, however, generate both $T_1 \times T_2 \times \mathbb{N} \rightarrow T_3$ and $T_2 \times T_1 \times \mathbb{N} \rightarrow T_3$. Otherwise, it would not be able to have functions recursing on both T_1 and T_2.
- For each of these function types, IsaWannaThm generates a set of candidate structurally recursive functions with this type. IsaWannaThm's function naming convention is the letter f with Greek letters as subscripts. The example of a new recursive function given in Figure 1 is f_α.
- The left-hand sides of the new function definitions are generated by a simple scheme. IsaCoSy is then used to generate a candidate right-hand sides. The example function definition in Figure 1 is:

  $$\begin{aligned} f_\alpha(C_a(x,y),z) &= z \\ f_\alpha(C_b(x),y) &= C_b(f_\alpha(x,y)) \end{aligned}$$

  These definitions consist of a number of base case and step case equations. In our example the first equation is a base case and the second is a step case. There is a one-to-one correspondence between these cases and the number and structure of the recursive definition of the type of the function's first argument. Recall that the recursive definition of T is:

  $$T \quad = \quad C_a(bool, bool) \mid C_b(T)$$

  and note the correspondence with the first arguments of f_α in the left-hand-sides of its two defining equations. The scheme used to generate the left-hand-side (*head*) of each case consists of the function name, a constructor term in the first argument position and distinct variables in the remaining argument positions. While the left-hand side of the function definition is generated by this simple scheme, IsaCoSy is used to generate the right-hand-side (*body*) of each case. It uses its grammar of well-formed terms to incrementally generate all possible terms of the required type. It must use the function being defined somewhere in each step case body. It can also use constructor functions and previously defined recursive functions.
- Many of these candidate definitions are not well-founded, i.e., they define non-terminating functions. We use Isabelle's function package [11] to filter out any which are not proved to be well-founded, thus ensuring consistency.

This process generates a potentially very large set of recursively defined functions. In fact the set of functions is so large that we cannot store it in a computer's memory. We use continuations to make this part of IsaWannaThm's evaluation lazy. Thus, we only generate a tiny fraction of the space of functions at any one time.

Again, upper limits are set on the number of a function's arguments and the size of its case bodies, to prevent the generated functions becoming too complex for successful theorem proving.

C. 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 is given in Figure 1. In §III, we describe how theorems of these theories are automatically generated by the IsaCoSy system.

As we use Isabelle's function package as a filter, we ensure that only terminating recursive functions are included in the theory. Theories where all axioms are such recursive definitions 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, in an inconsistent theory, all formulae are theorems.

To ensure that TheoryMine's theorems are always novel, with respect to previously know theories, 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.

In the very unlikely event that TheoryMine would rediscover some datatype and functions (not excluded by our current heuristics) which would yield a theorem that already has a name, e.g. in a mathematics textbook, the customer will be offered two freshly generated theorems for free as compensation. 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 planet Earth.

III. IsaCoSy

IsaCoSy was developed by Moa Johansson during her PhD at the University of Edinburgh, [10]. It creates implicitly universally quantified equations in a recursive theory by generating irreducible terms, then filtering out most non-theorems using the counter-example finder QuickCheck [5]. 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 [10].

The example given in Figure 1 is:

$$f_\alpha(x, f_\alpha(y,z)) \quad = \quad f_\alpha(y, f_\alpha(x,z))$$

Note that x and y are commuted, but only in the context of z. To see that f_α is not commutative in general, consider for instance:

$$\begin{aligned} f_\alpha(C_a(t,f), C_a(f,t)) &= C_a(f,t) \\ &\neq C_a(t,f) \\ &= f_\alpha(C_a(f,t), C_a(t,f)) \end{aligned}$$

IsaCoSy generates equations where the left-hand side is always greater or equal to the right-hand side according to a simple size based measure. The equations where the left-hand side is greater can thus be viewed as rewrite rules. All terms generated by IsaCoSy are guaranteed to be irreducible both by the recursive definitions of the theory's functions and by those previously generated theorems[1] which can be considered as rewrite rules (commutativity theorems, for example, are not rewrite rules and thus excluded). 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 be 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 merit. 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 [10]. 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 implicitly universally quantified equations, so cannot generate, for instance, theorems containing conditionals or 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 from 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[2].

As we saw in §II-B, IsaCoSy is also used to generate the bodies of the definitions of recursive functions. Its ability to generate all irreducible terms from the grammar of the term language is simply adapted to this additional application.

The conjecture generation and counter-example checking in IsaCoSy is what takes the longest in the TheoryMine system. The runtimes for IsaCoSy vary a great deal, from minutes to hours. This depends on properties and complexity of the theory being explored and the size bounds on conjectures generated. Theories involving polymorphism or many commutative functions take longer to explore, especially if we also want to look for large conjectures [10]. However, small conjectures in monomorphic theories can typically be generated relatively quickly. For the purpose of TheoryMine, we do not consider theories with polymorphism. We typically let the system run for some given amount of time to populate our database with

[1]We have also experimented with using unproven but unfalsified conjectures, since in practice these have usually 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.

[2]Although, none of them is likely to earn anyone a Fields Medal.

whatever new theorems it has discovered in that time. It is not necessary to discover all theorems of a theory, as we easily can generate new datatypes, functions and theories.

IV. 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 [9]. It uses proof planning [6] to guide Isabelle in an inductive proof of input conjectures. In particular, it uses rippling [7] to guide the step cases of inductive proofs, by manipulating the induction conclusion so that it matches the induction 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.

V. 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 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 improves on Isabelle's automation by automatically choosing which tactics to use in proofs by induction.

VI. 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 terminating recursive functions ensures that the theories produced 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 merit.
- 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 of some intrinsic merit.

References

[1] Material world. BBC Radio 4, 15/04/2010. http://www.bbc.co.uk/programmes/b00ryl03#synopsis.

[2] Unique Christmas ideas: Gifts for people who are difficult to buy for. The Guardian, 19/11/2010. www.theguardian.com/lifeandstyle/gallery/2010/nov/19/christmas-gifts-difficult-to-buy-for.

[3] Your own maths theorem for £15. The Herald, 16/11/2010. www.heraldscotland.com/news/education/your-own-maths-theorem-for-15-1.1068654.

[4] J. Aron. Mathematic immortality? Name that theorem. New Scientist, 03/12/2010. www.newscientist.com/article/dn19809-you-too-can-get-that-pythagoras-feeling#.VG2t60t994M.

[5] S. Berghofer and T. Nipkow. 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, 2004. IEEE Computer Society.

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

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

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

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

[10] M. Johansson, L. Dixon, and A. Bundy. Conjecture synthesis for inductive theories. *Journal of Automated Reasoning*, 47:251–289, 2011.

[11] A. Krauss. Partial recursive functions in higher-order logic. In U. Furbach and N. Shankar, editors, *Automated Reasoning*, volume 4130 of *LNCS*, pages 589–603. Springer, Oct 2006.

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

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