# CADE-18 Workshop:
# Problems and Problem Sets

Contents:



Organizers: Geoff Sutcliffe, Jeff Pelletier, Christan Suttner

# Extracting Test Problems from Real Applications

John Harrison

Intel, USA
johnh@ichips.intel.com

**Abstract.** The HOL Light theorem prover has a number of automated subsystems, e.g., a model elimination procedure for first order logic with equality, and arithmetic provers for linear and non-linear arithmetic. The sub-problems that are dealt with by these components can easily be extracted to give a good selection of the relatively easily decidable problems that arise in "real" applications, such as formalizing mathematics and performing industrial verifications. These can then be used as test problems for other automated provers, and possibly incorporated into standard test suites such as TPTP. We have already made available some test problems generated in this way. This simple approach has the disadvantage that the problems tend to be relatively easy, and self-selected for the particular methods used in HOL's own provers. However, since HOL is an LCF-style prover, it is essentially trivial to capture all proofs in the system, regardless of whether they are wholly or partly automated. Using this technique, we can generate realistic test problems of essentially arbitrary difficulty.

# Integrating HR and tptp2X into MathWeb to Compare Automated Theorem Provers

Jürgen Zimmer and Andreas Franke
Universität des Saarlandes
D-66041 Saarbrücken, Germany
Email: {jzimmer|afranke}@mathweb.org

Simon Colton
Division of Informatics
University of Edinburgh, UK
Email: simonco@dai.ed.ac.uk

Geoff Sutcliffe
Department of Computer Science
University of Miami, USA
Email: geoff@cs.jcu.edu.au

**Abstract**

The assessment and comparison of automated theorem proving systems (ATPs) is important for the advancement of the field. At present, the de facto assessment method is to test provers on the TPTP library of nearly 6000 theorems. We describe here a project which aims to complement the TPTP service by automatically generating theorems of sufficient difficulty to provide a significant test for first order provers. This has been achieved by integrating the HR automated theory formation program into the MathWeb Software Bus. HR generates first order conjectures in TPTP format and passes them to a concurrent ATP service in MathWeb. MathWeb then uses the tptp2X utility to translate the conjectures into the input format of a set of provers. In this way, various ATP systems can be compared on their performance over sets of thousands of theorems they have not been previously exposed to. Our purpose here is to describe the integration of various new programs into the MathWeb architecture, rather than to present a full analysis of the performance of theorem provers. However, to demonstrate the potential of the combination of the systems, we describe some preliminary results from experiments in group theory.

## 1 Introduction

Automated Theorem Proving (ATP) is concerned with the development and use of systems that automate sound reasoning: the derivation of conclusions that follow inevitably from facts. A key concern of ATP research is the development

of more powerful systems, capable of solving more difficult problems within the same resource limits. In order to build more powerful systems, it is important to understand which systems, and hence which techniques, work well for what types of problems. This knowledge is a key to further development, as it precedes any investigation into why the techniques and systems work well or badly. This knowledge is also crucial for users: given a specific problem, a user would like to know which systems are most likely to solve it. For classical first order ATP, empirical evaluation is a necessary tool for obtaining this knowledge.

One way to evaluate and compare ATP systems is to use the TPTP problem library [SS98], as discussed in §2.1 below. There are, however, some concerns with developers' continual use of the TPTP. The most significant concern is that ATP researchers who always use this library for testing their systems run the risk of producing systems that can solve only TPTP problems, and are weak on new problems or applications. One way to counter this concern is to keep adding new and increasingly more difficult theorems to the TPTP library. New problems are continually being added to the TPTP library, and in [CSar] we discuss how the HR program [CBW99] — as discussed in §3.1 below — has been used to generate novel theorems for this library.

We discuss here another way to counter the problem of researchers fine-tuning their systems to perform well on a library of problems. We intend to compliment the TPTP library with a new service available for ATP researchers. This service will generate a set of theorems over a number of domains, which are difficult enough to differentiate (in terms of efficiency) a set of provers. The analysis of why particular provers perform well over sets of (or individual) theorems will give insight into the strengths and weaknesses of the provers, which will drive their development. To achieve this, we have integrated the HR program [CBW99] and the tptp2X utility (see §3.2 below) into the MathWeb Software Bus [FK99]. In short, the integration is as follows: (i) HR generates a conjecture in TPTP format which is empirically true over a certain number of examples (ii) MathWeb employs the tptp2X utility to translate the theorems into the input formats of a number of ATPs and (iii) MathWeb invokes the provers to prove the theorems.

This project has three stages:

1. Integrate HR and tptp2X into MathWeb and demonstrate the system.
2. Make HR available as a service within MathWeb for ATP researchers.
3. Analyse the performance of a set of provers on sets of theorems.

We describe here the first stage of the project: how we have integrated HR and tptp2X into MathWeb in order to generate and prove thousands of conjectures using a variety of provers. To describe the integration, in §2 we discuss the TPTP library and MathWeb. Following this, in §3 we discuss the integration of the new additions to MathWeb, namely the E ATP system, HR and tptp2X. To demonstrate the potential of this system for ATP researchers, in §4 we describe some preliminary experiments using the provers Bliksem, E, Otter and Spass to prove thousands of theorems in group theory, with the results given in §5. We discuss the next stages of this project in §6.

# 2 Background

## 2.1 The TPTP Library

The TPTP (Thousands of Problems for Theorem Provers) Problem Library [SS98] is a library of test problems for ATP systems. It was developed in order to move the testing and evaluation of ATP systems from the previously ad hoc situation onto a firm footing. Since the first release in 1993, many researchers have used the TPTP library for testing their ATP systems and this is now the de facto standard for testing first order ATP systems.

Problems in the TPTP library are in full first-order form (FOF problems) or clause normal form (CNF problems). Each problem contains header information that identifies and describes the problem, provides information about occurrences of the problem in the literature and elsewhere, and gives the problem's ATP status and a table of syntactic characteristics of the problem. A problem may `include` standard axiom sets, and all problems contain their specific formulae. An important item of status information in each problem's header is its *difficulty rating*. This rating is computed using performance data from state-of-the-art ATP systems [SS01], and is a value in the range 0 to 1. Problems with a rating of 0 are *easy*, and can be solved by all state-of-the-art ATP systems. Problems with a rating between 0 and 1 are *difficult*. The rating value is a measure of the fraction of state-of-the-art ATP systems that fail to solve the problem within realistic resource limits. Problems with a rating of 1 are *unsolved* by any ATP system, in normal testing. The ratings are important, as they allow users to select problems according to their intentions.

The syntax of problems files is that of Prolog, which makes it trivial to manipulate the files using Prolog. In particular, the TPTP comes with the tptp2X utility (written in Prolog) that can convert problems from TPTP syntax to the syntax used by existing ATP systems, as discussed in §3.2. Access to the TPTP and related software is available at `www.cs.miami.edu/~tptp/`. In particular, the SystemOnTPTP interface [Sut00] allows a TPTP problem to be submitted in various ways to a range of ATP systems. In addition, the interface will recommend which ATP systems are most likely to be able to solve the problem. For example, for problems expressed in non-Horn pure equality, E-SETHEO csp01, Gandalf c-1.9c, SCOTT 6.0.0, and Spass 1.03 are currently recommended by SystemOnTPTP.

## 2.2 The MathWeb Software Bus

The MathWeb Software Bus (`www.mathweb.org/mathweb/`) is a platform for distributed automated reasoning that supports the connection of a wide range of mathematical services by a common software bus [FK99]. MathWeb provides the functionality to turn existing theorem proving systems, computer algebra systems (CAS), and other reasoning systems into mathematical services that are homogeneously integrated into a networked proof development environment. The environment thus gains the services from these particular modules, but each

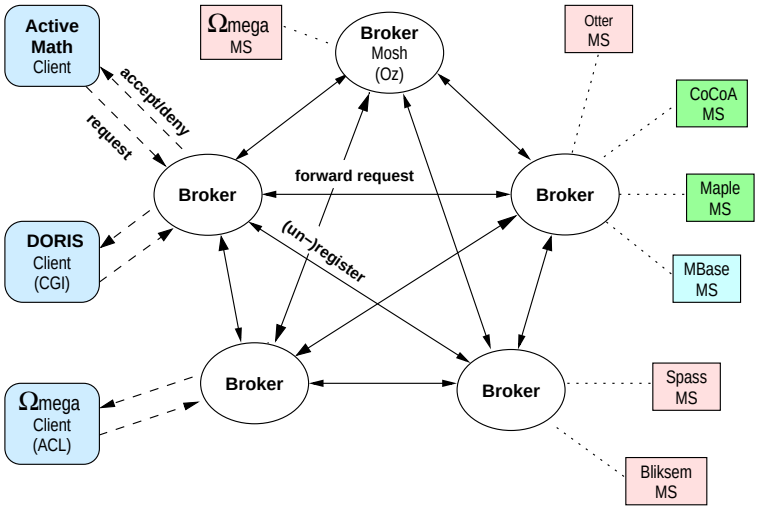



Figure 1: Current state of the MathWeb system

module in turn gains from using the features of other components in MathWeb. MathWeb is implemented in the multi-paradigm programming language Mozart (`www.mozart-oz.org`) which enables easy distribution of applications over a LAN or the Internet [Smo95]. This enabled us to create a stable network of mathematical services which is in every day use. Client applications can access 23 different reasoning and computation systems: computer algebra systems such as Maple, Magma and Cocoa, constraint solvers, mediators, model generators such as Mace and Satchmo. Moreover, MathWeb integrates nine automated theorem provers, such as Otter, Spass and Bliksem.

Mediators are mathematical services which transform mathematical knowledge from one format to another. Currently, MathWeb integrates mediators to translate (i) OpenMath formulae [CC98] into a variety of formats (ii) OmDoc documents [Koh00] into the logic of the Omega system [BCF+97], and (iii) ATP problems in DFG syntax into Otter input syntax. In addition to first order automated theorem provers, MathWeb also offers a concurrent ATP service which calls a selection of ATPs concurrently on a given problem. An application using this service can choose to see only the first result (i.e. the result of the fastest prover) or all the results. The latter enables a runtime comparison and/or several independent judgements about a given conjecture to be obtained.

The current architecture of the MathWeb system is depicted in figure 1. We see that local *brokers* provide routing and authentication information to the mathematical services (see [FK99, SHS98] for details). So called *meta-services*, offer the mathematical services (e.g., an ATP, CAS or a model checker) to their local broker. MathWeb brokers register with each other thus building a dynamic web of brokers. Client applications such as the Omega system, LOUI (a GUI

for Omega), or CGI-scripts connect to one of the MathWeb brokers and request services. If the requested service is not offered by a local meta-service, the broker forwards the request to all known brokers until the service is found. If the requested service is found, the client application receives a reference to a newly created *service object* and can send messages directly to the object. Some service objects such as Omega also act as clients, and request other services themselves.

# 3 Integration of New Systems into MathWeb

For this project, we have integrated three new systems into MathWeb:

- The automated theory formation system HR to generate theorems.
- The tptp2X utility to translate the theorems.
- The E automated theorem prover to prove the theorems.

HR was implemented as a new client and tptp2X as a mediator in order to test the ATP systems in MathWeb with conjectures supplied by HR, and we discuss these systems in this section. However, E was added as a new system to test, and we discuss this in §4, along with the other provers employed in the experiments. We also had to extend the existing MathWeb ATPs which were used in the experiments — namely Bliksem, Otter and Spass — to accept TPTP problem descriptions and to use the new tptp2X service. We discuss the technical problems which arose in the integration of the new systems and the adaptation of the current systems in §3.3.

## 3.1 The HR System

The HR program (named after mathematicians Hardy and Ramanujan) performs automated theory formation in domains of pure mathematics such as number theory, graph theory, and finite algebras, such as group theory and ring theory. The initial information about a domain supplied to HR include the axioms of the domain and optionally some initial concepts (e.g., multiplication and addition in number theory). The concepts are supplied with both a definition and some examples (e.g., triples of integers related by multiplication). In finite algebraic domains, HR can start with just the axioms of the theory, as it extracts initial concepts from these, e.g., given the identity axiom in group theory, HR extracts the concept of identity elements. HR operates by performing a theory formation step that attempts to invent a new concept from one (or two) old ones. Concept formation is facilitated using one of a set of general production rules that generate both a definition and set of examples for the new concept, from the definition and examples of the old concept(s). The *complexity* of a concept is measured as the number of production rule steps which were used to construct the concept. The 10 production rules are described in detail in [CBW99], [CBW00a] and [Col00], and include the following four:

- Compose rule: uses conjunction to join the definitions of two previous concepts
- Exists rule: introduces existential quantification
- Match rule: equates variables in a concept's definition
- Negate rule: negates predicates within a definition

A theory formation step will lead to either: (a) a concept that has no examples, (b) a concept that has exactly the same examples as a previous concept, or (c) a concept that has non-trivial examples that differ from those of all previously existing concepts. In the first case, there may be no examples for the concept because of the lack of data given to HR, or it may be because the definition of the concept is inconsistent with the axioms of the domain. Hence, HR makes a conjecture that no examples of the concept exist. In the second case, HR makes an if-and-only-if conjecture, stating that the definitions of the new concept and the previous one are equivalent. In the last case, the concept is simply added to the theory.

When HR makes conjectures in finite algebraic domains, it can invoke the Otter theorem prover [McC94] to attempt to prove the conjecture. If Otter fails, HR invokes the Mace model generator [McC01] to attempt to find a counterexample. If neither Otter nor Mace are successful, then the conjecture remains open. In cases where Otter proves an equivalence theorem, HR breaks this into a set of implication theorems, where a set of premise predicates imply a single goal predicate. Furthermore, HR extracts prime implicates from each implication theorem, i.e. it takes ever-larger subsets of the premises from the implication theorem and sees whether Otter can prove that they imply the goal.

## 3.2 The tptp2X Utility

The tptp2X utility is a multi-functional utility for reformatting, transforming, and generating TPTP problem files. In particular, tptp2X can be used to:

- Control the generation of TPTP problem files from TPTP generator files.
- Apply various transformations to TPTP problems.
- Convert problems in TPTP format to formats used by existing ATP systems.

The transformations currently available in tptp2X include conversion of FOF problems to CNF, random reordering of formulae and literals (to test sensitivity to the particular presentation in the TPTP), addition and removal of equality axioms from problems (as required by many ATP systems), and to apply Stickel's magic set transformation [Sti94]. The output format currently available from tptp2X include the Bliksem, Dedam, DFG, DIMACS, KIF, Otter, Protein, PTTP, Setheo, and Waldmeister formats. The core of tptp2X is written in Prolog, thus it can easily read and manipulate the Prolog format syntax of TPTP problems. The core has a modular construction, and it is easy for users to add new transformations and formats. The most common use of tptp2X is from a terminal command prompt, and a shell script interface is provided for this purpose. Direct use via Prolog is also possible, and various features of tptp2X have been optimized for this style of use.

### 3.3 Integration Details

We have integrated HR into MathWeb with the intention that HR will be both a client (utilising other services in MathWeb) and a service (utilised by other clients in MathWeb), with this latter aspect discussed in §6 below. Hence we wrote a MathWeb wrapper that communicates with HR via two sockets (one to handle service requests and the other to issue client commands). Due to the high number of conjectures HR typically produces in a session, we preferred the socket interface to the less efficient XML-RPC interface, which is also available in MathWeb. The high numbers of conjectures being passed around MathWeb caused many other problems, and we had to improve many aspects of MathWeb, for instance the handling of temporary files. HR was also improved. In particular, we wrote two new classes (HR is a Java program) to handle the interaction with MathWeb. The first class (MathWeb.class) abstracts the socket communication details and offers an easy way for HR to access MathWeb services, and for the user to choose which services to access. The second class (MathWebProver.class) is able to translate HR's conjectures into TPTP first order format, call MathWeb.class to employ tptp2X and the concurrent ATP service, and read the results, which are returned to HR as Java objects.

The integration of the E ATP system was straight-forward, as there is a standard MathWeb wrapper called ShellProver for automated theorem provers, which only has to be adjusted for the particular output of the prover at hand. First-order problems are written to (temporary) files and the ATPs are called as shell commands on the respective input file. Finally, the ShellProver wrapper analyses the output of the prover. The analysis of the output is the most difficult part of the integration of an ATP because automated theorem provers are typically not designed to produce machine readable output, but rather to inform a human user about their results. Hence we had to write bespoke algorithms within the prover-calling wrappers to determine the status of a theorem, extract a proof object and so on.

The integration of tptp2X was also a non-trivial task, because the standard tptp2X shell command was not efficient enough for our purposes (it took between 3 and 4 seconds to translate each theorem, as the Prolog interpreter was loaded, and tptp2X compiled each time). We therefore had to design and implement a tptp2X servlet based on a permanently running Prolog process with the pre-compiled tptp2X code that evaluates incoming transformation requests immediately. With this new service, we reduced the transformation time to 100-200 ms (for an average sized problem). This speedup was crucial, as HR produces thousands of conjectures in a short time, and we need $n$ tptp2X transformations for running $n$ provers in parallel on a single TPTP problem. We also upgraded how the runtime of the provers is recorded. All provers are started directly in the shell without any intermediate scripts and we use the Unix `time` command to record the CPU time of the prover process. We also recorded the CPU time that was used for operating system calls and added the two values. This gives an indication of the time required to prove a theorem, rather than how fast the prover is after intermediate processes have been undertaken.

# 4 Experiments

Our experiments were designed to show that a stable integration has been achieved and to highlight the potential usage of HR within MathWeb. Our aim here was to differentiate four first order theorem provers in terms of their efficiency proving a set of theorems generated by HR. The provers we employed are described in §4.1, and we describe the sessions in §4.2.

## 4.1 ATP Systems Employed

We used the latest publicly available versions of four first-order theorem provers, namely Bliksem 1.12, E 0.62, Otter 3.2, and Spass 1.03. These systems were used for the CASC system competition in 2001 [Sut01].

• Bliksem 1.12

Bliksem [DeN] implements the ordered resolution + superposition calculus. It supports many different orders, including reduction orders, A-orders, and non-liftable orders. Special attention has been given to resolution strategies that provide decision procedures for certain subsets of first order logic. Bliksem is able to transform first order formulae into clausal normal form, using different structural or non-structural clause transformations. The more recently added features of Bliksem include a posteriori orders, equality factoring and tautology elimination, optimized normal form transformations, non-unit demodulators, and equality subsumption (which takes into account commutativity of equality). High priority has been given to portability. The data structures have been carefully chosen after benchmark tests on the basic operations.

• E 0.62

E 0.62 [Sch01] is a purely equational theorem prover. The calculus used by E combines superposition (with selection of negative literals) and rewriting. No special rules for non-equational literals have been implemented, i.e., resolution is simulated via paramodulation and equality resolution. E 0.62 includes AC redundancy elimination and AC simplification for dynamically recognized associative and commutative equational theories, as well as simulated clause splitting. E is based on the DISCOUNT-loop variant of the given-clause algorithm, i.e., a strict separation of active and passive facts. Proof search in E is primarily controlled by a literal selection strategy, a clause evaluation heuristic, and a simplification ordering. Supported term orderings are several parameterized instances of Knuth-Bendix-Ordering (KBO) and Lexicographic Path Ordering (LPO). The most unique feature of the implementation is the maximally shared term representation. This includes parallel rewriting for all instances of a particular subterm. A second important feature is the use of perfect discrimination trees with age and size constraints for rewriting and unit-subsumption.

• Otter 3.2

Otter [McC94] is designed to prove theorems stated in first-order logic with equality. Otter's inference rules are based on resolution and paramodulation,

and it includes facilities for term rewriting, term orderings, Knuth-Bendix completion, weighting, and strategies for directing and restricting searches for proofs. Otter can also be used as a symbolic calculator and has an embedded equational programming system. Otter is a fourth-generation Argonne National Laboratory deduction system whose ancestors (dating from the early 1960s) include the TP series, NIUTP, AURA, and ITP.

• Spass 1.03

Spass [WAB+99] is an automated theorem prover for first-order logic with equality. It is a saturation based prover employing superposition, sorts and splitting. In contrast to many approaches to order-sorted clausal reasoning, the calculus enables sort predicates and equations to occur arbitrarily within clauses. Therefore, the sort theory is not separated from the problem clauses, but automatically and dynamically extracted. Spass also offers a variety of further inference and reduction rules including hyper resolution, unit resulting resolution, various variants of paramodulation and a terminator. Spass relies on an internal library supporting specific data structures and algorithms like, for example, indexing or orderings (KBO, RPOS).

## 4.2 Sessions with MathWeb

We ran HR for two sessions within MathWeb. In the first session, we supplied conjectures to all four provers. In the second session, we employed only Bliksem, E and Spass. The reason for the omission of Otter is that we wanted to perform a much more extensive test of the system in the second session, and had found that Otter was increasingly unable to prove the theorems produced. Otter repeatedly timing out at 120 seconds meant that the session was very slow, and so we removed Otter in order to increase the yield of theorems proved in a reasonable time. Both sessions were undertaken in the domain of group theory, and all conjectures HR produced were true of the groups up to order 8, as these were supplied to HR. The provers were given two sets of axioms from the TPTP library, namely `GRP004+0.ax` (group theory axioms) and `EQU001+0.ax` (axioms of equality). With these axioms, all the theorems produced were expressed in non-Horn pure equality. Each prover was run with the default settings and given a 120 second time limit for each theorem.

In the first session, we ran HR until it produced 1500 equivalence conjectures (with the four provers having attempted proofs). In the second session, we ran HR until it produced 12000 equivalence conjectures (again with proof attempts from the three provers). In both sessions, we ran HR with a random search, but in the first session, we employed a complexity limit of 6, with a complexity limit of 15 for the second session. Breadth first searches have been found to produce conjectures that are too simple, while depth first searches specialize the theory too much. In contrast, random searches — where, at each step, both concept and production rule are randomly chosen — tend to produce fairly complicated conjectures, without over-specializing the theory. We used only the exists, compose, negate and match production rules in these sessions.

# 5 Results

The first result to note is that the integration of HR and tptp2X into MathWeb is clearly stable, as we were able to complete a session where 12000 theorems were generated and proved. Moreover, there has been an increase in efficiency, with the four provers being called in roughly the same time as HR used to call just Otter. This is due in part to the new socket interface rather than the previous file interface, and in part to a slow implementation of HR reading the output files from Otter. For the two sessions, we recorded the number of theorems proved by each prover, the average time taken to prove those theorems which were proved, and we identified some theorems of interest, i.e. those which differentiate the provers in a significant way.

- Session 1

| Prover | No. Proved | Av. Time to Prove (ms) |
|---|---|---|
| Bliksem 1.12 | 1500 | 188.94 |
| E 0.62 | 1500 | 95.27 |
| Otter 3.2 | 1445 | 633.90 |
| Spass 1.02 | 1500 | 103.88 |

Table 1: Number proved and average time to prove - session 1

To produce 1500 theorems which were proved (or at least attempted) by all four provers took around 90 minutes on a Sun Ultra 10 workstation. The results from this session are given in table 1, and we see that Otter failed to prove 55 theorems. Of these, Otter timed out on 16, but returned 'no solution' to the other 39, and we are still investigating why Otter cannot find a solution to these. The 16 theorems which Otter timed-out on took around a third of the overall session time, which justifies our reason to remove Otter for the second session. For example, figure 2 shows a theorem (number 493) which Otter 3.2 could not prove in the 120 second limit, but all the other provers proved relatively quickly. For completeness, we present the theorem in the TPTP format which was passed to tptp2X, in addition to a more mainstream mathematical format.

```
include('Axioms/EQU001+0.ax').
include('Axioms/GRP004+0.ax').
input_formula(conjecture493,conjecture,(! [B,C,D] :
  ((equal(inverse(B),C) & equal(multiply(B,D),C) & equal(inverse(D),B)
  & ? [E,F] : (equal(inverse(E),F) & equal(multiply(E,B),F))
  & equal(multiply(D,C),B)) <=>
  (equal(inverse(D),C) & equal(inverse(B),D) & equal(multiply(C,D),B)))))).
```

---

$$\forall\, b, c, d\ (b^{-1} = c\ \&\ b{*}d = c\ \&\ d^{-1} = b\ \&\ \exists\, e, f\ (e^{-1} = f\ \&\ e{*}b = f)\ \&\ d{*}c = b$$
$$\iff d^{-1} = c\ \&\ b^{-1} = d\ \&\ c * d = b)$$

Figure 2: Theorem 493 in session 1, which Otter could not prove in 120 seconds

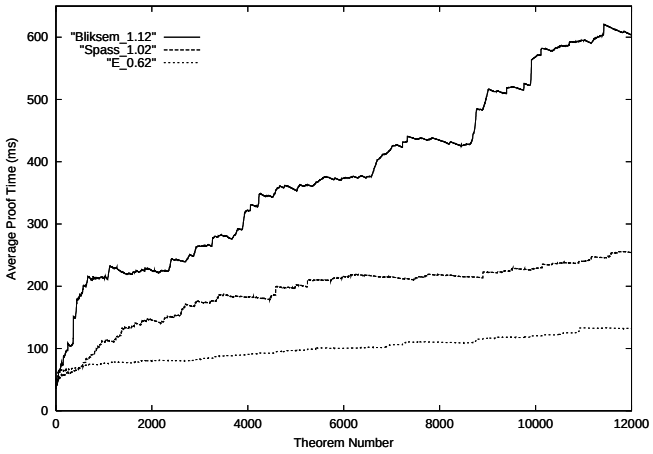



Figure 3: Average times to prove theorems

• Session 2

This session took around 14 hours on a Sun Ultra 10 to produce the required 12000 theorems. All were proved by at least one prover, with 70 exceptions. We have yet to determine which, if any, of the 70 were actually non-theorems. More interesting from the perspective of comparing and contrasting the provers, for each prover, there was at least one conjecture which it could not prove in 120 seconds, but both the others could. Specifically, there were 19 which were proved by E and Spass, but not by Bliksem, 4 which were proved by E and Bliksem, but not Spass and only 1 which Spass and Bliksem proved, but E did not. In appendix A, we provide examples of these theorems.

The average time taken to prove the theorems by each prover are given in table 2. These suggest that (with the default settings) E is most suited to problems of this type, followed by Spass and then Bliksem. Also, Bliksem failed to prove more theorems than E and Spass, although the differences are not great. These results correlate with SystemOnTPTP, which — as mentioned in §2.1 — recommends E (actually E-Setheo, a hybrid) and Spass, but not Bliksem, for non-Horn pure equality theorems such as those HR produced. We also recorded how the average time to prove a theorem changed as the session progressed (see figure 3). For all the provers, the average time to prove a theorem increased in general as HR's theory progressed. This suggests that, given enough time, HR can get to a stage where a theorem will, on average, take an arbitrarily long time to prove by all provers. This is important, as discussed in §6 below, although we need more experimentation to confirm this observation.

| Prover | No. Proved | Av. Time to Prove (ms) |
|---|---|---|
| Bliksem 1.12 | 11908 | 607.01 |
| E 0.62 | 11931 | 132.32 |
| Spass 1.02 | 11927 | 254.97 |

Table 2: Number proved and average time to prove - session 2

# 6 Conclusions and Future Work

In previous work, we have used the HR system to produce many thousands of group theory conjectures [CSar]. Human intervention was then required to employ automated theorem provers to identify those conjectures which were (a) theorems and (b) difficult for some of the provers. This has led to 184 theorems produced by HR being added to the TPTP library. Hence we have shown that HR can be used to improve the TPTP library and we are currently undertaking a project using HR to discover theorems in more complicated algebraic domains, in particular Zariski spaces, [MMS98]. We hope that results from this application will also find their way into the TPTP library.

In addition to improving the TPTP library, we have concentrated here on the potential of using HR to compliment the library. We have described the integration of HR, the tptp2X utility and automated theorem provers into the MathWeb software bus in such a way that HR generates, tptp2X translates and the ATP systems prove conjectures. The implementation is stable enough to process thousands of theorems, and initial experiments have demonstrated that HR can produce theorems which are hard to prove for some of the ATPs but not for others, which we suggest is a potential tool for differentiation the provers. Furthermore, we have shown that the system as a whole (i.e. the integration of HR, tptp2X and the provers in MathWeb) can identify the theorems which only one prover finds difficult. We have argued that the automated generation of such theorems can be used to identify strengths and weaknesses in particular provers, in much the same way as the TPTP library does. This in turn can be used to drive the development of provers.

It is important to stress that we draw no conclusions here about the efficiencies of the provers on the theorems HR generated. For each prover, there are many settings which can drastically improve settings, and we need to perform much more extensive testing before we can make any detailed comparisons of the provers. We have been more concerned here with detailing the integration of the various systems and to demonstrate that the integration is stable and has a potential application to the comparison of provers. By presenting the results from a session where 12000 theorems were stated and proved, we have shown that the system is indeed stable. Perhaps the most compelling demonstration of the potential to compare provers are the theorems given in appendix A. While these are syntactically fairly similar, the performance of the provers on them is strikingly different.

The next stage of this project is to offer HR as a service within MathWeb, both for users and for other applications. To allow other MathWeb applications to use HR as a service, we will define an interface which allows them to call HR's conjecture generation mechanism on a given set of axioms in a specific theory. We plan to offer a service based on some standard encoding of mathematical knowledge, such as OpenMath or OmDoc. We envisage users asking HR via MathWeb to provide them with a certain number of theorems with a certain average difficulty (in terms of proof time and/or number of provers which are successful) over a set of ATP systems. The fact that the average time to prove

the theorems generally increased for each prover as HR's theory progressed (as depicted in figure 3) is encouraging. This means that we can run HR for a certain amount of time before assessing the difficulty of the theorems it produces using the ATP systems. As discussed in [CSar], in a previous session, HR produced 46,000 syntactically different group theory conjectures in just 10 minutes on a 500Mhz Pentium processor. Hence it is plausible that we could run HR for a chosen amount of time, then start the provers and reasonably expect the average proof time to be similar to the one requested by the user.

We also intend to add more intelligent ways for HR to choose the theorems that it will pass to the provers for assessment of difficulty. In [CSar], we showed some corelation between the number of existential and universally quantified variables in a theorem and the difficulty of that theorem. HR also has certain measures of interestingness of both concepts and conjectures, and we will experiment to see if there is any corelation between the value of the measures of concepts in a theorem and/or the value of the measures of the overall theorem and the time to prove it for particular provers. In particular — as discussed in [CBW00b] — for equivalence conjectures, HR estimates the 'surprisingness' by looking at how different the two concepts conjectured to be equivalence are (in terms of how they were constructed).

The final stage of the project will be to systematically test a range of provers with different settings over large sets of conjectures produced in a variety of domains by HR within MathWeb. By offering HR as a MathWeb service, we hope to do this in conjunction with the developers of the ATP systems tested. Such large-scale testing of the ATP systems will compliment the TPTP library, providing a new tool for developers of ATP systems, which will hopefully contribute positively to the development of automated theorem proving in general.

## Acknowledgments

This work has been supported by EPSRC grant GR/M98012 and the Calculemus network (`www.eurice.de/calculemus`). Simon Colton is also affiliated with the Department of Computer Science at the University of York. We are grateful to the reviewers for their useful comments on this paper.

## References

[BCF+97] C. Benzmüller, L. Cheikhrouhou, D. Fehrer, A. Fiedler, X. Huang, M. Kerber, M. Kohlhase, K. Konrad, E. Melis, A. Meier, W. Schaarschmidt, J. Siekmann, and V. Sorge. ΩMEGA: Towards a mathematical assistant. In William McCune, editor, *Proc. of the 14th Conference on Automated Deduction*, number 1249 in LNAI, pages 252–255, Townsville, Australia, 1997. Springer Verlag.

[CBW99] S. Colton, A. Bundy, and T. Walsh. Automatic Concept Formation in Pure Mathematics. In T. Dean, editor, *Proceedings of the 16th Interna-*

*tional Joint Conference on Artificial Intelligence* , pages 183–190. Morgan Kaufmann, 1999.

[CBW00a] S. Colton, A. Bundy, and T. Walsh. Automatic Identification of Mathematical Concepts. In *Proceedings of the 17th International Conference on Machine Learning* , 2000.

[CBW00b] S. Colton, A. Bundy, and T. Walsh. On the Notion of Interestingness in Automated Mathematical Discovery. *International Journal of Human Computer Studies*, 53(3):351–375, 2000.

[CC98] Olga Caprotti and Arjeh M. Cohen. Draft of the Open Math standard. The Open Math Society, `www.nag.co.uk/projects/OpenMath/omstd/`, 1998.

[Col00] S. Colton. *Automated Theory Formation in Pure Mathematics*. PhD thesis, University of Edinburgh, Edinburgh, Scotland, 2000.

[CSar] S. Colton and G Sutcliffe. Automatic Generation of Benchmark Problems for Automated Theorem Proving Systems. In *Proceedings of the 7th Symposium on Artificial Intelligence and Mathematics* , 2002 (to appear).

[DeN] H. DeNivelle. Bliksem Resolution Prover. www.mpi-sb.mpg.de/ nivelle.

[FK99] Andreas Franke and Michael Kohlhase. System description: MathWeb, an agent-based communication layer for distributed automated theorem proving. In Harald Ganzinger, editor, *Proceedings of the 16th Conference on Automated Deduction*, number 1632 in LNAI, pages 217–221. Springer Verlag, 1999.

[Koh00] Michael Kohlhase. OMDoc: An open markup format for mathematical documents. Seki Report SR-00-02, Fachbereich Informatik, Universität des Saarlandes, 2000. `http://www.mathweb.org/omdoc`.

[McC94] W.W. McCune. Otter 3.0 Reference Manual and Guide. Technical Report ANL-94/6, Argonne National Laboratory, Argonne, USA, 1994.

[McC01] W.W. McCune. MACE 2.0 Reference Manual and Guide. Technical Report ANL/MCS-TM-249, Argonne National Laboratory, Argonne, USA, 2001.

[MMS98] R. McCasland, M. Moore, and P. Smith. An introduction to Zariski spaces over Zariski topologies. *Rocky Mountain Journal of Mathematics*, 28:1357–1369, 1998.

[Sch01] S. Schulz. System Abstract: E 0.61. In R. Gore, A. Leitsch, and T. Nipkow, editors, *Proceedings of the International Joint Conference on Automated Reasoning*, number 2083 in Lecture Notes in Artificial Intelligence, pages 370–375. Springer-Verlag, 2001.

[SHS98] M. Kohlhase S. Hess, Ch. Jung and V. Sorge. An implementation of distributed mathematical services. In Arjeh Cohen and Henk Barendregt, editors, *6th CALCULEMUS and TYPES Workshop*, Eindhoven, The Netherlands, July 13–15 1998.

[Smo95] G. Smolka. The Oz programming model. In Jan van Leeuwen, editor, *Computer Science Today*, volume 1000 of *LNCS*, pages 324–343. Springer-Verlag, Berlin, 1995.

[SS98] G. Sutcliffe and C.B. Suttner. The TPTP Problem Library: CNF Release v1.2.1. *Journal of Automated Reasoning*, 21(2):177–203, 1998.

[SS01] G. Sutcliffe and C.B. Suttner. Evaluating General Purpose Automated Theorem Proving Systems. *Artificial Intelligence*, 131(1-2):39–54, 2001.

[Sti94] M.E. Stickel. Upside-Down Meta-Interpretation of the Model Elimination Theorem-Proving Procedure for Deduction and Abduction. *Journal of Automated Reasoning*, 13(2):189–210, 1994.

[Sut00] G. Sutcliffe. SystemOnTPTP. In D. McAllester, editor, *Proceedings of the 17th International Conference on Automated Deduction*, number 1831 in Lecture Notes in Artificial Intelligence, pages 406–410. Springer-Verlag, 2000.

[Sut01] G Sutcliffe. The CADE-17 ATP System Competition. *Journal of Automated Reasoning*, 27(3):227–250, 2001.

[WAB+99] C. Weidenbach, B. Afshordel, U. Brahm, C. Cohrs, T. Engel, E. Keen, C. Theobalt, and D. Tpoic. System Description: SPASS Version 1.0.0. In H. Ganzinger, editor, *Proceedings of the 16th International Conference on Automated Deduction*, number 1632 in Lecture Notes in Artificial Intelligence, pages 378–382. Springer-Verlag, 1999.

# Appendix A - Example Theorems

The following examples are in TPTP format and require TPTP axiom files `EQU001+0.ax` and `GRP004+0.ax`.

- Proved by E in 0.9 seconds, Spass in 0.2 seconds, but not by Bliksem:

```
! [B,C,D] : ((equal(multiply(C,B),D) & equal(multiply(C,D),B) & ? [E,F] :
(equal(inverse(E),F) & equal(multiply(E,B),F)) & equal(inverse(B),D) &
equal(multiply(D,B),C)) <=> (equal(multiply(B,C),D) &
equal(multiply(C,B),D) & equal(multiply(C,D),B) & equal(multiply(B,D),C) &
? [E,F] : (equal(inverse(E),F) & equal(multiply(E,B),F)) &
equal(inverse(B),D) & ? [G,H] : (equal(inverse(G),H) &
equal(multiply(G,D),H)) & equal(multiply(D,C),B))).
```

- Proved by E in 1.4 seconds, by Bliksem in 5.0 seconds, but not by Spass:

```
! [B,C,D] :
((equal(multiply(B,C),D) & equal(inverse(B),C) & equal(multiply(B,D),C) &
? [E,F] : (equal(inverse(E),F) & equal(multiply(E,B),F)) &
? [G] : (~(equal(G,identity))) & equal(inverse(D),D) ) <=>
(equal(inverse(B),C) & equal(multiply(C,B),D) & equal(multiply(B,D),C) &
? [E,F] : (equal(inverse(E),F) & equal(multiply(E,B),F)) &
? [G] : (~(equal(G,identity))) & equal(multiply(D,B),C))).
```

- Proved by Spass in 0.9 seconds, by Bliksem in 16.1 seconds, but not by E:

```
! [B,C,D] : ((equal(inverse(B),C) & equal(multiply(B,D),C) &
? [E,F] : (equal(inverse(E),F) & equal(multiply(E,B),F)) &
? [G] : (~(equal(G,identity)))) <=> (equal(inverse(B),C) &
equal(multiply(B,D),C) & ? [E,F] : (equal(inverse(E),F) &
equal(multiply(E,B),F)) & ? [G] : (~(equal(G,identity))) &
equal(multiply(D,B),C) & ? [H,I] : (equal(inverse(H),I) &
equal(multiply(H,C),I)))).
```

# The Cyclic Complete Mappings Counting Problems

Jieh Hsiang *, YuhPyng Shieh and YaoChiang Chen
Dept. of Computer Science and Information Engineering,
National Taiwan University, Taipei 106, Taiwan
hsiang@csie.ntu.edu.tw
{arping,johnjohn}@turing.csie.ntu.edu.tw

**Abstract**

A *complete mapping* of a group $(G, +)$ is a permutation $f(x)$ of $G$ with $f(0) = 0$ such that $-x + f(x)$ is also a permutation of $G$. Given a group $G$, the *Complete Mappings Counting Problem* is to find, if any, the number of complete mappings of $G$. Complete mapping problems are ideal for testing the strength of propositional solvers.

In this paper we describe various types of complete mapping problems, and their relationship with variations of the $n$-queen problems. We also present several forms of *symmetry operators* which, in addition to being theoretically interesting on their own, are crucial for improving the efficiency of the provers. Several classes of challenge problems for propositional provers are given, so are the transformations of these problems into propositional format.

## 1 Overview

A *complete mapping* of a group $(G, +)$ is a permutation $f(x)$ of $G$ with $f(0) = 0$ where 0 is the identity of $(G, +)$ such that $-x + f(x)$ is also a permutation.

For example, $(1, 2, 4)(3, 6, 5)$ is a complete mapping of $(Z_7, +)$.

| $x$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | cycle_structure |
|---|---|---|---|---|---|---|---|---|
| $f(x)$ | 0 | 2 | 4 | 6 | 1 | 3 | 5 | (1,2,4)(3,6,5) |
| $-x + f(x)$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | |

*Research supported in part by Grant NSC 90-2213-E-002-110 of the National Science Council of the Republic of China.

The concept of complete mappings was first introduced by Mann [15], and used to construct orthogonal Latin squares. It was later studied by many people (e.g., [14, 19, 5, 10, 16]) under different names.

A *strong complete mapping* is a complete mapping $f(x)$ such that $x+f(x)$ is also a permutation. The term strong complete mapping was first used by Hsu and Keedwell [12]. They gave a construction for strong complete mappings of odd order abelian groups. Earlier studies also include [2] and [9]. A strong complete mapping can be seen as a solution of the *toroidal* $n$-queen problem.

The relationship between the complete mapping problems and the $n$-queen problems is worth investigating. In this paper we present a natural extension of the toroidal $n$-queen problem (which, in itself, is a natural extension of the $n$-queen problem), which we call the *toroidal-semi $n$-queen problem*. We also establish correspondences between the class of complete mapping problems and the $n$-queen problems.

Both $n$-queen problems and the complete mapping problems are good challenge problems for propositional provers. They can easily be coded into propositional expressions, but solving them requires more than sophisticated data structures and clever programming. It also needs a good understanding of how search works and how to eliminate redundancy in the search space. Indeed, it would not have been possible for our own prover to produce some impressive results (such as showing that the number of complete mappings in $Z_{23}$ is 19,686,730,313,955) if not for the symmetry cutting strategies and partition techniques designed [13, 22] to utilize the symmetry operators in search.

A *symmetry operator* $\Pi$ on complete mappings is a function from $G^G$ to $G^G$ such that $f$ is a complete mapping if and only if $\Pi(f)$ is also a complete mapping. The operators $\{R, A, H_\alpha, T_c\}$ were described implicitly in Singer(1960) [24] for cyclic groups. In 1991, Hsu [11] used a group of operators spanned by $\{R, A\}$ to classify the set of complete mappings on a cyclic group $Z_n$. Moreover, Hsiang, Hsu, and Shieh [23] offered the numbers of complete mappings for all abelian groups $G$ with $|G| \leq 19$ and the following ones $CM(n)$ for cyclic group $Z_n$. (Remark: $CM(n) = 0$ for all even numbers n.)

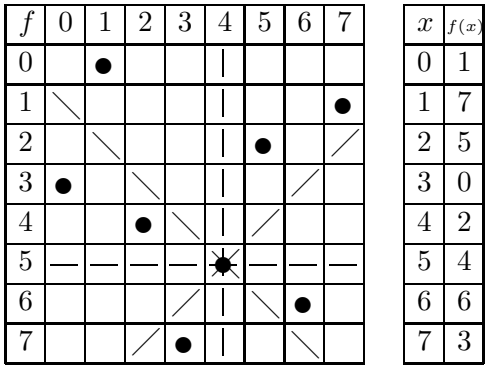
| $n$ | 5 | 7 | 9 | 11 | 13 |
|---|---|---|---|---|---|
| $CM(n)$ | 3 | 19 | 225 | 3441 | 79259 |
| $n$ | 15 | 17 | 19 | 21 | 23 |
| $CM(n)$ | 2424195 | 94417089 | 4613520889 | 275148653115 | 19686730313955 |

The rest of the paper is organized as follows. In Section 2 we give the

basic definitions and list known results of various types of $n$-queen problems and complete mapping problems, including some new ones. Section 3 introduces the symmetry operators and their properties. Section 4 defines the challenge problems for propositional provers, and Section 5 transforms the problems into propositional expressions.

## 2 Complete mappings and the $n$-queen problems

The famous *$n$-queen problem* [3, 4] can be described as follows: Place $n$ queens on a $n$ by $n$ chessboard, one queen on each square, so that no queen attacks any other. That is, there exists at most one queen on the same row, column and diagonal. There are many different perspectives of the $n$-queen problem described in Erbas, Sarkeshik, and Tanik [7]. One of the most basic is to view a solution of $n$-queen problem as a function $f(x)$ from $Z_n$ to $Z_n$ such that $f(x) = y$ if and only if position $(x, y)$ is occupied by a queen.



| $f$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 0 | | ● | | | | | | |
| 1 | | | | | | | | ● |
| 2 | | | | | | ● | | |
| 3 | ● | | | | | | | |
| 4 | | | ● | | | | | |
| 5 | | | | | ● | | | |
| 6 | | | | | | | ● | |
| 7 | | | | ● | | | | |

| $x$ | $f(x)$ |
|---|---|
| 0 | 1 |
| 1 | 7 |
| 2 | 5 |
| 3 | 0 |
| 4 | 2 |
| 5 | 4 |
| 6 | 6 |
| 7 | 3 |

Figure 1: A solution of 8-queen problem.

**Definition 1 ($n$-queen problem)** *Let $Z_n = \{0, 1, 2, \cdots, n-1\}$. A solution of the $n$-queen problem is a permutation $f(x)$ from $Z_n$ to $Z_n$ such that $\forall_{i \neq j \in Z_n} i + f(i) \neq j + f(j)$ and $\forall_{i \neq j \in Z_n} - i + f(i) \neq -j + f(j)$ under the natural number addition. We use $\widehat{Q}(n)$ to denote the set of solutions of $n$-queen problem and $Q(n)$ the cardinality of $\widehat{Q}(n)$.*

Consider a modular chessboard, that is, a chessboard where the diagonals continue on the other side as shown in Figure 2. This is a variation of the $n$-queen problem called the *modular $n$-queen problem*. The concept of modular

chessboards were introduced by Pólya [20]. There are different names for the modular $n$-queen problem. We adopt the term *toroidal* $n$-queen problem [21] in this paper. Another similar class of problems is the *toroidal-semi* $n$-*queen problem* where the diagonal "wrapping around" is assumed in only one direction.

**Definition 2 (toroidal $n$-queen problem)** *A solution of the toroidal* $n$-*queen problem is a permutation* $f(x)$ *from* $Z_n$ *to* $Z_n$ *such that (under the cyclic group* $(Z_n, +)$*),* $x + f(x)$ *and* $-x + f(x)$ *are both permutations. We use* $\widehat{TQ}(n)$ *to denote the set of solutions of the toroidal* $n$*-queen problem and* $TQ(n)$ *the cardinality of* $\widehat{TQ}(n)$*.*

**Definition 3 (toroidal-semi $n$-queen problem)** *A solution of the toroidal-semi* $n$*-queen problem is a permutation* $f(x)$ *from* $Z_n$ *to* $Z_n$ *such that (under the cyclic group* $(Z_n, +)$*),* $-x + f(x)$ *is a permutation. We use* $\widehat{TSQ}(n)$ *to denote the set of solutions of toroidal-semi* $n$*-queen problem and* $TSQ(n)$ *the cardinality of* $\widehat{TSQ}(n)$*.*

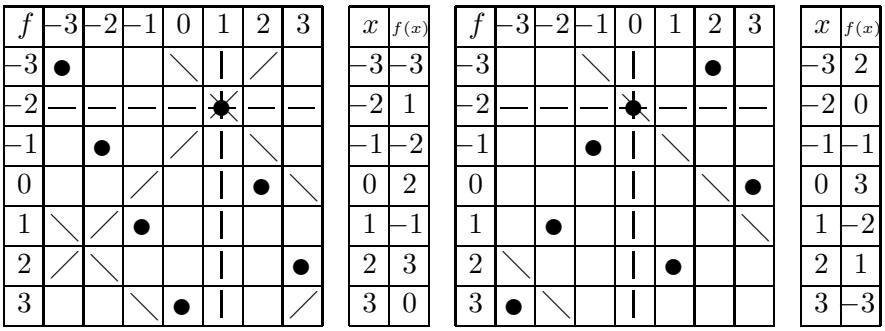



Figure 2: The left is one solution of *toroidal* 7-queen problem and the right is one solution of *toroidal-semi* 7-queen problem.

**Definition 4 (complete mapping problem)** *A* complete mapping *of a group* $(G, +)$ *is a permutation* $f(x)$ *of* $G$ *with* $f(0) = 0$ *such that* $-x + f(x)$ *is also a permutation of* $G$*. A complete mapping is called* strong *if* $x + f(x)$ *is also a permutation of* $G$*. Since in this paper we only discuss cyclic groups, we may assume* $G = Z_n$*. We use* $\widehat{SCM}(n)$ *and* $\widehat{CM}(n)$ *to denote the set of (strong) complete mappings of* $Z_n$ *and* $SCM(n), CM(n)$ *the cardinalities of* $\widehat{SCM}(n)$ *and* $\widehat{CM}(n)$*, respectively.*

By the above definitions, if $G$ is a cyclic group $Z_n$, then a solution $f(x)$ of the toroidal-semi $n$-queen problem is also a solution of the complete mapping problem if and only if $f(0) = 0$. There is a similar correspondence between the toroidal $n$-queen problem and the strong complete mapping problem. In Figure 3 we list some examples. For example, $f_1(x)$ is a solution of the (toroidal)-(semi) 11-queen problem and a (strong) complete mapping of $Z_{11}$, and $f_4(x)$ is a solution of the 11-queen problem and the toroidal-semi 11-queen problem but not a solution of the toroidal 11-queen problem and not a (strong) complete mapping of $Z_{11}$.

| $x$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | Q | TQ | TSQ | SCM | CM |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $f_1(x)$ | 0 | 2 | 4 | 6 | 8 | 10 | 1 | 3 | 5 | 7 | 9 | $\surd$ | $\surd$ | $\surd$ | $\surd$ | $\surd$ |
| $f_2(x)$ | 1 | 3 | 5 | 7 | 9 | 0 | 2 | 4 | 6 | 8 | 10 | $\surd$ | $\surd$ | $\surd$ | $\times$ | $\times$ |
| $f_3(x)$ | 0 | 2 | 5 | 8 | 1 | 7 | 10 | 3 | 6 | 4 | 9 | $\surd$ | $\times$ | $\surd$ | $\times$ | $\surd$ |
| $f_4(x)$ | 1 | 3 | 6 | 9 | 2 | 8 | 0 | 4 | 7 | 5 | 10 | $\surd$ | $\times$ | $\surd$ | $\times$ | $\times$ |
| $f_5(x)$ | 0 | 2 | 6 | 9 | 7 | 10 | 1 | 3 | 5 | 8 | 4 | $\surd$ | $\times$ | $\times$ | $\times$ | $\times$ |
| $f_6(x)$ | 0 | 2 | 1 | 5 | 8 | 10 | 9 | 4 | 3 | 7 | 6 | $\times$ | $\times$ | $\surd$ | $\times$ | $\surd$ |
| $f_7(x)$ | 1 | 3 | 2 | 6 | 9 | 0 | 10 | 5 | 4 | 8 | 7 | $\times$ | $\times$ | $\surd$ | $\times$ | $\times$ |

Figure 3: Examples under $Z_{11}$

**Proposition 1** *Given a cyclic group $Z_n$,*

1. *a solution $f(x)$ to the toroidal-semi $n$-queen problem is a complete mapping if and only if $f(0) = 0$,*
2. *a solution $f(x)$ to the toroidal $n$-queen problem is a strong complete mapping if and only if $f(0) = 0$,*
3. *$TSQ(n) = n \times CM(n)$ and $TQ(n) = n \times SCM(n)$,*
4. *$TQ(n) < TSQ(n)$ and $SCM(n) < CM(n)$,*
5. *$TQ(n) < Q(n)$.*

## 2.1 The existence problems

Given $Z_n$, the *existence* problem asks whether there exists a (toroidal)-(semi) $n$-queen solution or a (strong) complete mapping. The existence problems of the (toroidal)-(semi) $n$-queen problem have been completely solved. As

a consequence of Proposition 1, the (strong) complete mapping existence problems are also solved. We list the results here.

**Theorem 2 (The toroidal $n$-queen existence problem [20])** *$TQ(n) = 0$ if and only if $2 \mid n$ or $3 \mid n$.*

**Theorem 3 (The complete mapping existence problem [18])** *A finite abelian group $G$ admits complete mappings if and only if its Sylow 2-subgroup is trivial or non-cyclic. This implies $CM(n) = 0$ if and only if $2 \nmid n$.*

**Theorem 4 ($n$-queen existence problem [25])** *There are solutions to the $n$-queen problem for all $n \geq 4$.*

### 2.2 The counting problems

The counting problems, on the other hand, are mostly open, due to the tremendous complexity. The best known result for the $n$-queen problem is $n = 23$. It was first solved by Pion and Fourre [1] (A000170) using bitwise instructions, depth-first-search and was done in a distributed environment. (The citation [1], a website with an impressive collection of number sequences, will be used extensively in the rest of this paper. The indicator A000170 refers to the problem number given in that URL.) The best TSQ indicated in [1] (A006717) is $TSQ(17)$, done by Wanless. In last year, we ([22]) extended it to $TSQ(23)$ (by successfully computing $CM(23) = \frac{TSQ(23)}{23}$). In order to obtain these results, we used *symmetry operators*, to be introduced in the next section, and partition strategies based on the symmetry operators to partition the search space. Engelhardt [1] (A007705) used symmetry operators and depth-first-search to compute $TQ(29)$ successfully. The followings table (Table 1) are the known sequences of $Q(n)$, $CM(n)$, $TSQ(n)$, $TQ(n)$, and $SCM(n)$.

## 3 Symmetry operators

A *symmetry operator* of the (toroidal)-(semi) $n$-queen problem or the (strong) complete mapping problem is a bijection $\Pi$ from the solution space to the solution space such that $f(x)$ is a solution if and only if $\Pi(f)(x)$ is also a solution. For example, $R(f) = f^{-1}$, and $R_{90^\circ}(f) = (-1-f)^{-1}$ are two symmetry operators for the $n$-queen problem. The set of symmetry operators can usually be defined through spanning a set of *basic* symmetry operators, and we call the set a *symmetry operator group*. For example, $span\{R, R_{90^\circ}\} =$

| $n$ | $Q(n)$ | $CM(n)$ | $TSQ(n)$ | $TQ(n)$ | $SCM(n)$ |
|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 1 | 3 | 0 | 0 |
| 4 | 2 | 0 | 0 | 0 | 0 |
| 5 | 10 | 3 | 15 | 10 | 2 |
| 6 | 4 | 0 | 0 | 0 | 0 |
| 7 | 40 | 19 | 133 | 28 | 4 |
| 8 | 92 | 0 | 0 | 0 | 0 |
| 9 | 352 | 225 | 2025 | 0 | 0 |
| 10 | 724 | 0 | 0 | 0 | 0 |
| 11 | 2680 | 3441 | 37851 | 88 | 8 |
| 12 | 14200 | 0 | 0 | 0 | 0 |
| 13 | 73712 | 79259 | 1030367 | 4524 | 348 |
| 14 | 365596 | 0 | 0 | 0 | 0 |
| 15 | 2279184 | 2424195 | 36362925 | 0 | 0 |
| 16 | 14772512 | 0 | 0 | 0 | 0 |
| 17 | 95815104 | 94471089 | 1606008513 | 140692 | 8276 |
| 18 | 666090624 | 0 | 0 | 0 | 0 |
| 19 | 4968057848 | 4613520889 | 87656896891 | 820496 | 43184 |
| 20 | 39029188884 | 0 | 0 | 0 | 0 |
| 21 | 314666222712 | 275148653115 | 5778121715415 | 0 | 0 |
| 22 | 2691008701644 | 0 | 0 | 0 | 0 |
| 23 | 24233937684440 | 19686730313955 | 452794797220965 | 128850048 | 5602176 |
| 24 | ? | 0 | 0 | 0 | 0 |
| 25 | ? | ? | ? | 1957725000 | 78309000 |
| 26 | ? | 0 | 0 | 0 | 0 |
| 27 | ? | ? | ? | 0 | 0 |
| 28 | ? | 0 | 0 | 0 | 0 |
| 29 | ? | ? | ? | 605917055356 | 20893691564 |
| 30 | ? | 0 | 0 | 0 | 0 |

Table 1: The known sequences of $Q(n), CM(n), TSQ(n)$, and $TQ(n)$.

$\{Identity, R_{90^\circ}, {R_{90^\circ}}^2, {R_{90^\circ}}^3, R, R_{90^\circ} \circ R, {R_{90^\circ}}^2 \circ R, {R_{90^\circ}}^3 \circ R\}$ is a symmetry operator group spanned from the basic operators $R$ and $R_{90^\circ}$. The symmetry operators generated from the basic operators are also called *composite operators*.

The above 8 symmetry operators are all known ones for the $n$-queen problem ([17, 6]). Singer (1960) [24] presented $6 \times n \times \phi(n)$ operators spanned by $\{R, A, H_\alpha, T_c\}$ for complete mappings. Later in this section we will introduce an operator $R_{45^\circ}$ for the toroidal $n$-queen problem and strong complete mapping problem that we believe is new.

A set of symmetry operators define a notion of equivalence classes. It can reduce the problem of looking for all solutions into one that looks only for the minimal one in each class. Huang [13] designed a *symmetry-cutting* strategy that utilizes symmetry operators to significantly reduce the search space in his propositional prover. We improved upon his method ([22]) and successfully applied it to the complete mapping counting problems. Without using the symmetry operators and the symmetry-cutting methods, we do not think that computing $TSQ(23)$ is possible. Here we list a summary of symmetry operators, and then describe them in the following propositions and theorems.

| Problem | Basic symmetry operators | Number of known operators |
|---|---|---|
| $Q(n)$ | $R, R_{90^\circ}$ | 8 |
| $CM(n)$ | $R, A, T_c, H_\alpha$ | $6 \times n \times \phi(n)$ |
| $TSQ(n)$ | $R, A, TS_{(c,d)}, H_\alpha$ | $6 \times n^2 \times \phi(n)$ |
| $SCM(n)$ | $R, R_{45^\circ}, T_c, H_\alpha$ | $8 \times n \times \phi(n)$ |
| $TQ(n)$ | $R, R_{45^\circ}, TS_{(c,d)}, H_\alpha$ | $8 \times n^2 \times \phi(n)$ |

**Proposition 5 ([17, 6])** *The following are basic symmetry operators for the $n$-queen problem:* reflection $R$ *and* rotation $R_{90^\circ}$.

1. $R(f)(x) = f^{-1}(x)$.
2. $R_{90^\circ}(f)(x) = (-1 - f)^{-1}(x)$.

*Sometimes we also represent the behavior of $R$ as $R(x, y) = (y, x)$, that is, $R(f)(x) = R(x, f(x)) = (f(x), x) = f^{-1}(x)$. So we rewrite them as follows.*

1. $R(x, y) = (y, x)$ *and* $R(f)(x) = f^{-1}(x)$
2. $R_{90^\circ}(x, y) = (-1 - y, x)$ *and* $R_{90^\circ}(f)(x) = (-1 - f)^{-1}(x)$

**Lemma 6 ([17, 6])** *The basic symmetry operators ($R$, and $R_{90^\circ}$) for the $n$-queen problem satisfy the following properties. These properties are parts of one proof of Theorem 7.*

1. $R^2 = {R_{90^\circ}}^4 = id$.
2. $R \circ R_{90^\circ} = {R_{90^\circ}}^{-1} \circ R$.

*Here id is the identity. That is, $id(f) = f$ for all $f \in Z_n^{Z_n}$.*

**Theorem 7 ([17, 6])** *There are 8 symmetry operators for n-queen problem spanned by R, and $R_{90^\circ}$.*

**Proposition 8 ([24, 8, 23])** *The following are basic symmetry operators for the complete mapping problem of the cyclic group $Z_n$:* reflection $R$, operator $A$, homologies $H_\alpha$, *and* translations $T_c$.

1. $R(x, y) = (y, x)$ *and* $R(f)(x) = f^{-1}(x)$.
2. $A(x, y) = (-x + y, -x)$ *and* $A(f)(x) = -(-id + f)^{-1}(x)$ *where id is the identity function* $id(x) = x$.
3. $H_\alpha(x, y) = (\alpha x, \alpha y)$ *and* $H_\alpha(f)(x) = \alpha f(\alpha^{-1}x)$ *where* $\alpha \in Z_n$ *and* $gcd(\alpha, n) = 1$.
4. $T_c(x, f(x)) = (x - c, f(x) - f(c))$ *and* $T_c(f)(x) = f(x + c) - f(c)$ *where* $c \in Z_n$.

**Lemma 9 ([8, 23])** *The basic symmetry operators (R, A, $H_\alpha$'s, and $T_c$'s) for the complete mapping problem have the following properties. These properties are parts of one proof of Theorem 10.*

1. $R^2 = A^3 = {T_1}^n = ID$.
2. $H_\alpha \circ H_\beta = H_{\alpha \cdot \beta}$.
3. $T_c \circ T_d = T_{c+d}$.
4. $R \circ A = A^2 \circ R$.
5. $R \circ H_\alpha = H_\alpha \circ R$.
6. $(R \circ T_c)(f) = (T_{f(c)} \circ R)(f)$.
7. $A \circ H_\alpha = H_\alpha \circ A$.
8. $(A \circ T_c)(f) = (T_{-c+f(c)} \circ A)(f)$.
9. $H_\alpha \circ T_c = T_{c \cdot \alpha} \circ H_\alpha$.

**Theorem 10 ([23])** *There are* $6 \times n \times \phi(n)$ *symmetry operators for complete mapping problem for cyclic group* $Z_n$ *spanned by* $R$, $A$, $H_\alpha$*'s, and* $T_c$*'s where* $\phi(n) = |\{x \mid 0 < x < n, gcd(x, n) = 1\}|$.

**Proposition 11** *The following are basic symmetry operators for the toroidal-semi n-queen problem:* reflection $R$, operator $A$, homologies $H_\alpha$ *and toroidal shiftings* $TS_{(c,d)}$.

1. $R$, $A$, *and* $H_\alpha$*'s are defined as complete mapping problem (Definition 8).*
2. $TS_{(c,d)}(x, y) = (x - c, y - d)$ *and* $TS_{(c,d)}(f)(x) = f(x + c) - d$ *where* $c, d \in Z_n$.

**Lemma 12** *The basic symmetry operators (*$R$, $A$, $H_\alpha$*'s and* $TS_{(c,d)}$*'s) for the toroidal-semi n-queen problem have the following properties. These properties are parts of one proof of Theorem 13.*

1. *Relations between* $R$, $A$, *and* $H_\alpha$*'s are the same as those in the complete mapping problem (Lemma 9).*
2. $TS_{(c,d)} \circ TS_{(a,b)} = TS_{(a+c,b+d)}$.
3. $R \circ TS_{(c,d)} = TS_{(d,c)} \circ R$.
4. $A \circ TS_{(c,d)} = TS_{(d-c,-c)} \circ A$.
5. $H_\alpha \circ TS_{(c,d)} = TS_{(\alpha c,\alpha d)} \circ H_\alpha$.

**Theorem 13** *There are* $6 \times n \times n \times \phi(n)$ *symmetry operators for the toroidal-semi n-queen problem spanned by* $R$, $A$, $H_\alpha$*'s and* $TS_{(c,d)}$*'s.*

Let $S_d(x, y) = (x, y - d)$. We note that $span\{T_c, S_d \mid c, d \in Z_n\} = span\{TS_{(c,d)} \mid c, d \in Z_n\}$ because the only difference between the toroidal-semi $n$-queen problem and the complete mapping problem is the constraint $f(0) = 0$.

**Proposition 14** *The following are basic symmetry operators for the strong complete mapping problem:* reflection $R$, translations $T_c$*'s,* homologies $H_\alpha$ *and* rotation $R_{45^\circ}$.

1. $R$, $H_\alpha$*'s, and* $T_c$*'s are as defined in the complete mapping problem (Proposition 8).*

2. $R_{45^\circ}(x,y) = (\frac{x-y}{\sqrt{2}}, \frac{x+y}{\sqrt{2}})$ *and* $R_{45^\circ}(f)(x) = \sqrt{2}^{-1}(id+f)(id-f)^{-1}(\sqrt{2}x)$.

**Lemma 15** *The basic symmetry operators (R, $R_{45^\circ}$, $H_\alpha$'s, and $T_c$'s) for the strong complete mapping problem have following properties. These properties are parts of one proof of Theorem 16.*

1. *Relations between R, $T_c$'s, and $H_\alpha$'s are the same as for complete mappings (Lemma 9).*
2. ${R_{45^\circ}}^8 = ID$.
3. $R \circ R_{45^\circ} = {R_{45^\circ}}^{-1} \circ R$.
4. $H_\alpha \circ R_{45^\circ} = R_{45^\circ} \circ H_\alpha$.
5. $(R_{45^\circ} \circ T_c)(f) = (T_{\frac{c-f(c)}{\sqrt{2}}} \circ R_{45^\circ})(f)$.

**Theorem 16** *There are $8 \times n \times \phi(n)$ symmetry operators for strong complete mapping problem spanned by R, $T_c$'s, $H_\alpha$'s and $R_{45^\circ}$.*

In the above statements, $\sqrt{2}$ is one solution of $x^2 = 2$ in $Z_n^*$. Since $\sqrt{2}$ does not exist for all $Z_n^*$, we can also use $R'_{45^\circ}(x,y) = (x-y, x+y)$ and $R'_{45^\circ}(f)(x) = (id+f) \circ (id-f)^{-1}(x)$ to replace $R_{45^\circ}$. It is easy to prove that $span\{R_{45^\circ}, H_\alpha \mid \alpha \in Z_n, gcd(\alpha, n) = 1\} = span\{R'_{45^\circ}, H_\alpha \mid \alpha \in Z_n, gcd(\alpha, n) = 1\}$ when $n$ is an odd number. It is also interesting to observe that the order of reflection $R$ is 2, the order of translation $T_1$ is $n$, the number of homologies $H_\alpha$ is $\phi(n)$, and the order of rotation $R_{45^\circ}$ is 8. Thus one may tend to think that there are $16 \times n \times \phi(n)$ symmetry operators. However, note that $R_{180^\circ}(x,y) = (R_{45^\circ})^4(x,y) = (-x,-y) = H_{-1}(x,y)$. So there are only $8 \times n \times \phi(n)$ different symmetry operators. We further note that in our discussion here $(R_{45^\circ})^2(x,y) = R_{45^\circ} \circ R_{45^\circ}(x,y) = (-y,x)$, which is different from the $R_{90^\circ}(x,y) = (-1-y,x)$ in the $n$-queen problem.

**Definition 5** *The following are basic symmetry operators for the toroidal n-queen problem:* reflection *R,* rotation $R_{45^\circ}$, homologies $H_\alpha$, *and* toroidal shifting $TS_{(c,d)}$.

1. *R, $R_{45^\circ}$, and $H_\alpha$'s are defined as in the strong complete mapping problem (Definition 14).*
2. *$TS_{(c,d)}$'s are defined as in the toroidal-semi n-queen problem (Proposition 11).*

**Lemma 17** *The basic symmetry operators ($R$, $R_{45^\circ}$, $H_\alpha$'s, and $TS_{(c,d)}$'s) for the toroidal $n$-queen problem have following properties. These properties are parts of one proof of Theorem 18.*

1. *Relations between $R$, $H_\alpha$'s, $R_{45^\circ}$ are the same as ones in the strong complete mapping problem (Lemma 15).*
2. *Relations between $R$, $H_\alpha$'s, $TS_{(c,d)}$'s are the same as ones in the toroidal-semi $n$-queen problem (Lemma 12).*
3. $TS_{(c,d)} \circ R_{45^\circ} = R_{45^\circ} \circ TS_{(\frac{-c-d}{\sqrt{2}}, \frac{-c+d}{\sqrt{2}})}$.

**Theorem 18** *There are $8 \times n \times n \times \phi(n)$ symmetry operators for the toroidal $n$-queen problem spanned by $R$, $R_{45^\circ}$, $H_\alpha$'s and $TS_{(c,d)}$'s.*

# 4 Challenge problems for propositional provers

Let $P$ be a symbol in $\{Q, TSQ, TQ, CM, SCM\}$, $\Pi$ be a symmetry operator, and $\Gamma$ be a symmetry operator group. We are interested in three types of questions.

1. The first one is to evaluate the number of solutions of the original problem. We use $P(n)$ to denote them. For example, $Q(n)$ is the number of solutions of the $n$-queen problem.
2. The second one is to evaluate the number of solutions $f(x)$'s which are fixpoints under the operator $\Pi$ ($\Pi(f) = f$). We use $P_\Pi(n)$ to denote them. For example, $Q_{R_{90^\circ}}(n)$ is the number of fixpoint solutions under $R_{90^\circ}$.
3. The last one is to evaluate the number of equivalence classes up to a symmetry operator group $\Gamma$. (Since $\Gamma$ defines an equivalence relation on solution space , we are interested in the number of equivalence classes.) We use $P^\Gamma(n)$ to denote them. For example, $Q^{span\{R, R_{90^\circ}\}}(n)$ is the number of equivalence classes of $n$-queen problem for the symmetry operator group $span\{R, R_{90^\circ}\}$.

**Definition 6** *We define the symbols for the fixpoints of the symmetry operators.*

1. $Q_\Pi(n) = |\{f \in \widehat{Q}(n) \mid \Pi(f) = f\}|$ *for* $\Pi \in span\{R, R_{90^\circ}\}$
2. $TQ_\Pi(n) = |\{f \in \widehat{TQ}(n) \mid \Pi(f) = f\}|$ *for* $\Pi \in span\{R, R_{45^\circ}, TS_{(c,d)}, H_\alpha\}$

3. $TSQ_{\Pi}(n) = |\{f \in \widehat{TSQ}(n) \mid \Pi(f) = f\}|$ *for* $\Pi \in span\{R, A, TS_{(c,d)}, H_{\alpha}\}$

4. $CM_{\Pi}(n) = |\{f \in \widehat{CM}(n) \mid \Pi(f) = f\}|$ *for* $\Pi \in span\{R, A, T_c, H_{\alpha}\}$

5. $SCM_{\Pi}(n) = |\{f \in \widehat{SCM}(n) \mid \Pi(f) = f\}|$ *for* $\Pi \in span\{R, R_{45^\circ}, T_c, H_{\alpha}\}$

We are interested in the fixpoints defined by the basic symmetry operators. Shieh et al. [23] (2000) gave a solution for $T_c$ by reducing $CM_{T_c}(n)$ and $SCM_{T_c}(n)$ to $CM(c)$ and $SCM(c)$, as shown in the following theorem.

**Theorem 19 ([23])** *We have* $CM_{T_c}(c \times a) = \phi_2(a) \times a^{c-1} \times CM(c)$ *and* $SCM_{T_c}(c \times a) = \phi_3(a) \times a^{c-1} \times SCM(c)$ *where* $\phi_2(a) = |\{x \in Z_a \mid (x, a) = (x-1, a) = 1\}|$ *and* $\phi_3(a) = |\{x \in Z_a \mid (x+1, a) = (x, a) = (x-1, a) = 1\}|$.

We calculate $CM_R(n)$, $CM_A(n)$, $CM_{H_{\alpha}}(n)$ and obtained the following results. For complete mappings fixed under reflection R ($CM_R$), Horton calls them starters and calculates $CM_R(n)$ for $n \leq 28$ in [1](A006204) and [9].

| $n$ | 7 | 9 | 11 | 13 | 15 | 17 |
|---|---|---|---|---|---|---|
| $CM_R(n)$ | 3 | 9 | 25 | 133 | 631 | 3857 |
| $n$ | 19 | 21 | 23 | 25 | 27 | 29 |
| $CM_R(n)$ | 25905 | 188181 | 1515283 | 13376125 | 128102625 | 1317606101 |

(Remark: $CM_A(n) = 0$ if $2 \mid n$ or $n \equiv 2 \pmod 3$.)

| $n$ | 1 | 7 | 13 | 19 | 25 | 31 | 37 | 43 | 49 |
|---|---|---|---|---|---|---|---|---|---|
| $\frac{CM_A(n)}{2^{\frac{n-1}{3}}}$ | 1 | 1 | 5 | 52 | 1055 | 31814 | 1403925 | 83999589 | 6567620752 |

| $n$ | 3 | 9 | 15 | 21 | 27 | 33 | 39 | 45 |
|---|---|---|---|---|---|---|---|---|
| $\frac{CM_A(n)}{2^{\frac{n-3}{3}}}$ | 1 | 0 | 3 | 30 | 513 | 15996 | 718404 | 43148682 |

(Remark: We note that $CM_{H_{-1}}(n) = CM_{R_{180^\circ}}(n)$.)

| $n$ | 3 | 5 | 7 | 9 | 11 | 13 | 15 | 17 | 19 | 21 |
|---|---|---|---|---|---|---|---|---|---|---|
| $CM_{R_{180^\circ}}(n)$ | 1 | 3 | 5 | 21 | 69 | 319 | 1957 | 12513 | 85445 | 656771 |

| $n$ | 23 | 25 | 27 | 29 | 31 |
|---|---|---|---|---|---|
| $CM_{R_{180^\circ}}(n)$ | 5591277 | 51531405 | 509874417 | 5438826975 | 62000480093 |

| $n$ | 33 | 35 | 37 |
|---|---|---|---|
| $CM_{R_{180^\circ}}(n)$ | 752464463029 | 9685138399785 | 131777883431119 |

Szabo [1](A032522)(A033148) produced $Q_{R_{180^\circ}}(n)$ for $n = 1, 2, 3, \cdots, 32$ and $Q_{R_{90^\circ}}(n)$ for $n = 1, 2, 3, \cdots, 47$. In the following we also list some new results of $Q_{R_{90^\circ}}(n)$. The details will be described in forthcoming papers. (Remark: $Q_{R_{90^\circ}}(n) = 0$ if $n \equiv 2, 3 \pmod 4$.)

| $n$ | 48 | 49 | 52 | 53 |
|---|---|---|---|---|
| $\frac{Q_{R_{90^\circ}}(n)}{2^{\lfloor \frac{n}{4} \rfloor}}$ | 5253278 | 8551800 | 49667373 | 79595269 |

| $n$ | 56 | 57 | 60 | 61 |
|---|---|---|---|---|
| $\frac{Q_{R_{90^\circ}}(n)}{2^{\lfloor \frac{n}{4} \rfloor}}$ | 525731268 | 764804085 | 5932910966 | 8905825760 |

**Definition 7 (equivalence classes counting problem)** *Let $P$ be the problem $Q, TQ, TSQ, CM$ or $SCM$, and let $\Gamma$ be a symmetry operator group.*

1. *$\Gamma$ define an equivalence relation $\sim$ on solution set: $f \sim g$ if and only if there exists a $\Pi \in \Gamma$ such that $f = \Pi(g)$.*

2. *We define $P^{\Gamma}(n)$ be the number of equivalence classes.*

For example, $Q^{span\{R,R_{90^\circ}\}}(n)$ is the number of equivalence classes of the $n$-queen problem up to symmetry operator group $span\{R, R_{90^\circ}\}$. For simplicity, we may drop the word "span", and simply use $Q^{\{R,R_{90^\circ}\}}(n)$ for $Q^{span\{R,R_{90^\circ}\}}(n)$.

Engelhardt computed $Q^{\{R,R_{90^\circ}\}}(23)$ [1](A002562) and $TQ^{\{R,R_{90^\circ},TS_{(c,d)}\}}(29)$ [1](A053994) successfully.

To summarize all the above, we suggest the following challenge problems to the propositional provers in the CADE community. They are good test problems for incorporating symmetry elimination, an often encountered technique, in propositional problem solving. In the following we give the problems as well as the best known results, which are presented in the boxes.

Problem 1 Compute $Q(n)$, $TQ(n)(= n \times SCM(n))$, and $TSQ(n)(= n \times CM(n))$.

| |
|---|
| $Q(23) = 24233937684440$ in [1] (A000170) |
| $TQ(29) = 605917055356$ in [1] (A007705) |
| $TSQ(17) = 1606008513$ in [1] (A006717) |
| $CM(23) = 19686730313955$ in [22] |

Problem 2 Compute $Q_{R_{90^\circ}}(n)$, and $Q_{R_{180^\circ}}(n)$.

| |
|---|
| $Q_{R_{90^\circ}}(45) = 1795233792$ in [1] (A033148) |
| $Q_{R_{90^\circ}}(61) = 291826098503680$ in this paper |
| $Q_{R_{180^\circ}}(32) = 181254386312$ in [1] (A032522) |

Problem 3 Compute $TQ_{R_{45^\circ}}(n)(= SCM_{R_{45^\circ}}(n))$,
$TQ_{R_{90^\circ}}(n)(= SCM_{R_{90^\circ}}(n))$, and $TQ_{R_{180^\circ}}(n)(= SCM_{R_{180^\circ}}(n))$.

Problem 4 Compute $TSQ_R(n)(= n \times CM_R(n))$, $CM_A(n)$, $TSQ_A(n)$ and
$TSQ_{R_{180^\circ}}(n)(= CM_{R_{180^\circ}}(n))$.

| |
|---|
| $CM_R(27) = 128102625$ in [1] (A006204) |
| $CM_R(29) = 1317606101$ in this paper |
| $CM_A(49) = 430415593603072$ in this paper |
| $CM_{R_{180^\circ}}(37) = 131777883431119$ in [22] |

Problem 5 Compute $Q^{\{R,R_{90^\circ}\}}(n)$.

| |
|---|
| $Q^{\{R,R_{90^\circ}\}}(23) = 3029242658210$ in [1] (A002562) |

Problem 6 Compute $TQ^{\{R,R_{90^\circ},TS_{(c,d)}\}}(n)$.

| |
|---|
| $TQ^{\{R,R_{90^\circ},TS_{(c,d)}\}}(29) = 90120677$ in [1] (A054500) |

Problem 7 Compute $TQ^{\{R,R_{45^\circ}\}}(n)$,
$TQ^{\{R,R_{45^\circ},TS_{(c,d)},H_\alpha\}}(n)(= SCM^{\{R,R_{45^\circ},T_c,H_\alpha\}}(n))$,
and $TSQ^{\{R,A,TS_{(c,d)},H_\alpha\}}(n)(= CM^{\{R,A,T_c,H_\alpha\}}(n))$.

## 5 Presenting the problems in propositional logic

In this section we give a translation of the problems proposed in Section 4 into propositional logic. We use the propositional symbols $x_{i,j}$ to denote positions. That is, $x_{i,j} = true$ if and only if the place (i,j) is occupied by a queen. Let $P$ be a symbol in $\{Q, TSQ, TQ, CM, SCM\}$, $\Pi$ be a symmetry

operator, and $\Gamma$ be a symmetry operator group. We use $\phi[P, n]$, $\phi[P_\Pi, n]$, and $\phi[P^\Gamma, n]$ to denote the propositional expressions corresponding to the counting problems $P(n)$, $P_\Pi(n)$, and $P^\Gamma(n)$, respectively. In other words, computing the numbers $P(n)$, $P_\Pi(n)$, and $P^\Gamma(n)$ is the same as computing the numbers of assignments that satisfy $\phi[P, n]$, $\phi[P_\Pi, n]$, and $\phi[P^\Gamma, n]$, respectively.

**Proposition 20** *The number of satisfiable assignments for the propositional expression $\phi[Q, n]$ is the same as $Q(n)$.*

1. $\phi[F, n] = \bigwedge_{i \in Z_n} (\bigvee_{j \in Z_n} x_{i,j})$.
2. $\phi[Horizontal, n] = \bigwedge_{i,j \in Z_n} (x_{i,j} \Rightarrow \bigwedge_{j \neq k \in Z_n} \neg x_{i,k})$.
3. $\phi[Vertical, n] = \bigwedge_{i,j \in Z_n} (x_{i,j} \Rightarrow \bigwedge_{i \neq k \in Z_n} \neg x_{k,j})$.
4. $\phi[Diagonal_1, n] = \bigwedge_{i \in Z_n} \bigwedge_{i+j, j \in Z_n}$
   $((x_{i+j,j} \Rightarrow \bigwedge_{j \neq k, i+k \in Z_n} \neg x_{i+k,k}) \bigwedge (x_{j,i+j} \Rightarrow \bigwedge_{j \neq k, i+k \in Z_n} \neg x_{k,i+k}))$.
5. $\phi[Diagonal_2, n] = \bigwedge_{i \in Z_n} \bigwedge_{i-j, j \in Z_n} ((x_{i-j,j} \Rightarrow \bigwedge_{j \neq k, i-k \in Z_n} \neg x_{i-k,k})$
   $\bigwedge (x_{n-1-j,i+j} \Rightarrow \bigwedge_{j \neq k, i+k \in Z_n} \neg x_{n-1-k,i+k}))$.
6. $\phi[Q, n] = \phi[F, n] \bigwedge \phi[Horizontal, n] \bigwedge \phi[Vertical, n]$
   $\bigwedge \phi[Diagonal_1, n] \bigwedge \phi[Diagonal_2, n]$.

**Example 1** *Take Q(3) as an example.*

| | |
|---|---|
| $\phi[Q, 3] =$ | |
| $(x_{0,0} \vee x_{0,1} \vee x_{0,2}) \wedge (x_{1,0} \vee x_{1,1} \vee x_{1,2}) \wedge (x_{2,0} \vee x_{2,1} \vee x_{2,2})$ | $\phi[F, 3]$ |
| $\wedge(\neg x_{0,0} \vee \neg x_{0,1}) \wedge (\neg x_{0,0} \vee \neg x_{0,2}) \wedge (\neg x_{0,1} \vee \neg x_{0,2})$ | $\phi[Horizontal, 3]$ |
| $\wedge(\neg x_{1,0} \vee \neg x_{1,1}) \wedge (\neg x_{1,0} \vee \neg x_{1,2}) \wedge (\neg x_{1,1} \vee \neg x_{1,2})$ | $\phi[Horizontal, 3]$ |
| $\wedge(\neg x_{2,0} \vee \neg x_{2,1}) \wedge (\neg x_{2,0} \vee \neg x_{2,2}) \wedge (\neg x_{2,1} \vee \neg x_{2,2})$ | $\phi[Horizontal, 3]$ |
| $\wedge(\neg x_{0,0} \vee \neg x_{1,0}) \wedge (\neg x_{0,0} \vee \neg x_{2,0}) \wedge (\neg x_{1,0} \vee \neg x_{2,0})$ | $\phi[Vertical, 3]$ |
| $\wedge(\neg x_{0,1} \vee \neg x_{1,1}) \wedge (\neg x_{0,1} \vee \neg x_{2,1}) \wedge (\neg x_{1,1} \vee \neg x_{2,1})$ | $\phi[Vertical, 3]$ |
| $\wedge(\neg x_{0,2} \vee \neg x_{1,2}) \wedge (\neg x_{0,2} \vee \neg x_{2,2}) \wedge (\neg x_{1,2} \vee \neg x_{2,2})$ | $\phi[Vertical, 3]$ |
| $\wedge(\neg x_{0,0} \vee \neg x_{1,1}) \wedge (\neg x_{0,0} \vee \neg x_{2,2}) \wedge (\neg x_{1,1} \vee \neg x_{2,2})$ | $\phi[Diagonal_1, 3]$ |
| $\wedge(\neg x_{0,1} \vee \neg x_{1,2}) \wedge (\neg x_{1,0} \vee \neg x_{2,1})$ | $\phi[Diagonal_1, 3]$ |
| $\wedge(\neg x_{0,2} \vee \neg x_{1,1}) \wedge (\neg x_{0,2} \vee \neg x_{2,0}) \wedge (\neg x_{1,1} \vee \neg x_{2,0})$ | $\phi[Diagonal_2, 3]$ |
| $\wedge(\neg x_{0,1} \vee \neg x_{1,0}) \wedge (\neg x_{1,2} \vee \neg x_{2,1})$ | $\phi[Diagonal_2, 3]$ |

The main difference between the transformations of $\phi[Q, n]$ and $\phi[TQ, n]$ or $\phi[TSQ, n]$ is the way the addition operator in the variable indices $i$ and $j$ is defined. For the $n$-queen problem the addition ia natural addition. For the others it is the cyclic group addition $(Z_n, +)$.

**Proposition 21** *The number of satisfiable assignments in the expressions* $\phi[TQ,n]$, $\phi[TSQ,n]$, $\phi[CM,n]$ *and* $\phi[SCM,n]$ *is the same as* $TQ(n)$, $TSQ(n)$, $CM(n)$, *and* $SCM(n)$, *respectively.*

1. $\phi[TDiagonal_1,n] = \bigwedge_{i\in Z_n}\bigwedge_{j\in Z_n}(x_{i+j,j} \Rightarrow \bigwedge_{j\neq k\in Z_n}\neg x_{i+k,k})$.
2. $\phi[TDiagonal_2,n] = \bigwedge_{i\in Z_n}\bigwedge_{j\in Z_n}(x_{i-j,j} \Rightarrow \bigwedge_{j\neq k\in Z_n}\neg x_{i-k,k})$.
3. $\phi[TQ,n] = \phi[F,n]\bigwedge\phi[Horizontal,n]\bigwedge\phi[Vertical,n]$ $\bigwedge\phi[TDiagonal_1,n]\bigwedge\phi[TDiagonal_2,n]$.
4. $\phi[TSQ,n] = \phi[F,n]\bigwedge\phi[Horizontal,n]\bigwedge\phi[Vertical,n]$ $\bigwedge\phi[TDiagonal_1,n]$.
5. $\phi[CM,n] = \phi[TSQ,n]\bigwedge x_{0,0}$.
6. $\phi[SCM,n] = \phi[TQ,n]\bigwedge x_{0,0}$.

The basic symmetry operators $R$, $R_{90^\circ}$, $R_{45^\circ}$, $TS_{c,d}$, $T_c$, $H_\alpha$, and $A$, can be regarded as functions from $Z_n \times Z_n$ to $Z_n \times Z_n$. (For example, $R(x,y) = (y,x)$).

We remark that the operators $T_c$'s depend on the solution $f(x)$ since $T_c(x,y) = (x-c, y-f(c))$. Thus they cannot be used directly in the formulation of the fixpoint sets that we are going to formulate. This problem can be circumvented by using Theorem19, which reduced the computation of $CM_{T_c}$ and $SCM_{T_c}$ to that of $CM(c)$ and $SCM(c)$, respectively.

We now give the transforation for the fixpoint counting problems.

**Proposition 22** *Let* $\Pi$ *denote a basic symmetry operator mentioned above other than* $\Pi \neq T_c$, *where* $c \in Z_n$. *Let* $P$ *be a symbol in* $\{Q, TSQ, TQ, CM, SCM\}$.

1. $\phi[f = \Pi(f), n] = \bigwedge_{i,j\in Z_n}(x_{i,j} \Leftrightarrow x_{\Pi(i,j)})$.
2. $\phi[f = T_c(f), n] = \bigwedge_{v\in Z_n}(x_{c,v} \Rightarrow (\bigwedge_{i,j\in Z_n}(x_{i,j} \Leftrightarrow x_{i-c,j-v})))$.
3. $\phi[P_\Pi, n] = \phi[P,n]\bigwedge\phi[f = \Pi(f), n]$.
4. $\phi[P_{T_c}, n] = \phi[P,n]\bigwedge\phi[f = T_c(f), n]$.

In order to solve the equivalence classes counting problems, we use a lexicographical ordering on the solutions: $f < g$ if $\exists_{k\in Z_n}((\forall_{j<k\in Z_n} f(j) = g(j)) \wedge f(k) < g(k))$. For each class, we choose the smallest solution to be the representative. Then counting the number of equivalence classes is the same as counting the number of the representatives. Now we show the transformation of the equivalence classes counting problems as $\phi[P^\Gamma, n]$ in the following.

**Proposition 23** *Let $P$ be a symbol in $\{Q, TSQ, TQ, CM, SCM\}$ and $\Gamma$ be a symmetry operator group. For any composite symmetry operator $\Pi$ (not including those defined using $T_c$), we have $\phi[f \leq \Pi(f), n]$, and finally we define $\phi[P^\Gamma, n]$.*

1. $\phi[f < \Pi(f), k, n] = (\bigwedge_{u<k\in Z_n} \bigwedge_{v\in Z_n} (x_{u,v} \Leftrightarrow x_{\Pi^{-1}(u,v)})) \bigwedge (\bigwedge_{i\in Z_n} (x_{k,i} \Rightarrow \bigvee_{i\leq j\in Z_n} x_{\Pi^{-1}(k,j)}))$.
2. $\phi[f < \Pi(f), n] = \bigvee_{k\in Z_n} \phi[f < \Pi(f), k, n]$.
3. $\phi[f \leq \Pi(f), n] = \phi[f < \Pi(f), n] \bigvee \phi[f = \Pi(f), n]$.
4. $\phi[P^\Gamma, n] = \phi[P, n] \bigwedge (\bigwedge_{\Pi\in\Gamma} \phi[f \leq \Pi(f), n])$.

# 6 Discussion

In this paper we proposed seven series of challenge problems for propositional provers. The series involve variations of the complete mapping problems and the $n$-queen problems. We also established, in this paper, the relationship between the complete mapping problems and the $n$-queen problems.

These proposed challenges are interesting in several aspects. First, they are *counting* problems, not existence problems. Thus a straightforward implementation of Davis-Putnam like procedures may not be sufficient. Second, they require sophisticated search techniques, such as symmetry elimination and partitioning, in order to trim the search space effectively. Third, the computational cost grows almost exponentially with every increment of $n$. Of the three better known series, $Q$, $CM$, and $TQ$, the current magic numbers are $Q(24)$, $CM(25)$, and $TQ(31)$.

In the following we provide some data of our experiments. For the $n$-queen problem, it took our program 20 days on a Pentium IV 1.8GHz PC to compute $Q(22)$ successfully. We estimate that we can get $Q(24)$ in 1266 days using the same machine. Another way of making estimations is via the number of computing cycles. We ran the program of Pion and Fourre on the same machine, and made the following comparisons. Note that for the $n$-queen problems ($Q$), our program uses about 1/10 of the cycles as theirs. The saving is even more significant for the complete mapping problems ($CM$) due to the symmetry elimination and partition strategies that we employed. (In our estimation the cycle-difference between our program and Pion/Fourre will increase from 100 times to 1000 when $n$ is 23.)

| Problem | Pion and Fourre [1](A000170) | Partition Strategies |
|---|---|---|
| $Q(16)$ | $4.7 \times 10^5$megacycles/1 min | $4.7 \times 10^4$megacycles/26 sec |
| $Q(18)$ | $5.7 \times 10^6$/ 53 min | $1.4 \times 10^6$/13 min |
| $Q(20)$ | $3.5 \times 10^8$/ 2 days | $5.2 \times 10^7$/8 hours |
| $Q(22)$ | —— | $3.2 \times 10^9$/20 days |
| $Q(24)$ | $1.9 \times 10^{12}$/12104 days(estimate) | $1.9 \times 10^{11}$/1266 days(estimate) |
| $CM(17)$ | $4.2 \times 10^6$/237 sec | $9.0 \times 10^3$/5 sec |
| $CM(19)$ | $2.3 \times 10^7$/220 min | $2.4 \times 10^5$/134 sec |
| $CM(21)$ | —— | $8.5 \times 10^6$/ 80 min |
| $CM(23)$ | —— | $2.8 \times 10^8$/ 43 hours |
| $CM(25)$ | —— | $9.8 \times 10^9$/ 63 days (estimate) |

A number of results reported in this paper are new. While the numbers have already been given elsewhere in the paper, we now give a table summarizing these results.

| Problem Set | Our results | Note |
|---|---|---|
| 1. $CM(n)$ | $n = 19 \sim 23$ | $CM(n) = 0$ for $n \equiv 0 \pmod 2$ |
| 2. $Q_{R_{90^\circ}}(n)$ | $n = 49 \sim 61$ | $Q_{R_{90^\circ}}(n) = 0$ for $n \equiv 2, 3 \pmod 4$ |
| 4. $CM_R(n)$ | $n = 29$ | $CM_R(n) = 0$ for $n \equiv 0 \pmod 2$ |
| 4. $CM_A(n)$ | $n = 1 \sim 49$ | $CM_A(n) = 0$ for $n \equiv 0, 2, 4, 5 \pmod 6$ |
| 4. $CM_{R_{180^\circ}}(n)$ | $n = 1 \sim 37$ | $CM_{R_{180^\circ}}(n) = 0$ for $n \equiv 0 \pmod 2$ |

As another table, we list the largest number that we achieved in each series, together with the computing time.

| Problem | Value | Megacycles | P4 1.8G |
|---|---|---|---|
| $CM(23)$ | 19686730313955 | $2.8 \times 10^8$ | 43 hours |
| $CM_R(29)$ | 1317606101 | $1.1 \times 10^8$ | 17 hours |
| $CM_A(49)$ | 6567620752 | $1.9 \times 10^8$ | 30 hours |
| $CM_{R_{180^\circ}}(37)$ | 131777883431119 | $1.5 \times 10^9$ | 10 days |
| $Q_{R_{90^\circ}}(61)$ | 291826098503680 | $2.1 \times 10^8$ | 33 hours |

The binary of our program is publically available at http://turing.csie.ntu.edu.tw/∽arping/cm, together with a program that transforms the problems indicated in this paper to propositional expressions so that other researchers can test their own ATP systems.

## References


[1] *http://www.research.att.com/∼njas/sequences/.*

[2] B. A. Anderson, *Sequencings and houses*, Congr. Numer., 43 (1984), pp. 23–43.

[3] Anonymous, *Unknown*, Berliner Schachgesellschaft, 3 (1848), p. 363.

[4] W. W. R. Ball., *Mathematical Recreations and Essays*, MacMillan and Co., 1926, p. 113.

[5] R. C. Bose, I. M. Chakravarti, and D. E. Knuth, *On methods of constructing sets of mutually orthogonal Latin squares using a computer I*, Technometrics, 2 (1960), pp. 507–516.

[6] P. Cull and R. Pandy, *Isomorphism and the n-queens problem*, SIGCSE Bulletin, 26, pp. 29–36.

[7] C. Erbas, S. Sarkeshik, and M. Tanik, *Different perspectives of the n-queens problem*, Proceedings of the 15th Anniversary of the ASME ETCE Conference, Computer Applications Symposium, (1992).

[8] A. B. Evans, *Orthomorphism graphs of groups*, vol. 1535 of Lecture Notes in Mathematics, Springer-Verlag, 1991.

[9] J. D. Horton, *Orthogonal starters in finite Abelian groups*, Discrete Math., 79 (1990), pp. 265–278.

[10] D. F. Hsu, *Cyclic neofields and combinatorial designs*, vol. 824 of Lecture Notes in Mathematics, Springer-Verlag, 1980.

[11] D. F. Hsu, *Orthomorphisms and near orthomorphisms*, in Graph Theory, Combinatorics, and Applications (Y. Alavi eds.), John Wiley and Sons, Inc., 1991, pp. 667–679.

[12] D. F. Hsu and A. D. Keedwell, *Generalized complete mappings, neofields, sequenceable groups and block designs II*, Pacific J. of Math., 117 (1985), pp. 291–312.

[13] G. S. Huang, *Search Reduction Techniques and Applications to Problems in Combinatorics*, PhD thesis, National Taiwan University, 1999.

[14] D. M. Johnson, A. L. Dulmage, and N. S. Mendelsohn, *Orthomorphisms of groups and orthogonal Latin squares I*, Canad. J. Math., 13 (1961), pp. 356–372.

[15] H. B. Mann, *The construction of orthogonal Latin squares*, Ann. Math. Statistics, 13 (1942), pp. 418–423.

[16] H. Niederreiter and K. H. Robinson, *Complete mappings of finite fields*, J. Austral. Math. Soc. Ser. A, 33 (1982), pp. 197–212.

[17] A. T. Olson., *The eight queens problem*, Journal of Computers in Mathematics and Science Teaching, 12, pp. 93–102.

[18] L. J. Paige, *A note on finite abelian groups*, Bull. Amer. Math. Soc., 53 (1947), pp. 590–593.

[19] ——, *Complete mappings of finite groups*, Pacific J. Math., 1 (1951), pp. 111–116.

[20] G. Pólya, *Uber die 'doppelt-periodischen' Losungen des n-Damen-Problems*, Mathematische Unterhaltungen und Spiele., (1918), pp. 364–374.

[21] I. Rivin, I. Vardi, and P. Zimmermann, *The n-queens problem*, vol. 101 of Amer. Math. Monthly, 1994, pp. 629–639.

[22] Y. Shieh, *Partition Strategies for #P-Complete Problem with Applications to Enumerative Combinatorics*, PhD thesis, National Taiwan University, 2001.

[23] Y. P. Shieh, J. Hsiang, and D. F. Hsu, *On the enumeration of abelian k-complete mappings*, vol. 144 of Congressus Numerantium, 2000, pp. 67–88.

[24] J. Singer, *A class of groups associated with Latin squares*, Amer. Math. Monthly, 67 (1960), pp. 235–240.

[25] A. Yaglom and I. Yaglom, *Challenging Mathematical Problems with Elementary Solutions*, Holden-Day, 1964, pp. 76–101.

# A Benchmark Template for Substructural Logics

John Slaney
Computer Sciences Laboratory
The Australian National University
Canberra, ACT, Australia
John.Slaney@anu.edu.au

June 4, 2002

**Abstract**

This paper suggests a benchmark template for non-classical propositional reasoning systems, based on work done some twenty years ago in the investigation of relevant logic. The idea is simple: generate non-equivalent binary operations in the language of the logic and use an automated reasoning system to decide which ones satisfy given algebraic properties. Of course, problem classes generated in this way are not in any sense uniformly distributed: indeed, they are highly structured and have special features such as a low ratio of variables to length. Nonetheless, they have the character of theorem proving "in the field", and should be part of the evaluation equipment for systems dealing with a wide range of nonclassical logics.

## 1 Substructural reasoning: the impossible takes a little longer

This paper is a response to an implicit challenge in the description of this workshop:

> 'As the use of ATP systems expands to harder problems and new domains, it is important to place new problems and problem types in public view...
>
> 'There will be no limitation to classical logics – problems and problem sets in other logics, e.g., modal and relevance logics, are of interest...'
>
> PaPS 2002 CFP

My first recommendation to the automated reasoning community, addressed especially to those of its members with interests in non-classical logics, is to work on automating inference in a family of substructural logics which includes

linear logic, the relevant logics, Lambek's associative calculus, affine logic and others. This is not merely a wish that more logicians and more system builders would share my enthusiasm for these logics. They are of independent interest for their applications (well-rehearsed in the case of linear logic, but also present in other cases—see [7] for an account and many entry points into the literature) and call for a community effort similar to that which has gone into SAT solving for classical logic, and more recently into modal and temporal logics.

Automated theorem proving for some such substructural systems is enormously hard [13, 5]—indeed, some of the *propositional* systems such as linear logic and the original Anderson-Belnap relevant logics are undecidable—and proof search in those logics is quite different from what is classically familiar, so directing some of the effort of system developers into the deduction and decision problems for these logics is itself a goal in line with the sentiment expressed above. I also believe that the step up in richness as we move from classical to substructural reasoning will lead us to devise new algorithms and heuristics which will ultimately feed back into the rest of what we do. Certainly, it leads us to think differently in some respects: while boolean 3-SAT problems with several hundred variables are routine fare for modern solvers, there are tough decision questions for formulae of relevant logic which fit on one line of a page and contain only 3 variables.

So the change of perspective is good for us, and anyway climbing fresh mountains is excellent exercise. However, the literature lacks good benchmarks for automated reasoning of this type. A major reason for this is that there is little uniformity across the canvassed range of non-classical systems: what is trivial given the proof mechanisms of one logic may be a severe test for those of another. The challenge raised by the Call for Papers, therefore, is to devise a benchmark template which can deliver worthwhile problem sets for a vast range of propositional logics, with various sets of connectives and no uniform features such as normal forms either for formulae or for inferences.

## 2 Logics

In order to sharpen the goal a little, it is necessary to delineate the class of logics for which the technique is intended to work. Therefore, let logic $\mathcal{L}$ be a formal system built up as follows. There is a denumerable set $\mathcal{P}$ of basic propositions (or "propositional variables") and a finite set $\mathcal{C}$ of finitary connectives. The basic propositions and nullary connectives ("constants") make up the set $\mathcal{A}$ of atoms. For $c \in \mathcal{C}$, let $A(c)$ be the adicity (or "arity") of $c$. The set $\mathcal{F}$ of formulae is defined as usual:

$$\mathcal{F} \;=\; \mathcal{P} \;\mid\; c(\mathcal{F}^{A(c)}),\; c \in \mathcal{C}$$

To sharpen still further, we are especially interested in a class of substructural logics in the vicinity of linear logic. Their connectives include the constants $\mathsf{t}$, $\mathsf{f}$, $\top$ and $\bot$, negation $\neg$, conjunction and disjunction $\wedge$ and $\vee$, their intensional counterparts fusion and fission $\circ$ and $+$, and implication operators $\rightarrow$ and $\leftarrow$. Other connectives such as modalities (the "exponentials" of linear logic) may

be added, as is done for the example in Section 6 below, but for the purposes of this introduction we omit them. So the language of these systems is

$$\mathcal{A} \;=\; \mathcal{P} \mid \mathsf{t} \mid \mathsf{f} \mid \top \mid \bot$$
$$\mathcal{F} \;=\; \mathcal{A} \mid \neg\mathcal{F} \mid \mathcal{F} \wedge \mathcal{F} \mid \mathcal{F} \vee \mathcal{F} \mid \mathcal{F} \circ \mathcal{F} \mid \mathcal{F} + \mathcal{F} \mid \mathcal{F} \rightarrow \mathcal{F} \mid \mathcal{F} \leftarrow \mathcal{F}$$

Next, since $\mathcal{L}$ is supposed to be a propositional *logic*, it should be a theory about what follows from what in virtue of properties of the connectives. Hence there is some notion of implication intended to capture the validity of inferences in $\mathcal{L}$. That is, there are structures of some kind built up out of the formulae—the structures may be simply sets as in classical and intuitionist logic, or multisets as in linear logic, or sequences as in the Lambek associative calculus, or more exotic objects such as trees with several different labelled branching operations, as in display logic for instance. Then there is a relation $\vdash$ between structures, satisfying some conditions related to deducibility—classically, enough conditions to make it a consequence relation in the Gentzen-Tarski sense, but in the general setting perhaps some weaker analogue of those.

In the particularly intended logics, the structures are binary trees. and the implication relation is that given in Appendix A. For linear logic, the trees can be "flattened" into multisets by adding structural rules of associativity and commutativity. One effect of this is identify the two implication connectives. For the relevant logic LR, add to linear logic the left and right rules of contraction:

$$\frac{\Gamma[X, X] \vdash \Delta}{\Gamma[X] \vdash \Delta} \qquad \frac{\Gamma \vdash \Delta[X, X]}{\Gamma \vdash \Delta[X]}$$

To convert linear logic into affine logic, add instead the weakening rules:

$$\frac{\Gamma[X] \vdash \Delta}{\Gamma[X, Y] \vdash \Delta} \qquad \frac{\Gamma \vdash \Delta[X]}{\Gamma \vdash \Delta[X, Y]}$$

This is not the place to lay down the law about what should or should not count as a logic. The weak conditions to be met by $\mathcal{L}$ are only that implication should be closed under uniform substitution of formulae for variables, and that it should give rise at least to the special case in which the structures related by $\vdash$ amount to single formulae: it should make sense to ask whether formula $\alpha$ implies formula $\beta$ in $\mathcal{L}$. The relation between $\alpha$ and $\beta$ such that $\alpha$ implies $\beta$ should be reflexive and transitive, and the corresponding equivalence relation (that between $\alpha$ and $\beta$ such that $\alpha$ implies $\beta$ and $\beta$ implies $\alpha$) should be a congruence on the formula algebra of $\mathcal{L}$: that is, provable equivalents should be replacable as subformulae in any context, preserving the relation $\vdash$.

These very weak conditions do not constrain $\mathcal{L}$ to be much like the standard logics in the literature, and certainly do not suffice to place it within the scope of familiar automated reasoning techniques. It is not stipulated that $\vdash$ should satisfy any form of the Cut rule beyond transitivity for single-formula structures, or an interpolation theorem. Its sequent calculus or tableaux formulations need

not be workable, therefore. It may also be inaccessible to translation methods, unless it has some reasonable semantics into which its sequents may be translated. The range of mountains which automated reasoners are invited to scale, therefore, is as varied and challenging as we could wish. It follows that the problem sets to be described below cannot be focussed on any specific proof technique or even family of techniques. Problem sets should be generated for any $\mathcal{L}$ in the class and for any approach to automating it, in such a way as to strike the middle ground of difficulty: it should be routine to find problems that are feasible without being trivial.

# 3 Benchmarks

Devising suitable benchmark problem sets for non-classical logics is far from easy. Even for classical logic, it is in some respects an unsolved problem: we know about random $k$-SAT, of course, but we are also familiar with the complaint that it is not a "typical" benchmark and that more meaningful measures result from real-world problems (arising in hardware verification, for example) or from hybrid randomised structured problems. In non-classical cases the situation is worse. There are three main techniques in the literature for generating benchmarks:

1. Encodings of meaningful problems may be used. These may come from standard application domains in software engineering; more often, they tend to be rather artificial puzzles, like the queens problem or blocks-world planning, presumably because these are easy to understand, trivial to code, and readily scalable. The disadvantage of these scalable "toy" problems is that they are typically too easy: a solution can be found in polynomial time, and enumerating all solutions is harder only because there are many of them. Hence the toys usually fail to force solvers to confront the serious difficulty of proof search and model search in the chosen logics, while the problems based on realistic applications do not scale well and do not migrate well from one substructural logic to another.

   Well-chosen real problems, however, are in the end the best of benchmarks, so there is no need to disparage them, provided they meet the three criteria of scalability, intrinsic hardness and adjustability to different logics.

2. Purely random problems may be generated, in the style of the random $k$-SAT models for classical SAT. In practice, experiments in this direction have mainly been confined to normal modal logics [3, 4, 6] and QSAT, where it is relatively easy to determine the parameters within which to randomise, though even in the normal modal case, it has proved far from trivial to devise suitable definitions. In more general settings, where formulae may not have suitable normal forms, for instance, it appears still less trivial. For instance, should logical theorems like $p \rightarrow p$ be allowed to occur as proper subformulae in the generated problems? It is in the spirit

of relevant or linear logic to say yes, but this may greatly affect difficulty, so there is no univocally indicated answer.

The great advantage of uniformly distributed random problems is that they permit statistically sophisticated experiments to be performed. Their main disadvantage, apart from the fact that for the logics in question they do not exist, is that the results obtained have to be interpreted *very* carefully in the light of the parameterisation decisions.

3. Another approach is to define particular sequences of problem instances, where the difficulty may be stepped up to whatever level constitutes a challenge for a particular solver just by increasing the number of variables. A classic example is the sequence of formulae in variables $p_1, \ldots, p_{2n-1}$:

$$\left( \bigwedge_{i<2n-1} ((p_i \leftrightarrow p_{i+1}) \rightarrow \bigwedge_{k<2n} p_k) \wedge ((p_{2n-1} \leftrightarrow p_1) \rightarrow \bigwedge_{k<2n} p_k) \right) \rightarrow \bigwedge_{k<2n} p_k$$

due to de Bruijn and commonly used to test provers for intuitionist logic.

Problem sequences of this sort obviously offer scalable tests, and may be designed to stress provers by emphasising certain features (e.g. their heuristics for choosing which of two disjuncts to expand first, or their dependence on a loop check). A drawback is that they are unsystematic: they have to be designed afresh for every logic of interest, and there is no good set of principles as to what features are worth emphasising by this means. Indeed, there is a danger that sequences could be designed to show off the good points of particular solvers, especially where there is a trick such as a special way of ordering rule applications or a preprocessing step that renders the problems in a sequence efficiently solvable.[1]

Perhaps the best and most careful attempt to provide good benchmarks for non-classical provers is represented by TANCS, the competition associated with TABLEAUX [14]. That uses a combination of random problem generation and special sequences designed to stress the features most commonly found to be critical to prover performance, in a system-neutral way. In TANCS-2000, the logics in which problem sets were provided were mainly PSPACE-hard modal logics, and the EXPTIME Converse PDL. It is especially good to see such a competition concentrating on logics with hard decision problems. The TANCS model might conceivably be adapted to the wide range of substructural logics noted in the present paper, but since the logics differ so widely and since their complexity is extreme, such an adaptation would present difficulties.

[1] The de Bruijn sequence is an example of a sequence that falls to a specific trick. It is a fairly trivial fact about intuitionist logic that in the minimal Kripke model falsifying a sequent, the succedent is *true* at all worlds except for the base one. Provers which exploit this fact can solve the de Bruijn problems very easily; those which do not typically find them much harder. But the presence or absence of *that* feature of a prover can be established by inspection of the code and so hardly warrants a benchmark.

# 4 Historical Interlude: The case of LR

Another way forward is suggested by some old work on relevant logic, carried out mainly by Thistlewaite in collaboration with McRobbie and Meyer in the early 1980s [11]. That work did not arise out of nothing. It was motivated by an attempt to solve a long-standing open problem: the decision problem for the Anderson-Belnap relevant logic R [1, 2]. R is a propositional logic which differs from the additive-multiplicative fragment of linear logic by validating both contraction (noted above) and the distribution of extensional ("additive") conjunction and disjunction over each other:

$$A \wedge (B \vee C) \vdash (A \wedge B) \vee (A \wedge C)$$

The main reasons for wanting distribution to hold were firstly that it captures a common reasoning form which does not seem to be any sort of a "fallacy of relevance", and secondly that it is semantically natural. However, it complicates the logic a good deal proof-theoretically, to the point that the decision procedures which are available for LR ("Lattice R", or R without distributivity) cannot be adapted to decide R itself.[2]

Meyer's plan at the time was to prove R *undecidable* by showing that the word problem for semigroups could be coded into it. That required that the logic contain a free associative operation, somehow definable in terms of the connectives. "Free" here means that it does not satisfy any algebraic laws other than those which follow from its associativity. Thus, for instance, while all of the primitive connectives $\wedge$, $\vee$, $\circ$ and $+$ are indeed associative, they are also all commutative, which rules out their freedom. The idea was to find candidate binary operations, such as

$$f(A, B) \;=\; (A + A) \circ (\neg B \vee (A \circ B))$$

prove them associative in R, and pass them through a sanity check to ensure that they do not fail freedom in any straightforward way. Any operation that passed all the tests could be investigated in detail, with a view to showing it to be really a free associative operation, thus establishing that R must be undecidable. In the event, Meyer's plan did not succeed, for two reasons: first, all the operations found to be associative in R (such as $f$ above) were also proved to be associative in LR, which showed they were not *free* associative since LR is decidable; second, the undecidability of R was shown independently, by Urquhart [12], before the project was completed.

However, although the plan failed, it did generate interesting outcomes in the form of a theorem prover KRIPKE based on analytic tableaux (or in another

[2] This is a convenient opportunity to commend the logic R (as opposed to LR) as a subject for research in automation. The undecidability of the propositional logic seems to have discouraged work on it in the past, but obviously we should not regard undecidability as an insuperable obstacle—after all, it does not inhibit work on first order logic. For first order theories, indeed, undecidability is a virtue: a logic that can represent the halting problem for Turing machines is undecidable, while one which cannot is a pretty crummy logic.

view on the sequent calculus) and highly optimised for LR, and in the form of a series of hard problems for it to solve. In the case of the above operation $f$, for example, KRIPKE had to prove the two sequents:

$$f(p, f(q, r)) \vdash f(f(p, q), r)$$
$$f(f(p, q), r) \vdash f(p, f(q, r))$$

It is reported in [11] that the first of these was proved in around 5 minutes (probably on a Sun 370 or similar hardware), the final proof tree after compression by collapsing repeated subproofs having 97 nodes. The second could not be decided by KRIPKE at the time of [11]. Altogether, 16 interesting candidate formulae were investigated intensively using KRIPKE. The resulting 32 sequents became known as ASSET (Asociativity Set) and KRIPKE was developed and refined using ASSET as a testbed. On the hardware of the day, it proved 14 of the 32 within a time limit of one hour per proof search, leaving the other 18 undecided. Later work by Riche [8] resulted in automatically generated proofs of 10 more of the 18. All 32 were given human-generated proofs by McRobbie using KRIPKE as an assistant. These sequents, then, constituted a stern test for relevant logic theorem provers at least as late as 1991, and should still be addressed by any system whose performance on such logics is claimed to be acceptable.

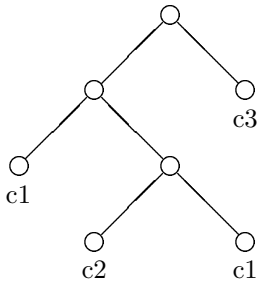
# 5 The General Case: Logical Groupoids

Provided $\mathcal{L}$ has at least one connective of arity greater than 1, it allows binary operations to be defined. In classical propositional logic, there are, up to equivalence, only 16 such operations. Similar bounds apply to other finitely many-valued logics. In weaker logics like LR or affine logic or any of their subsystems, however, there are infinitely many. Every binary operation defined by a formula in two variables gives rise to a groupoid on the formula algebra of $\mathcal{L}$ (and on any other algebra modelling $\mathcal{L}$) and the question arises of what algebraic properties each such operation satisfies, or equivalently what its logic is. That is, at the simplest, what inferences $\alpha \vdash \beta$ are valid in $\mathcal{L}$ where $\alpha$ and $\beta$ are built out of atoms using only the operation in question. The Modest Proposal of the present paper is that we mine this inexhaustible seam for suitably hard problems in every non-classical propositional logic of interest meeting the weak conditions outlined above.

Let $A(p, q)$ be a binary formula of logic $\mathcal{L}$ (i.e. a formula in two variables $p$ and $q$) and let $\tau$ be a (finite) binary tree whose terminal nodes are coloured from a set $\{\mathsf{c}_1, \ldots, \mathsf{c}_k\}$ of colours. Then the formula $\mathcal{F}^A_\tau$ is defined recursively:

1. Where $\tau$ is a single node $n$ coloured with $\mathsf{c}_i$, $\mathcal{F}^A_\tau$ is the atom $p_i$.
2. Where $\tau$ is a tree with root node $n$ and nonempty left and right subtrees $\tau_1$ and $\tau_2$, $\mathcal{F}^A_\tau$ is $A(\mathcal{F}^A_{\tau_1}, \mathcal{F}^A_{\tau_2})$. That is, it is $A$ with the two subformulae $\mathcal{F}^A_{\tau_1}$ and $\mathcal{F}^A_{\tau_2}$ substituted uniformly for the variables $p$ and $q$ respectively.

For example, let $A(p, q)$ be the formula $(p \circ q) \vee \neg p$, and let $\tau$ be the tree




Then $\mathcal{F}_\tau^A$ is the formula

$$(((p1 \circ ((p_2 \circ p_1) \vee \neg p_2)) \vee \neg p_1) \circ p_3) \vee \neg((p1 \circ ((p_2 \circ p_1) \vee \neg p_2)) \vee \neg p_1)$$

The general scheme for devising problem sets is as follows:

1. Select two (finite) binary trees $\tau_1$ and $\tau_2$.
2. Begin enumerating the formulae in two variables (say, $p$ and $q$). For each formula $A$ in the enumeration, problem $\langle \tau_1, \tau_2, A \rangle$ is to decide the $\mathcal{L}$ validity of the sequent $\mathcal{F}_{\tau_1}^A \vdash \mathcal{F}_{\tau_2}^A$.
3. Stop enumerating when the problems become too hard for the provers under test.

A nice refinement of the procedure is to omit problems $\langle \tau_1, \tau_2, A \rangle$ where $A$ is equivalent (in $\mathcal{L}$) to some formula $B$ which occurs earlier in the enumeration. This requires decisions as to equivalence or non-equivalence at the generating stage, giving rise to more $\mathcal{L}$ problems, though relatively easy ones since the formula $A$ is (in the interesting cases) much shorter than $\mathcal{F}_\tau^A$. Of course, equivalent formulae need not give rise to equally hard problems for theorem provers, but removal of "duplicates" is a convenient way of thinning out the list of formulae without omitting anything essential.

A useful technique for speeding up equivalence testing is to partition the formulae into semantically based equivalence classes according to their behaviour (their "truth tables") in one or more small algebraic models of $\mathcal{L}$. For example, the following two 3-element algebras are both model structures for linear logic. The constants $\top$ and $\bot$ are interpreted as values 2 and 0 respectively, and both algebras have the same matrices for $\neg$, $\wedge$ and $\vee$:

| $A$ | $\neg A$ |
|---|---|
| 0 | 2 |
| 1 | 1 |
| 2 | 0 |

| $\wedge$ | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 2 | 0 | 1 | 2 |

| $\vee$ | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 0 | 1 | 2 |
| 1 | 1 | 1 | 2 |
| 2 | 2 | 2 | 2 |

The difference between the two, RM3 and Ł3, concern the intensional constants $\mathsf{t}$ and $\mathsf{f}$, and the intensional connectives $\circ$ and $+$:

RM3:

| $\circ$ | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 2 |
| 2 | 0 | 2 | 2 |

$\mathsf{t} = 1$

$\mathsf{f} = 1$

| $+$ | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 0 | 0 | 2 |
| 1 | 0 | 1 | 2 |
| 2 | 2 | 2 | 2 |

Ł3:

| $\circ$ | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 |
| 2 | 0 | 1 | 2 |

$\mathsf{t} = 2$

$\mathsf{f} = 0$

| $+$ | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 0 | 1 | 2 |
| 1 | 1 | 2 | 2 |
| 2 | 2 | 2 | 2 |

$A$ and $B$ cannot be equivalent in linear logic unless they have exactly the same truth tables in both RM3 and Ł3, so if they differ on that simple semantic test there is no need to search for a proof of equivalence. Across a wide range of substructural logics, such small models are readily available via tools such as MaGIC [9] and render the equivalence test in the generating phase rather trivial in comparison with the benchmark itself.

A recommended method of using the benchmark template with provers for a particular logic is to experiment with different choices of $\tau$ until some are found which result in problems of a "reasonable" hardness reasonably quickly, and then to generate binary operations in levels, saturating each level (number of connective occurrences) before proceeding to the next. It is then possible to stop at the level that begins to stress the prover(s) in hand, and to pick out the hardest problems from that level to form a fairly coherent test set. The procedure, though not of course the problems, will be uniform across logics and provers.

# 6 An example

By way of illustration, the following results were obtained by running the linear logic solver provided with the Logics Workbench [15] on two sets of problems for propositional linear logic.[3] The first problem was to decide the associativity of binary operations definable by formulae in negation normal form containing (apart from sentential constants and negation in literals) exactly three connective occurrences. For example, the formula

$$?((a \circ \top) + b)$$

defines such an operation, and to decide whether it is associative is to prove or disprove the two sequents

$$?((?((p_0 \circ \top) + p_1) \circ \top) + p_2) \quad \vdash \quad ?((p_0 \circ \top) + ?((p_1 \circ \top) + p_2)) \qquad (1)$$

$$?((p_0 \circ \top) + ?((p_1 \circ \top) + p_2)) \quad \vdash \quad ?((?((p_0 \circ \top) + p_1) \circ \top) + p_2) \qquad (2)$$

[3] All times were obtained on a Sparc Ultra 250.

Clearly, the sequents generated in this way are not particularly long, and in fact most of them can be decided by the LWB solver in less than 0.01 seconds. Some, however, are surprisingly difficult.

In order to set the prover a slightly greater challenge, the experiment was re-run with the same set of binary operations but with the equation

$$(a \star b) \star (a \star c) = (a \star c) \star (b \star c)$$

in place of associativity. The sequents still have only three variables, but the operation being tested occurs more often and with more nesting than before.

The language of linear logic has the connectives and constants noted above, except that because of commutativity, the two arrows collapse into one, which anyway is defined away in terms of $\circ$ and negation, and the usual linear logic modalities (or "exponentials") $!$ and $?$ are added. For this note, I have transcribed the perverse linear logic notation into the more generally familiar one used here (see Appendix B for details). Formulae were built up from atoms (variables and constants) by applying the connectives $\wedge$, $\vee$, $\circ$, $+$, $\neg$, $!$ and $?$ in any order, up to a limit of three connective occurrences. The actual problems were generated automatically, and each direction of each equivalence formulated as a sequent like the two above. Certain formulae were eliminated from the list at generation time as being trivially equivalent to simpler ones: those containing subformulae of the form $A \wedge A$ or $A \vee A$, and those containing subformulae which could be simplified by absorption laws for constants ($A \wedge \top$, $A \wedge \bot$, $A \vee \top$, $A \vee \bot$, $A \circ \mathsf{t}$, $A \circ \bot$, $A + \mathsf{f}$, $A + \top$ and their converses which are equivalent to them by commutativity). Each problem was converted by the LWB solver to negation normal form and then to "commutative-associative normal form" before solution.

Altogether, 666 binary operations were generated, giving rise to 13032 sequents in each problem set, each sequent with just 3 variables of course. The first problem set, that of proving or disproving associativity, was attempted twice, first with the default "Cdnr" setting of 3 and then with it set to 4. This setting controls the maximum depth of the proof search in terms of applications of the (left side) contraction-dereliction rule:

$$\frac{A,\ ?A,\ \Gamma}{?A,\ \Gamma}$$

whose uncontrolled operation could make proof searches infinite (linear logic being undecidable). With the Cdnr setting of 3, the solver proved 3656 of the 13032 sequents, disproved 7989, and failed to reach a decision on the other 1387. With Cdnr set to 4, just 16 of the previously undecided sequents were proved; the number of disproved ones stayed the same. Thus even in this very simple problem set, around 10% of problems cannot be solved by the LWB prover with a search depth of less than 5. Running the solver with a search depth greater than 4 takes much longer—at least on the order of hours for this problem. Hence, despite the apparent simplicity of the formulae, a nontrivial proportion

of them are genuinely hard for this prover.[4]

The times taken (with the higher Cdnr setting) are also interesting. For almost every sequent that was proved, a proof was found in 0.02 seconds or less: only 4 required more, and the hardest was proved in 1.05 seconds. In other words, they are all easy problems with this Cdnr setting. Some of the disproved sequents took longer, however: 79 of them took 0.1 seconds or more, and 3 of them more than a second. The sequents that were neither proved nor disproved, however, were much harder. 121 of the "fail" results took more than a second, and 16 of them more than 10 seconds. The hardest of all was the example given above, which caused the prover to run for 95 seconds before terminating in failure to reach a decision.

For reference, here are 12 three-connective binary operations whose associativity in linear logic is surprisingly hard for the LWB prover to decide:

1. $?((a \circ b) + \mathsf{t})$
2. $?a + (b \vee \mathsf{t})$
3. $?(a + b) \circ \top$
4. $!a \circ (b \wedge \mathsf{f})$
5. $(a \circ a) + ?b$
6. $?a + (b \circ \mathsf{f})$
7. $?((a \circ b) + \bot)$
8. $?a \wedge (b \circ \top)$
9. $(a \circ b) + ?b$
10. $(?a \vee b) + a$
11. $(!a \wedge b) \circ a$
12. $?((a \circ \top) + b)$

The results on the second problem set were quite similar overall, except that the demonstrations of unprovability were as hard as the failures to reach a decision. The prover was run with the default Cdnr setting of 3 only, since to run with higher settings would have taken too long for the experiment. 2508 of the 13032 sequents were proved, 8540 disproved, leaving 1984 undecided without allowing deeper proof searches. Where proofs exist, they seem to be easy to find: only 3 cases required more than 0.1 seconds. However, 139 of the disproofs took more than a second, 16 of them requiring more than 10 seconds and the hardest

[4]The numbers have to be taken rather loosely, since they represent "raw" data. More intensive cleaning-up would have deleted what were essentially duplicate results, where the normal-forming process has produced trivially equivalent problems—in some cases even alphabetic variants. As an illustration of the problem generation method, however, the broad features of the example are indicative enough.

37 seconds. 111 of the "fails" took over a second, and 24 over 10 seconds, the hardest taking 59 seconds. Undoubtedly, the times would have been much greater had a Cdnr setting of 4 been used.

Finally, here are 13 binary operations $\star$ for which the question of whether $(x \star y) \star (x \star z)$ is equivalent in linear logic to $(x \star z) \star (y \star z)$ is particularly challenging for LWB:

1. $(!a \wedge a) \circ b$
2. $(!a + \bot) \circ b$
3. $(a \vee b) + ?a$
4. $(a \wedge b) \circ !a$
5. $?(a + b) \vee \mathsf{f}$
6. $?(a + b) \vee a$
7. $!(a \circ b) \wedge a$
8. $?(a + b) \vee \mathsf{t}$
9. $?a + (b \circ \top)$
10. $!(a \circ b) \wedge \mathsf{t}$
11. $?(a + b) \circ \top$
12. $!(a \circ b) \wedge \mathsf{f}$
13. $?a + (b \circ b)$

# 7 Conclusion

The scheme presented in this note is of course not the only approach to benchmarking provers for non-classical logics, whether for evaluating rival automated systems or for software development purposes. Probably no all-purpose benchmark for such a wide range of logics as those considered here will ever be useful. Nonetheless, as noted, the "logical groupoid" template adjusts itself quite smoothly to very different logics and very different proof methods. It is moreover easy to implement and reproduce, from a very compact description, and largely impervious to "tinkering" to tilt the results towards one system rather than another. It is hard to see what greater virtues such a scheme could have.

# A Sequent Calculus rules for the main target substructural logics

The following rules are for the positive (negation-free) logics. Negative particles may be introduced in a number of ways: as a boolean operation, by inferrential definition as in minimal logic, as a primitive operation on atoms only, as is often done in linear logic, or in other ways. Not all of them are easy to treat in the proof-theoretic style of this note, so they are omitted from the present account.

Bunches of formulae on the left and right of '$\vdash$' are to be read as binary trees, written for the purposes of stating the rules in a one-dimensional form with '$X, Y$' representing the tree with principal subtrees $X$ and $Y$. A formula is a one-node tree. We write '$\Gamma(X)$' to represent a tree with $X$ in some distinguished position as a subtree, and '$\Gamma(Y)$' for the result of replacing that occurrence of $X$ by $Y$. We formulate the logics first without sentential constants, beginning with the axiom scheme:

$$A \vdash A$$

The logical rules for connectives are as follows:

$$\frac{\Gamma(A) \vdash X}{\Gamma(A \wedge B) \vdash X}(\wedge\vdash) \qquad \frac{\Gamma(B) \vdash X}{\Gamma(A \wedge B) \vdash X}(\wedge\vdash) \qquad \frac{X \vdash \Gamma(A) \qquad X \vdash \Gamma(B)}{X \vdash \Gamma(A \wedge B)}(\vdash\wedge)$$

$$\frac{\Gamma(A) \vdash X \qquad \Gamma(B) \vdash X}{\Gamma(A \vee B) \vdash X}(\vee\vdash) \qquad \frac{X \vdash \Gamma(A)}{X \vdash \Gamma(A \vee B)}(\vdash\vee) \qquad \frac{X \vdash \Gamma(B)}{X \vdash \Gamma(A \vee B)}(\vdash\vee)$$

$$\frac{\Gamma(A, B) \vdash X}{\Gamma(A \circ B) \vdash X}(\circ\vdash) \qquad \frac{X \vdash \Gamma(A) \qquad Y \vdash \Gamma(B)}{X, Y \vdash \Gamma(A \circ B)}(\vdash\circ)$$

$$\frac{\Gamma(A) \vdash X \qquad \Gamma(B) \vdash Y}{\Gamma(A+B) \vdash X, Y}(+\vdash) \qquad \frac{X \vdash \Gamma(A, B)}{X \vdash \Gamma(A+B)}(\vdash+)$$

$$\frac{X \vdash A \qquad \Gamma(B) \vdash Y}{\Gamma(A{\to}B, X) \vdash Y}(\to\vdash) \qquad \frac{X, A \vdash B}{X \vdash A{\to}B}(\vdash\to)$$

$$\frac{X \vdash A \qquad \Gamma(B) \vdash Y}{\Gamma(X, B{\leftarrow}A) \vdash Y}(\leftarrow\vdash) \qquad \frac{A, X \vdash B}{X \vdash B{\leftarrow}A}(\vdash\leftarrow)$$

# B Linear logic notation

The following is the "translation manual" for linear logic. There are three versions: the operators used by Girard and other linear logicians (LL), those used by everyone else (EE), and those required as ASCII input by the Logics Workbench (LWB). The mapping between them is as follows:

| LL | EE | LWB |
|---|---|---|
| $\top$ | $\top$ | `top` |
| $\bot$ | `f` | `bot` |
| 1 | `t` | `1` |
| 0 | $\bot$ | `0` |
| & | $\wedge$ | `&` |
| $\otimes$ | $\circ$ | `X` |
| $\parr$ | $\vee$ | `\|` |
| $\oplus$ | $+$ | `+` |
| $\multimap$ | $\rightarrow$ | `--o` |
| $\circ\!\!-\!\!\circ$ | $\leftrightarrow$ | `o--o` |

# References


[1] A. Anderson and N. Belnap. *Entailment: The Logic of Relevance and Necessity* Vol. 1. Princeton University Press, 1975.

[2] J. M. Dunn. Relevance Logic and Entailment. In D. Gabbay, and F. Günthner (eds), *Handbook of Philosophical Logic*, Vol 3. Dordrecht, Reidel, 1986.

[3] F. Giunchiglia and R. Sebastiani. Building Decision Procedures for Modal Logics from Propositional Decision Procedures—The Case Study of Modal K$^*$. *Proceedings of CADE-13* (1996): 583–597.

[4] U. Hustadt and R. Schmidt. On Evaluating Decision Procedures for Modal Logics. *Proceedings of IJCAI-15* (1997): 202–207.

[5] P. Lincoln. Deciding Provability of Linear Logic Formulas. In Girard, Lafont, and Regnier (eds), *Advances in Linear Logic*. Cambridge University Press, 1995: 109-122.

[6] P. Patel-Schneider and R¿ Sebastiani. A New System and Methodology for Generating Random Modal Formulae. *Proceedings of IJCAR* (2001): 464–468.

[7] G. Restall. *An Introduction to Substructural Logics*. Routledge, 2000.

[8] J. Riche. *Decidability, Complexity and Automated Reasoning in Relevant Logic*. PhD thesis, Australian National University, Canberra 1991.

[9] J. Slaney. MaGIC, matrix generator for implication connectives: Notes and guide. Technical report TR-ARP-11-95, Automated Reasoning Project, Australian National University, 1995.

[10] G. Sutcliffe and C. Suttner. The TPTP Problem Library: The CNF Release v.1.2.1. *Journal of Automated Reasoning* 21 (1998) 177-203.

[11] P. Thistlewaite, M. McRobbie and R. Meyer. *Automated Theorem-Proving in Non-Classical Logics*. London, Pitman, 1988.

[12] A. Urquhart. The Undecidability of Entailment and Relevant Logic. *Journal of Symbolic Logic* 49 (1984): 1059–1073.

[13] A. Urquhart. The Complexity of Decision Procedures in Relevance Logic. In Dunn and Gupta (eds), *Truth or Consequences, Essays in Honour of Nuel Belnap*. Dordrecht, Kluwer, 1990: 61–76.

[14] TANCS: The Tableaux Non Classical (Modal) Sysyems Comparison. `http://www.dis.uniroma1.it/~tancs/`

[15] The Logics Workbench. `http://www.lwb.unibe.ch/`

# More First-order Test Problems in Math and Logic*

*Z. Ernst*, Department of Philosophy, University of Wisconsin
*B. Fitelson*, Department of Philosophy, Stanford University
*K. Harris*, Department of Computer Science, University of Chicago
*W. McCune*, MCS Division, Argonne National Laboratory
*R. Padmanabhan*, Department of Mathematics, University of Manitoba
*R. Veroff*, Department of Computer Science, University of New Mexico
*L. Wos*, MCS Division, Argonne National Laboratory

May 28, 2002

## 1 Introduction

This paper contains a collection of theorems, nontheorems, and conjectures in first-order and equational logic. These problems arose in our work on applications of automated deduction to mathematics and logic. Some originated in our work, and others were sent to us as challenge problems or open questions.

There is no unifying theme to the mathematics and logic in the problems. It is simply a set of challenging problems that might be useful in testing and developing theorem provers and related programs.

We have OTTER [9] proofs (available on request) for all of the problems listed as "theorem". That is not to say we have a general strategy that causes OTTER to find the proofs. In many cases very specialized strategies, including lots of domain knowledge from the users, were used to find the proofs.

We have finite countermodels for all of the problems listed as "nontheorem". These are not as difficult as the theorems; in fact, most can be proved automatically by SEM[24] or MACE's [11] successor.

These problems are available on the Web at `www.mcs.anl.gov/~mccune/papers/paps-2002`. Let us know if you solve any of the open ones!

## 2 Problems

### 2.1 Condensed Detachment Problems

**xcb-reflex** [23, 3] Prove reflexivity from formula XCB by condensed detachment. This was long thought to be a nontheorem and was first proved by Fitelson. Status: theorem (unsatisfiable).

```
-P(e(x,y)) | -P(x) | P(y).       % condensed detachment
P(e(x,e(e(e(x,y),e(z,y)),z))).   % XCB
-P(e(A,A)).                      % denial of reflexivity
```

---

**xcb** [23, 22] Show that formula XCB is a single axiom for the equivalential calculus by deriving the well known single axiom WN by condensed detachment. This was the last remaining open candidate of length 11. It was first proved to be a single axiom by Wos in April 2002. Status: theorem (unsatisfiable).

*This work was Supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, U.S. Department of Energy, under Contract W-31-109-Eng-38.

```
-P(e(x,y)) | -P(x) | P(y).       % condensed detachment
P(e(x,e(e(e(x,y),e(z,y)),z))).   % XCB
-P(e(e(A,e(B,C)),e(C,e(A,B)))).  % denial of single axiom WN
```

---

**yqe** [8, 16] Is formula YQE a single axiom for the right group calculus? Any countermodels must be infinite. Status: open.

```
-P(e(x,y)) | -P(x) | P(y).                 % condensed detachment
P(e(e(x,y),e(e(x,z),e(y,z)))).             % YQE
-P(e(A,e(A,e(e(B,C),e(e(B,D),e(C,D)))))).  % denial of single axiom L2'
```

---

**mv-dist-star** [4, 18] From the axioms MV1–MV4 (many-valued sentential calculus), derive the distributivity properties AK1, AK2, KA1, and KA2. This is four separate problems. Status: all are theorems (unsatisfiable).

```
-P(i(x,y)) | -P(x) | P(y).      % condensed detachment

P(i(x,i(y,x))).                 % MV1
P(i(i(x,y),i(i(y,z),i(x,z)))).  % MV2
P(i(i(i(x,y),y),i(i(y,x),x))).  % MV3
P(i(i(n(x),n(y)),i(y,x))).      % MV4

-P(i(i(i(A,n(i(i(n(B),n(C)),n(C)))),n(i(i(n(B),n(C)),n(C)))),n(i(i(n(i(i(A,B),B)),
                n(i(i(A,C),C))),n(i(i(A,C),C)))))).  % AK1
-P(i(n(i(i(n(i(i(A,B),B)),n(i(i(A,C),C))),n(i(i(A,C),C)))),i(i(A,n(i(i(n(B),n(C)),
                n(C)))),n(i(i(n(B),n(C)),n(C)))))).  % AK2
-P(i(n(i(i(n(A),n(i(i(B,C),C))),n(i(i(B,C),C)))),i(i(n(i(i(n(A),n(B)),n(B))),
                n(i(i(n(A),n(C)),n(C)))),n(i(i(n(A),n(C)),n(C)))))).  % KA1
-P(i(i(i(n(i(i(n(A),n(B)),n(B))),n(i(i(n(A),n(C)),n(C)))),n(i(i(n(A),n(C)),
                n(C)))),n(i(i(n(A),n(i(i(B,C),C))),n(i(i(B,C),C)))))).  % KA2
```

---

**twoval-luka-23** [6] Show that the formula Luka-23 is a single axiom for two-valued logic by deriving the Łukasiewicz 3-basis. Status: theorem (unsatisfiable).

```
-P(i(x,y)) | -P(x) | P(y).          % condensed detachment
P(i(i(i(x,y),i(i(i(n(z),n(u)),v),z)),i(w,i(i(z,x),i(u,x))))).  % Luka-23

% denial of Lukasiewicz 3-basis
-P(i(i(A,B),i(i(B,C),i(A,C)))) | -P(i(i(n(A),A),A)) | -P(i(A,i(n(A),B))).
```

---

**twoval-mer-21** [21] Show that the formula Mer-21 is a single axiom for two-valued logic by deriving the Łukasiewicz 3-basis. Status: theorem (unsatisfiable).

```
-P(i(x,y)) | -P(x) | P(y).          % condensed detachment
P(i(i(i(i(i(x,y),i(n(z),n(u))),z),v),i(i(v,x),i(u,x)))).  % Mer-21

% denial of Lukasiewicz 3-basis
-P(i(i(A,B),i(i(B,C),i(A,C)))) | -P(i(i(n(A),A),A)) | -P(i(A,i(n(A),B))).
```

---

**sheffer-org-d23** [5] Show that formula ORG-D23 is a single axiom for propositional calculus in terms of the Sheffer stroke by deriving Nicod's single axiom. The nice property of ORG-D23 is that it is organic; that is, no subformula is a theorem. Status: theorem (unsatisfiable).

```
-P(d(x,d(y,z))) | -P(x) | P(z).  % detachment rule for the Sheffer stroke
P(d(d(x,d(y,z)),d(d(x,d(y,z)),d(d(u,z),d(d(z,u),d(x,u)))))).  % ORG-D23

% denial of Nicod's original single axiom
-P(d(d(A,d(B,C)),d(d(E,d(E,E)),d(d(F,B),d(d(A,F),d(A,F)))))).
```

---

**mingle-bci** [1] Show that if the mingle formula is added to the logic BCI, the Karpenko formula can be derived by condensed detachment. Status: theorem (unsatisfiable).

```
-P(i(x,y)) | -P(x) | P(y).          % condensed detachment
P(i(i(x,y),i(i(z,x),i(z,y)))).      % B
P(i(i(x,i(y,z)),i(y,i(x,z)))).      % C
P(i(i(i(i(i(x,y),y),x),z),i(i(i(i(i(y,x),x),y),z),z))).  % mingle

% denial of Karpenko formula
-P(i(i(A,i(i(B,B),A)),i(i(i(A,B),B),i(i(B,A),A)))).
```

---

**mingle-concise** [2] Show that the mingle axiom can be derived from the three formulas given below by condensed detachment. This gives a simpler basis for the system RM→. Status: theorem (unsatisfiable).

```
-P(i(x,y)) | -P(x) | P(y).           % condensed detachment
P(i(i(x,y),i(i(y,z),i(x,z)))).      % suffixing
P(i(x,i(i(x,y),y))).                % assertion
P(i(i(i(i(i(x,y),z),i(y,x)),z),z)).  % candidate

% denial of mingle axiom
-P(i(i(i(i(i(A,B),B),A),C),i(i(i(i(i(B,A),A),B),C),C))).
```

---

**intuit-imp** [17] Show that the candidate formula is not a single axiom for intuitionistic implication by finding a model in which the required property below fails. Status: nontheorem (satisfiable).

```
-P(i(x,y)) | -P(x) | P(y).               % condensed detachment
P(i(i(x,y),i(i(y,i(i(z,x),u)),i(x,u)))).  % candidate
-P(i(A,i(B,A))).                        % required property
```

---

## 2.2 Equational Bases for Boolean Algebra

**ba-dn1** [14] Show that equation DN-1 is a single axiom for Boolean algebra in terms of disjunction and negation by deriving the Huntington 3-basis. Status: theorem (unsatisfiable).

```
n(n(n(x + y) + z) + n(x + n(n(z) + n(z + u)))) = z.    % DN-1

% denial of Huntington 3-basis
B + A != A + B | (A + B) + C != A + (B + C) | n(n(A) + B) + n(n(A) + n(B)) != A.
```

---

**sheffer-sh1** [14, 20] Show that equation Sh-1 is a single axiom for Boolean algebra in terms of the Sheffer stroke by deriving the Meredith 2-basis. Status: theorem (unsatisfiable).

```
f(f(x,f(f(y,x),x)),f(y,f(z,x))) = y.  % Sh-1

% denial of Meredith 2-basis
f(f(A,A),f(B,A)) != A | f(A,f(B,f(A,C))) != f(f(f(C,B),B),A).
```

---

**sheffer-mstar** [14] Show that each of the equations below is too weak to be a single axiom for Boolean algebra in terms of the Sheffer stroke. This is 7 separate problems. Status: all are nontheorems (satisfiable).

```
f(f(y,f(f(x,z),y)),f(x,f(z,y))) = x.   % M5A
f(f(f(y,f(x,z)),y),f(x,f(z,y))) = x.   % M5B
f(f(f(y,f(x,y)),y),f(x,f(y,z))) = x.   % M6A
f(f(y,f(y,f(x,y))),f(x,f(y,z))) = x.   % M6B
f(f(f(y,f(x,x)),y),f(x,f(y,z))) = x.   % M6C
f(f(y,f(y,f(x,x))),f(x,f(y,z))) = x.   % M6D
f(f(f(y,f(z,x)),y),f(x,f(z,y))) = x.   % M8A

% denial of Meredith 2-basis
f(f(A,A),f(B,A)) != A | f(A,f(B,f(A,C))) != f(f(f(C,B),B),A).
```

---

**sheffer-cstar** [14, 20] Which of equations C1–C16 are single axioms for Boolean algebra in terms of the Sheffer stroke? There are 16 problems here: take each of the candidates and pair it with the denial. Status: all are open.

```
f(f(y,f(f(x,y),y)),f(x,f(y,z))) = x.  % C1
f(f(y,f(y,f(x,y))),f(x,f(z,y))) = x.  % C2
f(f(y,f(y,f(y,x))),f(x,f(z,y))) = x.  % C3
f(f(y,f(y,f(y,x))),f(x,f(y,z))) = x.  % C4
f(f(y,f(y,f(x,z))),f(x,f(z,y))) = x.  % C5
f(f(y,f(y,f(z,x))),f(x,f(y,z))) = x.  % C6
f(f(y,f(y,f(x,x))),f(x,f(z,y))) = x.  % C7
f(f(f(y,f(y,x)),y),f(x,f(z,y))) = x.  % C8
f(f(f(y,f(x,x)),y),f(x,f(z,y))) = x.  % C9
f(f(f(y,f(x,z)),y),f(x,f(y,z))) = x.  % C10
f(f(f(y,f(z,x)),y),f(x,f(y,z))) = x.  % C11
f(f(f(y,f(y,x)),y),f(x,f(y,z))) = x.  % C12
f(f(f(f(y,x),y),y),f(x,f(z,y))) = x.  % C13
f(f(f(f(y,x),y),y),f(x,f(y,z))) = x.  % C14
f(f(f(f(y,x),z),z),f(x,f(y,z))) = x.  % C15
f(f(f(f(y,x),z),z),f(x,f(z,y))) = x.  % C16

% denial of Meredith 2-basis
f(f(A,A),f(B,A)) != A | f(A,f(B,f(A,C))) != f(f(f(C,B),B),A).
```

---

## 2.3 Equational Lattice Theory

**ol-e51** [10] Show that equation E51 does not necessarily hold in ortholattices. Status: nontheorem (satisfiable).

```
x ^ y = y ^ x.              % lattice axioms
(x ^ y) ^ z = x ^ (y ^ z).
```

```
x ^ (x v y) = x.
x v y = y v x.
(x v y) v z = x v (y v z).
x v (x ^ y) = x.

c(x) ^ x = Zero.            % add these for ortholattice
c(x) v x = One.
x ^ y = c(c(x) v c(y)).

% denial of E51
((A v c(B))^ (((A^B) v (c(A)^B)) v (c(A)^c(B)))) != ((A^B) v (c(A)^c(B))).
```

---

**ol-e62** [10] Show that equation E62 does not necessarily hold in ortholattices. Status: nontheorem (satisfiable).

```
x ^ y = y ^ x.              % lattice axioms
(x ^ y) ^ z = x ^ (y ^ z).
x ^ (x v y) = x.
x v y = y v x.
(x v y) v z = x v (y v z).
x v (x ^ y) = x.

c(x) ^ x = Zero.            % add these for ortholattice
c(x) v x = One.
x ^ y = c(c(x) v c(y)).

% denial of E62
A ^ (B v (A ^ (c(A) v (A ^ B)))) != A ^ (c(A) v (A ^ B)).
```

---

**ol-rw1** [15] Show that equation *3-68 does not necessarily hold in weak orthomodular lattices. Status: nontheorem (satisfiable).

```
x ^ y = y ^ x.              % lattice axioms
(x ^ y) ^ z = x ^ (y ^ z).
x ^ (x v y) = x.
x v y = y v x.
(x v y) v z = x v (y v z).
x v (x ^ y) = x.

c(x) ^ x = Zero.            % add these for ortholattice
c(x) v x = One.
x ^ y = c(c(x) v c(y)).

(c(x) ^ (x v y)) v (c(y) v (x ^ y)) = One.  % weak orthomodular law

A ^ (B v (A ^ (c(A) v (A ^ B)))) != A ^ (c(A) v (A ^ B)).  % denial of *3-68
```

---

**ol-rw2** [15] Show that equation 98A does not necessarily hold in ortholattices. Status: nontheorem (satisfiable).

```
x ^ y = y ^ x.              % lattice axioms
(x ^ y) ^ z = x ^ (y ^ z).
x ^ (x v y) = x.
```

```
x v y = y v x.
(x v y) v z = x v (y v z).
x v (x ^ y) = x.

c(x) ^ x = Zero.            % add these for ortholattice
c(x) v x = One.
x ^ y = c(c(x) v c(y)).

% denial of equation 98A.
A v (c(B) ^ (c(A) v (c(B) ^ (A v (c(B) ^ c(A)))))) !=
A v (c(B) ^ (c(A) v (c(B) ^ (A v (c(B) ^ (c(A) v (c(B) ^ A)))))).
```

---

**oml-mod** [15] Show that orthomodular lattices are not necessarily modular. This is well known, but it is a good test problem for finite model search. Status: nontheorem (satisfiable).

```
x ^ y = y ^ x.              % lattice axioms
(x ^ y) ^ z = x ^ (y ^ z).
x ^ (x v y) = x.
x v y = y v x.
(x v y) v z = x v (y v z).
x v (x ^ y) = x.

c(x) ^ x = Zero.            % add these for ortholattice
c(x) v x = One.
x ^ y = c(c(x) v c(y)).

x v (c(x) ^ (x v y)) = x v y.  % orthomodular law (OM)

A v (B ^ (A v C)) != (A v B) ^ (A v C).  % denial of modularity
```

---

**lattice-uc** [13] Consider uniquely complemented (UC) lattices. We are looking for weak properties that force them to be Boolean. Distributivity is well known to do the job, so we use it as our goal (denial). There are six problems here. For the open ones, all counterexamples are infinite, because all finite uniquely complemented lattices are Boolean. Status: three are theorems, three are open.

```
x ^ y = y ^ x.                          % lattice axioms
(x ^ y) ^ z = x ^ (y ^ z).
x ^ (x v y) = x.
x v y = y v x.
(x v y) v z = x v (y v z).
x v (x ^ y) = x.

x v c(x) = One.                         % complementation
x ^ c(x) = Zero.

x v y != One | x ^ y != Zero | c(x) = y.  % complements are unique

A ^ (B v C) != (A ^ B) v (A ^ C).       % denial of distributivity

% Take each of the following 6 with the preceding clauses.

% Each of these three gives a theorem.
x ^ (y v (z ^ (x v (y ^ z)))) = x ^ (y v (x ^ z)).          % 94-6
```

```
x ^ ((y ^ (z v (x ^ y))) v (z ^ (x v y))) = (x ^ y) v (x ^ z).  % 94-37
x ^ (y v (z v (u ^ (x v (y ^ z))))) = x ^ (y v (z v (x ^ u))).  % G61
% Each of these three is open.
x ^ (y v ((x v y) ^ (z v (y ^ (x v z))))) = x ^ (y v z).        % 94-3
x ^ (y v (z ^ (u v (x ^ (y v z))))) = x ^ (y v (z ^ (x v u))).  % F53
x ^ (y v (z ^ ((x ^ z) v (u ^ (y v z))))) = x ^ (y v ((x ^ z) v (z ^ u))). % G113
```

---

## 2.4 Miscellaneous Equality Problems

**cs-comm-ad** [7, 13] Assume a cancellative semigroup (CS) admits a commutator operation. Then the following three properties are equivalent: (1) commutator is associative; (2) commutator distributes over product; (3) the semigroup is nilpotent of class 2. This is a generalization of the corresponding theorem for group theory. The problem here is to prove (1) implies (2). Status: theorem (unsatisfiable).

```
(x * y) * z = x * (y * z).        % product is associative
x * y != x * z | y = z.           % left cancellation
y * x != z * x | y = z.           % right cancellation

x * y = y * (x * (x @ y)).        % CS admits commutator
(x @ y) @ z = x @ (y @ z).        % commutator is associative
(a * b) @ c != (a @ c) * (b @ c). % denial: commutator distributes over product
```

---

**cs-comm-dn** [7, 13] See the description of problem cs-comm-ad. The problem here is to prove that the distributivity property (2) implies the nilpotent property (3). Status: theorem (unsatisfiable).

```
(x * y) * z = x * (y * z).        % product is associative
x * y != x * z | y = z.           % left cancellation
y * x != z * x | y = z.           % right cancellation

x * y = y * (x * (x @ y)).        % CS admits commutator
(x * y) @ z = (x @ z) * (y @ z).  % commutator distributes over product
(a @ b) * c != c * (a @ b).       % denial: nilpotent class 2
```

---

**cs-comm-na** [7, 13] See the description of problem cs-comm-ad. The problem here is to prove that the nilpotent property (3) implies the associativity property (1). Status: theorem (unsatisfiable).

```
(x * y) * z = x * (y * z).        % product is associative
x * y != x * z | y = z.           % left cancellation
y * x != z * x | y = z.           % right cancellation

x * y = y * (x * (x @ y)).        % CS admits commutator
(x @ y) * z = z * (x @ y).        % nilpotent class 2
(a @ b) @ c != a @ (b @ c).       % denial: commutator is associative
```

---

**hbck** [12, 19] Axioms for the quasivariety HBCK are given below. Show that equation J follows. This result has been known for some time by a model-theoretic argument. The first first-order proof was found by Veroff in 2002. Status: theorem (unsatisfiable).

```
x * One = One.                    % M3
One * x = x.                      % M4
(x * y) * ((z * x) * (z * y)) = One.      % M5
x * y != One | y * x != One | x = y.      % M7
```

```
x * x = One.                                % M8
x * (y * z) = y * (x * z).                  % M9
(x * y) * (x * z) = (y * x) * (y * z).      % H

(((A * B) * B) * A) * A != (((B * A) * A) * B) * B.  % denial of J
```

---

# References


[1] Z. Ernst. Completions of $TV_{\rightarrow}$ from $H_{\rightarrow}$. *Bulletin of the Section of Logic*, 2002. To appear.

[2] Z. Ernst, B. Fitelson, K. Harris, and L. Wos. A concise axiomatization of $RM_{\rightarrow}$. *Bulletin of the Section of Logic*, 30(4):191–194, 2001.

[3] B. Fitelson, 1998. E-mail to L. Wos.

[4] B. Fitelson and K. Harris. Distributivity in $Ł_{\aleph_0}$ and other sentential logics. *J. Automated Reasoning*, 27(2):141–156, 2001.

[5] B. Fitelson and K. Harris. On propositional Sheffer axioms. Unpublished, 2001.

[6] B. Fitelson and L. Wos. Missing proofs found. *J. Automated Reasoning*, 27(2):201–225, 2001.

[7] A. G. Kurosh. *The Theory of Groups*, volume 1. Chelsea, New York, 1955.

[8] W. McCune. Automated discovery of new axiomatizations of the left group and right group calculi. *J. Automated Reasoning*, 9(1):1–24, 1992.

[9] W. McCune. Otter 3.0 Reference Manual and Guide. Tech. Report ANL-94/6, Argonne National Laboratory, Argonne, IL, 1994.

[10] W. McCune, January 1999. E-mail to N. Megill.

[11] W. McCune. MACE 2.0 Reference Manual and Guide. Tech. Memo ANL/MCS-TM-249, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL, June 2001.

[12] W. McCune and R. Padmanabhan. *Automated Deduction in Equational Logic and Cubic Curves*, volume 1095 of *Lecture Notes in Computer Science (AI subseries)*. Springer-Verlag, Berlin, 1996.

[13] W. McCune, R. Padmanabhan, and R. Veroff. Notes from automated reasoning workshop. Unpublished, April 2002.

[14] W. McCune, R. Veroff, B. Fitelson, K. Harris, A. Feist, and L. Wos. Short single axioms for Boolean algebra. Preprint MCS-P848-1000, Argonne National Laboratory, Argonne, IL, 2000. To appear in J. Automated Reasoning.

[15] M. Rose and K. Wilkinson. Application of model search to lattice theory. *Association for Automated Reasoning Newsletter*, 52, August 2001.

[16] D. Ulrich, 2001. E-mail to L. Wos.

[17] D. Ulrich, 2001. E-mail to B. Fitelson.

[18] R. Veroff. On distributivity in many-valued logic. Unpublished, 2001.

[19] R. Veroff. A first-order proof of the HBCK theorem. Unpublished, 2002.

[20] S. Wolfram, February 2000. E-mail to L. Wos and W. McCune.

[21] L. Wos. Conquering the Meredith single axiom. *J. Automated Reasoning*, 27(2):175–179, 2001.

[22] L. Wos, April 2002. E-mail to W. McCune.

[23] L. Wos, S. Winker, R. Veroff, B. Smith, and L. Henschen. Questions concerning possible shortest single axioms in equivalential calculus: An application of automated theorem proving to infinite domains. *Notre Dame J. Formal Logic*, 24(2):205–223, 1983.

[24] J. Zhang and H. Zhang. SEM: A system for enumerating models. In *Proc. IJCAI-95*, volume 1, pages 298–303. Morgan Kaufmann, 1995.

# Verification of Hardware Systems with First-Order Logic

Koen Claessen[1,2], Reiner Hähnle[1,2], and Johan Mårtensson[1]

[1] Safelogic AB
Stena Center 1C
S-412 92 Gothenburg
{koen|reiner|johan}@safelogic.se

[2] Chalmers University of Technology
Department of Computing Science
S-412 96 Gothenburg
{koen|reiner}@cs.chalmers.se

**Abstract.** The state of the art of automatic first order logic theorem provers is advanced enough to be useful in a commercial context. This paper describes a way in which first order logic and theorem provers are used at the Swedish formal verification company Safelogic, to formally verify properties of hardware systems. Two different verification methods are discussed, which both make use of translations of formalisms into first order logic. We draw some preliminary conclusions from our experiences and provide problems sets and benchmarks.

## 1 Introduction

A common way in which descriptions of hardware systems have been verified in the last decade, has been to translate finite-state instances of the system into a finite state machine, and to verify properties about this state machine by, for example, computing the reachable state space using Binary Decision Diagrams (BDDs) [1].

More recently, an alternative method has become popular, which translates the finite state machine into propositional logic, and uses a propositional theorem prover, or SAT solver, to verify properties [2,3]. Some form of time-level reasoning, such as temporal induction has to be added in this case. Certain properties that are out of reach for BDDs can then be solved by SAT solvers.

At Safelogic we decided to go one step further and to employ first order (FO) theorem proving for verification of hardware systems. In the following section we briefly motivate and discuss this decision. Then we proceed to describe two different methods we used for generating first order verification conditions from hardware designs and their requirements. This is done in Section 3 and 4, respectively. We discuss some domain-specific "tricks" we did to make the generated problems easier to prove. In Section 5 we present a FIFO circuit, as an example that is simple, yet incorporates many of the challenges that come up

during verification of hardware systems occurring at our industrial partners and customers. This is followed by a brief concluding section. The first order logic problems discussed in this paper as well as some experimental results can be found at `http://www.safelogic.se/problems/PaPS`.

## 2 Why First Order Logic?

At first glance, the decision to use first order theorem proving for verification of hardware designs may seem non-obvious. After all, hardware systems have a finite number of states. It seems overkill and unnecessarily inefficient to use an undecidable logic. In the following we argue that FO theorem proving has a number of important advantages that can easily outweigh the drawbacks.

*Lemmas* One advantage of using theorem proving methods over, for example, state machine building methods is that it is possible to add lemmas. Lemmas can be suggested by a human user or a lemma-finding heuristic to make a problem easier. Lemmas can also be used automatically to perform hierarchical verification.

*Compact Proof Objects* Verification with theorem proving may result in a compact proof that in itself is a certificate of correctness. In contrast to this, SAT solvers usually don't yield proof objects at all while BDD-based model checkers often yield very large structures.

*Proofs for Generic Designs* State machine building and propositional theorem proving have the common limitation that only instances of a system description can be verified. In general, a hardware description can be generic in input sizes, number of registers, etc. However, in order to verify properties, one is forced to choose a particular combination of such parameters. On the other hand, in FO logic it is possible to prove properties for generic designs.

*State Space vs. Search Space* Verification methods based on state space exploration often face the problem of state space explosion. In this case theorem proving may offer a workable alternative. Although there are complexity issues also with theorem proving, they mostly have little to do with the size of the set of reachable states. We do not represent the state space other than indirectly. In FO theorem proving compact proofs often do exist. The challenge is to find these proofs in an infinite set of possible derivations. State space explosion and complexity due to large proof search spaces are in many cases orthogonal causes of infeasibility, hence, theorem proving and state space exploration can be regarded as complementary methods of verification.

*Independence of Tools and Problems* One drawback of BDD-based methods is that algorithms and data structures usually have to be carefully matched against the requirements property language at hand. Modern property languages (such

as [5], but also the language sketched below) tend to be rich and complex. It is a non-trivial problem to design efficient model checkers for them. The translation approach to FO logic is inherently more flexible and modular. New features in the property language can be implemented and evaluated quickly. SAT solvers also offer flexibility, but the inexpressiveness of propositional logic imposes severe restrictions and leads to blow-up during translation.

*Future Research Potential* First order theorem proving tools are now approaching a state of maturity comparable to SAT solvers and model checkers. One has to concede, though, that FO theorem proving is no "pushbutton technology" (which is not strictly true of model checking either): tuning is of critical importance and expert knowledge is required to get out useful results from FO theorem proving engines. Much less research has been spent in the FO theorem proving community on problem generation and compilation of domain-specific knowledge than on calculi and their implementation. (Possibly, the vision from early AI days of theorem provers as general problem solvers still lingers on.) Our experiences suggest that considerable gains are to be had, if FO theorem proving experts and verification domain experts work closely together.

*Limitations* Theorem proving is an affirmation of FO logical consequence, because theorem provers enumerate the set of such consequences. It is well known that the satisfiable FO formulas are not recursively enumerable, so there is no general complete procedure for establishing non-consequence. Worse, even truth in the intended system models cannot be effectively determined, because usually there are no complete axiomatizable FO theories characterizing them [6]. Certain sublanguages of FO logic are decidable [7], but unfortunately the decidable subclasses do not pertain to the problems considered here.

Certain temporal properties are not provable in FO logic, but in practice the addition of suitable induction schemata goes a long way. This, however, decreases the degree of automation. In Section 5 below we argue why even FO logic without induction is quite useful.

Summing up, requirements verification by means of FO automatic theorem proving is only a partial method in a sense, but still extremely useful. It does not give a yes or no answer to any question about the system that can be rendered as a FO formula, but in cases where the property actually holds it often gives a positive answer, and in addition to this, a proof of this fact.

*Summary* Most current work in hardware verification employs propositional logics as property description languages. On the other end of the spectrum, interactive theorem proving is used for verification of complex hardware designs, sometimes involving higher order logics. This involves a lower degree of automation and significantly increases the demands on the skill and education of users.

By using first order logic we keep a middle path and retain a lot of the benefits of using a theorem proving method. At the same time we have the possibility of verifying hardware descriptions at a much higher level of abstraction than with propositional logic.

There are, of course, also practical limitations due to complexity issues. There are intractable problems in automatic theorem proving, and we have encountered such problems.

## 3 Requirements Verification by Translation to First Order Logic

### 3.1 Requirements Verification

Requirements verification, in contrast to verification methods like equivalence checking and refinement (see Section 4) consists in establishing whether a certain property holds with respect to a certain system or not. A property of a system in this sense is a partial description of the system in some logical formalism. Systems are often specified algorithmically in some imperative formalism. How the system will behave (in the long run) is not directly stated in such a specification, it is something that, if at all, can only be computed from the specification. In contrast to this a property description is not an algorithm but a statement about what is true of the system and its behavior.

*Example 1.* To illustrate this we will look at a simple counter in pseudo-code (resembling VHDL).

```
begin
  if reset = '1' then
    v <= 0;
  elsif clock = '1' and clock_edge then
    if v < max then
      v <= v + 1;
    else
      v <= 0;
    end if;
  end if;
end
```

This system is assumed to be sensitive to changes in the `reset` and `clock` signals. If there is a positive flank[3] of the `reset` signal (or a flank of the clock signal while `reset` is high) the variable $v$ is set to 0, if there is a positive flank of the `clock` signal when `reset` is low then if $v <$ `max`, $v$ is set to $v + 1$ otherwise $v$ is set to 0.

The specification does not say anything directly about the behavior of the system over time. A typical behavioral property one would like to know is:

> If started with a `reset` will $v$ ever have a value exceeding the value of `max`? (1)

[3] A *flank* is the change of a signal value to its opposite between two successive time points. A *positive* flank is a change from low to high value.

Further examples of properties are: Will a certain value of $v$ recur in some regular way (e.g. every $n$ time steps, where $n$ is a multiple of $\texttt{max} + 1$)? Will $v$ assume every value in the set $[0..\texttt{max}]$? Will $v$ assume the value 0 infinitely many times if run for ever? And so on.

Verifying a system with respect to behavioral properties of this kind amounts to establishing that the properties in question are true of (all) systems that conform to the system specification. This is, in effect, the notion of logical consequence. We have chosen to verify systems by establishing logical consequence by means of automatic theorem proving in first order logic.

**Definition 1 (First order logic).** *The language of* first order (FO) logic *is inductively defined by:*

$$\phi := P \mid P(t_1, \ldots, t_n) \mid \neg\phi \mid \phi \wedge \phi \mid \exists x \phi \ ,$$

*where $P$ is a member of a nonempty set of* predicate symbols*, $x$ is a variable, and the $t_i$ are terms inductively defined by*

$$t := v \mid c \mid f(t_1, \ldots, t_n) \ ,$$

*where $v$ is a* variable*, $c$ is a* constant *and $f$ is a* function symbol. Disjunction $\vee$*,* implication $\Rightarrow$*,* equivalence $\Leftrightarrow$*, and* universal quantification $\forall$ *are defined from these in the usual way.*

*The semantics is defined in the standard way. The equality predicate $=$ is interpreted as equality on the domain elements. A first order approximation of the natural numbers (written as $0, 1, \ldots$) including the function symbols $+$, $*$, and inequality predicates $<$, $\leq$ is built-in as well.*

It is well known that built-in support of the particular semantics of equality and other symbols can greatly improve the efficiency of theorem provers (as opposed to the explicit addition of axioms). At the moment we use theorem provers that have built-in support for efficient handling of equality and we add explicit axioms for the remaining function and predicate symbols. It would be very desirable to integrate special rules besides those for equality, but it is not obvious how to do this for current high-performance theorem provers.

### 3.2 Translation into First Order Logic

Notwithstanding the principal difference between algorithmic specification and behavioral description, both can be expressed in FO logic. We developed a translator from a subset of the hardware system specification or description language VHDL (see, e.g., [8]) into first order logic.

*Example 2.* A simple example of translation from VHDL into FO logic follows. It is similar (though simplified) to what the translator would output when translating the VHDL equivalent of the specification in Example 1.

$$\begin{aligned}
&\forall t\,(\texttt{reset}(t) \Rightarrow v(t) = 0 \wedge \\
&\quad (\neg\texttt{reset}(t+1) \Rightarrow \\
&\quad\ (\neg \max \leq v(t) \Rightarrow v(t+1) = v(t) + 1 \wedge \\
&\quad\quad \max \leq v(t) \Rightarrow v(t+1) = 0) \\
&\quad )))
\end{aligned}$$

Here the binary `reset` signal is represented by means of a unary predicate symbol with the same name (the signal `clock` is assumed to define the time-step and does not occur other than indirectly in the translation output). These predicates take a time argument. If $t$ is a term denoting a time point, then the intended semantics of $\texttt{reset}(t)$ is that the `reset` signal is high at that time point. The integer valued signal (variable) $v$, on the other hand, is represented by a unary function symbol with the same name. Intuitively, $v$ is interpreted as a function from time points to integers.

*Property specification language* We use a linear time temporal property specification language developed at Safelogic that extends FO logic with embedded temporal constructs that are useful for specifying behavioral properties. Like other recent property specification languages [5] it contains a rich set of constructs, which is motivated by practical usefulness in expressing the required properties of industrial designs.

Formulas in this specification language are then translated into FO logic along with the system that is to be verified. For example, the formula

$$v \leq \texttt{max}\ \texttt{whenever}\ (\neg\texttt{reset}\ \texttt{since}\ \texttt{reset})$$

in the property specification language, expressing property (1) translates into the following FO formula:

$$\begin{aligned}
&\forall t_0\,(\exists t_1\,(t_1 \leq t_0 \wedge \\
&\qquad \texttt{reset}(t_1) \wedge \\
&\qquad \forall t_2\,((t_1 < t_2 \wedge t_2 \leq t_0) \Rightarrow \neg\texttt{reset}(t_2)) \\
&\qquad ) \Rightarrow v(t_0) \leq \texttt{max})
\end{aligned}$$

If one can establish that the formula expressing the property is a logical consequence of the formula expressing the system specification (possibly, with the help of additional axioms or lemmas), then one has shown that the property formula is true of every system that realizes the system description.

*Axioms* Formulas that are needed besides the formulas specifying the system are e.g., those axiomatizing discrete time. At this point we only translate synchronous designs in which there is a common clock signal for the whole design

and assignments are assumed to take place only at a (positive) flank of this signal. We are thus able to conceptualize time as a discrete succession of time points, i.e. essentially a structure isomorphic to that of $\mathbb{N}$ or $\mathbb{Z}$ with the linear order $<$. Additionally, we have a kind of polymorphic addition on those structures for taking us a certain number of time steps along the time line. We need theories that axiomatize the different types of the design and also theories that axiomatize the operations that are defined on objects of those types. Important examples are theories of integers, bit vectors, etc., with $<$, addition and multiplication, and other arithmetical operators on those objects.

We experimented with different FO theories for such structures. It is well known that these structures are not FO characterizable (up to isomorphy), there are not even complete axiomatizable FO theories for most of them [6], so we have to use incomplete approximations. For $\mathbb{Z}$ with $<$, addition, and multiplication, we simply used standard theories containing irreflexivity, transitivity and trichotomy for $<$, straightforward inductive definitions of $+$ (and, where needed $*$), associative and commutative laws, definition of $-$ in terms of $+$, as well as axioms relating $+$ and $<$ such as:

$$\forall n\, \forall m\, \forall k\, (n < m \Rightarrow n + k < m + k)$$

$$\forall n\, \forall m\, (m < n \Rightarrow \exists k\, (m + k = n))$$

Our axiomatization is not minimal. Redundant axioms were added liberally provided that they sped up the theorem proving process.

We start out with formulas produced by the VHDL-translation, add the arithmetical axioms, the other axioms, and finally the negation of the requirement to be verified. All these formulas are converted to clause normal form required by the translation procedures that come with the resolution based theorem provers we use. In many cases we apply additionally disjunctive splitting (sometimes asymmetrical) on the negated requirement (see Section 4).

*Problem Characteristics* Much can be done to make proof search more effective. Designers of ATP systems furnish those systems with a large number of search heuristics. Some of these are especially suited for problems generated from industrial designs. We observed that such problems are often structured quite differently from the problems theorem provers are usually optimized for (including the majority of the problems found in the TPTP problem library[4]).

One such difference is that mathematical problems are often characterized by few and small axioms, a minimal axiomatization, and often fairly long proofs, whereas the industrial problems we have worked with show the opposite characteristics: the axioms are numerous, lengthy, redundant, and typical proofs (when found) are quite short, although we have also encountered longer proofs.

*Redundancy* Redundancy of problem formulations stems from several sources, and we think it should be given more attention in automated theorem proving.

[4] `http://www.cs.miami.edu/~tptp/`

First, the axiomatization of a theory might be redundant in the sense that the set of axioms is not minimal. For complex theories, minimality can be hard to show. Second, as is typical for interactive theorem proving (but also for challenging problems in automated theorem proving), one tries to tackle problems with elusive proofs by proving lemmas first that constitute intermediate steps. These lemmas are then added to the axioms. Third, even when the requirements specification and theories are expressed minimally, only a small subset of them might actually be needed to prove a particular claim. We have found this situation very often in our problems.

In our experience, FO theorem provers do not behave very robustly with respect to redundancy, so it would be important to identify parts of a problem formulation that are not required to prove a given conjecture. If theories are hierarchically defined and fulfill certain restrictions, it is possible to reduce the number of axioms by analyzing the symbols occurring in the conjecture [9]. Unfortunately, this kind of analysis is not implemented in current ATP systems. Otherwise, one has to resort to the users' insight into a problem domain who might be able to pinpoint properties that are unlikely to play a role in the proof. But even for this the user interface of current ATP systems forces one one to add and remove comments tediously by hand in the input files.

## 4 Verification by Refinement

A different way of specifying the behavior of a system than giving an explicit model consists in using the notion of refinement. One starts with a complete specification of all admissible/correct behaviors of a system. Then one *refines* this specification, i.e., one makes decisions on how to implement certain aspects, and on which particular behavior we want (if there is a choice).

In systems of non-trivial size, refinement steps are iterated, eventually leading to an implementation. For each refinement step, of course, one has to show that it is correct, i.e., all behaviors permitted by a refinement are also permitted by its specification.

One practical way of employing the refinement methodology in circuit verification uses a so-called reference implementation of the circuit which is believed to be correct (e.g., [10]). The actual implementation of the circuit is then shown to be a correct refinement of the reference implementation.

For the following somewhat informal presentation of verification refinement we borrowed some notions from the B-method [11]. For simplicity, we assume that our behavioral description language is similar to a simple imperative programming language.

**Definition 2 (System description).** *A system description $S$ consists of four parts: a finite number of* state variables $v_1^S, \ldots, v_n^S$*; a behavioral description $T^S$ of a circuit called the* transition*; a logical formula $I^S$ called the* invariant*; and a logical formula $B^S$ called the* initialization*.*

We use system descriptions to describe specifications, refinements and implementations. The idea is that a system description $S$ corresponds to a circuit, the state of which is represented by the state variables $v_i^S$, the behavior of which is described by $T^S$, and in every reachable state, the invariant $I^S$ should hold. The initial states of the circuit are described by the formula $B^S$.

*Correctness of specification* When we create a system description $S$ that corresponds to a specification, we need, of course, to prove that the invariant in fact is an invariant. This is done by proving the following two formulas, which together amount to an inductive proof:

$$B^S \Rightarrow I^S \qquad \text{(S-init)}$$

$$I^S \Rightarrow [T^S]I^S \qquad \text{(S-trans)}$$

We use the notation $[P]F$, where $P$ is a behavioral description and $F$ is a formula, to express "after performing $P$, the formula $F$ must hold", in other words, $F$ is a postcondition of $P$.

Hence, the formula (S-init) above can be read as: "The invariant holds for any initial state". The second formula (S-trans) above means: "If the invariant holds, then the transition will take us to a state where the invariant holds again".

*Establishing postconditions* The operator $[\_]$ is formally defined by a set of rewrite rules. These rules are applied automatically. Here are some example rules:

$$\begin{aligned}
[x := e]F &\longrightarrow F[e/x] \\
[P\ ;\ Q]F &\longrightarrow [P][Q]F \\
[\mathsf{if}\ G\ \mathsf{then}\ P\ \mathsf{else}\ Q]F &\longrightarrow (G \Rightarrow [P]F) \wedge (\neg G \Rightarrow [Q]F)
\end{aligned}$$

And so on. Further, we use abbreviation $\langle P\rangle F$ to mean $\neg[P]\neg F$, which can be read as "when we perform $P$, the formula $F$ possibly holds".

*Example 3.* As an example, consider the following system specification $S$ of a 2-bit counter circuit, which has one state variable $v$:

$$\begin{aligned}
T^S &\equiv \mathsf{if}\ v < 3\ \mathsf{then}\ v := v + 1\ \mathsf{else}\ v := 0 \\
I^S &\equiv 0 \le v \le 3 \\
B^S &\equiv v = 0
\end{aligned}$$

The initialization part (S-init) of correctness amounts to:

$$v = 0 \Rightarrow 0 \le v \le 3$$

The transition part (S-trans) amounts to:

$$0 \le v \le 3 \Rightarrow [\mathsf{if}\ v < 3\ \mathsf{then}\ v := v + 1\ \mathsf{else}\ v := 0]0 \le v \le 3$$

After some rewriting, this becomes:

$$0 \le v \le 3 \Rightarrow (v < 3 \Rightarrow 0 \le v + 1 \le 3 \wedge (\neg(v < 3) \Rightarrow 0 \le 0 \le 3))$$

Both parts are easily shown to be valid under most standard FO integer axiomatizations.

*Correctness of Refinement* Suppose we have a system description $S$, which is a specification, and a system description $R$, which is a refinement of $S$. The invariant $I^R$ of $R$ should not only express the invariant of the transition $T^R$, but also how the state variables of $R$ relate to the state variables of $S$. The proof obligation of showing that $R$ really is a refinement of $S$ amounts to proving the following two formulas:

$$B^R \;\Rightarrow\; (B^S \;\Leftrightarrow\; I^R) \qquad \text{(R-init)}$$

$$(I^S \;\wedge\; I^R) \;\Rightarrow\; [T^R]\langle T^S\rangle I^R \qquad \text{(R-trans)}$$

The first formula, (R-init), stipulates that initializing the state in the refinement establishes the invariant in correspondence with the initialization of the specification. The second formula, (R-trans), says that if both systems are in a corresponding good state, then for any $R$-transition there exists an $S$-transition such that the systems are in a corresponding state again, i.e., $R$ only makes transitions which are permitted by $S$.

*Example 4 (Example 3 cont'd).* Consider the following system specification $R$, which is a refinement of the 2-bit counter circuit specified above. It has state variables $b_0$ and $b_1$ (we assume to have standard operators $\mathsf{inv}$ and $\mathsf{xor}$ of bit arithmetic).

$$\begin{aligned} T^R &\;\equiv\; b_1 := (b_0 \;\mathsf{xor}\; b_1) \;;\; b_0 := \mathsf{inv}(b_0) \\ I^R &\;\equiv\; 0 \le b_0, b_1 \le 1 \;\wedge\; v = b_0 + 2 * b_1 \\ B^R &\;\equiv\; b_0 = 0 \;\wedge\; b_1 = 0 \end{aligned}$$

The initialization part (R-init) of correctness amounts to:

$$(b_0 = 0 \;\wedge\; b_1 = 0) \;\Rightarrow\; (v = 0 \;\Leftrightarrow\; (0 \le b_0, b_1 \le 1 \;\wedge\; v = b_0 + 2 * b_1))$$

The transition part R-trans amounts to:

$$\begin{aligned} &(0 \le v \le 3 \;\wedge\; 0 \le b_0, b_1 \le 1 \;\wedge\; v = b_0 + 2 * b_1) \;\Rightarrow \\ &\quad [b_1 := (b_0 \;\mathsf{xor}\; b_1) \;;\; b_0 := \mathsf{inv}(b_0)]\langle \mathsf{if}\; v < 3 \;\mathsf{then}\; v := v + 1 \;\mathsf{else}\; v := 0\rangle \\ &\qquad\qquad (0 \le b_0, b_1 \le 1 \;\wedge\; v = b_0 + 2 * b_1) \end{aligned}$$

Both parts can be rewritten and simplified to valid formulas.

*Automation* All of the above formulas become formulas in FO logic after (automatic) rewriting. The proving process can be automated by producing the proof obligations together with a suitable theory about integers and other operations in a suitable input format to a FO theorem prover, and searching for a proof. However, even for simple systems this turned out to be not that easy. We apply a number of "tricks" on the resulting FO formulas to render them provable more easily by a theorem prover.

First, we implemented a simplifier that works on the formulas *before* clausifying them. In addition, we perform case splitting and special axiomatizations.

*Case splitting* Formulas stemming from refinement followed by rewriting can become very large. Luckily, in many cases one can identify smaller parts that deal with a particular aspect of a proof obligation independently. For example, the invariant is often a big conjunction, hence, a proof obligation relating to an invariant can be split up into the elements of the conjunction. Likewise, a transition often consists of a number of nested $\mathsf{if}-\mathsf{then}-\mathsf{else}$ statements, leading to a number of control paths which can be considered separately.

The following rewrite rules for case splitting generalize these considerations:

$$\begin{array}{rcl} A \wedge B & \longrightarrow & A\,,\ B \\ A \Rightarrow (B \wedge C) & \longrightarrow & A \Rightarrow B\,,\ A \Rightarrow C \\ A \Leftrightarrow B & \longrightarrow & A \Rightarrow B\,,\ B \Rightarrow A \end{array}$$

We found that, typically, a proof obligation can be split up into 100 to 1000 separate proof obligations, and 40–80% can usually be simplified away by the simplifier, hence, they generate no FO theorem proving problem at all.

*Axiomatization of arrays* One datatype occurring often in hardware descriptions is the array type. We deal with arrays in the following way. When an array update of the form

$$a[i] := e$$

occurs, it is treated as if it were the statement

$$a := \mathsf{override}(a, i, e)$$

Furthermore, indexing in an array $a[i]$ is translated into $\mathsf{index}(a, i)$. The core of the axiomatization of the theory of arrays looks as follows:

$$\mathsf{index}(\mathsf{override}(A, I, X), I) = X \qquad \text{(Index-1)}$$

$$I \neq J \Rightarrow \mathsf{index}(\mathsf{override}(A, I, X), J) = \mathsf{index}(A, J) \qquad \text{(Index-2)}$$

Unfortunately, this theory turned out to be very difficult to deal with for current FO theorem provers. Normally, when encountering an indexing operation in an overridden array, the theorem prover should initiate a case split on the two cases amounting to (Index-1) and (Index-2). It is not possible to configure current FO theorem provers in such a way that this happens.

To express the possibility of this case split explicitly in the logic a special $\mathsf{if}-\mathsf{then}-\mathsf{else}$ connective was used in [12] in a similar situation. We chose a somewhat different solution which turns out to be useful in other contexts as well: an $\mathsf{if}-\mathsf{then}-\mathsf{else}$ on *terms* in FO logic (not to be confused with the $\mathsf{if}-\mathsf{then}-\mathsf{else}$ in the behavioral descriptions introduced earlier). This allows us to embed formulas inside terms!

The meaning of $\mathsf{if}-\mathsf{then}-\mathsf{else}$ on the term-level is given by the following rewrite rule. For any predicate symbol $P$:

$$P(\ldots \mathsf{if}\ F\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 \ldots) \longrightarrow (F \Rightarrow P(\ldots e_1 \ldots)) \wedge (\neg F \Rightarrow P(\ldots e_2 \ldots))$$

Using the new $\mathsf{if} - \mathsf{then} - \mathsf{else}$ construct, one can axiomatize arrays differently:

$$\mathsf{index}(\mathsf{override}(A, I, X), J) \;=\; \mathsf{if}\ I = J\ \mathsf{then}\ X\ \mathsf{else}\ \mathsf{index}(A, J) \qquad \text{(Index)}$$

Now, the equality above can be seen as yet another rewrite rule which can be used to demodulate the proof obligations. For each array building construct we have a rewrite rule similar to (Index), which says what happens when indexing is performed on that construct. After demodulation, all occurrences of index and the other array operators, like in this case override, disappear from the proof obligation. If we do this before case splitting, this leads to a lot of particularly powerful case splits.

## 5 Example: Requirements Verification of a FIFO Buffer

We applied the method sketched in Section 3 successfully to verify various properties of VHDL designs from industry. Unfortunately, we cannot disclose any of this work at this point; instead, we exemplify our method with a simple (although far from trivial) VHDL design of a FIFO buffer.

*The FIFO Buffer* The FIFO buffer is implemented as an array `mem` of length `fifo_length`, the elements of which are bit vectors of length `fifo_width`. The parameters `fifo_length` and `fifo_width` are generic positive integer parameters which need not be instantiated in FO verification, which is a great advantage. Properties can thus be verified uniformly for FIFOs of any length and width.

The design also has integer valued signals `level`, `wr_level` and `rd_level` that keep track of the number of items recorded in the FIFO, the write index, and the read index in the main array `mem`. (See `http://www.safelogic.se/problems/PaPS` for the VHDL code of this design). The in signal `reset` initializes these three signals to 0.

In addition to the binary `clock` signal the incoming signals are the binary `wr` and `rd` signals used for controlling whether a value should be written into the FIFO or whether a value should be read from (and thus be taken away from) the FIFO.

The `data_in` bit vector of length `fifo_width` carries the value that is written into the FIFO. In addition to `level` there are the out signals `data_out` which is a bit vector of length `fifo_width` that carries the value that is read off the FIFO, and the binary signals `rd_error` and `wr_error`.

When there is a successful write the value of `data_in` should have been written to the `mem` array at the index indicated by `wr_level` and the latter signal should have been incremented (modulo `fifo_length`), so that the next successful write will take place at the next index of `mem`

Successful reads write the value at index `rd_level` of `mem` to `data_out`. Then `rd_level` is incremented in a similar fashion to `wr_level`.

The `level` signal keeps track of the number of values that have been successfully written to and not yet been successfully read from the FIFO, in other words,

the number of values in the buffer. Provided that simultaneous read is not implemented, writing should not be allowed as soon as `level` reaches `fifo_length`, as in this case every index of the `mem` array holds a written value that is not already read from the FIFO. On the other hand, reading should not be allowed when `level` is 0, because then there is no written value in `mem` that is not already read.

*Properties of the FIFO Buffer* Obviously, there is some control logic to verify with respect to this design. We exemplify this by giving some of the desirable requirements in our property language. Most of these properties can be verified without using induction of any kind, since they are robust with respect to reachable states. Put equivalently, they hold in all possible states of the design regardless of whether the signals were properly initialized. These are properties like:

$$\begin{aligned}&\texttt{next level} = \texttt{level} + 1\\ &\quad\texttt{whenever}\\ &\qquad\qquad \neg\texttt{reset} \wedge\\ &\qquad\qquad \texttt{level} < \texttt{fifo\_length} \wedge\\ &\qquad\qquad \neg\texttt{rd} \wedge\\ &\qquad\qquad \texttt{wr}\end{aligned}$$

This states that whenever `level` is less than `fifo_length`, the `rd` signal is low and the `wr` signal high, then `level` is incremented. This is true of the design regardless of how signals like `level` are initialized.

There are also properties not being robust in the above sense, for example:

$$(\texttt{level} \leq \texttt{fifo\_length})\ \texttt{always}$$

namely the property that `level` should never assume a value greater than `fifo_length`. This was verified with induction of step length 1 over the succession of time points, with an arbitrary time point when `reset` is high as base case. In effect the following two requirements were proved:

$$(\texttt{next level} \leq \texttt{fifo\_length})\ \texttt{whenever reset} \qquad \text{(Base Case)}$$

namely that at a time point following a `reset`, `level` should not be greater than `fifo_length`. which is the base case and then the step case:

$$\begin{aligned}&\texttt{next}(\texttt{level} \leq \texttt{fifo\_length})\\ &\texttt{whenever}\ (\texttt{level} \leq \texttt{fifo\_length})\end{aligned} \qquad \text{(Step Case)}$$

namely that if `level` is not greater than `fifo_length` at any time point then this is also true of the next time point.

*Challenging requirements* Even with respect to this relatively simple design there are behavioral properties on a higher level in the sense that they not only relate adjacent time points (or time points with a given distance). For example, they might say something about the behavior of the design in a non-deterministic environment, such as the property of being a FIFO: if $v$ is successfully written to the FIFO before $v'$ then $v$ should be read from the FIFO before $v'$. Such properties can be formalized in FO logic and in our property specification language. They involve time spans of flexible length. Specification and verification of such properties is work in progress.

*Significance of the Example* The design described above is too simple and regular to be really representative of the examples emanating from industry that we have been working with. Nevertheless, requirements similar to those given above tend to recur in more complicated designs as well.

Requirements of this kind turned out to be feasible to verify with the theorem provers we use. (Experimental results with different theorem provers can be found at `http://www.safelogic.se/problems/PaPS`.) And this holds not only for this simple FIFO but also for more complicated real-life systems we have been working with.

One recurring aspect of FO verification that is already visible in the FIFO is that parameterized designs can often be verified simultaneously for all possible values of the parameters. This works, because often the value of the parameters do not affect the proof. This is an aspect of a more general phenomenon: such aspects of a design that do not affect the proof of a property need not be brought into the verification process. In this way, the complexity of a problem may often be cut down.

What is absent from the FIFO example is data manipulation. Data are only shuffled around in the FIFO but not modified or inspected. Other absent features, which are ubiquitous in real-world VHDL designs, are type conversions and polymorphic application of operators: bit vectors may be added to or compared to integers, for example.

All recursive functions and properties are representable in the arithmetical theories we use [6], but it offers a considerable challenge to represent the needed functions and predicates in a way that is suitable for automatic theorem proving.

## 6 Conclusion

In this paper we argued for using first order theorem proving in verification of hardware systems. There are a number of strong advantages to use expressive logics such as FO logic, despite its main drawback, which is semi-decidability. We believe there is considerable unexplored potential from FO theorem proving in this area, in particular, if theorem proving specialists and domain experts work together on problem representation and translation.

We had to learn that existing methods to axiomatize data structures, to translate from temporal to first order logic, to compute normal form, and the

configuration possibilities of existing theorem provers do not suffice for automatic verification of interesting properties.

On the other hand, it was possible to adapt the existing technologies in such a way that interesting results could be proved. On the web page corresponding to this paper we contribute some first order problems that give a flavor of what is required. It is our hope that the FO theorem proving community finds them stimulating.

## References


1. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. The MIT Press, Cambridge, Massachusetts (1999)
2. Abdulla, P.A., Bjesse, P., Eén, N.: Symbolic reachability analysis based on SAT-solvers. In Graf, S., Schwartzbach, M., eds.: Tools and Algorithms for the Construction and Analysis of Systems TACAS. Volume 1785 of LNCS., Springer-Verlag (2000) 411–425
3. Biere, A., Cimatti, A., Clarke, E., Zhu, Y.: Symbolic model checking without BDDs. In Cleaveland, W., ed.: Tools and Algorithms for the Construction and Analysis of Systems. Volume 1579 of LNCS., Springer-Verlag (1999) 193–207
4. Robinson, A., Voronkov, A., eds.: Handbook of Automated Reasoning. Elsevier Science B.V. (2001)
5. Armoni, R., Fix, L., Flaisher, A., Gerth, R., Ginsburg, B., Kanza, T., Landver, A., Mador-Haim, S., Singerman, E., Tiemeyer, A., Vardi, M.Y., Zbar, Y.: The ForSpec temporal logic: A new temporal property specification language. In Katoen, J.P., Stevens, P., eds.: Proc. Tools and Algorithms for the Construction and Analysis of Systems TACAS, Grenoble, France. Volume 2280 of LNCS., Springer-Verlag (2002) 296–311
6. Boolos, G.S., Jeffrey, R.C.: Computability and Logic. Cambridge University Press (1989)
7. Fermüller, C.G., Leitsch, A., Hustadt, U., Tammet, T.: Resolution decision procedures. [4] chapter 25 1791–1849
8. Ashenden, P.J.: The Designer's Guide to VHDL. Morgan Kaufmann, San Francisco (1996)
9. Reif, W., Schellhorn, G.: Theorem proving in large theories. In Bibel, W., Schmitt, P., eds.: Automated Deduction: A Basis for Applications. Volume III. Kluwer (1998) 225–242
10. Jones, R.B., O'Leary, J.W., Seger, C.J.H., Aagaard, M.D., Melham, T.F.: Practical formal verification in microprocessor design. IEEE Design & Test of Computers **18** (2001) 16–25
11. Abrial, J.R.: The B Book: Assigning Programs to Meanings. Cambridge University Press (1996)
12. Ahrendt, W., Beckert, B., Hähnle, R., Menzel, W., Reif, W., Schellhorn, G., Schmitt, P.H.: Integration of automated and interactive theorem proving. In Bibel, W., Schmitt, P., eds.: Automated Deduction: A Basis for Applications. Volume II. Kluwer (1998) 97–116

# Title to be Announced

Johannes Schumann

NASA Ames, USA
schumann@ptolemy.arc.nasa.gov

**Abstract.** To be Provided