Applications of a Global Workspace Framework to Mathematical Discovery

John Charnley and Simon Colton
Combined Reasoning Group, Department of Computing
Imperial College, United Kingdom

Abstract

Systems which combine various forms of reasoning such as deductive inference and symbolic manipulation have repeatedly been shown to be more effective than stand-alone systems. In general, however, the combined systems are ad-hoc and designed for a single task. We present a generic framework for combining reasoning processes which is based on the theory of the Global Workspace Architecture. Within this blackboard-style framework, processes attached to a workspace propose information to be broadcast, along with a rating of the importance of the information, and only the most important is broadcast to all the processes, which react accordingly. To begin to demonstrate the value of the framework, we show that the tasks undertaken by previous ad-hoc systems can be performed by a configuration of the framework. To this end, we describe configurations for theorem discovery and conjecture making respectively, which produce comparable results to the previous ICARUS and HOMER systems. We further describe a novel application where we use a configuration of the framework to identify potentially interesting specialisations of finite algebras.

Keywords: global workspace, combined reasoning, conjecture making, finite algebra

1 Introduction

Stand-alone Artificial Intelligence systems for performing specific types of reasoning – such as deduction in theorem provers, symbolic manipulation in computer algebra systems or induction in machine learning systems – have steadily become more powerful. It is therefore no surprise that researchers have investigated how one might fruitfully combine such reasoning systems so that the whole is more than a sum of the parts. In general, such combinations have been ad-hoc in nature and designed with a specific task in mind. For the field of combining reasoning system to progress, we believe it is imperative for more generic frameworks to be developed and experimented with. As described below, the cognitive science theory of a Global Workspace Architecture has been proposed as a model which captures the massively-parallel reasoning capabilities of mammalian brains. It could therefore provide a basis for a generic computational framework within which reasoning systems can be combined. We describe here such a generic framework and its implementation. We demonstrate two applications of it, namely to theorem discovery and conjecture making in domains of pure mathematics, which have been performed previously by bespoke combined systems. We also give details of a new application to finite algebra, where we use a configuration of the framework to automatically identify specialisations of algebra classes which could be fruitful areas for mathematical investigation.

As described in §2, our framework allows for multiple processes to reason about information in disparate ways and communicate via a blackboard-style global workspace. In each round of a session, a single piece of information is broadcast to each process attached to the workspace. Each process may or may not reason about the broadcast information in order to produce novel information. For instance, if a conjecture is broadcast, a deductive process may try to prove the conjecture, whereas a model-generation process may seek a counterexample. Any process which does produce novel information will propose it for broadcast, along with a numerical rating of its value, which the process itself determines.

Sutcliffe G., Colton S., Schulz S. (eds.); Proceedings of ESARM 2008, pp. 1-16

The framework then broadcasts only the highest rated proposal at the start of the next round, and the session progresses. Developers of combined reasoning systems can configure the framework to perform particular tasks by specifying how certain processes will reason about certain types of information; what novel information they will output; and how proposals should be ranked.

Automated theory formation, as performed by the HR system [6], has been at the heart of a number of ad-hoc combined reasoning systems. In particular, the TM system [9] used HR, the Otter theorem prover [19] and the MACE model generator [20] to discover and prove alternative true theorems to a given false conjecture. In the ICARUS system [5], HR was used in combination with Otter and the CLPFD constraint solver inside Sicstus Prolog [4] to reformulate constraint satisfaction problems. In the HOMER system [7], HR was used in conjunction with Otter and the Maple computer algebra package [24] in order to make conjectures about computer algebra functions. Also, HR is part of the system developed by Sorge et al. [8] which integrates computer algebra routines, theorem proving, SAT-solving and machine learning, and has been used to discover and prove novel classification theorems in algebraic domains of pure mathematics. As an initial test of our implemented framework, it seemed sensible to see whether it could be configured to perform similar tasks to those performed by some of these ad-hoc systems.

As described in §3 and §4 respectively, we have managed to configure the framework to perform automated theorem discovery in quasigroup theory (as previously undertaken by HR as part of the ICARUS system) and to perform automated conjecture making in number theory (as previously undertaken by the HOMER system). For each configuration, we describe the processes, the types of information which are broadcast, and how this information is rated. We also provide qualitative and quantitative comparisons with ICARUS and HOMER respectively. In §5, we describe a new application for combined reasoning systems in the domain of finite algebra. In this application, we seek to identify specialisations of finite algebras which are simple in definition yet have mathematical properties to suggest they may be interesting areas for mathematical investigation in their own right. The configuration we developed for this task is a modified version of the one we used in §3 for automated theorem discovery in quasigroup theory. We applied this to three different classes of finite algebra and, in each case, the system identified an interesting specialisation.

A Global Workspace Architecture is essentially a model of combined serial and parallel information flow, wherein specialist sub-processes compete and co-operate for access to a global workspace [1]. A specialist sub-process can perform any cognitive process such as perception, planning, problem solving, etc. If allowed access to the global workspace, the information generated by such a specialist is broadcast to the entire set of specialists, thereby updating their knowledge of the current situation. In recent years, a substantial body of evidence has been gathered to support the hypothesis that the mammalian brain is organised via such global workspaces [2]. Moreover, this theory can be used to explain aspects of human cognition such as conscious and unconscious information processing, and can be applied to challenges such as the frame problem [23]. From an engineering point of view, the global workspace architecture has aspects in common with Newell's blackboard architectures for problem solving [21]. AI software agents based on the Global Workspace Architecture have been successfully implemented [15, 16], and in some cases applied to problem solving, e.g., constraint solving [17]. Another framework which combines disparate reasoning systems is the OANTS [3] architecture of Benzmüller et al. That framework has much overlap with the GWA, as proactive software agents which determine they are relevant to a central proof object are allocated resources to investigate, and they bid to amend the current state based upon their findings. Also, in [12], Fisher describes a concurrent approach to theorem proving using broadcasts, where agents each hold a subset of clauses to a given proof task. In that approach, agents respond to clausal broadcasts by updating their set of clauses and broadcasting any new information to all, or subgroups of, other processes. This has been adapted to problem-solving applications, such as negotiation [13] and planning [14]. What distinguishes the GWA framework from other approaches is the simplicity

of the architecture and the restrictions it places upon communication. Our framework effectively limits processes to one broadcast message for each processing result and this message is not guaranteed to be received by any other process. Such messages represent only a tiny fraction of the current state of the overall system and so the workspace has a much reduced scope in comparison to the blackboard in a traditional blackboard architecture. We are interested in addressing whether – despite these simplifying restrictions – we are able to construct useful combined reasoning systems using this framework, or indeed whether the framework actually simplifies the often difficult process of integrating disparate AI systems.

2 A Framework for Combining Reasoning Systems

The architecture defined by our framework is inspired by the Global Workspace Architecture [1]. Each of the processes attached to the global workspace performs either some type of reasoning (e.g., by encapsulating a theorem prover or a computer algebra system) or a useful administrative task such as checking for redundancy in outputs. The framework defines how processing takes place on a round-by-round basis. In addition, it outlines rules which all attached processes must follow. A round starts with the broadcast of some reasoning *artefact* (e.g., a conjecture, proof, example, etc.) which each attached process may ignore or may react to in various ways. Specifically, a process may do one or more of the following:

- Construct a *proposal* for broadcast, consisting of a reasoning artefact and a numerical (heuristic) value of importance that the process ascribes to that artefact.
- Detach itself from the framework.
- Attach new processes to the framework.

At the end of each round, various processes will have been added to and removed from the global workspace, and a set of broadcast proposals will have been submitted to the framework. At the start of the next round, the framework chooses the proposal with the highest importance value, and broadcasts the reasoning artefact from that proposal. In the case where multiple proposals have equal heuristic value, one is chosen from them randomly. All non-broadcast proposals are discarded and will not be considered for broadcast later unless they are re-proposed. Currently, all reasoning artefacts are broadcast as string arrays, examples of which are presented in the next two sections.

To create a combined system, a developer must create a *configuration* of the framework, by defining:

- The reasoning artefacts that may be broadcast on the workspace.
- The processes that may be attached to the workspace and their behaviour, which must conform to the framework rules. In particular, how each process reacts to broadcasts, the processing or reasoning they perform, the proposals they can make and the method they use in determining the heuristic rating of importance.
- The starting state, i.e. the initially attached processes.

We have developed the GC toolkit, which enables developers to easily configure combinations of reasoning systems for particular tasks within the framework. GC, which takes its name from global-workspace and combining, allows users to develop their configurations into full system implementations. It includes the core code for the round-by-round processing and a number of pre-coded processes which encapsulate specific reasoning tasks. For example, the toolkit currently provides a process which appeals to the Prover9 theorem prover [18] in attempts to prove broadcast conjectures. Users can choose and adapt

processes from GC's pre-coded selection for use in their configurations or they can develop their own processes with the aid of libraries provided in the toolkit.

The GC toolkit is implemented in Java. Developers who wish to create new processes do so by extending the toolkit's *WorkspaceProcess* class. This class defines an abstract method *react* which the developer must override with the behaviour they want for their process. The *react* method must implement how the process behaves in response to different broadcasts and the processing that should be performed. The *react* method should return whatever proposals the developer wishes the process to make, and the developer must also define how the process should rate the importance of its proposals. The developer can also define parameters for the process. For example, a process to read and broadcast the contents of a file benefits from a parameter to indicate the file path. If a developer wishes to encapsulate an external reasoning system then the *react* method should be a wrapper for that system, making appropriate external calls. The toolkit includes a graphical interface for creating configurations of the framework. It allows users to drag and drop processes into configurations, set their parameters and run the resulting configurations. The code that runs the workspace simply calls the *react* method for all currently attached processes, collates their returned proposals and selects the highest rated.

Our framework is quite straightforward and we have only deviated from the underlying GWA theory by allowing the termination and spawning of processes. We believe that maintaining this low complexity has several benefits for clarity of design and extensibility, and we are investigating how capable this simple framework can be. However, there are many areas of the framework where we see potential for future development. For example, currently, all processing is performed on a single processor in a serial fashion. Moreover, processing is synchronous, because all processes are given the chance to react before a new round is begun, and it may be that parallel or asynchronous variants of the framework could be more effective. Further, once started, the behaviour of the system will be determined by three factors: the configuration chosen by the user, the values of parameters they have set and the randomness in choosing between equal proposals in a given round. Currently, once processing has begun, the user cannot intervene in any way other than to pause or stop the system. We have introduced control processes and control broadcasts, in response to which some processes detach from the workspace. This allows us to terminate some aspects of processing when a specific processing goal has been reached. However, we have not yet considered any more sophisticated intervention schemes. As yet, we have only considered relatively simple importance rating schemes. In general, processes will assign the same importance to all broadcasts of the same type. We have enhanced this slightly in some applications where we enable processes to determine the importance of their proposals based upon the specifics of their proposal. For example, in §3.4, the importance rating is based partly upon the number of predicates and variables included in a concept defintion. These, and other, aspects of the framework, where we see the potential for future development are discussed in §6.

In the next sections, we present configurations of the framework to demonstrate its potential for combining reasoning systems. In the first configuration, we develop the core automated theory formation processes which achieves similar behaviour to the concept production rules, conjecture making routines and third-party interactions which drive the HR system. These processes appeal to both the Yap Prolog system [11] and Prover9. In the second configuration, we improve the theory formation processes, and enable an interaction with the Maple computer algebra system [24]. In §5, we discuss a novel application of the theory formation configuration to investigating interesting specialisation classes of finite algebras.

3 Configuration 1: Quasigroup Theorem Discovery

Constraint solving is a very successful area of Artificial Intelligence research, and constraint solvers are regularly used for solving industrial-strength problems in, for example, scheduling or bin-packing.

The way in which a constraint satisfaction problem (CSP) is specified is crucial to the success of the solver, as different specifications can lead to radically quicker solving times. For this reason, there has been much research into reformulating constraint satisfaction problems. Our contribution has been to introduce a combined-reasoning system for finding additional constraints which are implied by the CSP specification. With the ICARUS system [5], using the QG-quasigroup standard CSP benchmark set, we showed that it was possible to substantially increase efficiency when solving CSP problems by automatically discovering implied constraints. ICARUS is an ad-hoc system which employs the HR program to make conjectures empirically about the solutions to a CSP, then Otter to show that the conjectures are true, and finally uses the CLPFD constraint solver [4] in Sicstus Prolog to determine which proved theorems increase the solver efficiency when added as implied constraints.

With the configuration described here, we intended to show that the framework could combine machine learning and theorem proving processes to discover implied constraints about QG-quasigroups similar to those found by ICARUS. In future, we also plan to attach constraint solving processes, so that the framework will be able to determine – like ICARUS – which implied constraints reduce solving time. As described in §3.1 to §3.4 below, to configure the framework for this task, we specified five different types of broadcastable artefacts, some concept forming, conjecture making and explanation finding processes and a fairly straightforward scheme for ascribing importance values to bids for artefacts to be broadcast. In §3.5, we show that the configured framework does indeed produce similar results to ICARUS.

In overview, the configured framework was required to invent new concepts (initially built from a set of user-supplied background concepts), by manipulating and combining existing concept definitions. Concept defintions are first order logic statements in a Prolog-readable format. Each concept definition partitions the example set into two; those for which the definition holds and those for which it does not. By comparing the sets of examples for which definitions hold, the system makes empirical conjectures which relate the concepts. Some of these conjectures can be proven to follow from the axioms of the domain of investigation. In simple outline, the system starts with Definition broadcasts for the background concept definitions. The DefinitionReviewer acts as a filter, so that only unique Definitions become Concepts. The ExampleFinder process finds examples for Concepts and the EquivalenceReviewer process filters out those Concepts that are equivalent to previous Concepts. When a Concept has passed this filter, a NewConcept is broadcast. DefinitionCreator processes react to NewConcept broadcasts by manipulating their definitions to create new Definitions. NewConcept broadcasts also feed into conjecture making. In particular, ImplicationMaker processes compare the example sets of NewConcepts and propose Conjectures, which may be proved by the Prover process, creating an Explanation proposal. Full details of broadcastable artefacts and the processes are given below.

3.1 Broadcast Artefacts

To represent the concepts, conjectures and proofs, we specified five main types of broadcast artefacts, below. As noted in §2, broadcasts take the form of string arrays, and we use the notation $[e_0 : e_1 : \ldots : e_n]$ to represent such an array of strings. In general, the first element of a broadcast string array indicates the type of broadcast. The remaining elements are the specifics of that broadcast.

1. Definition, in the form [def:D], where D is a Prolog-readable definition of a concept.

2. Concept, in the form [conc:D:E], with D as above and E being a list of examples which satisfy that concept definition.

3. NewConcept, in the form [new:D:E], with D and E as above. The distinction between Concept and NewConcept facilitates equivalence reviewing as described below.

4. Conjecture, in the form [conj:D_1:D_2:K], where D_1 and D_2 are concept definitions and K is a keyword

indicating the type of conjecture. In this configuration, K is limited to being either *im*, which denotes that D_1 is conjectured to imply D_2; or *eq*, denoting D_1 is conjectured to be equivalent to D_2.

5. Explanation, in the form [exp:D_1:D_2:K:R:P], where D_1, D_2 and K represent a conjecture, as above. R is a keyword indicating the type of explanation; either *true*, which indicates that the conjecture has been proved, for example by a prover process; or *false*, where the conjecture has been refuted, for instance a model generator has found a counter-example. P gives details of the proof or refutation, as appropriate.

An example *Definition* from quasigroup theory is [def:m(Q,A,A,B),m(Q,B,B,A)], which describes elements of quasigroups satisfying the equation $A*A=B \wedge B*B=A$. The $m/4$ predicate defines the multiplication table for a quasigroup; e.g. $m(Q_0,0,1,2)$ states that $0*1=2$ for quasigroup instance Q_0. The *NewConcept* artefact would be [new:m(Q,A,A,B),m(Q,B,B,A):((Q_0,0,1),(Q_0,1,0),(Q_1,2,1),...)], with an associated example set. These are tuples of valid bindings for the definition variables; $\{Q_0, Q_1\}$ are identifiers for different quasigroups and *{0,1,2}* are the elements of those quasigroups. The conjecture $\forall A\,B\,(A*B=B*A \leftrightarrow A=B)$ would be represented as [conj:m(Q,A,B,_C),m(Q,B,A,_C):e(Q,A,B):eq], in a *Conjecture* artefact where the *e/3* predicate, *e(Q,A,B)* is the equality of two elements within a particular algebra instance ($A=B$ in quasigroup Q). A proof of this conjecture in an *Explanation* artefact would be as follows; [exp:m(Q,A,B,_C),m(Q,B,A,_C):e(Q,A,B):eq:true:proof_text].

We make use of *flags* attached to broadcast artefacts to control the workspace. In particular, *Definition* and *Concept* artefacts have an attached complexity flag indicating the number of definitions, including itself, that have been used in creating that definition. This allows us to place an upper limit upon processing. For example, the definition *m(Q,A,A,B),m(Q,B,B,A)* would have a complexity of 3; in addition to itself, this definition has used the background definition of multiplication *m(Q,A,B,C)*, and the concept of squaring an element *m(Q,A,A,B)*. For clarity, we have omitted these flags from the formal definitions of the broadcasts, above.

3.2 Processes

We developed the following types of processes for use in the Quasigroup Theorem Discovery configuration:

1. DefinitionCreator processes propose new *Definitions*. They each encapsulate a different concept formation method, akin to production rules in HR. They react to *NewConcept* broadcasts, [new:D:E]. Some formation methods involve modifying a single concept definition, where they attempt to create a new definition from D. Others combine two definitions, in which case they remember D, by spawning a clone process that reacts to *NewConcept* broadcasts, [new:*D'*:*E'*:*C'*], by attempting to combine D and *D'*.

2. DefinitionReviewer, reacts to *Definition* broadcasts, [def:D], and removes redundancy by checking whether D has been seen before. If not, it proposes for broadcast [conc:D:$\emptyset$], i.e. a concept with definition D and an empty example set.

3. ExampleFinder, encapsulates a Prolog database containing examples for the initial background concepts. All concept definitions are Prolog terms and *ExampleFinder* can generate example sets for new concepts by querying Prolog with the definition. *ExampleFinder* reacts to *Concept* broadcasts with empty example sets, [conc:D:$\emptyset$], by generating an example set E. If E is non-empty, it proposes [conc:D:E].

4. EquivalenceReviewer, checks each new concept to identify and filter out those having the same example set as a previously developed concept. This removes a great deal of duplicated effort as the further development of each concept would give equivalent results. The process reacts to *Concept* broadcasts [conc:D:E] by proposing for broadcast [new:D:E]. Also, the process reacts to broadcast [new:D_1:E_1] by spawning a clone process P. For any future *Concept* broadcast [conc:D_2:E_2], if P finds that $E_2=E_1$, it

proposes an equivalence conjecture between them: [conj:D_1:D_2:eq]. A higher iportance value is allocated to equivalence conjecture proposals than to concept proposals. In the case where E2 = E1, both [conj:D_1:D_2:eq] and [new:D_2:E_2] will be proposed and the conjecture will be broadcast as it has higher importance. [new:D_2:E_2] will only be broadcast in the case where no spawned process identifies equivalence.

5. ImplicationMaker, compares the example sets of two *NewConcept* broadcasts. It reacts to the first *NewConcept*, [new:D_1:E_1], (where $E_1 \neq \emptyset$), by spawning a clone process, P, which itself reacts to future *NewConcept* broadcasts [new:D_2:E_2]. In particular, if P finds that $E_1 \subset E_2$, it proposes [conj:D_1:D_2:im] (or [conj:D_2:D_1:im] if $E_2 \subset E_1$).

6. Prover processes encapsulate the Prover9 theorem prover with axioms for the domain under investigation. It attempts to prove conjectures in any *Conjecture* broadcast, [conj:D_1:D_2:K], and proposes [exp:D_1:D_2:K:true:P], whenever a proof, P, is found.

The definition methods embodied by the *DefinitionCreator* processes are either *unary* or *binary*. Unary methods act upon one definition. An example of a unary method is called *variable freeing*. Given the starting concept $m(Q,A,A,B)$, i.e., the arity 3 concept of pairs of elements for which $A * A = B$, freeing the variable A would result in the concept $m(Q,_A,_A,B)$, which is the arity 2 concept of elements B, which are the square of some element. In addition, there are unary concept forming processes which unify variables, for example creating $m(Q,A,A,A)$ from $m(A,B,C,D)$. Other methods include grounding, which involves variable instantiation, and methods combining definitions as conjunctions with themselves with different variable orderings. The most common binary methods involve either creating a conjunction of two previous definitions and unifying their variables in a particular manner, or creating a conjunction of a previous concept and the negation of another previous concept. These methods are inspired by HR's concept formation methods, as described in [6].

DefinitionCreator processes spawn other processes to repeat-propose the definitions they create. This means that the definition is not forgotten if it is not immediately broadcast. Several other process types, for example the ImplicationMaker and Prover also use such an approach. This repetition addresses the question of residual memory, which is not really tackled by GWA theory.

3.3 Initial State

At the start of a session, a number of processes are attached to the global workspace. Several *DefinitionCreator* processes are attached as they each embody a different definition creation method and allow us to develop a theory in different ways. We attach *DefinitionCreator* processes for each of the definition formation methods described above. In addition, we include several *parameterisations* of each method, whereby the same formation method operates upon different sub-sets of variables. In addition to the DefinitionCreator processes, we attach one instance of each of the other processes defined above. We also specified a process which proposes BackgroundConcept and BackgroundAxiom artefacts for the domain, which starts the theory formation session.

3.4 Importance Rating Scheme

The framework requires that each proposal is given a numerical rating and it chooses the highest for broadcast. However, for the experiments described below, we used a simple scheme; Definitions are scored at 100, Concepts with no examples at 200, Concepts with examples at 250. Any NewConcept broadcasts are given the value 300 and Conjectures 400 meaning, in particular, equivalence Conjectures will get precedence over NewConcepts. Explanations are valued at 500. Having Explanations ranked the most highly ensures that any proved theorems are broadcast immediately, which seemed sensible

given the purpose of the application – to discover proved theorems about quasigroups. In addition, we also experimented with using the complexity measure to further rank Definitions, whereby the base importance of 100 is reduced by the definition's complexity. This means that simpler definitions are more likely to win the competition for broadcast. Not surprisingly, we have found that this produces simpler theories in step-limited sessions. There are many different rating schemes we could experiment with, and we plan to do so.

3.5 Illustrative Results

As mentioned previously, we want to compare the configured framework against the ICARUS system, as described in [5], where the application domain was QG-quasigroups, which are Latin squares with additional axioms, e.g., QG3-quasigroups have the additional axiom that $\forall a,b((a*b)*(b*a)=a)$. In its investigation into QG3 quasigroups, as reported in [5], ICARUS discovered three theorems which were turned into efficiency-gaining implied constraints: (i) $\forall a\,b\,(a*a=b \leftrightarrow b*b=a)$ (ii) $\forall a\,b\,((a*b=b*a) \rightarrow a=b)$ (iii) $\forall a\,b\,((a*a=b*b) \rightarrow a=b)$. Using the configuration described above, in a run of 777 broadcast rounds, lasting 61.3 seconds on a 3.2GHz Intel Pentium IV with 1GB of RAM, our configuration generated 24 Concept broadcasts, after equivalence checking, and created and proved 159 conjectures, including the three above found by ICARUS. These results mean that if our configuration were used to replace the HR system in the ICARUS system, then we would certainly achieve the same results. We have similarly ran the configured framework with QG4 and QG5 quasigroups, and found that it also produced comparable results to ICARUS, although we omit details here.

As an illustrative example, we will describe how our system discovered and proved result (i) above. The background concepts we supplied were the multiplication operator *m(Q,A,B,C)* which states that *A*B=C* for $A,B,C \in Q$; and equality *e(Q,A,A) :- m(Q,A,_,_)*, which states that all elements of Q are equal to themselves. The workspace was configured according to the initial state described above. Examples for the background concepts were given to the ExampleFinder and axioms for *QG3 quasigroups* to the Prover process. For clarity, by *propose* below, we mean that a process made and repeatedly proposed a proposal for broadcast. We also omit complexity information.

In a fairly early processing round, a Definition artefact for *m(Q,A,B,C)* was broadcast. DefinitionReviewer reacted to this definition and confirmed that it was novel, thus proposing it in a Concept artefact with no examples. In the next round, this was broadcast, and ExampleFinder reacted by obtaining examples of the concept from its Prolog database and proposing a Concept with these examples. EquivalenceReviewer reacted to the broadcast by proposing a NewConcept with that definition and this was subsequently broadcast as the concept was not equivalent to any previous concept. EquivalenceReviewer reacted again to this broadcast NewConcept by spawning a clone process to review future Concepts for equivalence with *m(Q,A,B,C)*. In that same round, the broadcast also triggered the DefinitionCreator process encapsulating the *Unify-[0,1,1,2]* method. The process unified the second and third variables of this definition to generate, and propose, a new Definition, [def:m(Q,A,A,B)]. In a similar process to the above, this became a NewConcept and an EquivalenceReviewer process was spawned to compare it to future concepts. The DefinitionCreator process with the *conjoin-[[0,0],[1,2],[2,1]]* method then proposed [def:m(Q,A,A,B), m(Q,B,B,A)]. In later rounds, this was elevated to a Concept and examples were found for it. At this stage, the EquivalenceReviewer process spawned earlier identified that the example set for this concept was identical to that of *m(Q,A,A,B)* and proposed the equivalence conjecture [conj:m(Q,A,A,B):m(Q,A,A,B),m(Q,B,B,A):eq]. This was broadcast in the next round and the *Prover* process reacted by proving the conjecture, using the axioms it was supplied with. The ranking scheme favours explanation artefacts above all others so the proof of this was broadcast. The proven theorem states $\forall a\,b(a*a=b \leftrightarrow a*a=b \wedge b*b=a)$ which is logically equivalent to that above.

The configuration described above was our second major attempt at getting the framework to achieve

results comparable to ICARUS. Our first attempt generated many repeat broadcasts and therefore performed much unnecessary processing. The main reason for this was that the same concept was formed in a number of different ways. Also, there were instances where a chain of definition creation methods applied to a definition resulted in that same definition, creating many repetitive loops. This redundancy meant that the first system took 25,000 rounds (during an hour of running time) to produce the three interesting theorems above, found amongst 15,000 other conjectures. In response, we added the DefinitionReviewer process to stop the repetition. Later, we designed the EquivalenceReviewer and made the distinction between Concept and NewConcept, to remove equivalent concepts. The results from the final version of this configuration represent a vast improvement upon those early results.

4 Configuration 2: Number Theory Conjecture Making

One of the purposes of using computer algebra packages is to gain a better understanding of certain mathematical functions, and the calculation of outputs for inputs can often lead to the formation of conjectures about the function. With the HOMER system, we combined the HR machine learning system with the Maple computer algebra package so that HR could automatically discover conjectures about some user-supplied Maple functions. In [7], we applied this approach to number theory, using some functions which included a Boolean primality test, and $\tau(n)$ and $\sigma(n)$ which calculate the number and sum of divisors of n respectively. We found that, due to the inherent connectivity between concepts in number theory, the main problem was the production of too many conjectures which followed from the definitions. Hence, we enabled HOMER to use the Otter theorem prover as a filter, i.e., any conjectures proved by Otter were discarded, as they were highly likely to follow trivially from the concept definitions.

To produce equivalent results to HOMER, using the GC framework, we built a second configuration on top of the one described above. We used the same importance rating scheme as before, but we changed the artefacts slightly and the processes slightly more. In overview, the new configuration effectively passes conjectures through a series of filters which halt the progress of the conjecture if it fails a test indicating that it will be uninteresting. In addition to filtering, we also wanted to mimic the way in which HOMER splits conjectures into smaller ones, e.g., an equivalence is split into two implication conjectures. To achieve the filtering and splitting through the normal workspace-process mechanism, we changed the Conjecture reasoning artefact to be of the form [conj:D_1:D_2:K:L]. Here, L is a number in $\{0,1,2,3\}$ which indicates the level of conjecture filter through which the conjecture has passed. The levels are as follows: (0) indicates that the conjecture has not been split or filtered, (1) indicates that the conjecture has been split into implicates (2) indicates that the conjecture cannot be proven trivially from details of the background examples and function definitions (3) indicates that the conjecture still cannot be proven after considering some of the other conjectures the user has chosen as axioms – this rules out conjectures which are specialisations of some interesting conjectures identified by the user. All conjectures initially proposed by the processes spawned by EquivalenceReviewer and ImplicationMaker are assigned level 0.

When configuring the concept forming processes, we took into account the infinite nature of the number line. That is, in order to make empirically correct conjectures which relate concepts, the example generation must be sound, i.e., the generation of examples for two logically equivalent concepts should always result in the same finite example set. For efficiency purposes, we restricted our consideration to working just with the numbers 1 to 50, but this introduces some problems. For example, $\sigma(28) = 56$, so, with no further information, the result of $\tau(\sigma(28))$ cannot be calculated. We resolved this problem by storing background function values to cover a much larger range of integers whilst restricting the generation of examples to 1 to 50. We introduced the notion of a *generator variable*, which is one from which all other variables in a definition can be generated, by being the output from a function to

which the inputs are known or can themselves be generated. All background concepts are supplied with relevant function input and output information, and the concept forming production rules use this to avoid inventing ambiguous definitions.

In addition to upgrading the concept forming processes, we made a few changes to some other existing processes. In particular, using the notion of generator variables, DefinitionReviewer was improved to also check definitions for empirical testability. In this configuration, it rejects those definitions for which example sets cannot be soundly generated. Also, we have enabled the ExampleFinder process to use the Maple computer algebra system to calculate examples for background concepts which represent number theory functions. Moreover, the ImplicationMaker process can now be given a minimum example set size, and if a concept has fewer examples than this threshold, no conjectures will be made for that concept. This stops the formation of conjectures such as $\tau(a) = 1 \leftrightarrow \sigma(a) = 1$, as the concepts on both the left and right hand side are only satisfied by the number 1. For this configuration, we implemented two types of Prover process. The level 1 Prover reacts to level 1 conjectures using background examples and function definitions in proof attempts. In contrast, the level 2 Prover reacts to level 2 conjectures and attempts a proof using the same axioms together with axioms supplied by the user. In each case, the prover proposes an Explanation artefact if they are successful and a Conjecture artefact of one level higher if they fail.

We also added processes to split equivalence conjectures into two implications, and a method by which conjectures concerning concepts with small example sets – which are likely to be uninteresting – are avoided. In particular, we added these processes:

RightConjectureSplitter and **LeftConjectureSplitter**
The first of these processes reacts to level 0 Conjecture broadcasts (both *im* and *eq*) by proposing a level 1 *im* conjecture with the same definition list. LeftConjectureSplitter performs similarly, but reverses the members of the definition list, and only in response to *eq* conjectures. This procedure increases the overall number of conjectures but does lead to more pertinent conjectures, e.g., the non-obvious side of an equivalence conjecture.

ConjectureApplicability
This reacts to a Concept with a small set of examples satisfying its definition. It proposes a conjecture that the given examples are the only ones possible for that definition. For example, it would propose [conj:sigma(A,1):e(A,1):eq] on seeing that *1* is the only number for which $\sigma(A) = 1$. These conjectures prevent further development of these concepts in a similar manner to the EquivalenceReviewer.

4.1 Illustrative Results

As for the experiments with the HOMER system, we used the $\sigma(n)$, $\tau(n)$ and $isprime(n)$ background functions together with the notion of equality. We ran the system to completion with a complexity limit of 6. Under a similar experimental set up, as reported in [7], HOMER created 48 concepts, whereas our configuration created 97, of which 38 were found in HOMER's corpus of 48. We identified four reasons why the extra ten concepts were not created by our system. Firstly, three were not produced due to timing differences in the equivalence checking. Equivalence checking works in similar ways in each system, i.e., any new concepts with the same example set as a previous one are discarded and an equivalence conjecture is raised. Hence, the time at which equivalent concepts are proposed determines which is discarded and which is kept. For example, both systems found the following equivalence: $\forall a\, \sigma(a) = 1 \leftrightarrow \tau(a) = 1$. However, our configuration chose to explore the $\tau(a) = 1$ branch, discarding $\sigma(a) = 1$, whereas HOMER did the opposite.

Secondly, variable ordering accounted for some of the difference in the results. Our system considered $\tau(a) = b \wedge \sigma(b) = a \wedge \tau(b) = b$ to be equivalent to $\sigma(a) = b \wedge \tau(a) = a$. Depending on how the

variables are ordered, the example sets of each can be written [[1,1],[2,3]] or [[1,1],[3,2]]. This ordering difference led to HOMER treating this concept as new whereas our system discarded it (which highlights a deficiency in the underlying HR system). Thirdly, one concept produced by HOMER, namely $\tau(a) = 2 \wedge isprime(2) \wedge \sigma(2) = a$, would not be considered valid by the DefinitionReviewer in our system. In this formula, a is always the result of $\sigma(2)$, i.e., 3. So, this formula is essentially variable free, and hence not really a concept definition (which again highlights a deficiency in HR). Finally, two concepts were not produced by our configuration due to differences in its calculation of complexity – HOMER counts re-used concepts only once towards the complexity, whereas our system counts them with multiplicity. Hence, our system counted certain concepts as complexity 7, whereas HOMER counted them as complexity 6. When the configuration is run with a complexity limit of 7 then these concepts are produced.

On the other hand, our system produced a number of concepts that were not produced by HOMER. Some of these were ignored by HOMER for good reason, e.g., our system produced several concepts in the form $\sigma(a) = b \wedge (\exists c\,(\sigma(b) = c)))$. Here, c indicates the existence of a functional result for sigma(b). Such formulae are uninteresting extensions to earlier concepts. However, other concepts produced by our system and not HOMER were valid and potentially interesting. It is difficult to determine why HOMER missed these concepts, and we are still investigating this. Looking instead at the conjectures made by the two systems, we note that our configuration created 669 level 1 conjectures, i.e., after splitting valid equivalence conjectures into implicates. The first filter – which excluded conjectures if they were provable from the background definitions – removed 331 conjectures, leaving 338. By comparison, the HOMER system created 137 conjectures, of which Otter proved 43, using the same methods, leaving 94. The main reason for the difference in these numbers is the relative number of concepts that the two systems produced, which naturally meant that the configured framework produced more conjectures.

In a similar manner to that adopted for HOMER, we reviewed the 338 remaining conjectures. In particular, we looked at the first 10 conjectures and added the following as axioms (after re-combining some of the split implications):

$$\forall a\,(\sigma(a) = 1 \rightarrow a = 1) \qquad \forall a\,(\sigma(a) = a \leftrightarrow \sigma(a) = 1)$$
$$\forall a\,(\tau(a) = 1 \rightarrow a = 1) \qquad \forall a\,(\tau(a) = a \rightarrow (a = 1 | a = 2))$$
$$\forall a\,(\tau(a) = 2 \leftrightarrow isprime(a))$$

After running the process again, our configuration filtered out all but 66 conjectures (a reduction of around 90%). A similar kind of reduction was achieved by the users of HOMER, hence we can claim that our configured generic framework is capable of producing comparable results to the bespoke, ad-hoc HOMER system. Importantly, the most interesting (proved by hand) result from the HOMER experiments was: $isprime(\sigma(a)) \rightarrow isprime(\tau(a))$, which our system re-discovered.

5 Investigating Specialisations of Finite Algebras

In addition to showing that our Global Workspace approach can be used to re-implement existing combinations of reasoning systems, we can also use the system for novel applications, one of which we describe here. In finite algebra, it is common for a specialisation of an algebra to be studied in its own right, for instance, Cyclic groups, Abelian quasigroups [22], and so on. As there are thousands of possible specialisations, it is an interesting question to try to predict which one it would be fruitful to study.[1] It is naturally very difficult to predict in advance whether a domain will be worthy of study, so we restrict ourselves to an automated reasoning setting. In particular, we say that a specialisation of an algebra is

[1]Note that Toby Walsh originally suggested that we perform a similar analysis, using the HR theory formation system, but this project wasn't undertaken.

more interesting than another if a theory formed about the former specialisation contains theorems which on average are more difficult to prove by an automated prover than the latter.

The GC configuration for this application was similar to that used for quasigroup theorem discovery above, with a small number of amendments. Firstly, we introduced a MaceAxiomPopulator process, which uses the Mace model generator to produce background concepts and examples for the domain. The process reacts to the broadcast of a BackgroundAxiom by generating models for that axiom up to a user defined algebra size limit. It extracts the algebra operators from the models and generates example sets for them by interpreting the models. It then proposes these operators and examples sets together as a BackgroundConcept artefact, which initiates theory formation as before. This new process allows us to perform theory formation with only the axioms of the domain as a starting point. This greatly improves efficiency as, previously, a user was required to generate examples independently and convert them into an appropriate format. Our next amendment was to configure the system with two Prover processes; one of which uses the axioms of the domain in attempts to prove conjectures and another which holds no axioms. If this second prover can prove a conjecture then we know it is trivially true. This allows us to distinguish, potentially, more interesting conjectures. If the trivial prover can prove a conjecture then we know that the Prover with the domain axioms will also be able to prove it. Consequently, we weight the importance rating scheme toward the trivial proofs to ensure they are broadcast. We also introduced a MaceRefuter process which appeals to Mace to find counter-examples to broadcasts conjectures. This refutation process has access to higher orders of algebra and so can sometimes refute conjectures that have been generated by being empirically true for smaller orders.

Starting with a fairly unstructured algebraic domain such as quasigroup theory, we used the system to generate 100 concepts which can be interpreted as specialisations, expressed in a Prolog style in terms of the multiplication operator only. Then, for each specialisation, we started GC with the original axioms of the algebraic domain, with the specialisation definition also added as an axiom. We then ran the system until it generated and proved 100 implication theorems, and recorded the average length of the proof produced by the Prover9 prover. In Table 1, we present the top ranked specialisations from three domains: semigroups, quasigroups and star algebras (which have the single axiom: $\forall x,y,z((x*y)*z = y*(z*x))$, see [10]). We also performed a correlation analysis over the specialisations. In particular, for each specialisation, we calculated a crude measure of the complexity of its definition by adding the number of existential variables to the number of multiplication predicates present (note that this value is presented in the comp. column of Table 1. We correlated this with the rank of the specialisation in Table 1, using the R^2 goodness-of-fit measure. A high level of correlation would indicate that it is possible to predict to some extent which algebra specialisations are going to be interesting, without having to perform an in-depth theory for each specialisation. We found a positive correlation of 0.25, 0.03 and 0.15 in the quasigroup, semigroup and star algebra domains respectively. While the lack of correlation with semigroups is surprising, it was not surprising that for the other two algebras, more complex definitions produce more complex (in terms of average proof length) theories.

Bearing in mind that the balance of syntactic simplicity and semantic complexity is often regarded as a maxim in pure mathematics – for instance, consider Fermat's Last Theorem, which is very easily stated, but very difficult to prove – we looked for a specialisation of each algebra which had a high rank, yet a low complexity for its definition. In each case, we found suitable candidates. In particular, for semigroups, the specialisation (m(A,_B,_C,_B)),\+(m(A,_C,_D,_B)) has a definitional complexity of only 5 (it has three free variables, namely _B, _C and _D, and two occurrences of the multiplication operator), but it is ranked 92nd out of the 100 specialisations, as it produces a theory with a proof length of 6.54 on average (with the best achieved being 7.76). The definition describes the notion of semigroups for which an element is the right identity of an element that does not appear on its row of the multiplication table. For quasigroups, the specialisations ranked 94th and 100th each have short definitions with a complexity of 7. The 100th ranked is ((m(A,_B,_C,_C),m(A,_D,_D,_D)),\+(m(A,_E,_E,_B))), which alludes to the notion

Semigroups

rank	apl	comp.	definition
91	6.50	7	((m(A,_B,_C,_B)),\+(((m(A,_B,_D,_B)),\+(m(A,_D,_E,_D))))
92	6.54	5	((m(A,_B,_C,_B)),\+(m(A,_C,_D,_B)))
93	6.54	10	((m(A,_B,_C,_D)),\+(((m(A,_C,_E,_C)),\+(((m(A,_C,_F,_C)),\+(m(A,_F,_G,_C))))))
94	6.59	9	((m(A,_B,_C,_D)),\+(((m(A,_E,_F,_E)),\+(m(A,_G,_G,_F))))
95	6.63	9	((m(A,_B,_C,_D)),\+(((m(A,_E,_D,_F)),\+(m(A,_G,_G,_E))))
96	6.64	8	((m(A,_B,_C,_D)),\+(((m(A,_D,_E,_D)),\+(m(A,_E,_F,_E))))
97	6.67	10	((m(A,_B,_C,_D)),\+(((m(A,_E,_E,_C)),\+(((m(A,_C,_F,_C)),\+(m(A,_F,_G,_C))))))
98	6.68	10	((m(A,_B,_C,_D)),\+(((m(A,_E,_F,_G)),\+(m(A,_H,_F,_H))))
99	6.80	11	((m(A,_B,_C,_D)),\+(((m(A,_E,_E,_F)),\+(((m(A,_F,_G,_F)),\+(m(A,_G,_H,_F))))))
100	7.76	8	((m(A,_B,_C,_B)),\+(m(A,_D,_D,_C))),m(A,_E,_F,_C)

Quasigroups

rank	apl	comp.	definition
91	6.65	10	((m(A,_B,_C,_D)),\+(((m(A,_E,_F,_F)),\+(((m(A,_E,_G,_E)),\+(m(A,_G,_E,_E))))))
92	6.81	10	((m(A,_B,_C,_D)),\+(((m(A,_E,_E,_E)),\+(((m(A,_F,_G,_F)),\+(m(A,_G,_G,_E))))))
93	6.93	9	((m(A,_B,_C,_C)),\+(m(A,_D,_D,_B))),m(A,_E,_E,_E),m(A,_F,_B,_F)
94	6.98	7	((m(A,_B,_B,_B),m(A,_C,_D,_C)),\+(m(A,_E,_E,_D)))
95	7.12	9	(((m(A,_B,_C,_B)),\+(m(A,_C,_D,_D))),\+(m(A,_E,_E,_C))),m(A,_F,_F,_F)
96	7.85	8	(((m(A,_B,_C,_B)),\+(m(A,_D,_D,_C))),\+(((m(A,_E,_C,_E)),\+(m(A,_C,_E,_E))))
97	8.05	9	(((m(A,_B,_C,_C)),\+(m(A,_D,_D,_B))),\+(((m(A,_E,_E,_E)),\+(m(A,_F,_F,_B))))
98	8.21	9	((m(A,_B,_C,_B)),\+(((m(A,_C,_D,_D)),\+(((m(A,_E,_C,_E)),\+(m(A,_F,_F,_C))))))
99	8.67	7	(((m(A,_B,_C,_B)),\+(m(A,_C,_D,_D))),\+(m(A,_E,_E,_C)))
100	9.61	7	((m(A,_B,_C,_C),m(A,_D,_D,_D)),\+(m(A,_E,_E,_B)))

Star algebras

rank	apl	comp.	definition
91	6.40	10	(((m(A,_B,_C,_D)),\+(m(A,_C,_E,_C))),\+(((m(A,_F,_F,_F)),\+(m(A,_C,_G,_F))))
92	6.42	11	((m(A,_B,_C,_D)),\+(((m(A,_E,_F,_G)),\+(((m(A,_G,_H,_G)),\+(m(A,_G,_G,_H)))))))
93	6.42	12	((m(A,_B,_C,_D)),\+(((m(A,_E,_F,_G)),\+(((m(A,_H,_I,_F)),\+(m(A,_I,_F,_F)))))))
94	6.51	9	((m(A,_B,_C,_D)),\+(((m(A,_E,_E,_D)),\+(((m(A,_F,_D,_F)),\+(m(A,_F,_F,_D)))))))
95	6.51	10	((m(A,_B,_C,_D)),\+(((m(A,_E,_C,_E)),\+(((m(A,_F,_F,_F)),\+(m(A,_C,_G,_F)))))))
96	6.60	9	((m(A,_B,_C,_D)),\+(((m(A,_E,_F,_G)),\+(m(A,_F,_G,_G)))))
97	6.73	8	((m(A,_B,_C,_D)),\+(((m(A,_E,_F,_E)),\+(m(A,_E,_E,_F)))))
98	7.10	10	((m(A,_B,_C,_D)),\+(m(A,_E,_E,_C))),((m(A,_F,_C,_C)),\+(m(A,_G,_G,_F)))
99	7.46	9	((m(A,_B,_C,_D)),\+(m(A,_E,_E,_D))),m(A,_D,_D,_F),m(A,_F,_D,_F)
100	8.55	8	((m(A,_B,_C,_B),m(A,_D,_E,_C)),\+(m(A,_F,_F,_C)))

Table 1: Highest ranked specialisations for quasigroups, semigroups and star algebras. apl is the average proof length, comp. is the sum of the variable and predicate counts.

of quasigrioups having at least one idempotent element , and an element which is the left identity of some element but does not appear on the central diagonal of the multiplication table ,i.e. it is not the square of some other element. This specialisation produced the theory with the highest average proof length of 9.61. Interestingly, the specialisation ranked 94th is very similar to the 100th but describing an element that is a right identity. For star algebras, the 97th ranked specialisation is interesting. It represents a specialisation for which an implication should hold. Namely, for all elements x and y, if $x * y = x$ then $y * y = x$. We further investigated this specialisation by drawing a graph to indicate how it relates to the other star algebra specialisations. In figure 1, the arrows all indicate an implication conjecture/theorem that the algebras satisfying the definition of the first specialisation all satisfy the definition of the second specialisation. Solid arrows are all true, i.e., can be proved from the background axioms. The arrows with a dotted line are only empirically true, i.e., Prover9 could not prove the implications, and MACE could not find counterexamples, even when we re-ran them for several minutes. These are therefore interesting open conjectures and we will continue to investigate them.

6 Conclusions and Further Work

We have used the cognitive science theory of the Global Workspace Architecture (GWA) to devise and implement a generic computational framework within which disparate reasoning systems can be fruitfully combined. We have shown how the framework can be configured in order to achieve similar results to the ICARUS and HOMER combined reasoning systems which were implemented in an ad-hoc way for specific purposes. We have also demonstrated a novel application of a configuration of our system to investigating algebra specialisations. We applied this system to three classes of finite algebra and, in

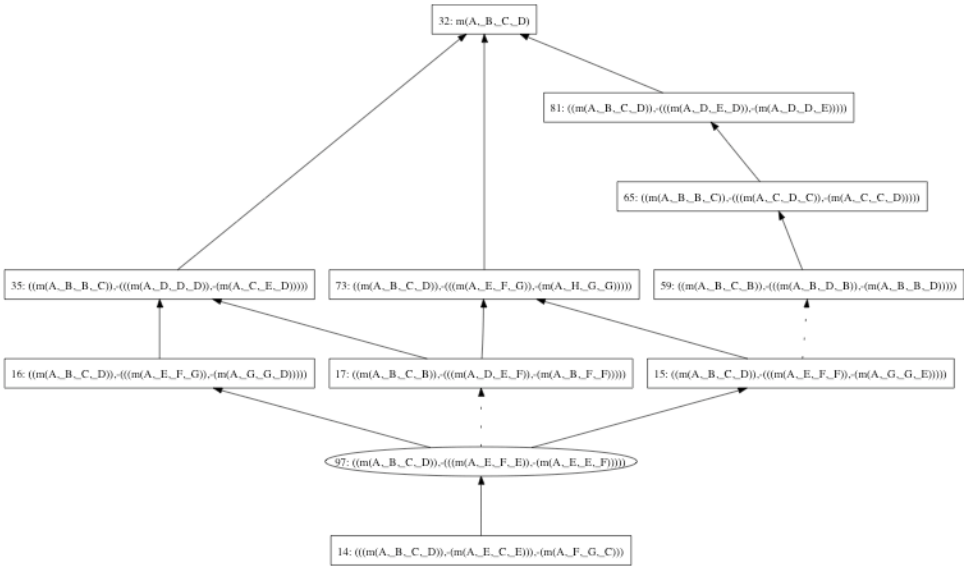

Figure 1: Graph showing the implication relationships between the specialisation of interest (circled) and other specialisations of star algebras.

each case, it identified a syntactically simple, although semantically quite complex, specialisation of that algebra that may be worthy of further mathematical investigation.

While the value of using disparate AI systems in combination has been repeatedly demonstrated, the combined reasoning systems themselves are in general ad-hoc and purpose-specific. We believe that in order for the field of combined reasoning to progress, there needs to be much more research into generic frameworks within which different AI problem solving methods can be integrated. We hope that the framework presented here will eventually become one of many that researchers can turn to when they need to harness the power of more than one reasoning approach in combination.

While the functionality that the framework provides is fairly straightforward, we have shown that it is possible to configure it to achieve behaviour which combines induction, deduction and calculation. Producing similar results to those of previous ad-hoc systems has been the first milestone in the development and testing of the framework. We intend to further demonstrate the potential of the system by adding constraint solving processes, so that the full functionality of the ICARUS system can be achieved. We will further add model generation, SAT-solving and planning processes so that results similar to those from other combined reasoning systems – in particular the TM system [9] and the algebraic classification system described in [8] – can be produced. We also want to show that using the framework can reduce some of the development time in building combined reasoning systems, and we will do this by configuring the framework to apply to new problems by achieving novel reasoning combinations. In addition, we will consider configurations of similarly skilled processes working as a portfolio.

While much of our effort to date has been spent on writing processes which achieve theory formation like the HR system, it is important to note how differently the configured frameworks presented here perform when compared to the very linear approach in HR. In particular, one of the potential benefits of a framework based on the Global Workspace Architecture is the ability to distribute the reasoning it performs over many processors. We plan to implement a distributed version of the framework, and to determine the efficiency gains to be made.

Currently, our framework operates in a synchronous manner, where all processes are given time to react before a new round is begun. Relaxing this requirement may yield efficiency gains as some

processes react much faster than others. It may be possible, for instance, for the reasoning to proceed for many rounds while a difficult theorem is proved on one processor. Furthermore, all processes are given equal resources, and these could be adapted based upon perceived difficulty or importance of a particular process' task. We have only scratched the surface of the experimental possibilities that the framework provides.

We chose quite straightforward importance rating schemes for the two configurations in order to achieve similar results to HOMER and ICARUS. However, it is not clear that these are the best possible schemes. In our experience with the HR system, it often pursues avenues of a theory which ultimately lead to little of interest. It will be interesting for us to experiment with different setups of processes and schemes to see which configurations of the framework are more fruitful, or which solve problems more efficiently. We have only touched upon grading importance based upon the specifics of a proposal. We could extend the sophistication of processes so that they consider other factors such as previous broadcasts by other processes or by specialist control processes. This could also encompass such things as information from the user or some external guiding system giving real-time feedback about the progress of the system.

In addition to using the framework to advance the field of combining reasoning systems, we hope to use it to shed light on the area of Global Workspace Architectures. We have already had to deviate on two occasions from the standard theory of the GWA, in particular by allowing processes to terminate themselves and to add new processes to the workspace. With a computational model of the GWA, we hope to make the theory more concrete and more useful as a tool both for cognitive scientists and AI researchers.

7 Acknowledgements

We would like to thank the anonymous reviewers for their thorough and helpful comments.

References

[1] B Baars. *A cognitive theory of consciousness*. Cambridge University Press, 1988.

[2] B Baars. The conscious access hypothesis: Origins and recent evidence. *Trends in Cognitive Science*, 6(1):47–52, 2002.

[3] C Benzmüller, V Sorge, M Jamnik, and M Kerber. Combined reasoning by automated cooperation. *Journal of Applied Logic*, doi:10.1016/j.jal.2007.06.003, 2007.

[4] M Carlsson, G Ottosson, and B Carlson. An open-ended finite domain constraint solver. In *Proc. Programming Languages: Implementations, Logics, and Programs*, 1997.

[5] J Charnley, S Colton, and I Miguel. Automatic generation of implied constraints. In *Proceedings of the 17th ECAI*, 2006.

[6] S Colton. *Automated Theory Formation in Pure Mathematics*. Springer-Verlag, 2002.

[7] S Colton. Automated conjecture making in number theory using HR, Otter and Maple. *Journal of Symbolic Computation*, 2004.

[8] S Colton, A Meier, V Sorge, and R McCasland. Automatic generation of classification theorems for finite algebras. In *Automated Reasoning — 2nd International Joint Conference, IJCAR 2004*, pages 400–414, 2004.

[9] S Colton and A Pease. The TM system for repairing non-theorems. In *Proceedings of the IJCAR'04 Disproving workshop*, 2004.

[10] S Colton, P Torres, P Cairns, and V Sorge. Managing automatically formed mathematical theories. In *Proceedings of MKM*, 2006.

[11] A Faustino da Silva and V Santos Costa. Design, implementation, and evaluation of a dynamic compilation framework for the yap system. In *ICLP*, pages 410–424, 2007.

[12] M Fisher. An open approach to concurrent theorem-proving. *Parallel Processing for Artificial Intelligence*, 3, 1997.

[13] M Fisher. Characterising simple negotiation as distributed agent-based theorem-proving. In *Fifth International Conference on Multi-Agent Systems (ICMAS)*, 2000.

[14] M Fisher and M Wooldridge. Distributed problem-solving as concurrent theorem-proving. In *Multi-Agent Rationality - MAAMAW*, 1997.

[15] S Franklin. Ida: A conscious artifact? *Consciousness Studies*, 10:47–66, 2003.

[16] S Franklin and A Graesser. A software agent model of consciousness. *Consciousness and Cognition*, 8:285–301, 1999.

[17] A Kelemen, S Franklin, and Y Liang. Constraint satisfaction in "conscious" software agents - a practical application. *Applied Artificial Intelligence*, 19:491–514, 2005.

[18] W McCune. Prover9. http://www.cs.unm.edu/ mccune/prover9/.

[19] W McCune. The OTTER user's guide. Technical Report ANL/90/9, Argonne National Laboratories, 1990.

[20] W McCune. A Davis-Putnam program and its application to finite first-order model search. Technical report, ANL/MCS-TM-194, 1994.

[21] A Newell. Some problems of basic organization in problem-solving systems. In *Proceedings of the second conference on self-organizing systems*, page 393.342, 1962.

[22] J Schwenk. A classification of Abelian quasigroups. *Rendiconti di Matematica, Serie VII*, 15:161–172, 1995.

[23] M Shanahan and B Baars. Applying global workspace theory to the frame problem. *Cognition*, 98 no.2:157–176, 2005.

[24] Waterloo Maple. *Maple Manual at http://www.maplesoft.on.ca*.

Automated Parameterisation of Finite Algebras

Simon Colton
Department of Computing
Imperial College, United Kingdom

Volker Sorge
School of Computer Science
University of Birmingham, United Kingdom

Abstract

In previous work we have designed an automatic bootstrapping algorithm to classify finite algebraic structures by generating properties that uniquely describe and discriminate different equivalence classes. One of the drawbacks of the approach was that during the classification a large number of different discriminating properties were generated. This made it particularly difficult to compare classifying properties for different sizes of algebraic structures. To minimise the overall number of properties needed we have now experimented with parameterising structures by counting the number of elements with particular properties. Isomorphism classes are then discriminated by the different number of elements with the same property. With this new approach we can now classify large numbers of algebraic structures using only a small number of properties. We were able to construct and prove parameterisations for algebraic structures like loops of size 6 and groups of size 8. However, the approach is currently limited as translating counting arguments into pure first order or propositional logic, often makes for prohibitively long problem formulations.

1 Introduction

In previous work, we have automated the construction of fully verified qualitative classification theorems for finite algebraic structures of a given size, satisfying a given axiom set. For instance, given the axioms of group theory and specification of size 6, our system would: invent the concepts of Abelian and non-Abelian groups; prove that these concepts classify groups of size 6 into isomorphism classes; and prove that no other isomorphism classes exist. We use a bootstrapping technique which involves the combination of many automated reasoning systems, including theorem provers, model generators, SAT solvers, machine learning and computer algebra systems. Without going into detail (see [3] for this), a major part of the bootstrapping algorithm is to take two non-equivalent examples of the algebra and find an invariant property that one has and the other doesn't, which proves that they cannot be equivalent. When such a property is found, a branching point on a classification tree is produced. The leaves of the final tree represent concepts which define equivalence classes. Using this approach, we have been able to construct the first qualitative classification theorems for loops and quasigroups up to both isomorphism [3] and isotopism [6], e.g., for loops of size 6, which have 109 isomorphism classes and 22 isotopism classes.

A quasigroup is generally a non-associative structure with a Latin square property. Loops have in addition a unit element. Isotopism is an equivalence relation, which is a generalisation of isomorphism, where two quasigroups $(Q_1, \circ)$ and $(Q_2, \star)$ are isotopic if and only if there exist three bijections α, β, γ, such that $\alpha(x) \star \beta(y) = \gamma(x \circ y)$ for all $x, y \in Q_1$. The quantitative classifications for quasigroups and loops up to size 10 with respect to isomorphism and isotopism, i.e., the number of different equivalence classes, have been known for some time [7, 4]. Similar to establishing the quantitative results, qualitative classification can realistically only be done automatically, considering the sheer number of structures to consider. For instance, there are 1411 isomorphism classes for quasigroups of order 5 but already 1130531 different classes of size 6 quasigroups.

While this approach was successful, there are three main drawbacks to continuing in this way, namely (i) the number of equivalence classes (both isomorphism and isotopism) explodes as the size of the

Sutcliffe G., Colton S., Schulz S. (eds.); Proceedings of ESARM 2008, pp. 17-22

algebra increases (e.g., there are more than 100 million loops of size 8 up to isomorphism), (ii) the reasoning tools we use have difficulty with certain required theorems about algebras of size 8 or more, and (iii) the classification trees produced for larger sizes bear little resemblance to those for smaller sizes for a given axiomatisation. With respect to this final point, we have considered data-mining the trees to find concepts which appear in multiple classification trees. However, it seems more sensible to first experiment with more proactive approaches to generating classification results, by initially imposing a tree structure which uses common concepts, before filling in the rest of the tree.

Drawing on existing mathematical results, we note that there are 14 groups of size 8 or smaller up to isomorphism. Moreover, they are usually classified either in terms of a parameterisation consisting of a family that they belong to and their size, e.g., the cyclic group of order 5 (C_5), the dihedral group of order 8 (D_4), etc., or in terms of a cross product of such parameterised groups, e.g., $C_4 \times C_4$. While we have performed some experiments with cross products, we concentrate here on the automatic generation of parameterisations of finite algebraic structures. Our approach aims to look at parameterisations of algebraic structures in terms of a list of set sizes, where each set contains elements of the structures with particular properties. For instance, groups up to size 6 can be classified up to isomorphism by using a parameterisation in terms of two coefficients: the number of elements and the number of self-inverse elements (x s.t. $x = x^{-1}$).

This choice is motivated by the desire to use counting, as this is an important tool in producing classification results, yet has been missing from our previous approaches. The approach also reduces the number of different properties needed to fully classify structures of a particular size. Moreover, we have chosen to focus our first experiments on concepts which count the number of elements of a particular type because there is a standard – if cumbersome – way of formalising such set-size results in first order logic, which enables us to get proofs of our results from automated theorem provers. In sections §2 and §3, we describe how we generated and then proved parameterisations of loops, groups and quasigroups. While we concentrate purely on counting elements here, we have also already experimented with counting substructures [6] and will consider counting sub-algebras and comparing their sizes in future work. In §5, we discuss further future work and, in particular, we use the results from our experiments presented in this paper to suggest improvements to our bootstrapping approach.

2 Generating Classifying Parameterisations

Suppose we start with a set of algebraic structures $A = \{A_1, \ldots, A_n\}$ and a list of *element-type concepts* $C = \{c_1, \ldots, c_k\}$. An element-type concept is a boolean test on an element in an algebraic structure, for instance whether the element is idempotent ($x * x = x$). We then define the profile of a given $a \in A$ with respect to C as: $P(a) = \langle |\{x \in a : c_1(x)\}|, \ldots, |\{x \in a : c_k(x)\}| \rangle$. We further say that C represents an *element-type parameterisation* of A if no pair of algebraic structures in A have the same profile. If A contains representatives of each isomorphism class up to a certain size n for a specific algebraic structure, then the parameterisation can be used to classify that structure up to size n, and this classification can be proved (see next section).

We constructed such classifying parameterisations for loops up to size 5, groups up to size 8 and quasigroups up to size 4 as follows (for clarity, we will use the groups up to size 8 as an illustrative example). We started with a set of groups, A, with each member being a representative of a different isomorphism class, and all the isomorphism classes covered. We used A in the background knowledge for the HR automated theory formation system. Details of how HR works can be found in [2]. For our purposes here, HR is a concept generator, i.e., given some background concepts such as the multiplication operator in groups, HR will invent concepts such as commutativity, etc. In particular, HR is able to generate hundreds of element-type concepts.

Domain	Max size	Classes	Achieved	Steps	Element-types	Classifiers	Time (s)
Groups	8	14	14	1000	32	3	28
Loops	5	11	11	1000	32	3	10
Loops	6	120	86	40000	736	11	2903
Quasigroups	4	42	42	10000	523	5	215

Table 1: Details of parameterisations achieved in loop, group and quasigroup theory

We initially ran HR for 1000 theory formation steps. This value could be increased if no full classification could be found in the given number of steps. Each step attempts to construct a new concept, but may result in a conjecture being made instead. From the resulting theory, we extracted the set, C, of element-type concepts and we used these to automatically construct a parameterisation P as follows: The first concept in the parameterisation list is chosen as the overall size of the algebraic structure, largely for reasons of comprehensibility. We then check the parameterisation against A, and remove from A any structures a for which the profile of a is different from all the others. We then iteratively add to P the concept $c \in C$ which differentiates the largest number of pairs of structures from A. Note that we say c differentiates a_1 and a_2 iff $|\{x \in a_1 : c(x)\}| \neq |\{x \in a_2 : c(x)\}|$. Each time a new concept is added, P is checked against A, and – as before – any structure which has a unique profile is removed. This iteration continues until either A is empty (in which case a full parameterisation has been constructed), or there are no concepts left to try. In the output, each structure a is presented with only the concepts needed to distinguish it from the others. The conjunction of this set of concepts is a classifying concept for the isomorphism class represented by a.

We ran the same experiment for loops up to size 5. For quasigroups up to size 4, we increased the number of theory formation steps to 10000, and for loops up to size 6, we increased it to 40000. The results are presented in table 1.

We see that the method was able to produce full parameterisations in a reasonable time (on a 2.1GHz machine) for the groups, quasigroups and loops to size 5 datasets. However, it only achieved a partial classification of 86 of the 120 loop classes up to size 6. Note the the *Element-types* column above describes the total number of element-types produced by HR, while the *Classifiers* columns describes the number of these which were used in the parameterisation. The group theory parameterisation was particularly simple, in terms of counting 3 elements types, namely (i) elements themselves (ii) self-inverse elements and (iii) elements which appear on the diagonal of the multiplication table. The sizes 1 to 5 loop theory parameterisation also required counting only 3 element types: (a) elements themselves (b) elements on the diagonal of the multiplication table and (c) elements, x such that $\exists y\ (y*x = id \wedge y*y = x)$. We think it is an achievement to be able to classify all 42 quasigroups up to size 4 by counting only 5 element types, and to classify 86 of the 120 loops up to size 6 by counting 11 element types.

3 Proving Isomorphism Classes

As we saw above, for each algebraic structure satisfying a set of axioms $\mathscr{R}$, HR produces a conjunction of concepts which can be used to classify the structure. Each concept expresses a boolean test of the structure, namely whether the number of elements with a particular logic property is a particular number. For example, HR returns the cyclic group G of order 4 as the multiplication table given below, together with the axioms of group theory and the single property $2 : x^{-1} = x$, i.e., that there are exactly two self-inverse elements in G.

To prove that the conjunctions of set sizes represent classifying concepts, we reuse some of the

technology implemented for the original bootstrapping algorithm as presented in [3] and its adaptations to work with satisfiability modulo theories solvers detailed in [5].

G	a	b	c	d
a	a	b	c	d
b	b	a	d	c
c	c	d	b	a
d	d	c	a	b

In a first step, we translate the set-size properties into full first order logic by expressing the counting argument in a formal way. For instance, in our example we define a property P as

$$\exists x,y.x \neq y \wedge x^{-1} = x \wedge y^{-1} = y \wedge (\forall z.z^{-1} = z \rightarrow (z = x \vee z = y)).$$

We then need to prove two types of problems: (1) proving that the given conjunction of set-size properties is an invariant under isomorphism for a particular type of algebraic structure and therefore serves as a discriminant, and (2) that the discriminant uniquely defines an isomorphism class for algebraic structures of a given size.

Problems of type (1) are formalised independently of the size of the structures that were involved during the generation of the properties in question. Thus the property is shown to be a discriminant for structures of arbitrary size. The problems are easy to formalise as $\forall A_1,A_2.\mathscr{R}(A_1) \wedge \mathscr{R}(A_2) \wedge P(A_1) \wedge \neg P(A_2) \rightarrow A_1 \ncong A_2$, where $\mathscr{R}$ is as mentioned above the set of axioms describing the algebraic structure in question. They can be expressed in first order logic by considering the sets A_1 and A_2 as arbitrary but different constants and formulating their axiomatisations with disparate operations. Proving these theorems is relatively easy and we used the first order prover Spass [8]. Problems of type (2) are less trivial since they are essentially second order problems: we have to show that all algebraic structures that have property P are also isomorphic to the representant, in our case G, or more formally:

$$\forall A.[\mathscr{R}(A) \wedge P(A)] \rightarrow [\exists \phi.\text{bijective}(\phi) \wedge \text{homomorphic}(\phi) \wedge \phi(A) = G].$$

However, since we are in a finite domain, we can explicitly formulate the problem in propositional logic: We give G in terms of its elements and multiplication table and then formulate all possible bijective mappings from an arbitrary structure A onto the elements of G. However, since the number of mappings to consider is $n!$, where n is the size of the structures A and G, the technique quickly becomes infeasible, even for small n. We therefore use a computer algebra device by restricting the mappings to consider a generating system of G, i.e., a set of elements that can generate all other elements of the structure together with all generating equations. In our example, G is the cyclic group of size 4, and thus the generating system is of the form: $\langle \{c\}, \{a = ((c*c)*c)*c, b = c*c, c = c, d = (c*c)*c\} \rangle$. The number of bijective mappings to consider is then 4 instead of $4! = 24$. (For a detailed discussion of these techniques see [5].) While the problem formulation can still be relatively lengthy, we found that we could solve problems up to size 8 using CVC-3 [1].

4 Results and Limitations

In our experiments, we were successful in fully automatically generating and proving the necessary theorems for quasigroups of up to size 4, loops up to size 5 (as presented in §2), and groups up to size 8. We could also show 86 loops 6 problems for which HR could return parameterisations. The most challenging bit when solving these problems was to produce the problem formulations due to their size. However, once formulations could be produced they were shown to be valid by CVC-3 in less than 1 minute.

Despite several optimisations to our routines to more efficiently produce larger problem formalisations, it is currently infeasible to even generate problems beyond those for size 8 groups and size 6 loops. This has essentially two reasons:

Firstly, we use a naive formalisation for counting arguments in first order logic. For example, one of the properties that uniquely determines one of the loops 6 isomorphism classes is that there exists exactly three elements b, such that

$$\exists c,d,e\,(d*d=c\wedge\neg((\exists f\,(f*f=d)))\wedge b*b=e\wedge e*e=c) \tag{1}$$

To express this argument in standard first order logic we have to explicitly state that

1. there exist three elements that have property (1),
2. these three elements are all different, and
3. there do not exist any other elements with property (1).

The latter is formulated in first order logic by stating that all other elements that have property (1) have to be equal to one of the three original ones. Thus (1) becomes the following larger formula:

$$\begin{aligned}\exists x_0,x_1,x_2 \quad & (x_0\neq x_1)\wedge(x_0\neq x_2)\wedge(x_1\neq x_2)\\ \wedge\quad & \exists c,d,e\,(d*d=c\wedge\neg((\exists f\,(f*f=d)))\wedge x_0*x_0=e\wedge e*e=c)\\ \wedge\quad & \exists c,d,e\,(d*d=c\wedge\neg((\exists f\,(f*f=d)))\wedge x_1*x_1=e\wedge e*e=c)\\ \wedge\quad & \exists c,d,e\,(d*d=c\wedge\neg((\exists f\,(f*f=d)))\wedge x_2*x_2=e\wedge e*e=c)\\ \wedge\quad & \forall y\,(\exists c,d,e\,(d*d=c\wedge\neg((\exists f\,(f*f=d)))\wedge y*y=e\wedge e*e=c)\\ & \qquad\rightarrow(y=x_0)\vee(y=x_1)\vee(y=x_2))\end{aligned} \tag{2}$$

The resulting problems are beyond the power of current first order systems, even for algebraic structures of small size (cf. [5]). As alternative we therefore employ SMT solvers, that can deal much more effectively with large problems in finite domains. However, they generally require ground formulas as input. While they can be simply generated by eliminating the quantifiers over the finite domain, formulas and problems become very quickly unwieldily large.

This problem can be partially overcome as CVC-3 allows for problem formulations involving quantifications, where the quantifiers are restricted over a type that represents the finite set of our algebraic structure. Unfortunately, CVC-3's calculus is incomplete for quantified input formulas, which often leads to a non-conclusive result when given problem formulations containing quantifiers. Thus the only alternative is again to supply the quantifier free problem. But, for example, eliminating the quantifiers of formula (2) over a domain of six elements leads to a formula that is of size 61 MB. And while CVC-3 can deal with problems of several hundred MB in size, generating the actual input becomes quickly infeasible. We therefore currently adopt the strategy to only generate fully grounded problems if CVC-3 can not find a proof given the quantified problem. Unfortunately our experience so far shows that for larger problems CVC-3 only rarely succeeds on the quantified problems.

5 Conclusions and Future Work

We have shown that it is possible to both derive and prove novel element-type classifying parameterisations of algebraic domains. These classifications have an advantage of simplicity and homogeneity over our previously constructed classifications, because they require understanding of only a handful of

element-type concepts. Also, the classification of smaller algebraic structures share concepts with those of larger structures: another improvement on our previous results.

This work suggest the following improvement to our existing bootstrapping method [3]. Given an axiomatisation of an algebraic domain, A, we will attempt to build a classification tree for size 1 algebras, then size 2, and so on, until computational limits are reached. The method should proceed as usual, but only invent invariants which count the size of sets of elements. Furthermore, suppose that every time the tree for algebras of size n is produced, the system adds to a global list, I, all the new invariant properties it has needed to invent to achieve the classification. Then, before attempting to produce the classification tree for size $n+1$, the system should pro-actively check to see whether any p and k such that $p \in I$ and $k \in \{0,\ldots,n+1\}$ is a classifying concept, i.e., the concept of an algebraic structure, S, being such that $|\{x \in S : p(x)\}| = k$ uniquely classifies structures of type A and size $n+1$. The pairs (p,k) for which this is true will form the basis of the classification tree for size $n+1$, while for every pair (p,k) for which this is not true, the system could see whether a conjunction of pairs of properties provides a classifying concept, then conjunctions of triples of properties, and so on.

So far we have used the bootstrapping algorithm only to show each of our parameterisations forms describes indeed an isomorphism class for the algebraic structures of that particular size. That is, we start the algorithm given the parameterisation, and it terminates returning a single equivalence class. However, we have not yet integrated the use of parameterisation into the general bootstrapping process, which will be our next step. If we can also prove results about cross products of algebras, we hope to produce trees which not only perform the classification, but which are simpler and more in line with those produced by mathematicians. In particular, we hope to discover *families* of loops and quasigroups, where a family is essentially a parameterisation such as those described above and a way of choosing parameters which guarantees that the resulting concept will describe an isomorphism class for all sizes. In this fashion, it is not impossible that automated classification tools could begin to have an impact on research mathematics.

References

[1] C Barrett and S Berezin. CVC Lite: A new implementation of the cooperating validity checker. In *Computer Aided Verification, 16th Int. Conf., 2004*.

[2] S Colton. *Automated Theory Formation in Pure Mathematics*. Springer, 2002.

[3] S Colton, A Meier, V Sorge, and R McCasland. Automatic generation of classification theorems for finite algebras. In *Proceedings of IJCAR* , 2004.

[4] McKay, B. D., A. Meynert, and W. Myrvold: 2002, 'Counting Small Latin Squares'. In: *European Women in Mathematics International Workshop on Groups and Graphs*. Varna, Bulgaria, pp. 67–72.

[5] A Meier and V Sorge. Applying sat solving in classification of finite algebras. *Journal of Automated Reasoning*, 35(1–3):201–235, 2005.

[6] V Sorge, A Meier, R McCasland, and S Colton. The automatic construction of isotopy invariants. In *Proceedings of IJCAR*, 2006.

[7] H. Pflugfelder. Quasigroups and Loops: Introduction. Heldermann Verlag, 1990.

[8] C Weidenbach, U Brahm, T Hillenbrand, E Keen, C Theobald, and D Topic. SPASS version 2.0. In Andrei Voronkov, editor, *Proceedings of CADE–18*, 2002.

Proof Analysis with HLK, CERES and ProofTool: Current Status and Future Directions

Stefan Hetzl, Alexander Leitsch, Daniel Weller, Bruno Woltzenlogel Paleo
Vienna University of Technology
Vienna, Austria

Abstract

CERES, HLK and ProofTool form together a system for the computer-aided analysis of mathematical proofs. This analysis is based on a proof transformation known as cut-elimination, which corresponds to the elimination of lemmas in the corresponding informal proofs. Consequently, the resulting formal proof in atomic-cut normal form corresponds to a direct, i.e. without lemmas, informal mathematical proof of the given theorem.

In this paper, we firstly describe the current status of the whole system from the point of view of its usage. Subsequently, we discuss each component in more detail, briefly explaining the formal calculi (**LK** and **LKDe**) used, the intermediary language *HandyLK*, the *CERES* method of cut-elimination by resolution and the extraction of Herbrand sequents. Three successful cases of application of the system to mathematical proofs are then summarized. And finally we discuss extensions of the system that are currently under development or that are planned for the short-term future.

1 Introduction

Proof synthesis and analysis form the very core of mathematical activity. While the application of automated reasoning techniques to proof synthesis is the focus of large research fields such as *automated theorem proving* and *interactive theorem proving*, the corresponding use of automated reasoning techniques in proof analysis has received considerably less attention. Nevertheless, the importance of proof analysis to mathematics should not be underestimated. Indeed, many mathematical concepts such as the notion of group or the notion of probability were introduced by analyzing existing mathematical arguments [Pol54a, Pol54b].

CERES, HLK and ProofTool are three computer programs that form a system for the computer-aided analysis of mathematical proofs. HLK is responsible for the formalization of mathematical proofs; CERES is responsible for transformations of formal proofs and for the extraction of relevant information from them; and ProofTool is responsible for the visualization of formal proofs.

The most useful proof transformation implemented by CERES for the purpose of proof analysis is cut-elimination. The elimination of cuts from the formalized proofs corresponds to the elimination of lemmas from the original informal proofs. By interpreting the resulting (cut-free or in atomic-cut normal form) proof, a mathematician can obtain a new direct informal proof of the theorem under consideration. The elimination performed by CERES follows the *CERES* method [BL00], which relies on the *resolution calculus* and is essentially different from reductive methods such as the first algorithm for cut-elimination introduced by Gentzen [Gen69]. On the other hand, for the extraction and storage of relevant information from proofs, CERES currently uses the ideas of *characteristic clause set* and *Herbrand sequent*.

In this paper we start with an overview of the system, followed by a discussion of some details of each of the three component programs. Three successful cases of application of the system to real mathematical proofs are then summarized. And finally, we present our plans for the further development and extension of the system.

Sutcliffe G., Colton S., Schulz S. (eds.); Proceedings of ESARM 2008, pp. 23-41

2 Overview of the Architecture and Workflow

Figure 1 sketches how HLK, CERES and ProofTool can be used by a mathematician to analyze existing mathematical proofs and obtain new ones.

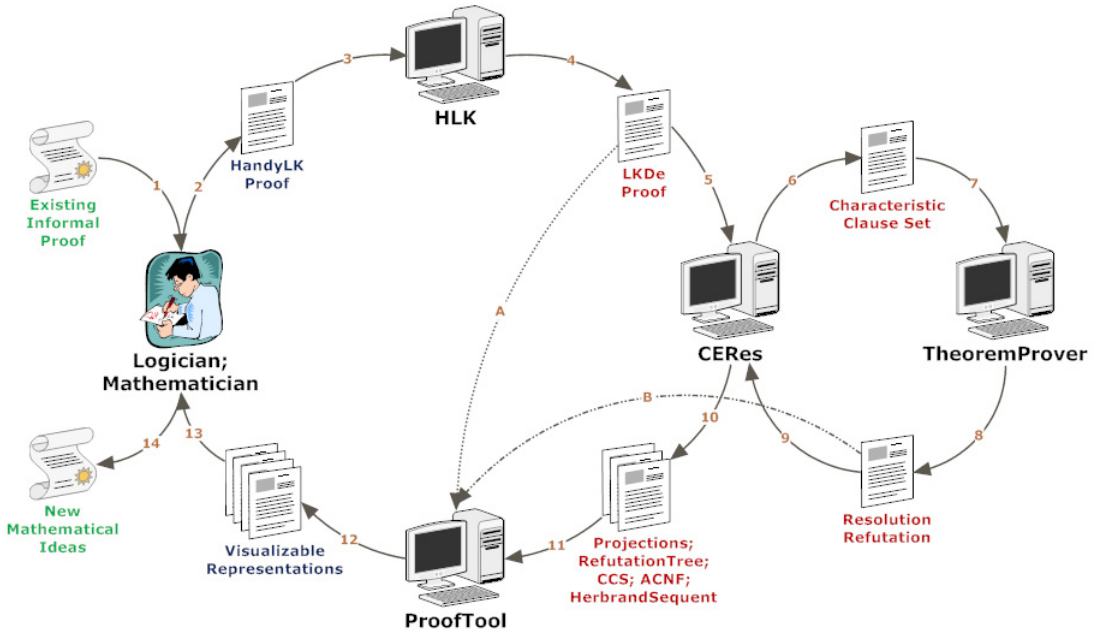

Figure 1: Working with HLK, CERES and ProofTool.

According to the labels in the edges of Figure 1, the following steps are executed within the system and in interaction with the user:

1. The user (usually a mathematician) selects an interesting informal mathematical proof to be transformed and analyzed. Informal mathematical proofs are proofs in natural language, as they usually occur in mathematics.
2. The user writes the selected proof in *HandyLK*, an intermediary language between natural mathematical language and formal calculi.
3. The proof written in *HandyLK* is input to HLK, the compiler of *HandyLK*.
4. HLK generates a formal proof in sequent calculus **LKDe**.
5. The formal proof is input to CERES, which is responsible for cut-elimination and various other proof-transformations that serve as pre- and post-processing, such as proof skolemization, upward shifting of equality rules and Herbrand sequent extraction.
6. CERES extracts from the formal proof a *characteristic clause set*, which contains clauses formed from ancestors of cut-formulas in the formal proof.
7. The characteristic clause set is then input to a *resolution theorem prover*, e.g. Otter[1] or Prover9[2].

[1]Otter Website: http://www-unix.mcs.anl.gov/AR/otter/
[2]Prover9 Website: http://www.cs.unm.edu/ mccune/prover9/

8. The resolution theorem prover outputs a refutation of the characteristic clause set.
9. CERES receives the refutation, which will be used as a skeleton for the transformed proof in atomic-cut normal form (ACNF).
10. CERES outputs the grounded refutation in a tree format and the characteristic clause set. Moreover it extracts projections from the formal proof and plugs them into the refutation in order to generate the ACNF. The projections and the ACNF are also output. A Herbrand sequent that summarizes the creative content of the ACNF is extracted and output as well.
11. All outputs and inputs of CERES can be opened with ProofTool.
12. ProofTool, a graphical user interface, renders all proofs, refutations, projections, sequents and clause sets so that they can be visualized by the user.
13. The information displayed via ProofTool is analyzed by the user.
14. Based on his analysis, the user can formulate new mathematical ideas, e.g. a new informal direct proof corresponding to the ACNF.

2.1 Implementation

HLK[3], CERES[4] and ProofTool[5] are free and open-source programs written in ANSI-C++ (the C++ Programming Language following the International Standard 14882:1998 approved as an American National Standard), and they are therefore likely to run on all operating systems. However, at this time only Linux and Mac OS X are supported. Nevertheless, to make it easier for the system to be used also by users of unsupported operating systems or even by users of Linux and Mac OS X who are not so familiar with software development, a virtual machine containing the whole system pre-installed is also available for download.

CERES and ProofTool use XML files satisfying the *proofdatabase* DTD[6] for the storage of formal logical structures (e.g. the original proof, the ACNF, the projections, the Herbrand sequent).

3 HLK for Proof Formalization

In order to analyze mathematical proofs in an automated way, we must firstly write down the proof according to the rules of a formal logical calculus, so that the formalized proofs are then well-defined data-structures, suitable to be manipulated automatically by computers. On the one hand, these data structures should be as simple as possible, to allow theoretical logical investigations as well as the easy implementation of algorithms. On the other hand, they should be as rich and close to the natural language of informal mathematical proofs as possible. Therefore, the chosen formal calculus should optimize a trade-off between simplicity and richness of the data structures.

With this in mind, the sequent calculus **LK** was the initial choice. It is a very well-known and well-studied logical calculus, for which proof-theoretical results abound. The specific variant that we use is detailed in Subsection 3.1. To bring it closer to the natural language of informal mathematics, we extended it with definition and equality rules. The resulting calculus is called **LKDe** and its details are discussed in Subsections 3.2 and 3.3.

[3] HLK Website: http://www.logic.at/hlk
[4] CERES Website: http://www.logic.at/ceres
[5] ProofTool Website: http://www.logic.at/prooftool
[6] ProofDatabase DTD: http://www.logic.at/ceres/xml/4.0/proofdatabase.dtd

However, **LKDe** is still uncomfortable to be used directly to write proofs down, because its data structures for proofs are nested and contain many repetitions of formulas in the context of ancestor sequents. To solve this problem *HandyLK* was created to serve as an intermediary language, as explained in detail in Subsection 3.4.

3.1 The Sequent Calculus LK

A *sequent* is a pair of sequences of formulas. The sequence in the left side of the sequent is called its *antecedent*; and the sequence in the right is its *succedent*. A sequent can be interpreted as the statement that at least one of the formulas in the succedent can be proved from the formulas in the antecedent. A sequent calculus *proof* or *derivation* is a tree where the leaf-nodes are *axiom sequents* and the edges are inferences according to the *inference rules* of the sequent calculus. Axiom sequents are arbitrary atomic sequents specified by a *background theory* (e.g. the reflexivity axiom scheme $\vdash t = t$ for all terms t, expressing the reflexivity of equality) or tautological atomic sequents (i.e. $A \vdash A$).

There are many sequent calculi. Our variant uses the following rules:

- **Propositional-rules:**

$$\frac{\Gamma \vdash \Delta, A \quad \Pi \vdash \Lambda, B}{\Gamma, \Pi \vdash \Delta, \Lambda, A \wedge B} \wedge : r \qquad \frac{A, \Gamma \vdash \Delta}{A \wedge B, \Gamma \vdash \Delta} \wedge : l1 \qquad \frac{A, \Gamma \vdash \Delta}{B \wedge A, \Gamma \vdash \Delta} \wedge : l2$$

$$\frac{A, \Gamma \vdash \Delta \quad B, \Pi \vdash \Lambda}{A \vee B, \Gamma, \Pi \vdash \Delta, \Lambda} \vee : l \qquad \frac{\Gamma \vdash \Delta, A}{\Gamma \vdash \Delta, A \vee B} \vee : r1 \qquad \frac{\Gamma \vdash \Delta, A}{\Gamma \vdash \Delta, B \vee A} \vee : r2$$

$$\frac{\Gamma \vdash \Delta, A \quad B, \Pi \vdash \Lambda}{A \to B, \Gamma, \Pi \vdash \Delta, \Lambda} \to : l \qquad \frac{A, \Gamma \vdash \Delta, B}{\Gamma \vdash \Delta, A \to B} \to : r$$

$$\frac{A, \Gamma \vdash \Delta}{\Gamma \vdash \Delta, \neg A} \neg : r \qquad \frac{\Gamma \vdash \Delta, A}{\neg A, \Gamma \vdash \Delta} \neg : l$$

  In the rules $\wedge : l1, \wedge : l2, \vee : r1, \vee : r2$, we say that the formula B is a *weak subformula* of A.

- **Quantifier-rules:**

$$\frac{\Gamma \vdash \Delta, A\{x \leftarrow \alpha\}}{\Gamma \vdash \Delta, (\forall x)A} \forall : r \qquad \frac{A\{x \leftarrow t\}, \Gamma \vdash \Delta}{(\forall x)A, \Gamma \vdash \Delta} \forall : l$$

$$\frac{\Gamma \vdash \Delta, A\{x \leftarrow t\}}{\Gamma \vdash \Delta, (\exists x)A} \exists : r \qquad \frac{A\{x \leftarrow \alpha\}, \Gamma \vdash \Delta}{(\exists x)A, \Gamma \vdash \Delta} \exists : l$$

  For the $\forall : r$ and the $\exists : l$ rules, the *eigenvariable condition* must hold: α can occur neither in Γ nor in Δ nor in A. For the $\forall : l$ and the $\exists : r$ rules the term t must not contain a variable that is bound in A.

- **Weakening-rules:**

$$\frac{\Gamma \vdash \Delta}{\Gamma \vdash \Delta, A_1, \ldots, A_n} w : r \qquad \frac{\Gamma \vdash \Delta}{A_1, \ldots, A_n, \Gamma \vdash \Delta} w : l$$

  where $n > 0$

- **Contraction-rules:**

$$\frac{\Gamma \vdash A_1^{(m_1)}, \ldots, A_n^{(m_n)}}{\Gamma \vdash A_1, \ldots, A_n} c(m_1, \ldots, m_n) : r \qquad \frac{A_1^{(m_1)}, \ldots, A_n^{(m_n)} \vdash \Delta}{A_1, \ldots, A_n \vdash \Delta} c(m_1, \ldots, m_n) : l$$

  where $m_i > 0$ for $1 \leq i \leq n$ and $A^{(k)}$ denotes k copies of A.

- **Permutation-rules:**

$$\frac{A_1,\ldots,A_n \vdash \Delta}{B_1,\ldots,B_n \vdash \Delta}\ \pi(\sigma):l \qquad \frac{\Gamma \vdash A_1,\ldots,A_n}{\Gamma \vdash B_1,\ldots,B_n}\ \pi(\sigma):r$$

  where σ is a permutation which is interpreted as a function specifying bottom-up index mapping, i.e. $B_i = A_{\sigma(i)}$.

- **Cut-rule:**

$$\frac{\Gamma \vdash \Delta, A \quad A, \Pi \vdash \Lambda}{\Gamma, \Pi \vdash \Delta, \Lambda}\ \text{cut}$$

3.2 The Sequent Calculus LKe

Equality is widely used in real mathematical proofs. Carrying the axioms of equality along the proof within the antecedent of the sequents would render large, redundant and unreadable proofs. Therefore, **LK** was extended in [BHL$^+$06] to **LKe** with the following rules:

- **Equality-rules:**

$$\frac{\Gamma \vdash \Delta, s=t \quad \Pi \vdash \Lambda, A[s]_\Xi}{\Gamma, \Pi \vdash \Delta, \Lambda, A[t]_\Xi} = (\Xi):r1 \qquad \frac{\Gamma \vdash \Delta, s=t \quad A[s]_\Xi, \Pi \vdash \Lambda}{A[t]_\Xi, \Gamma, \Pi \vdash \Delta, \Lambda} = (\Xi):l1$$

$$\frac{\Gamma \vdash \Delta, t=s \quad \Pi \vdash \Lambda, A[s]_\Xi}{\Gamma, \Pi \vdash \Delta, \Lambda, A[t]_\Xi} = (\Xi):r2 \qquad \frac{\Gamma \vdash \Delta, t=s \quad A[s]_\Xi, \Pi \vdash \Lambda}{A[t]_\Xi, \Gamma, \Pi \vdash \Delta, \Lambda} = (\Xi):l2$$

  where Ξ is a set of positions in A and s and t do not contain variables that are bound in A.

3.3 The Sequent Calculus LKDe

Definition introduction is a simple and very powerful tool in mathematical practice, allowing the easy introduction of important concepts and notations (e.g. groups, lattices, ...) by the introduction of new symbols. Therefore, **LKe** was extended in [BHL$^+$06] to **LKDe** with the following rules:

- **Definition-rules:** They correspond directly to the *extension principle* in predicate logic and introduce new predicate and function symbols as abbreviations for formulas and terms. Let A be a first-order formula with the free variables $x_1,\ldots,x_k$,denoted by $A(x_1,\ldots,x_k)$, and P be a *new* k-ary predicate symbol (corresponding to the formula A). Then the rules are:

$$\frac{A(t_1,\ldots,t_k), \Gamma \vdash \Delta}{P(t_1,\ldots,t_k), \Gamma \vdash \Delta}\ d:l \qquad \frac{\Gamma \vdash \Delta, A(t_1,\ldots,t_k)}{\Gamma \vdash \Delta, P(t_1,\ldots,t_k)}\ d:r$$

  for arbitrary sequences of terms $t_1,\ldots,t_k$.

3.4 *HandyLK*

In this subsection, we will focus on some particular features of *HandyLK*'s syntax[7]. These features make it more comfortable to write down proofs in *HandyLK* and then automatically translate them with HLK to **LKDe** instead of writing them directly and manually in **LKDe**.

When typing proofs in a naive notation for **LKDe** (e.g. a LISP-like notation for tree structures), two problems are encountered. Firstly, one would have to repeat many formulas that are not changed by the application of inference rules, i.e. those formulas that occur in the contexts Γ, Δ, Π and Λ and are simply carried over from the premises to the conclusions. Secondly, large branching proofs quickly become

[7] *HandyLK* full syntax specification: http://www.logic.at/hlk/handy-syntax.pdf

unreadable due to the nesting of subproofs within subproofs. The partial *HandyLK* code below shows how the first problem is avoided in the system's intermediary language for proof formalization.

```
define proof \varphi_1
  proves
    all x ( not P(x) or Q(x) ) :- all x ex y ( not P(x) or Q(y) );

  with all right
    :- ex y ( not P(\alpha) or Q(y) );
  with all left
    not P(\alpha) or Q(\alpha) :- ;
  with ex right
    :- not P(\alpha) or Q(\alpha);

  continued auto propositional
    not P(\alpha) or Q(\alpha) :- not P(\alpha) or Q(\alpha);
;
```

The code above starts by declaring a proof and the end-sequent that it derives. Then it lists the rules that have to be applied successively to the end-sequent in a bottom up way. For each inference, only the auxiliary formulas have to be specified. Context formulas are handled automatically by HLK.

Another useful feature shown above is the *HandyLK* command "**continued auto propositional**". It specifies that the current sequent can be derived from tautological axioms by application of propositional rules. In such cases, HLK is able to produce the proof automatically.

For the example above, HLK generates the following **LKDe** proof:

$\varphi_1 :=$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{P(\alpha)\vdash P(\alpha)}{\vdash P(\alpha),\neg P(\alpha)}\,\neg:r}{\neg P(\alpha)\vdash\neg P(\alpha)}\,\neg:l}{\neg P(\alpha)\vdash\neg P(\alpha)\vee Q(\alpha)}\,\vee:r1 \quad \dfrac{Q(\alpha)\vdash Q(\alpha)}{Q(\alpha)\vdash\neg P(\alpha)\vee Q(\alpha)}\,\vee:r2}{\neg P(\alpha)\vee Q(\alpha)\vdash\neg P(\alpha)\vee Q(\alpha),\neg P(\alpha)\vee Q(\alpha)}\,\vee:l}{\neg P(\alpha)\vee Q(\alpha)\vdash\neg P(\alpha)\vee Q(\alpha)}\,c:r}{\neg P(\alpha)\vee Q(\alpha)\vdash(\exists y)(\neg P(\alpha)\vee Q(y))}\,\exists:r}{(\forall x)(\neg P(x)\vee Q(x))\vdash(\exists y)(\neg P(\alpha)\vee Q(y))}\,\forall:l}{(\forall x)(\neg P(x)\vee Q(x))\vdash(\forall x)(\exists y)(\neg P(x)\vee Q(y))}\,\forall:r$$

The other code below shows how *HandyLK* avoids the problem of nesting and branching. A premise of any inference can have a link to another proof instead of having the auxiliary subsequent of that premise (in the example below, the left branch has a link to the subproof φ_1 and the right branch, a link to φ_2). Hence, large subproofs can be specified somewhere else in the file, instead of appearing nested inside the proof.

```
define proof \varphi
  proves
    P(a), all x ( not P(x) or Q(x) ) :- ex y Q(y);

  with cut all x ex y ( not P(x) or Q(y) )
    left by proof \varphi_1
    right by proof \varphi_2;
;
```

Finally it should be noted that *HandyLK* allows parameterizing meta-terms and meta-formulas in proofs. Hence it is thus capable of expressing recursive definitions of proof schemata, enabling the user to define infinite proof sequences[8]. In the code below, for example, a recursive proof is defined, parameterized by the meta-term n. The proof at level n can refer to the proof at level $n-1$, $pr[n-1]$, in the same way that the recursive function fr at level n, $fr[n]$, refers to $fr[n-1]$.

```
define function f of type nat to nat;

define recursive function fr( x ) to nat
  at level 0 by x
  at level n by f( fr[n - 1](x) )
;

define recursive proof pr
  at level 0 proves
    P(a) :- P( fr[0](a) );

  ...

  at level n proves
    P(a), all x ( P(x) impl P( f(x) ) ) :- P( fr[n](a) );

  ...
;
```

HLK can output the **LKDe** proof either as a LaTeX file or as an XML file satisfying to the proofdatabase DTD.

4 CERES

After the proof has been formalized in **LKDe** with HLK, it can be transformed with the CERES system. Among the transformations that can be performed by CERES the most interesting one is cut-elimination, which produces a proof in atomic-cut normal form. The ACNF is mathematically interesting, because cut-elimination in formal proofs corresponds to the elimination of lemmas in informal proofs. Hence the ACNF corresponds to an informal mathematical proof that is analytic in the sense that it does not use auxiliary notions that are not already explicit in the theorem itself.

To perform cut-elimination, CERES uses a method that employs the resolution calculus to eliminate cuts in a global way, instead of the more traditional local reductive methods [Gen69]. The *CERES* method has been described in various degrees of detail and with different emphasis in [BL00, BHL$^+$05, BHL$^+$06, BHL$^+$] and hence it is just sketched in Subsection 4.1.

Although the ACNF is mathematically interesting, it is too large and full of redundant information. Its direct analysis and interpretation by humans is therefore too difficult. To overcome this, an algorithm to extract the essential information of proofs has been recently implemented within CERES. It performs *Herbrand sequent extraction* and is sketched in Subsection 4.2.

[8] The Exponential Proofs (http://www.logic.at/ceres/examples/index.html) are a good example of *HandyLK* being used to define infinite proof sequences with parameterized meta-terms and meta-formulas

4.1 The *CERES* Method

The *CERES* method transforms any **LKDe**-proof with cuts into an *atomic-cut normal form* (ACNF) containing no non-atomic cuts. The remaining atomic cuts are, generally, non-eliminable, because **LKDe** admits non-tautological axiom sequents.

The transformation to ACNF via Cut-Elimination by Resolution is done according to the following steps:

1. Construct the (always unsatisfiable [BL00]) *characteristic clause set* of the original proof by collecting, joining and merging sets of clauses defined by the ancestors of cut-formulas in the axiom sequents of the proof.

2. Obtain from the characteristic clause set a grounded resolution refutation, which can be seen as an **LKe**-proof by exploiting the fact that the resolution rule is essentially a cut-rule restricted to atomic cut-formulas only and by mapping paramodulations to equality-inferences.

3. For each clause of the characteristic clause set, construct a *projection* of the original proof with respect to the clause.

4. Construct the ACNF by plugging the projections into the corresponding clauses in the leaves of the grounded resolution refutation tree (seen as an **LKe**-proof) and by adjusting the resulting proof with structural inferences (weakening, contractions and permutations) if necessary. Since the projections do not contain cuts and the refutation contains atomic cuts only, the resulting **LKDe** proof will indeed be in atomic-cut normal form.

Theoretical comparisons of the *CERES* with reductive methods can be found in [BL00, BL06]. For illustration, consider the following example:

$\varphi =$

$$\dfrac{\begin{array}{cc}\varphi_1 & \varphi_2\end{array}}{(\forall x)(P(x) \to Q(x)) \vdash (\exists y)(P(a) \to Q(y))}\ cut$$

$\varphi_1 =$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{P(u)^\star \vdash P(u) \quad Q(u) \vdash Q(u)^\star}{P(u)^\star, P(u) \to Q(u) \vdash Q(u)^\star}\to : l}{P(u) \to Q(u) \vdash (P(u) \to Q(u))^\star}\to : r}{P(u) \to Q(u) \vdash (\exists y)(P(u) \to Q(y))^\star}\exists : r}{(\forall x)(P(x) \to Q(x)) \vdash (\exists y)(P(u) \to Q(y))^\star}\forall : l}{(\forall x)(P(x) \to Q(x)) \vdash (\forall x)(\exists y)(P(x) \to Q(y))^\star}\forall : r$$

$\varphi_2 =$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{P(a) \vdash P(a)^\star \quad Q(v)^\star \vdash Q(v)}{P(a), (P(a) \to Q(v))^\star \vdash Q(v)}\to : l}{(P(a) \to Q(v))^\star \vdash P(a) \to Q(v)}\to : r}{(P(a) \to Q(v))^\star \vdash (\exists y)(P(a) \to Q(y))}\exists : r}{(\exists y)(P(a) \to Q(y))^\star \vdash (\exists y)(P(a) \to Q(y))}\exists : l}{(\forall x)(\exists y)(P(x) \to Q(y))^\star \vdash (\exists y)(P(a) \to Q(y))}\forall : l$$

The crucial information extracted by *CERES* is the characteristic clause set, which is based on the ancestors of the cut-formulas (marked here by $\star$). For this proof, the set is $\mathrm{CL}(\varphi) = \{P(u) \vdash Q(u);\ \vdash$

$P(a);\ Q(v) \vdash \}$. This set is always unsatisfiable, here the resolution refutation δ of $\mathrm{CL}(\varphi)$

$$\dfrac{\dfrac{\vdash P(a) \quad P(u) \vdash Q(u)}{\vdash Q(a)}\, R \qquad Q(v) \vdash}{\vdash}\, R$$

does the job. By applying the most general unifier σ of δ, we obtain a ground refutation $\gamma = \delta\sigma$:

$$\dfrac{\dfrac{\vdash P(a) \quad P(a) \vdash Q(a)}{\vdash Q(a)}\, R \qquad Q(a) \vdash}{\vdash}\, R$$

This will serve as a skeleton for a proof in ACNF. To complete the construction, we have to compute the projections to the clauses that are used in the refutation, this essentially works by leaving out inferences on ancestors of cut-formulas and applying σ:

$\varphi(C_1) =$

$$\dfrac{\dfrac{\dfrac{P(a) \vdash P(a) \quad Q(a) \vdash Q(a)}{P(a), P(a) \to Q(a) \vdash Q(a)}\, \to : l}{P(a), (\forall x)(P(x) \to Q(x)) \vdash Q(a)}\, \forall : l}{P(a), (\forall x)(P(x) \to Q(x)) \vdash (\exists y)(P(a) \to Q(y)), Q(a)}\, w : r$$

$\varphi(C_2) =$

$$\dfrac{\dfrac{\dfrac{\dfrac{P(a) \vdash P(a)}{P(a) \vdash P(a), Q(v)}\, w : r}{\vdash P(a) \to Q(v), P(a)}\, \to : r}{\vdash (\exists y)(P(a) \to Q(y)), P(a)}\, \exists : r}{(\forall x)(P(x) \to Q(x)) \vdash (\exists y)(P(a) \to Q(y)), P(a)}\, w : l$$

$\varphi(C_3) =$

$$\dfrac{\dfrac{\dfrac{\dfrac{Q(a) \vdash Q(a)}{P(a), Q(a) \vdash Q(a)}\, w : l}{Q(a) \vdash P(a) \to Q(a)}\, \to : r}{Q(a) \vdash (\exists y)(P(a) \to Q(y))}\, \exists : r}{Q(a), (\forall x)(P(x) \to Q(x)) \vdash (\exists y)(P(a) \to Q(y))}\, w : l$$

Finally, we combine the projections and the ground refutation:

$\varphi(\gamma) =$

$$\dfrac{\dfrac{\dfrac{\varphi(C_2) \qquad \varphi(C_1)}{B \vdash C, P(a) \quad P(a), B \vdash C, Q(a)}\, cut \qquad \varphi(C_3)}{B, B \vdash C, C, Q(a) \qquad Q(a), B \vdash C}\, cut}{\dfrac{B, B, B \vdash C, C, C}{B \vdash C}\, \text{contractions}}$$

where $B = (\forall x)(P(x) \to Q(x))$, $C = (\exists y)(P(a) \to Q(y))$. Clearly, $\varphi(\gamma)$ is a proof of the end-sequent of φ in ACNF.

4.2 Herbrand Sequent Extraction

Our motivation to devise and implement Herbrand sequent extraction algorithms was the need to analyze and understand the result of proof transformations performed automatically by the CERES system, especially the ACNF.

Definition 4.1 (Herbrand Sequents of a Sequent)**.** Let s be a sequent without free-variables and containing weak quantifiers only (i.e. universal quantifiers occur only with negative polarity and existential quantifiers occur only with positive polarity). We denote by s_0 the sequent s after removal of all its quantifiers. Any propositionally valid sequent in which the antecedent (respectively, succedent) formulas are instances (i.e. their free variables are possibly instantiated by other terms) of the antecedent (respectively, succedent) formulas of s_0 is called *a Herbrand sequent of* s.

Let s be an arbitrary sequent and s' a skolemization of s. Any Herbrand sequent of s' is a Herbrand sequent of s.

Theorem 4.1 (Herbrand's Theorem)**.** A sequent s is valid if and only if there exists a Herbrand sequent of s.

Proof. Originally in [Her30], stated for Herbrand disjunctions. Also in [Bus95] with more modern proof calculi. □

Herbrand's theorem guarantees that we can always obtain a Herbrand sequent from a correct proof, a possibility that was firstly exploited by Gentzen in his Mid-Sequent Theorem, which provides an algorithm for the extraction of Herbrand sequents based on the downward shifting of quantifier rules. However, Gentzen's algorithm has one strong limitation: it is applicable only to proofs with end-sequents in prenex form. Although we could transform the end-sequents and the proofs to prenex form, this would compromise the readability of the formulas and require additional computational effort. Prenexification is therefore not desirable in our context, and hence, to overcome this and other limitations in Gentzen's algorithm, we developed and implemented another algorithm.

To extract a Herbrand sequent of the end-sequent s of an **LKDe**-proof φ without cuts containing quantifiers, the implemented algorithm executes two transformations:

1. Ψ ([WP08]): produces a quantifier-rule-free $\mathbf{LKe_A}$-proof where quantified formulas are replaced by array-formulas containing their instances.

2. Φ (definition 4.3): transforms the end-sequent of the resulting $\mathbf{LKe_A}$-proof into an ordinary sequent containing no array-formulas. The result is a Herbrand sequent of the end-sequent of the original proof.

Definition 4.2 (Sequent Calculus $\mathbf{LKe_A}$)**.** The sequent calculus $\mathbf{LKe_A}$ is the sequent calculus **LKe** extended with the following array-formation-rules:

$$\frac{\Delta, A_1, \Gamma_1, \ldots, A_n, \Gamma_n, \Pi \vdash \Lambda}{\Delta, \langle A_1, \ldots, A_n \rangle, \Gamma_1, \ldots, \Gamma_n, \Pi \vdash \Lambda} \langle\rangle : l \qquad \frac{\Lambda \vdash \Delta, A_1, \Gamma_1, \ldots, A_n, \Gamma_n, \Pi}{\Lambda \vdash \Delta, \langle A_1, \ldots, A_n \rangle, \Gamma_1, \ldots, \Gamma_n, \Pi} \langle\rangle : r$$

The intuitive idea behind the computation of $\Psi(\varphi)$ for an **LKDe**-proof φ consists of omitting quantifier-inferences and definition-inferences and replacing unsound contraction-inferences by array-formation-inferences (omissions and replacements are done in a top-down way and formulas in the downward path of a changed formula are changed accordingly). More precisely, if a quantified formula is the main formula of a contraction, then this contraction will not be sound anymore after the omission of the quantifier-inferences, because its auxiliary formulas will now contain different instances of the quantified formula instead of being exact copies of the quantified formula. Hence, the unsound contraction inferences are replaced by array-formation inferences, which will collect all the instances of that quantified formula into an array formula. This idea has been formally defined in [WP08, HLWWP08]. However, the implemented algorithm follows an equivalent recursive definition, which is more natural to implement but quite technical, preventing a clean exposition of the fundamental idea of the transformation to $\mathbf{LKe_A}$.

Definition 4.3 (Φ: Expansion of Array Formulas)**.** The mapping Φ transforms array formulas and sequents into first-order logic formulas and sequents. In other words, Φ eliminates $\langle\ldots\rangle$ and can be defined inductively by:

1. If A is a first-order logic formula, then $\Phi(A) \doteq A$
2. $\Phi(\langle A_1,\ldots,A_n\rangle) \doteq \Phi(A_1),\ldots,\Phi(A_n)$
3. If $\Phi(A) = A_1,\ldots,A_n$, then $\Phi(\neg A) \doteq \neg A_1,\ldots,\neg A_n$
4. If $\Phi(A) = A_1,\ldots,A_n$ and $\Phi(B) = B_1,\ldots,B_m$, then $\Phi(A \circ B) \doteq A_1 \circ B_1,\ldots,A_1 \circ B_m,\ldots,A_n \circ B_1,\ldots,A_n \circ B_m$, for $\circ \in \{\wedge,\vee,\rightarrow\}$
5. $\Phi(A_1,\ldots,A_n \vdash B_1,\ldots,B_m) \doteq \Phi(A_1),\ldots,\Phi(A_n) \vdash \Phi(B_1),\ldots,\Phi(B_m)$

Example 4.1. *Let φ be the following* **LKDe***-proof:*

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[\varphi']}{P(0),P(0)\rightarrow P(s(0)),P(s(0))\rightarrow P(s^2(0))\vdash P(s^2(0))}\forall:l}{P(0),P(0)\rightarrow P(s(0)),(\forall x)(P(x)\rightarrow P(s(x))\vdash P(s^2(0))}\forall:l}{P(0),\forall x(P(x)\rightarrow P(s(x)),(\forall x)(P(x)\rightarrow P(s(x))\vdash P(s^2(0))}c:l}{P(0),\forall x(P(x)\rightarrow P(s(x))\vdash P(s^2(0))}\wedge:l}{P(0)\wedge(\forall x)(P(x)\rightarrow P(s(x))\vdash P(s^2(0))}$$

The **LKDe***-pseudoproof φ_c with unsound contractions, resulting from the omission of quantifier-inferences and definition-inferences is:*

$$\dfrac{\dfrac{\dfrac{[\varphi']}{P(0),P(0)\rightarrow P(s(0)),P(s(0))\rightarrow P(s^2(0))\vdash P(s^2(0))}c^*:l}{P(0),\nabla\vdash P(s^2(0))}\wedge:l}{P(0)\wedge\nabla\vdash P(s^2(0))}$$

where ∇ stands for the undeterminable main formula of the unsound contraction-inference, since it contains different auxiliary formulas.

The $\mathbf{LKe_A}$*-proof $\Psi(\varphi)$, after replacement of unsound contractions by array-formations, is:*

$$\dfrac{\dfrac{\dfrac{[\varphi']}{P(0),P(0)\rightarrow P(s(0)),P(s(0))\rightarrow P(s^2(0))\vdash P(s^2(0))}\langle\rangle:l}{P(0),\langle P(0)\rightarrow P(s(0)),P(s(0))\rightarrow P(s^2(0))\rangle\vdash P(s^2(0))}\wedge:l}{P(0)\wedge\langle P(0)\rightarrow P(s(0)),P(s(0))\rightarrow P(s^2(0))\rangle\vdash P(s^2(0))}$$

Let s be the end-sequent of $\Psi(\varphi)$. Then $\Phi(s)$, which is a Herbrand sequent extracted from φ, is:

$$\left(\begin{array}{c} P(0)\wedge(P(0)\rightarrow P(s(0))), \\ P(0)\wedge(P(s(0))\rightarrow P(s^2(0))) \end{array}\right)\vdash P(s^2(0))$$

4.2.1 Further Improvements of Herbrand Sequent Extraction

Preliminary tests of the implemented algorithm described above showed that the extracted Herbrand sequent still contained information that was irrelevant for the interpretation of the ACNF to obtain a new informal mathematical proof corresponding to the ACNF. Such irrelevant information occurred in the form of subformulas of the Herbrand sequent that were introduced by weakening-inferences or as the weak subformula of the main formula of some other inference. This problem was solved by omitting the weak subformulas in the construction of $\Psi(\varphi)$. This improvement implies that the extracted sequent is not strictly a Herbrand sequent anymore, because its formulas are not instances of the formulas of the original end-sequent. However, the extracted sequent is still valid and contains the desired relevant information about the used variable instantiations in φ.

5 ProofTool

ProofTool is the graphical user interface that displays the original formal proof, the skolemized proof, the characteristic clause set, the projections, the ACNF and the Herbrand sequent to the user, so that he can analyze them and obtain a better understanding of the original proof or a new direct informal proof for the same theorem based on the Herbrand sequent and on the ACNF. It allows the user to zoom in and out of the proofs and to navigate to subproofs defined by proof links. Although it is mostly intended for proof visualization, some simple forms of proof editing, e.g. formula substitution and splitting of proofs by the creation of new proof links, are also possible.

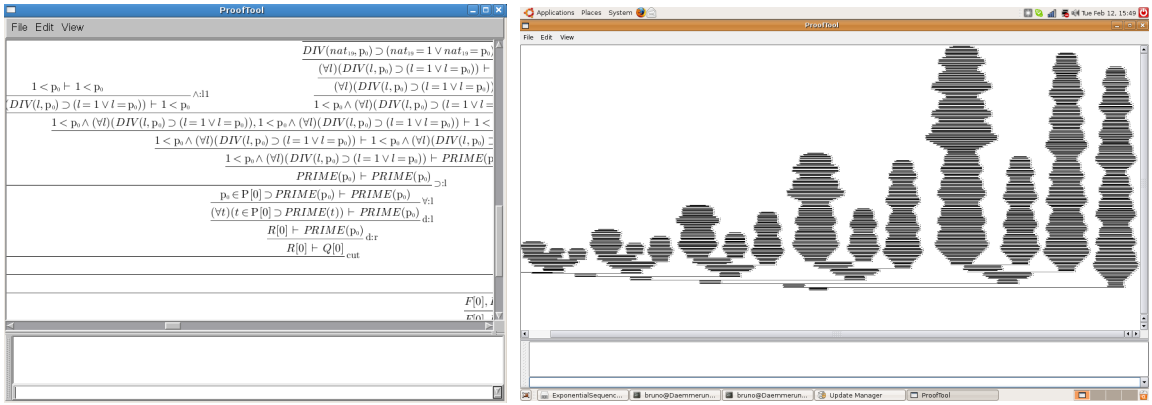

Figure 2: ProofTool Screenshots

6 Successful Experimental Cases

In this section, we summarize three successful cases in which the system was used for the analysis of mathematical proofs. All the examples below constitute interesting proofs for cut-elimination because they contain mathematical concepts in the cut-formulas which do not occur in the theorems shown. More information about each case can be found in the respective referred papers containing the detailed analysis.

6.1 The Tape Proof

The first proof that was analyzed using CERES (in [BHL+05] within **LK** and in [BHL+06] within **LKDe**) was originally defined in [Urb00]. This proof deals with the following situation: We are given an infinite

tape where each cell contains either '0' or '1'. We prove that on this tape there are two cells with the same value. The original proof goes on to show that on the tape, there are infinitely many cells containing '0' or infinitely many cells containing '1'. From this, it follows that there are two cells having the same value. Clearly, these lemmas are much stronger than what is actually needed to prove the theorem, and exactly these lemmas where eliminated by the use of CERES.

By using different resolution refinements (positive and negative hyperresolution), two different resolution refutations (resulting in different ACNFs) were produced. After analysis of the argument contained in the refutations, it turns out that the refutations differ not only formally, but also in their mathematical interpretation. This showed that for any particular proof ψ, CERES may produce several ACNFs of the end-sequent of ψ and each of them may contain a mathematically different argument. This example exhibited the diversity of analytic proofs that may be obtained by using CERES.

Furthermore, in [Het07], two different tape proofs with different cut-formulas were analyzed comparatively. It has been observed how the characteristic clause sets of each proof determine and restrict the kinds of mathematical arguments and direct informal mathematical proofs that can be obtained from the ACNFs of each proof. This shows the potential of CERES not only for the analysis of single proofs, but also for the comparative analysis of different proofs of the same theorem.

6.2 The Prime Proof

In [BHL$^+$], the system was used to analyze Fürstenberg's proof of the infinity of primes. This proof uses topological concepts in its lemmas and proves that there are infinitely many primes. A central part in the formalization of this proof was the possibility of expressing proof schemas in HLK— here, it was used to formalize the inductive parts of the proof (e.g. that the union of closed sets is closed). Each proof $\varphi(k)$ according to the proof schema φ expresses that there are more than k primes. The collection of all such proofs implies that there are infinitely many primes.

The refutation and analysis of the *characteristic clause set* extracted by CERES led to a direct informal proof of the infinity of primes, which corresponded to the well-known Euclid's argument for the infinity of primes. Thus CERES could essentially transform Fürstenberg's proof into Euclid's proof.

6.3 The Lattice Proof

In [HLWWP08], the usefulness of a Herbrand sequent for understanding a formal proof was demonstrated on a simple proof from lattice theory showing that $L1$-lattices (semi-lattices satisfying "inverse" laws) are $L2$-lattices (semi-lattices satisfying the absorption laws) by firstly showing that $L1$-lattices are $L3$-lattices (partially-ordered sets with greatest lower bounds and least upper bounds) and then showing that $L3$-lattices are $L2$-lattices.

The formalization of the proof with HLK resulted in a proof with 260 inferences where the concept of $L3$-lattice appears, as expected, as a cut-formula in a cut-inference. The elimination of cuts with CERES resulted in a proof in atomic-cut normal form (ACNF) containing 214 inferences and no essential cuts. The informal proof corresponding to the ACNF would be a direct mathematical argument proving that $L1$-lattices are $L2$-lattices. However, due to its size, the interpretation of the ACNF to extract this informal proof is hardly possible. On the other hand, the Herbrand sequent extracted from the ACNF contains only 6 formulas. Analyzing the Herbrand sequent alone, it was possible to obtain the desired informal direct proof.

7 Future Directions

The development of our systems focuses on the analysis of mathematical proofs. However, it could also be extended to other applications of the transformation and analysis of formal proofs, e.g. for the computation of interpolants in symbolic model checking [McM03, HJMM04].

In the following subsections, we will discuss a few improvements to the system formed by CERES, HLK and ProofTool that are currently under development or that shall be developed in the future.

7.1 The Proof Profile

The proof profile [Het07, Het08] is a refinement of the characteristic clause set which, in addition to the logical information about the cut-formulas, incorporates also combinatorial information about the structure of the proof. This leads to an optimization of the *CERES* method in the sense that atomic-cut normal forms generated by using the profile will always be at most the size of those generated by using the characteristic clause set, and furthermore, the profile is stronger in detecting certain kinds of redundancies which can yield even a non-elementary speed-up with respect to an atomic-cut normal form generated by using the characteristic clause set. Moreover, not only is the proof profile a practical optimization but it also gives important theoretical benefits: they have been used in [HL07] to obtain new results about the relation between simple proof transformations and cut-elimination. Therefore, CERES will be extended to make it possible to use the proof profile instead of the characteristic clause set.

7.2 Extension to Second-order Logic

Second-order logic extends first-order logic by allowing quantification over set variables. The reasons for considering an extension of the *CERES* method to second-order logic are twofold: first, the second-order induction axiom

$$(\forall X)(0 \in X \wedge (\forall x)(x \in X \rightarrow x' \in X) \rightarrow (\forall x)x \in X)$$

enables proofs by induction to be handled on the object level, whereas induction in first-order logic can only be handled by (non-tautological) rules or by proof schemas. This then allows the induction component of proofs to be handled in the same way as any other formula occuring in a proof. Secondly, a comprehension axiom scheme

$$(\exists X)(\forall x)(x \in X \leftrightarrow \varphi(x))$$

can be formulated in second-order logic, which allows one to assert the existence of sets defined by formulas $\varphi(x)$. Certainly, the definition of sets by formulas is a very natural mathematical operation: consider as an example the sentence

$$(\exists X)(\forall x)(x \in X \leftrightarrow (\exists z)2 * z = x)$$

which asserts the existence of the set of even numbers.

Note that the second-order axioms of induction and comprehension are preferable to the corresponding schemes of first-order arithmetic and set theory in this context: the goal of proof analysis by cut-elimination is to remove certain mathematical concepts from the proof. If, however, these concepts appear in instances of axiom schemes in the end-sequent of the proof with cuts, they will also appear in the end-sequent of the cut-free (or ACNF) proof. In the second-order formulation, preserving the end-sequent does not imply preserving the instances of the schemes.

As second-order logic distinguishes between set variables and individual variables, it does not suffer from Russel's paradox. Still, second-order arithmetic (even with restrictions on $\varphi(x)$ in the comprehension scheme) is powerful enough to prove a large part of ordinary mathematics (see [Sim99]).

Extending *CERES* to second-order logic is non-trivial: In first-order logic, proof skolemization is used as an essential technical tool to remove strong quantifier rules going into the end-sequent from the proof. In second-order logic, the corresponding transformation can only be applied in very special cases (we must restrict the way in which weak second-order quantifiers occur). Consider for example:

$$\dfrac{\dfrac{\dfrac{\dfrac{P(\beta,a)\vdash P(\beta,a)}{(\forall x)P(x,a)\vdash P(\beta,a)}\forall:l \quad \dfrac{P(\alpha,b)\vdash P(\alpha,b)}{(\forall z)P(z,b)\vdash P(\alpha,b)}\forall:l}{(\forall x)P(x,a)\vdash(\forall z)P(z,a)}\forall:r \quad \dfrac{P(\alpha,b)\vdash P(\alpha,b)}{(\forall z)P(z,b)\vdash(\forall z)P(z,b)}\forall:r}{(\forall x)P(x,a),(\forall z)P(z,a)\to(\forall z)P(z,b)\vdash(\forall x)P(x,b)}\to:l}{(\forall x)P(x,a),(\forall X)(X(a)\to X(b))\vdash(\forall x)P(x,b)}\forall^2:l\ \lambda x.(\forall z)P(z,x)}{(\forall X)(X(a)\to X(b))\vdash(\forall x)P(x,a)\to(\forall x)P(x,b)}\to:r$$

Skolemization of the proof would yield

$$\dfrac{\dfrac{\dfrac{\dfrac{P(s_2,a)\vdash P(s_2,a)}{(\forall x)P(x,a)\vdash P(s_2,a)}\forall:l \quad \dfrac{P(s_1,b)\vdash P(s_1,b)}{(\forall z)P(z,b)\vdash P(s_1,b)}\forall:l}{(\forall x)P(x,a),P(s_2,a)\to(\forall z)P(z,b)\vdash P(s_1,b)}\to:l}{(\forall x)P(x,a),(\forall X)(X(a)\to X(b))\vdash P(s_1,b)}\forall^2:l\ \lambda x.(\forall z)P(z,x)}{(\forall X)(X(a)\to X(b))\vdash(\forall x)P(x,a)\to P(s_1,b)}\to:r$$

where the $\forall^2:l$ rule application is clearly not sound. For this reason, we will investigate an extension of the *CERES* method that deals with proofs that have not been skolemized.

Also, resolution in second-order logic loses some nice properties of its first-order counterpart: first-order logic is semi-recursive and therefore we can always find a resolution refutation of the characteristic clause set. This is not the case for second-order logic. Still, there exist implementations of higher-order resolution provers (e.g. Leo, see [BK98]), which we plan to adapt for use with CERES in second-order logic.

7.3 Handling Non-skolemized Proofs and Elimination of Single Cuts

As mentioned in the previous section, skolemization of proofs in second-order logic is not always possible. Also in first-order logic, it would be advantageous to be able to apply the *CERES* method to non-skolemized proofs: consider, for example, a proof φ containing a proof ψ as a subproof. Applying *CERES* to the skolemization of ψ yields an ACNF ψ'. It is then not possible in general to put ψ' in place of ψ in φ: the end-sequent of ψ may contain formula occurrences that are ancestors of cut-formulas in φ, and skolemization of these formulas will prevent these cuts from being applied (as the cut-formula occurrences have different polarities). This leads to the fact that *CERES* currently only supports elimination of all cuts of a proof at once.

On the other hand, if *CERES* is able to handle non-skolemized proofs this immediatly gives rise to a method to eliminate a single uppermost cut ρ in φ by isolating the subproof ending with ρ and applying *CERES* to it.

Our experiments with applying CERES to Fürstenberg's proof on the infinity of primes showed that current automated theorem provers are too weak to handle the characteristic clause sets of larger proofs. With a method for eliminating single cuts, the problem of performing cut-elimination on a proof φ can then be reduced to a sequence of cut-eliminations on proofs ψ_i that contain only one non-atomic cut. The characteristic clause sets of the ψ_i will be less complex than that of φ, which will enable the theorem provers to find a refutation more easily.

The main problem that occurs when considering non-skolemized proofs with CERES are eigenvariable violations of the strong quantifier rules that appear in the projections. There are different ideas on how this problem may be resolved, a promising approach is the following: we restrict the form of the strong quantifier rules going into the end-sequent by using, instead of eigenvariables, skolem terms as *eigenterms* of the rules. In the resulting ACNF, we will then have violations of the eigenterm conditions of these rules by ancestors of cut-formulas. By a proof transformation (essentially reductive cut-elimination together with certain rule permutations) we can produce a valid ACNF. Intuitively, this works because reductive cut-elimination always shifts cuts upwards in a proof, so eventually the cut-formulas will be cut out at the top of the proof, and can not cause eigenterm violations below.

7.4 Extension to Superdeduction Calculi

The first logical calculi followed Hilbert's style of having few inference rules and many axiom schemas. The next generations of calculi such as natural deduction and sequent calculi substituted some axiom schemas by new inference rules. Formal proofs in these calculi were therefore closer to natural informal mathematical proofs, simply because the new inference rules corresponded more closely to some kinds of inferences that are usually done in informal mathematical argumentation. However, the new inference rules substituted only axioms that contained purely *logical* information about the behavior of connectives and quantifiers. Informal mathematical argumentation, on the other hand, contains many informal inferences based on axioms containing *mathematical* information about a concrete domain of mathematical practice. The trend in the evolution of formal calculi for the actual formalization of mathematical proofs is to incorporate such mathematical axioms into rules of inference, in the same way that natural deduction and sequent calculi incorporated purely logical axioms into their rules of inference.

The use of arbitrary initial sequents, equation-rules and definition-rules in **LKDe** can be seen as a small step within this trend. Another related but bigger step, though, was done with the proposal of *superdeduction rules*[BHK07], which roughly correspond to a rigorous combination of our definition-rules with other **LK** inferences in such a way that all auxiliary formulas of the superdeduction inference are atomic. More precisely, superdeduction inferences can be easily simulated in **LKDe** by substituting them by many **LK** inferences followed by a definition-inference. Nevertheless, proofs using superdeduction inferences are shorter, more readable and closer to informal mathematical proofs than pure **LKDe**-proofs.

To support superdeduction inferences, the *CERES* method and CERES, HLK and ProofTool will have to be extended to support rules of arbitrary arity, because they currently work with unary and binary rules only and superdeduction rules can have an arbitrary number of premises.

7.5 Better Herbrand Sequents

Our algorithm for Herbrand sequent extraction currently lacks support for definition-rules, because definition-inferences must be omitted in the transformation to $\mathbf{LKe_A}$. We plan to modify the algorithm, so that it reinserts defined formulas in the extracted Herbrand sequent, in order to further improve its readability. Better readability could also be achieved by post-processing and compressing long terms appearing in the formulas of Herbrand sequent. Furthermore, the usage of the Herbrand sequent as a guide for the construction of new informal direct mathematical proofs could be made easier by enriching the Herbrand sequent with the axiom links that were used in the proof, similarly to what is done in proof nets and atom flows.

7.6 Resolution Refinements for Cut-Elimination

The *CERES* method relies on a search for a refutation of the characteristic clause set. Although the characteristic clause set is known to be always unsatisfiable, the search space with unrestricted resolution or even with typical resolution refinements (e.g. hyper-resolution, unit-resolution, ...) can be so large that theorem provers like Otter and Prover9 do not find a refutation in a reasonable time. This occurs specially if the proof and its characteristic clause set are large or if the proof contains equality rules, in which case the refutation needs paramodulation.

We plan to develop resolution refinements that will exploit the structure of the proofs in order to restrict the search space and possibly even eliminate the need for search altogether.

7.7 Interactive Resolution Theorem Prover

In order to implement the planned resolution refinements, we have implemented a simple but flexible resolution theorem prover. It was designed with a focus on easy plugability of new refinements and on the possibility of interaction with the user. However, it supports only unrestricted resolution and paramodulation so far. It still needs to be tested with large clause sets and other refinements have to be implemented.

7.8 Improvements for HLK

The **continued auto propositional** feature of HLK is currently restricted to sequents that can be derived from *tautological* axiom sequents. However, since **LKDe** allows arbitrary atomic axioms specified by a background theory, it would be interesting if **continued auto propositional** were extended in order to generate proofs automatically also for sequents that can be propositionally derived from any arbitrary axioms, tautological or not.

7.9 Improvements for ProofTool

ProofTool supports global zooming and scrolling, which allow the user to navigate to different parts of the proof and analyze the structure of the proof in different degrees of detail. However, for very large proofs, navigating only via global zooming and scrolling has some disadvantages:

- If the user globally zooms in to see details of a part of the proof, he loses his view of whole proof. Moreover, if he needs to analyze details of various distant regions of the proof, he has to zoom in to one region, then zoom out and scroll to another far region, then zoom in again, and so on. This is sometimes very impractical. A solution to this problem could be the use of a magnifying glass, which would allow the user to locally zoom in, without losing his view of the whole proof, and to easily change the local zoom to other regions of the proof just by moving the magnifying glass.

- Sometimes the user needs to navigate to some specific regions of the proof, but its precise locations are not known a priori. Examples are:

  - Assume that the user is looking at the root inference of the subproof at the left branch of a binary inference. Then he might want to go to "the root inference of the subproof at the right branch of this binary inference".
  - If the user is looking at a certain formula occurrence, he might want to go to "the inferences that contain ancestors of this formula as their main occurrences"

In such situations, the user has to manually scroll and search for the desired regions. Ideally, this problem could be solved by some simple yet expressive enough query language for positions in proofs. In the short-term, keyboard shortcuts for typical queries could be implemented.

While the previous problems are mainly due to proof size, other problems arise from the existence of bureaucracy and trivial structural information in **LKDe**-proofs. Many features shall be added to ProofTool to overcome this. Unary structural inferences could be hidden; colors could be used to highlight different interesting objects (corresponding formulas, ancestor paths, inferences, subproofs) in the proof; and context formulas could be hidden.

Finally, ProofTool's editing capabilities could be extended to allow simple proof transformations to be done directly via the graphical user interface.

7.10 Alternatives for *HandyLK* and HLK

HandyLK is not the only language for the formalization of mathematical proofs. Other languages and systems, such as COQ[9], Isabelle[10], Mizar[11] and ForTheL[12], also aim at helping mathematicians in this task. We would like to study the possibility of integrating these languages and systems to CERES, as alternatives to *HandyLK* and HLK. We foresee two potential compatibility obstacles. On the theoretical side, these languages and systems might not use the same logics and calculi that CERES uses. On the implementation side, these alternatives systems must easily output a formal proof object, and this proof object must be easily convertible to the internal data structures that CERES uses.

8 Conclusion

As summarized in Section 6, the system formed by HLK, CERES and ProofTool has been applied in the transformation and analysis of real mathematical proofs. It constitutes therefore a successful example of automated reasoning being employed in mathematics. In this paper, an overview of the system was given, with a special emphasis on recently added features (e.g. Herbrand sequent extraction) and on extensions that are currently being developed. We showed the current status of our efforts in employing automated proof theoretical methods in mathematical analysis, and we pointed the directions that we intend to follow in order to bring these methods even closer to mathematical practice.

9 Acknowledgements

The authors would like to thank the anonymous referees for their interesting comments and suggestions.

References

[BHK07] Paul Brauner, Clement Houtmann, and Claude Krichner. Principles of Superdeduction. In *Twenty-Second Annual IEEE Symposium on Logic in Computer Science (LiCS)*, 2007.

[BHL+] Matthias Baaz, Stefan Hetzl, Alexander Leitsch, Clemens Richter, and Hendrik Spohr. Ceres: An Analysis of Fürstenberg's Proof of the Infinity of Primes. to appear in Theoretical Computer Science.

[9] COQ Website: http://coq.inria.fr/

[10] Isabelle Website: http://www.cl.cam.ac.uk/research/hvg/Isabelle/

[11] Mizar Website: http://mizar.org/

[12] ForTheL Website: http://nevidal.org.ua/

[BHL+05] Matthias Baaz, Stefan Hetzl, Alexander Leitsch, Clemens Richter, and Hendrik Spohr. Cut-Elimination: Experiments with CERES. In Franz Baader and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning (LPAR) 2004*, volume 3452 of *Lecture Notes in Computer Science*, pages 481–495. Springer, 2005.

[BHL+06] Matthias Baaz, Stefan Hetzl, Alexander Leitsch, Clemens Richter, and Hendrik Spohr. Proof Transformation by CERES. In Jonathan M. Borwein and William M. Farmer, editors, *Mathematical Knowledge Management (MKM) 2006*, volume 4108 of *Lecture Notes in Artificial Intelligence*, pages 82–93. Springer, 2006.

[BK98] Christoph Benzmüller and Michael Kohlhase. Leo — A Higher-Order Theorem Prover. In *Proc. CADE-15*, volume 1421, pages 139–143, Heidelberg, Germany, 1998. Springer-Verlag.

[BL00] Matthias Baaz and Alexander Leitsch. Cut-elimination and Redundancy-elimination by Resolution. *Journal of Symbolic Computation*, 29(2):149–176, 2000.

[BL06] Matthias Baaz and Alexander Leitsch. Towards a clausal analysis of cut-elimination. *Journal of Symbolic Computation*, 41(3–4):381–410, 2006.

[Bus95] S. R. Buss. On Herbrand's theorem. *Lecture Notes in Computer Science*, 960:195, 1995.

[Gen69] G. Gentzen. Untersuchungen über das logische Schließen. In M.E.Szabo, editor, *The Collected Papers of Gerhard Gentzen*, pages 68–131. North-Holland Publishing Company, Amsterdam - London, 1969.

[Her30] J. Herbrand. *Recherches sur la Theorie de la Demonstration*. PhD thesis, University of Paris, 1930.

[Het07] Stefan Hetzl. *Characteristic Clause Sets and Proof Transformations*. PhD thesis, Vienna University of Technology, 2007.

[Het08] Stefan Hetzl. *Proof Profiles. Characteristic Clause Sets and Proof Transformations*. VDM, 2008.

[HJMM04] Thomas A. Henzinger, Ranjit Jhala, Rupak Majumdar, and Kenneth L. Mcmillan. Abstractions from proofs. In *POPL '04: Proceedings of the 31st ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, volume 39, pages 232–244, New York, NY, USA, January 2004. ACM Press.

[HL07] Stefan Hetzl and Alexander Leitsch. Proof Transformations and Structural Invariance. In Stefano Aguzzoli, Agata Ciabattoni, Brunella Gerla, Corrado Manara, and Vincenzo Marra, editors, *Algebraic and Proof-theoretic Aspects of Non-classical Logics*, volume 4460 of *Lecture Notes in Artificial Intelligence*, pages 201–230. Springer, 2007.

[HLWWP08] Stefan Hetzl, Alexander Leitsch, Daniel Weller, and Bruno Woltzenlogel Paleo. Herbrand Sequent Extraction. to appear in the proceedings of Mathematical Knowledge Management (MKM) 2008, 2008.

[McM03] K. L. McMillan. *Interpolation and SAT-Based Model Checking*. Lecture Notes in Computer Science. Springer, 2003.

[Pol54a] G. Polya. *Mathematics and plausible reasoning, Volume I: Induction and Analogy in Mathematics*. Princeton University Press, 1954.

[Pol54b] G. Polya. *Mathematics and plausible reasoning, Volume II: Patterns of Plausible Inference*. Princeton University Press, 1954.

[Sim99] Stephen G. Simpson. *Subsystems of Second Order Arithmetic*. Springer-Verlag, Heidelberg, Germany, 1999.

[Urb00] Christian Urban. *Classical Logic and Computation*. PhD thesis, University of Cambridge, October 2000.

[WP08] Bruno Woltzenlogel Paleo. *Herbrand Sequent Extraction*. VDM-Verlag, Saarbruecken, Germany, 2008.

Automated Theorem Proving in Loop Theory

J. D. Phillips
Wabash College, USA

David Stanovský *
Charles University, Czech Republic

Abstract

In this paper we compare the performance of various automated theorem provers on nearly all of the theorems in loop theory known to have been obtained with the assistance of automated theorem provers. Our analysis yields some surprising results, e.g., the theorem prover most often used by loop theorists doesn't necessarily yield the best performance.

1 Introduction

Automated reasoning tools have had great impact on loop theory over the past decade, both in finding proofs and in constructing examples. It is widely believed that these achievements have transformed loop theory, both as a collection of deep results, as well as the mode of inquiry itself. Automated reasoning tools are now standard in loop theory.

To date, all automated proofs in loop theory have been obtained by Prover9 [McC05] or its predecessor Otter [McC03], and all models have been generated by SEM [ZZ] , Mace4 [McC05], and recently the Loops package for GAP [NV] which G. Nagy used to help find the first nonMoufang, finite simple Bol loop (definitions to follow in section 2), thus solving one of the oldest open problems in loop theory. The present paper is devoted to automated theorem proving. For model building, it seems that GAP/Loops is far better than general purpose automated reasoning tools, especially in certain of the more well known varieties of loops, as it exploits the underlying group theory with its fast algorithms. (Also, most interesting problems about finding finite loops either include properties that cannot be easily formalized in first order theory (such as simplicity), or are known to have lower bound at least several hundred elements.)

While [Phi03] is an introduction to automated reasoning for loop theorists, the present paper is intended as its complement: for computer scientists as an introduction to one of the areas in algebra, namely loop theory, in which automated reasoning tools have had perhaps the greatest impact. The paper is self-contained in that we don't assume the reader is familiar with loop theory.

Our goals are twofold. Firstly, we catalogue the loop theory results to date that have been obtained with the assistance of automated theorem provers. Secondly, we lay the groundwork for developing benchmarks for automated theorem provers on genuine research problems from mathematics. Toward that end, we create a library called QPTP (Quasigroup Problems for Theorem Provers) and test the problems on selected automated theorem provers. Note that we don't intend to mirror the TPTP library [SS98]. Rather, we select a representative subset of problems that mathematicians approached by automated reasoning in their research.

We now give a brief outline of the paper.

Section 2 contains a brief introduction to loop theory, with an emphasis on formal definitions (as opposed to motivation, history, applications, etc.). We think this self-contained introduction to loop theory is the right approach for our intended audience: computer scientists interested in applications of automated reasoning in mathematics. For a more rigorous introduction to the theory of loops see [Bel67], [Bru71], or [Pfl90].

Sutcliffe G., Colton S., Schulz S. (eds.); Proceedings of ESARM 2008, pp. 42-54

*This work is a part of the research project MSM 0021620839 financed by MŠMT ČR. The second author was partly supported by the GAČR grant #201/08/P056.

Section 3 contains a catalogue of all the theorems from loop theory that we used in our analysis. Taken together, the papers that contain these theorems—and we give full citations for all of them—constitute a complete list of those results in loop theory that have been achieved to date with the assistance of automated theorem provers.

Section 4 is devoted to the tests of selected theorem provers on these results.

Section 5 contains final thoughts as well as suggested directions for future work.

Additional information on our library, the problem files and the output files may be found on the website

`http://www.karlin.mff.cuni.cz/~stanovsk/qptp`

2 Basic Loop Theory

We call a set with a single binary operation and with a 2-sided identity element 1 a *magma*. There are two natural paths from magmas to groups, as illustrated in Figure 1.

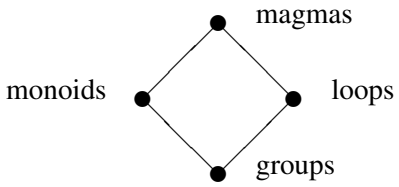

Figure 1: Two paths from magmas to groups.

One path leads through the *monoids*—these are the associative magmas, familiar to every computer scientist. The other path leads through the *loops*—these are magmas in which every equation

$$x \cdot y = z$$

has a unique solution whenever two of the elements x, y, z are specified. Since groups are precisely loops that are also monoids, loops are known colloquially as "nonassociative groups", and via this diagram, they may be thought of as dual to monoids. Many results in loop theory may by regarded as a generalization of results about groups.

As with the class of monoids, the class of loops is too large and general to yield many of its secrets to algebraic inquiry that doesn't focus on narrower subclasses. Here, we simply catalog a few of the most important of these subclasses (the abundant evidence arguing for their importance may be found in many loop theory sources).

First, a comment about notation: we use a multiplication symbol for the binary operation. We usually write xy instead of $x \cdot y$, and reserve $\cdot$ to have lower priority than juxtaposition among factors to be multiplied, for instance, $y(x \cdot yz)$ stands for $y \cdot (x \cdot (y \cdot z))$. We use binary operations $\backslash, /$ of *left* and *right division* to denote the unique solutions of the equation $x \cdot y = z$, ie., $y = x\backslash z$ and $x = z/y$. Loops can thus be axiomatized by the following six identities:

$$x \cdot 1 = x, \quad 1 \cdot x = x,$$

$$x\backslash(xy) = y, \quad x(x\backslash y) = y, \quad (yx)/x = y, \quad (y/x)x = y.$$

Loops without the unit element 1 are refered to as *quasigroups*; in the finite case, they correspond to Latin squares, via their multiplication table.

2.1 Weakening associativity

A *left Bol loop* is a loop satisfying the identity

$$x(y \cdot xz) = (x \cdot yx)z; \tag{lBol}$$

right Bol loops satisfy the mirror identity, namely

$$z(xy \cdot x) = (zx \cdot y)x. \tag{rBol}$$

In the sequel, if we don't specify right or left, and simply write "Bol loop", we mean a left Bol loop.

A left Bol loop that is also a right Bol loop is called *Moufang loop*. Moufang loops are often axiomatized as loops that satisfy any one of the following four equivalent (in loops) identities:

$$x(y \cdot xz) = (xy \cdot x)z, \quad z(x \cdot yx) = (zx \cdot y)x, \quad xy \cdot zx = x(yz \cdot x), \quad xy \cdot zx = (x \cdot yz)x.$$

Generalizing from the features common to both the Bol and the Moufang identities, an identity $\varphi = \psi$ is said to be of *Bol-Moufang type* if: (i) the only operation appearing in $\varphi = \psi$ is multiplication, (ii) the number of distinct variables appearing in φ, ψ is 3, (iii) the number of variables appearing in φ, ψ is 4, (iv) the order in which the variables appear in φ coincides with the order in which they appear in ψ. Such identities can be regarded as "weak associativity". For instance, in addition to the Bol and Moufang identities, examples of identities of Bol-Moufang type include the *extra law*

$$x(y \cdot zx) = (xy \cdot z)x, \tag{extra}$$

and the *C-law*

$$x(y \cdot yz) = (xy \cdot y)z. \tag{C}$$

There are many others, as we shall see. Some varieties of Bol-Moufang type are presented in Figure 2 (for a complete picture, see [PV05]).

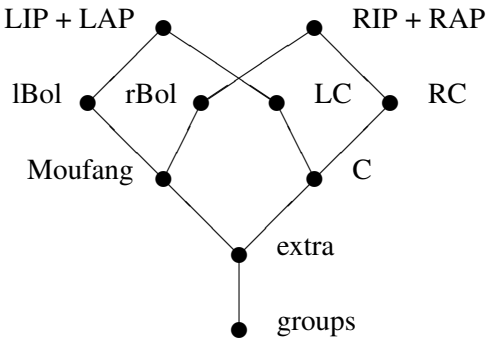

Figure 2: Some varieties of weakly associative loops.

For loops in which each element has a 2-sided inverse, we use x^{-1} to denote this 2-sided inverse of x. In other words,

$$x^{-1}x = xx^{-1} = 1.$$

In Bol loops (hence, also in Moufang loops), all elements have 2-sided inverses. In Moufang loops, inverses are especially well behaved; they satisfy the *anti-automorphic inverse property*

$$(xy)^{-1} = y^{-1}x^{-1}, \tag{AAIP}$$

a familiar law from the theory of groups. Bol loops don't necessarily satisfy the AAIP; in fact, the ones that do (left or right), are Moufang. Dual to the AAIP is the *automorphic inverse property*

$$(xy)^{-1} = x^{-1}y^{-1}. \tag{AIP}$$

Not every Bol loop satisfies the AIP, but those that do are called *Bruck loops*. Bruck loops are thus dual to Moufang loops, with respect to these two inverse properties, in the class of Bol loops.

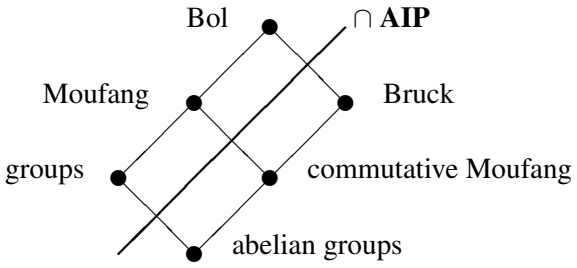

Figure 3: The role of AIP.

A loop is *power associative* if each singleton generates an associative subloop. Bol loops are power associative. Moufang loops satisfy the *flexible* law

$$x \cdot yx = xy \cdot x. \tag{flex}$$

Flexible Bol loops, either left or right, are Moufang. Left Bol loops satisfy both the *left inverse property*

$$x^{-1} \cdot xy = y \tag{LIP}$$

and the *left alternative property*

$$x \cdot xy = xx \cdot y. \tag{LAP}$$

The *right inverse property* (RIP) and the *right alternative property* (RAP) are defined in the obvious ways. The *inverse property* (IP) thus means both the RIP and the LIP, and a loop is called *alternative* if it is both RAP and LAP. Moufang loops and C-loops are alternative and have the inverse property. The *weak inverse property* is given by

$$(yx)\backslash 1 = x\backslash(y\backslash 1). \tag{WIP}$$

2.2 Translations

In a loop Q, the left and right translations by $x \in Q$ are defined by

$$L(x) : y \mapsto xy, \qquad R(y) : x \mapsto xy.$$

The *multiplication group*, $\mathrm{Mlt}(Q)$, of a loop Q is the subgroup of the group of all bijections on Q generated by right and left translations:

$$\mathrm{Mlt}(Q) = \langle R(x), L(x) : x \in Q \rangle.$$

The *inner mapping group* is the subgroup $\mathrm{Mlt}_1(Q)$ fixing the unit element 1. $\mathrm{Mlt}_1(Q)$ is generated by the following three families of mappings, thus rendering the definition equational, and fit for automated theorem provers:

$$T(x) = L(x)^{-1}R(x),$$

$$R(x,y) = R(xy)^{-1}R(y)R(x),$$
$$L(x,y) = L(yx)^{-1}L(y)L(x).$$

If Q is a group, then $\mathrm{Mlt}_1(Q)$ is the group of inner automorphisms of Q. In general, though, $\mathrm{Mlt}_1(Q)$ need not consist of automorphisms. But in those cases in which it does, the loop is called an *A-loop*. Groups and commutative Moufang loops are examples of A-loops.

A subloop invariant to the action of $\mathrm{Mlt}_1(Q)$ (or, equivalently, closed under $T(x)$, $R(x,y)$, $L(x,y)$) is called *normal*. Normal subloops are kernels of homomorphisms, and are thus analogous to normal subgroups in group theory. (In loops, there is no counterpart of the coset definition of a normal subgroup.)

A loop is called *left conjugacy closed* if the conjugate of each left translation by a left translation is again a left translation. This can be expressed equationally as

$$z \cdot yx = ((zy)/z) \cdot zx. \tag{LCC}$$

The definition of *right conjugacy closed* is now obvious, and is given equationally as

$$xy \cdot z = xz \cdot (z\backslash(yz). \tag{RCC}$$

A *conjugacy closed loop* (CC-loop) is a loop that is both LCC and RCC.

We end this section by defining two classes of loops that are closely related to both Moufang loops and A-loops. *RIF loops* are inverse property loops that satisfy

$$xy \cdot (z \cdot xy) = (x \cdot yz)x \cdot y. \tag{RIF}$$

ARIF loops are flexible loops that satisfy

$$zx \cdot (yx \cdot y) = z(xy \cdot x) \cdot y. \tag{ARIF}$$

2.3 Important subsets and subloops

The *commutant*, $C(Q)$, of a loop Q is the set of those elements which commute with each element in the loop. That is,

$$C(Q) = \{c : \forall x \in Q, cx = xc\}.$$

The commutant of a loop need not be a subloop. Even in those cases when the commutant is a subloop (for instance, in Moufang loops), it need not be normal (of course, the commutant in a group is normal, and in group theory it is called the center, as we shall see).

The *left nucleus* of a loop Q is the subloop given by

$$N_\lambda(Q) = \{a : a \cdot xy = ax \cdot y, \forall x, y \in Q\}.$$

The middle nucleus and the right nucleus, $N_\mu(Q)$ and $N_\rho(Q)$ respectively, are defined analogously; both are subloops. The *nucleus*, then, is the subloop given by

$$N(Q) = N_\lambda(Q) \cap N_\mu(Q) \cap N_\rho(Q).$$

The *center* is the normal subloop given by

$$Z(Q) = N(Q) \cap C(Q),$$

thus coinciding with the language from groups. $C(Q)$ need not have any relationship with $N(Q)$; that is, $C(Q) \cap N(Q) = Z(Q)$ can be trivial. The situation in Bol loops is strikingly different. In a (left) Bol loop

Q, $N_\lambda(Q) = N_\mu(Q)$, and this subloop need not have any relationship with $N_\rho(Q)$, i.e., the intersection can be trivial. Thus, in a Moufang loop, all nuclei coincide, and $N(Q)$ is a normal subloop. Moreover, if Q is Bruck, then $N_\lambda(Q) \leq C(Q)$.

The *commutator*, $[x,y]$ of x and y, in a loop Q is given by

$$xy = yx \cdot [x,y].$$

The *associator*, $[x,y,z]$ of x, y, and z, is given by

$$xy \cdot z = (x \cdot yz) \cdot [x,y,z].$$

The point is that the lack of associativity in loops provides a structural richness, part of which can be captured equationally, thus rendering loops excellent algebraic objects to investigate with automated theorem provers.

3 The Theorems

The present section catalogues all papers in loop theory to date whose results were obtained with the assistance of an automated theorem prover. All proofs were obtained by Prover9 or Otter. The proofs were always translated to human language and usually simplified (none of the papers presents a raw output from a prover), hence none of the results relies on soundness of Otter/Prover9. As far as we know, no automatically generated proof was found to be incorrect during translation.

In some cases, the main results weren't obtained directly with automated theorem provers. Instead, provers were used to prove key technical lemmas, or even just special cases, which in turn helped mathematicians find proofs of the main results. This is explained in more detail below, see, e.g., our description of [AKP06].

We list the papers in chronological order. From each paper, we choose up to five theorems for the QPTP library.

[Kun96a]. This is an important paper, because it was the first to use automated theorem provers in loop theory and, in fact, one of the first noneasy results in mathematics obtained automatically. The theorem says that a quasigroup satisfying any one of the four Moufang laws is, in fact, a loop, i.e., has a unit element. We analyze this result for each of the four Moufang identites. Note that the proof for the third and the fourth Moufang identities can by done relatively easily by hand, while the proof for the first and the second one was only discovered by Otter.

[Kun96b]. This is a sequel to the previous paper. The main result is the determination of which of the Bol-Moufang identities, implies, in a quasigroup, the existence of a unit element. We analyze three of these identities.

[Kun00]. There are many results in this paper proved by automated theorem provers. We analyze the following two: (1) If G is conjugacy closed, with $a,b \in G$ and $ab = 1$, then ba is in the nucleus of G. (2) If G is conjugacy closed, the commutant of G is contained in the nucleus.

[KKP02a]. The main result in this paper is that inverse property A-loops are Moufang; we analyze this result. This was one of the major long-standing open problems in loop theory, and perhaps the most important automated theorem proving success in loop theory. And it marks the point at which the number of loop theorists using automated theorem provers in their work jumped from one to three.

[KKP02b]. There are many results in this paper proved by automated theorem provers; we include the following four: (1) 2-divisible ARIF loops are Moufang, (2) flexible C-loops are ARIF, (3) Moufang loops are RIF, (4) RIF loops are ARIF.

[KK04]. There are many results in this paper in which automated theorem provers helped, e.g., finite nonassociative extra loops have nontrivial centers. We analyze the following result: In an extra loop, z commutes with $[x,y,t]$ if and only if t commutes with $[x,y,z]$ if and only if $[x,y,z][x,y,t] = [x,y,zt]$.

[KKP04]. There are many results in this paper proved by automated theorem provers. We include the following one: in CC-loops, associators are in the center of the nucleus.

[KP04]. The main result in this paper is that commutants of Bol loops of odd order are, in fact, subloops. Obviously, this is not a first order statement, however its proof relies on several lemmas proved by a theorem prover. We analyze the following one: If Q is a Bol loop, and if $a,b \in C(Q)$, then so too are a^2, b^{-1} and a^2b.

[KP05]. The main result in this paper is to give a basis for the variety of rectangular loops which consists of 7 identities, thus improving Krapež's pre-existing basis of 12 axioms [Kra00]. A *rectangular loop* is a direct product of a loop and a rectangular band. A *rectangular band* is a semigroup which is a direct product of a left zero semigroup and right zero semigroup. A *left* (*right*, resp.) *zero semigroup* is a semigroup satisfying $x \cdot y = x$ ($x \cdot y = y$, resp.). We analyze part of this result by showing that the identities

$$x\backslash(xx) = x,\ (xx)/x = x,\ x\cdot(x\backslash y) = x\backslash(xy),\ (x/y)\cdot y = (xy)/y,\ x\backslash(x(y\backslash y)) = ((x/x)y)/y,$$

$$(x\backslash y)\backslash((x\backslash y)\cdot(zu)) = (x\backslash(xz))\cdot u,\ ((xy)\cdot(z/u))/(z/u) = x\cdot((yu)/u)$$

imply each of the following identities (in algebras with three binary operations $\cdot, \backslash$, and $/$):

$$(x\backslash y)\backslash((x\backslash y)z) = x\backslash(xz),\ (x/y)\backslash((x/y)z) = x\backslash(xz),\ x(y\backslash(yz)) = xz,\ ((xy)/y)z = xz,$$

$$(x\cdot yz)/(yz) = (xz)/z,\ (x(y\backslash z))/(y\backslash z) = (xz)/z,\ (x(y/z))/(y/z) = (xz)/z.$$

[PV05]. The main result of this paper is the systematic classification of all varieties of loops axiomatized by a single identity of Bol-Moufang type, achieved to a large extent automatically. We include a typical result from [PV05]: in loops, the following two identities are equivalent (and thus both axiomatize the so-called variety of LC-loops): $x(y\cdot yz) = (x\cdot yy)z$ and $xx\cdot yz = (x\cdot xy)z$.

[AKP06]. One of the main results in this paper is that in a Bruck loop, elements of order a power of two commute with elements of odd order. Obviously, automated theorem provers can't prove this result directly, as it is a result about infinitely many positive integers. On the other hand, one may use automated theorem provers to generate proofs about *specific* integers, and then use these proofs to help construct the proof of the general result. The three specific cases we analyze here: in a (left) Bruck loop, elements of order 2^2 commute with elements of oder 3, elements of order 2^2 commute with elements of order 3^2, and elements of order 2^4 commute with elements of order 3^2. The three different cases give rise to clear performance differences between the automated theorem provers, as we shall see. We note that this property was used in [AKP06] in a proof of a deep decomposition theorem for Bruck loops. That is, this also was an important success for automated theorem provers in loop theory.

[KK06]. There are many results in this paper proved by automated theorem provers. We analyze the following results: for each c in a power associative conjugacy closed loop, c^3 is WIP (i.e., $c^3(xc)^{-1} = x^{-1}$ for every x), c^6 is extra (i.e., $c^6(x \cdot yc^6) = (c^6x \cdot y)c^6$ for every x, y) and c^{12} is in the nucleus. (Initially, the last property wasn't obtained directly by Prover9. Interestingly, other provers can do it.)

[Phi06]. The main result in this paper is that the variety of power associative, WIP conjugacy closed loops is axiomatized, in loops, by the identities $(xy \cdot x) \cdot xz = x \cdot ((yx \cdot x)z)$ and $zx \cdot (x \cdot yx) = (z(x \cdot xy)) \cdot x$. We analyze this result.

[PV06]. There are many results in this paper proved by automated theorem provers. We analyze the following two: (1) in C-loops, the nucleus is normal, and (2) in a commutative C-loop, if a has order 4 and b has order 9, then $a \cdot bx = ab \cdot x$ (this is one of the cases that led to a proof of the decomposition theorem for commutative torsion C-loops).

[KKP07]. An *F-quasigroup* is a quasigroup that satisfies the following two equations: $x \cdot yz = xy \cdot (x\backslash x)z$ and $zy \cdot x = z(x/x) \cdot yx$. The main result of the paper is that every F-quasigroup is isotopic to a Moufang loop. This was a long-standing open problem—it was the first open problem listed in Belousov's 1967 book [Bel67]. We analyze this result.

[KPV07]. There are many results in this paper proved by automated theorem provers. We analyze the following one: a C-loop of exponent four with central squares is flexible.

[KPV08]. There are many results in this paper proved by automated theorem provers. We include the following one: in a Bol loop, if c is a commutant element, then c^2 is in the left nucleus if and only if c is in the right nucleus.

[PV08]. The purpose of this paper is to find group-like axiomatizations for the varieties of loops of Bol-Moufang type. We include the following typical result: a magma with 2-sided inverses satisfying the C-law is a loop.

[CDKxx]. A *Buchsteiner loop* is a loop that satisfies the following identity: $x\backslash(xy \cdot z) = (y \cdot zx)/x$. These loops are closely related to conjugacy closed loops, and are closely related to loops of so-called Bol-Moufang type [DJxx]. The result from [CDKxx] that we analyze here is that in Buchsteiner loops, fourth powers are nuclear (i.e., $x^4 \in N(Q)$ for every $x \in Q$).

[KKPxx]. The main result in this paper is that in a strongly right alternative ring (with a unit element), the set of units is a Bol loop under ring multiplication, and the set of quasiregular elements is a Bol loop under "circle" multiplication. A *right alternative ring* is a set, R, with two binary operations, $+$ and $\cdot$, such that under $+$, R is an abelian group, under $\cdot$, R is a right alternative magma, and such that $\cdot$ distributes over $+$. A right alternative ring is *strongly right alternative* if $\cdot$ is a right Bol loop. A *unit* in an alternative ring is an element that has a two-sided inverse. The circle operation is given by $x \circ y = x + y + xy$. And finally, an element is *quasiregular* if it has a two-sided inverse under circle, e.g. $x \circ x' = x' \circ x = 0$. We analyze the following technical result: If a has a 2-sided inverse, then $R(a^{-1}) = R(a)^{-1}$ and $L(a)^{-1} = R(a)L(a^{-1})R(a^{-1})$.

[KVxx]. There are many results in this paper proved by automated theorem provers. We analyze the following one: in a commutative RIF loop, all squares are Moufang elements and all cubes are C-elements. An element a is a *Moufang element* if for all x and y, $a(xy \cdot a) = ax \cdot ya$. And it is a *C-element* if for all x and y, $x(a \cdot ay) = ((xa \cdot a)y$.

A remark on TPTP. The intersection of the TPTP and QPTP libraries is empty (by now). The only loop theory problem in TPTP is the equivalence of the four Moufang identities (GRP200 – GRP206). This result is included in the book [MP96], which demonstrates the power of Otter in selected areas of mathematics (the other loop theory problems in the book are several single axioms).

4 Benchmark tests

In the present section, we analyze the problems in the QPTP library by running them on selected automated theorem provers. Based on the results of the CASC competition in recent years [SS06], we chose the following five provers: E [Sch02], Prover9 [McC05], Spass [S], Vampire [RV02] and Waldmeister [Hil03].

We ran each prover on each file twice: with 3600 and 86400 seconds time limit (1 hour and 1 day). The problems from the library were translated to the TPTP syntax, and the input files for the provers were generated by the tptp2X tool. We ran the provers with their default settings, and we didn't tune any of the input files for a particular prover, thus obtaining conditions similar to the CASC competition.

Our results are presented in Figure 4. The names of the problems start with the code of the paper in the bibliography followed by the code of the selected result. In the case when there is no single obvious way to formalize the statement in first order theory, alternative axiomatizations are given. (For details, see the QPTP website.) Running time (i.e., the time it took to find a proof) is displayed in rounded seconds; a blank space means timeout, cross means that the problem is not equational and thus ineligible for Waldmeister. Running times over 360s (the time limit of the last CASC) are displayed in bold, running times over 1 hour in italic.

The total number of problems in QPTP is 80, of which 68 are equational. 71 problems were solved by at least one prover, 38 by all of them. The overall performance of the provers is summarized in Figure 5.

In our study, Waldmeister performed better than the other four provers on equational problems. The performances of E, Prover9 and Vampire seem to be similar (incomparable in the strict sense), although E can be quite fast on some difficult problems. Spass seems to be well behind the other provers.

In order to minimize bias in our study, we ignored basic parameter settings. While some provers work fully automatically (e.g., Vampire), other actually have *no* default setting (e.g., Waldmeister). In our case, "default" is defined by the output of tptp2X. In particular, this means the set(auto) mode for Prover9 and some explicit term ordering for Waldmeister.

In fact, term ordering is perhaps the most influential parameter. Waldmeister's default ordering is KBO with weights 1, while Prover9's default is LPO. If Prover9 is manually reset to KBO, it proves six additional files, but fails for two files that were proved with LPO; this way, Prover9's performance becomes closer to Waldmeister's. Another influential parameter is symbol ordering. Prover9 chooses a relatively smart one (inverse $>$ left/right division $>$ multiplication), while Waldmeister gets from tptp2X an alphabetical one (inverse $>$ left division $>$ multiplication $>$ right division). When reset to the smarter choice, it's on average much faster, although it fails to prove any additional problems. (Note that perhaps the smartest choice is division $>$ inverse $>$ multiplication.) Indeed, parameter setting deserves much greater attention, but this is beyond the scope of the present study.

file/prover	E 0.999-006	Prover9 1207	Spass 3.0	Vampire 8.0	Waldmeister 806
AKP06_1	0	11	**459**	6	0
AKP06_2	16	**1110**			74
AKP06_3					
CDKxx_1a					
CDKxx_1b					
CDKxx_1c					
KK04_1					*29919*
KK04_2					*29387*
KK04_3					*10322*
KK06_1a	57			**507**	59
KK06_1b	**922**				**592**
KK06_1c	*46277*				**570**
KK06_1d	*53534*				**560**
KK06_1e	*46687*				**554**
KKP02a_1	**3023**	*26735*			x
KKP02a_1alt1	**848**	*36852*		**553**	205
KKP02a_1alt2	**848**	*35016*		**500**	208
KKP02a_1alt3	**1001**	*24832*		**550**	213
KKP02a_1alt4	**1018**	*24242*		**584**	202
KKP02b_1	4	120	99	97	x
KKP02b_1alt1	9	205		**488**	8
KKP02b_1alt2	3	147	**413**	**484**	8
KKP02b_1alt3	9	208	*56918*	**491**	9
KKP02b_1alt4	9	190	*53195*	**485**	9
KKP02b_2	0	0	**475**	10	1
KKP02b_3	0	0	0	0	2
KKP02b_4a	31	**1462**		138	4
KKP02b_4b	0	0	0	0	0
KKP04_1a					
KKP04_1b					
KKP04_1c					
KKP04_2				**2856**	**580**
KKP07_1					**2265**
KKPxx_1	2	0	3	8	0
KKPxx_2a					
KKPxx_2b					
KP04_1	0	0	0	0	0
KP04_2	0	0	0	0	0
KP04_3	7	73	*72463*	270	2
KP05_1a	0	0	0	0	0
KP05_1b	0	0	0	0	0
KP05_1c	0	0	0	0	0
KP05_1d	0	0	0	0	0
KPV07_1	0	0	0	0	0
KPV08_1	0	0	0	0	0
KPV08_2	0	0	0	0	0
Kun00_1a	352	*7088*		*12482*	**705**
Kun00_1b	353	*7536*		*15412*	**736**
Kun00_1c	353	**3264**		*19762*	**706**
Kun00_1alt1		*38587*			**690**
Kun00_2	0	0	0	0	0
Kun96a_1	56	75		258	x
Kun96a_1alt1	128	112		218	3
Kun96a_1alt2	9	68		238	3
Kun96a_2	57	**1256**		285	x
Kun96a_2alt1	8	**398**		284	3
Kun96a_2alt2	51	**1282**		164	3
Kun96a_3	0	0	0	0	x
Kun96a_4	0	0	0	0	x
Kun96b_1	0	0	1	0	x
Kun96b_2	0	1	9	0	x
Kun96b_3	0	19	161	5	x
Kun96b_3alt1	0	7	125	28	0
Kun96b_3alt2	0	5	148	43	0
KVxx_1		357		**1692**	49
KVxx_2		**1705**		**3172**	95
Phi06_1a	72	29		14	21
Phi06_1b	46	2	*8632*	6	17
Phi06_2a	0	118	41	1	0
Phi06_2b	0	1	0	**473**	0
Phi06_2c	0	0	0	0	x
Phi06_3	0	41	**857**	9	0
PV05_1	0	1	19	6	0
PV05_2		9	5	1	0
PV06_1a	0	0	0	0	0
PV06_1b	0	0	0	0	0
PV06_1c	0	0	0	0	0
PV06_2	34	17		4	0
PV08_1a	0	0	1	0	x
PV08_1b	0	0	0	10	x

Figure 4: Detailed results.

prover	E 0.999	Prover9 1207	Spass 3.0	Vampire 8.0	Waldmeister 806
proofs in 360s	53	46	31	44	46
proofs in 3600s	59	53	35	57	56
proofs in 86400s	62	61	39	60	59
timeouts	18	19	41	20	9

Figure 5: Summary.

Finally, the reader may wonder about those theorems on which all provers were unsuccessful (these are indicated by blank entries in Figure 4). After all, these are theorems that were first proved with the assistance of an automated theorem prover (which was the sole criterion for inclusion in our study). Why were none of the provers able to find proofs in our study? Firstly, we didn't tune the input files. And, perhaps more importantly, we didn't use any advanced techniques in our study (e.g., Prover9's powerful hints strategy), which is often the way to obtain a new mathematical result.

5 Conclusions

While we hope our results are interesting to automated reasoning researchers (especially since they involve problems from an active area of mathematical research), they may not be *surprising* to these same researchers, informed as these researchers are by the CASC results over the past ten years. Our results, though, might surprise loop theorists, who are less familiar with most of the provers in our study. But again, we stress that some of these loop theory results were originally obtained using advanced Otter/Prover9 techniques such as the hints strategy, or sketches [Ver01]. Could these be implemented in other provers?

Since the various automated theorem provers have different strengths and weaknesses, loop theorists could profit by using a suite of theorem provers in their investigations. For instance, the result in [KKP07] was originally derived as a series of results, a number of steps eventually leading to the main theorem. In our study, Waldmeister proved it from scratch in 40 minutes. To state the obvious: some theorems will be missed if one uses only one automated theorem prover. On the other hand, the actual proofs themselves are, of course, of great importance, and the various automated theorem provers differ greatly in this regard. Some provers don't even give the proof, they simply indicate that they've found one, while others, notably Prover9, produce relatively readable proofs, and even include tools to simplify them further.

There are clearly many opportunities for future work. We intend to keep our catalogue of loop theory results as up-to-date as possible. We eventually hope to test our files on more automated theorem provers; in particular, on some instantiation based ones, to check the hypothesis that those are weak on algebraic problems (that always require a lot of computation with equality). Another obvious direction for future work is to analyze results in other domains, for instance quasigroups and other nonassociative algebras.

It would be immensely useful to compare the various provers' performance by using input files and techniques designed to obtain the best performance for each particular prover. In particular, and for example, what are the best first order descriptions of properties like "existence of a unit" (the formula $\exists x \forall y(xy = y \ \& \ yx = x)$, or the identities $(x/x)y = y, y(x/x) = y$, or the identities $(x\backslash x)y = y, y(x\backslash x) = y$), or of the Moufang property (in what ways might the fact that the four defining identities of this variety are equivalent impact "best performance" strategies amongst the various provers?).

A possible new direction of exploiting automated reasoning to prove new theorems about loops could be, first, to create a knowledge base of definitions and theorems in loop theory (those that can be expressed within the first order theory of loops) and then to apply tools for reasoning in large theories. Particularly, such a knowledge base would substantially differ from other recent projects such as the MPTP [Urb04]. The QPTP library can be viewed as the zeroth step towards such a library.

Acknowledgement. We thank Michael Kinyon and Bob Veroff for carefully reading, and then commenting on, an earlier version of this paper.

[AKP06] M. Aschbacher, M.K. Kinyon, and J.D. Phillips, Finite Bruck loops, *Transactions of the American Mathematical Society*, **358** (2006), 3061–3075.

[Bel67] V.D. Belousov, Foundations of the Theory of Quasigroups and Loops, Nauka, Moscow, 1967 (in Russian).

[Bru71] R. H. Bruck, A Survey of Binary Systems, third printing, corrected, *Ergebnisse der Mathematik und ihrer Grenzgebiete*, New Series **20**, Springer-Verlag, 1971.
[CDKxx] P. Csorgö, A. Drápal, M.K. Kinyon, Buchsteiner loops, submitted.
[DJxx] A. Drápal, P. Jedlička, On loop identities that can be obtained by nuclear identification, submitted.
[Hil03] T. Hillenbrand, Citius altius fortius: Lessons Learned from the Theorem Prover Waldmeister, in Dahn I., Vigneron L., Proceedings of the 4th International Workshop on First-Order Theorem Proving (Valencia, Spain), Electronic Notes in Theoretical Computer Science 86.1, Elsevier Science, 2003.
`http://www.waldmeister.org`
[Kra00] A. Krapež, Rectangular loops, *Publ. Inst. Math. (Beograd) (N.S.)*, **68(82)** (2000), 59–66.
[Kun96a] K. Kunen, Moufang quasigroups *Journal of Algebra*, **183** (1996) no. 1, 231–234.
[Kun96b] K. Kunen, Quasigroups, loops, and associative laws *Journal of Algebra*, **185** (1996) no. 1, 194–204.
[Kun00] K. Kunen, The structure of conjugacy closed loops, *Transactions of the American Mathematical Society*, **352** (2000) no. 6, 2889–2911.
[KK04] M.K. Kinyon and K. Kunen, The structure of extra loops *Quasigroups Related Systems*, **12** (2004), 39–60.
[KK06] M.K. Kinyon and K. Kunen, Power-associative, conjugacy closed loops, *Journal of Algebra*, **304** (2006), no. 2, 679–711.
[KKP02a] M.K. Kinyon, K. Kunen, and J.D. Phillips, Every diassociative A-loop is Moufang, *Proceedings pf the American Mathematical Society*, **17** 130 (2002), 619–624.
[KKP02b] M.K. Kinyon, K. Kunen, and J.D. Phillips, A generalization of Moufang and Steiner loops, *Algebra Universalis*, **48** (2002), 81–101.
[KKP04] M.K. Kinyon, K. Kunen, and J.D. Phillips, Diassociativity in conjugacy closed loops, *Communications in Algebra*, **32** (2004), 767–786.
[KKP07] T. Kepka, M.K. Kinyon, and J.D. Phillips, The structure of F-quasigroups, *Journal of Algebra*, 317 (2007), 435–461.
[KKPxx] M.K. Kinyon, K. Kunen, and J.D. Phillips, Strongly right alternative rings and Bol loops, *Publicationes Mathematicae Debrecen*, submittted.
[KP04] M.K. Kinyon and J.D. Phillips, Commutants of Bol loops of odd order, *Proceedings of the American Mathematical Society*, **132** (2004), 617–619.
[KP05] M.K. Kinyon and J.D. Phillips, Rectangular quasigroups and rectangular loops, *Computers and Mathematics with Applications* **49** (2005), **11–12**, 1679–1685.
[KPV07] M.K. Kinyon, J.D. Phillips, and P. Vojtěchovský, C-loops: extensions and constructions, *Journal of Algebra and its Applications*, **6 (1)**, (2007), 1–20.
[KPV08] M.K. Kinyon, J.D. Phillips, and P. Vojtěchovský, When is the commutant of a Bol loop a subloop? *Transactions of the American Mathematical Society*, 360 (2008), no. 5, 2393–2408.
[KVxx] M.K. Kinyon and P. Vojtěchovský, Primary decompositions in varieties of commutative diassociative loops, submitted.
MP96 W.W. McCune and R. Padmanabhan, *Automated Deduction in Equational Logic and Cubic Curves*, Springer-Verlag, 1996.
[McC03] W. W. McCune, *OTTER 3.3 Reference Manual and Guide*, Argonne National Laboratory Technical Memorandum ANL/MCS-TM-263, 2003;
`http://www.mcs.anl.gov/AR/otter/`
[McC05] W. W. McCune, *Prover9*, automated reasoning software, and *Mace4*, finite model builder, Argonne National Laboratory, 2005.
`http://www.prover9.org`
[NV] G. P. Nagy and P. Vojtěchovský, *LOOPS: a package for GAP* 4,
available at `http://www.math.du.edu/loops`

[Pfl90] H. O. Pflugfelder, Quasigroups and Loops: Introduction, *Sigma Series in Pure Mathematics* **7**, Heldermann Verlag Berlin, 1990.
[Phi03] J.D. Phillips See Otter digging for algebraic pearls, *Quasigroups and Related Systems*, **10** (2003), 95–114.
[Phi06] J.D. Phillips, A short basis for the variety of WIP PACC-loops, *Quasigroups and Related Systems*, **14** (2006), 73–80.
[PV05] J.D. Phillips and P. Vojtěchovský, The varieties of loops of Bol-Moufang type, *Algebra Universalis*, **54 (3)** (2005), 259–271.
[PV06] J.D. Phillips and P. Vojtěchovský, C-loops: an introduction, *Publicationes Mathematicae Debrecen*, **68/1–2** (2006), p. 115–137.
[PV08] J.D. Phillips and P. Vojtěchovský, A scoop from groups: new equational foundations for loops, *Commentationes Mathematicae Universitatis Carolinae*, 49/2 (2008), 279–290.
[RV02] A. Riazanov, A. Voronkov, The Design and Implementation of Vampire, *AI Communications* 15(2-3) (2002), pp.91-110.
[S] `http://spass.mpi-sb.mpg.de/`
[Sch02] S. Schulz, E: A Brainiac Theorem Prover, *AI Communications* 15(2-3), 111–126.
`http://www4.informatik.tu-muenchen.de/~schulz/WORK/eprover.html`
[SS98] G. Sutcliffe, C. Suttner, The TPTP Problem Library: CNF Release v1.2.1, *Journal of Automated Reasoning*, 21/2 (1998), 177–203.
[SS06] G. Sutcliffe, C. Suttner, The State of CASC, *AI Communications* 19/1 (2006), 35-48.
[Urb 04] Josef Urban: MPTP - Motivation, Implementation, First Experiments. *J. Autom. Reasoning* 33(3-4): 319-339 (2004)
[Ver01] R. Veroff, Solving open questions and other challenge problems using proof sketches, *J. Automated Reasoning* 27(2) (2001), 157–174.
[ZZ] J. Zhang, H. Zhang, SEM: a System for Enumerating Models,
`http://www.cs.uiowa.edu/~hzhang/sem.html`

Generating Loops with the Inverse Property

John Slaney, Asif Ali
Australian National University, Australia

Abstract

This is an investigation in the tradition of Fujita *et al* (IJCAI 1993), Zhang *et al* (JSC 1996), Dubois and Dequen (CP 2001) in which CP or SAT techniques are used to answer existence questions concerning small algebras. In this paper, we open the attack on IP loops, an interesting and under-investigated variety intermediate between loops and groups.

1 Introduction

Automated reasoning techniques, particularly those of propositional satisfiability (SAT) and finite domain (FD) constraint satisfaction, are obviously applicable to the problem of enumerating small algebraic structures, and so should standardly be used to answer existence questions at least concerning very small cases. In this paper, we report on an investigation of IP loops, a variety intermediate between loops and groups which has been known for many years but to our knowledge never explored in much detail. We used FINDER [Sla94] to enumerate all IP loops up to order 13, and the commutative ones of order 14, and obtained a number of new results prompted by observation of these algebras. The numbers of algebras were reported in a recent note [AS08] and the full list of small IP loops is freely available online [SA07].

1.1 Algebraic background

A *quasigroup* is a groupoid with left and right division operators $/$ and $\backslash$. That is, it satisfies the laws:

$$x(x\backslash y) = y$$
$$(x/y)y = x$$
$$x\backslash xy = y$$
$$xy/y = x$$

In the finite case, this amounts simply to satisfying the left and right cancellation laws:

$$xy = xz \Rightarrow y = z$$
$$xz = yz \Rightarrow x = y$$

That is, its "multiplication table" is a Latin square, each row and each column being a permutation of the elements. A quasigroup is a *loop* iff it has a (right and left) identity: an element e such that

$$ex = x = xe$$

for all x. Loops in general are so numerous that almost all work on them has concerned special cases. One of the earliest classes of loops to be investigated was that of *Steiner loops*, which satisfy the additional postulates

$$(xy)y = x$$
$$x(xy) = y$$

Clearly, in any loop, each element x has a left inverse—an element y such that $yx = e$—and a right inverse

Sutcliffe G., Colton S., Schulz S. (eds.); Proceedings of ESARM 2008, pp. 55-66

z such that $xz = e$. In the case of Steiner loops, both y and z are just x. A weaker condition, also satisfied by groups, is that the left and right inverse operations coincide, meaning that for every x there is an element x^{-1} such that

$$xx^{-1} = e = x^{-1}x$$

Note that $(x^{-1})^{-1} = x$.

A loop is said to have the *inverse property*, and is called an *IP loop*, iff it is a loop with inverse such that for all elements x and y

$$x^{-1}(xy) = y = (yx)x^{-1}$$

It is not hard to see that IP loops also satisfy the principle $(xy)^{-1} = y^{-1}x^{-1}$. A Steiner loop is an IP loop of exponent 2 (i.e. such that $x^2 = e$ for all x) and a group is simply an associative IP loop. Moufang loops, which have been studied intensively, are IP loops satisfying the identity

$$x(z(yz)) = ((xz)y)z$$

IP loops are of interest as a strong and natural generalisation of both groups and Steiner loops. Moreover, they correspond exactly to semiassociative relation algebras [Mad82] in the same sense that groups correspond to (associative) relation algebras.[1] It is therefore a little surprising that they have attracted comparatively slight attention from algebraists.

The smallest IP loop that is not a group is of order 7:

$*$	1	2	3	4	5	6	7
$e=1$	1	2	3	4	5	6	7
2	2	3	1	6	7	5	4
3	3	1	2	7	6	4	5
4	4	7	6	5	1	2	3
5	5	6	7	1	4	3	2
6	6	4	5	3	2	7	1
7	7	5	4	2	3	1	6

x	x^{-1}
1	1
2	3
3	2
4	5
5	4
6	7
7	6

This structure has proper subalgebras $\{1,2,3\}$, $\{1,4,5\}$ and $\{1,6,7\}$. Note that the order of these subloops does not divide the order of the loop, marking a significant difference between IP loops and groups.[2] Associativity fails in that, for instance, $(2*2)*4 = 3*4 = 7$ while $2*(2*4) = 2*6 = 5$.

2 Generating IP loops

It is frequently useful to enumerate small examples of a class of algebraic structures, so that by examining what exists, and observing places where no such structures exist, the mathematician can gain a "feel" for the objects in question. At the simplest, the spectrum (the set of numbers n for which such algebras of order n exist) can have its initial segment settled by enumeration. In some cases, this suffices to allow the entire spectrum to be determined; in others, it merely dispposes of some awkward questions and suggests a conjecture concerning the rest. In many cases, "off the shelf" reasoning systems suffice for the enumeration, making this an attractive application domain for automated reasoning.

[1] Let $G = \langle S, * \rangle$ be a groupoid. The field of sets consisting of the power set of L, with $*$ raised to sets in the obvious pointwise manner, is a relation algebra iff G is a group, and a semiassociative relation algebra iff G is an IP loop.

[2] A loop in which the order of every subloop divides the order of the loop is said to have the *weak Lagrange property*. It has the *strong Lagrange property* if every subloop has the weak property.

```
solve satisfy;
int N;
type element = 1..N;

array[element,element] of var element: star;

constraint
  forall (x,y in element) (star[star[star[y,x],y],y] = x);

N = 3;
```

Figure 1: Zinc encoding of quasigroup existence problem QG5(3)

2.1 History

Fujita *et al* [FSB93] used the ICOT group's 'Model Generation Theorem Prover', a propositional reasoner in the style of SATCHMO [MB88], and other tools including FINDER to solve open problems in the theory of quasigroups by proving the existence or nonexistence of quasigroup models of certain equations. During the 1990s, this work was taken up and extended, notably by Hantao Zhang and his collaborators through the SATO system [ZBH96] and by McCune, Stickel and others [McC, ZS00]. In the constraint programming community, there were interesting developments concerning efficient encodings [DD01] and in the SAT community concerning symmetry avoidance [Zha96, AH01]. Recently, it has been shown [APSS05] that preprocessing of SAT encodings using restricted variants of resolution can simplify some of the quasigroup existence problems to the point that stochastic local search (SLS) solvers can successfully prove existence (though not nonexistence, of course). Meanwhile, the related problem of quasigroup completion [GS97] has become a well-established benchmark constraint satisfaction problem, offering as it does a nice balance between the highly structured and the random. Benchmark collections of SAT, CSP and SMT problems now routinely contain problems about quasigroups.

2.2 Problem representation

The simplest way to represent existence problems about quasigroups, loops, groups or other groupoids for automated reasoning purposes is to cast them as finite domain CSPs where each entry $\langle x, y\rangle$ in the "multiplication table" of the groupoid is a CSP variable whose domain consists of the elements of the algebra. Take for example the problem QG5(3). This requires the matrix

*	1	2	3
1			
2			
3			

to be filled with nine entries chosen from the values $1\ldots3$, in such a way that they form a Latin square and that the equation $(yx.y)y = x$ holds for all x and y. In fact, if they satisfy the equation, the cancellation properties follow. In the CSP modelling language Zinc [dlBMRW06] for instance, this is directly expressible (see Figure 1). Other such languages for constraint programming make it similarly easy to state the problem.

The equation flattens to $\forall x\forall y\forall w\forall z((yx = w \wedge wy = z) \Rightarrow zy = x)$ which has 4 variables and therefore $3^4 = 81$ domain-grounded instances obtained by substituting the three possible values 1, 2, 3 for the variables. Each of those instances relates a triple (possibly with repetition) of entries in the table,

and correspondingly imposes a constraint of cardinality at most 3 on the variables of the CSP. In the straightforward SAT recension, each possible value assignment $a * b = c$ is represented by a propositional variable p_{abc}, and each domain-grounded instance of the flattened equation becomes a 3-clause on these variables. Other encodings are possible, of course, but the suggested one is standard. Once the problem is so encoded, any FD or SAT solver can be used to solve it. Theorem provers, whether based on resolution and its variants or on term rewriting, can also be used to make inferences on either the first order or propositional levels.

To generate IP loops, we need another array of decision variables representing the inverse function, and of course the appropriate equations. It is useful to add a few redundant constraints, to strengthen propagation. We added the fact that inverse is of period 2 and the duality equation $(xy)^{-1} = y^{-1}x^{-1}$. Since we wish to enumerate isomorphism classes, it is important to avoid generating too many isomorphic copies of the solutions, which means we need to break symmetries. In order to break some of the many symmetries in a simple way, we required the identity e to be the lowest-valued element and x^{-1} to be in the range $x-1 \ldots x+1$, with self-inverse elements coming first in the order.

2.3 FINDER is good enough

For our work, we used FINDER, which stands somewhere between the FD and SAT solver families. It represents the problem in the FD manner rather than explicitly rendering it into SAT, so for example its variable selection heuristic looks at the FD variables, not at specific values for them—typically it looks for the smallest available domain—but it reasons somewhat like a SAT solver rather than in typical FD style. In particular, it uses unit resolution on the ground constraints together with forward-checking as its notion of local consistency, and it learns nogoods.

FINDER is far from representing the state of the art in finite model building: we expect that similar results could be produced faster using more recent technology such as Paradox, which is based on the much more efficient SAT solver Minisat. It suffices for our purpose, however, as it can find the solutions faster than they can be checked for isomorphism and has completed the order 13 search in reasonable time.

To count the isomorphism classes, it is necessary either to reason in a sophisticated way about symmetries during the search[3] or to remove redundant solutions from the output in a postprocessing phase. We chose the latter: our postprocessor takes each generated IP loop in turn and tries to generate from it an isomorphic copy that comes earlier in the (row-major) lexicographic order. If it succeeds, the generated loop is discarded; if it fails, the loop in question is the canonical one of its class and is output. This isomorphism removal method is rather slow, but requires little memory. Indicated future research includes incorporating isomorphism detection into the search.

Generating the IP loops of orders up to 11 is easy. We confirmed our results by obtaining the same numbers with MACE [McC]. Order 12 caused more difficulties, taking unreasonably long for both FINDER and MACE with their default settings. With a small change to make FINDER more aggressive about deleting old nogoods, however, we were able to solve the order 12 problem in a matter of hours.

Order 13 was more challenging. Our first partially successful run took over a week without exhausting the search space. On closer examination, we found that almost all of this time was taken up by the postprocessor eliminating isomorphic copies. We therefore somewhat strengthened the symmetry-breaking constraints, in order to reduce the number of copies to be treated, and rewrote the postprocessor to search less naïvely for dominating copies. The result is that the order 13 problem can now be completed in less than a day on a fairly ordinary desktop machine.

[3]The GAP-ECLiPSe hybrid of Gent *et al* [GHKL03] does this, and, given a suitably efficient underlying solver, may be the preferred method if the present investigations are to be pressed beyond order 13.

	Basic		Enhanced	
size	time (sec)	solutions	time (sec)	solutions
7	0.00	10	0.00	4
8	0.03	128	0.01	50
9	0.11	488	0.01	64
10	2.51	8856	0.50	1294
11	39.30	128488	1.25	5008
12	3026.31	8956032	231.38	626888

Figure 2: Basic *versus* enhanced symmetry breaking: FINDER runtimes and numbers of solutions before postprocessing. The fewer (redundant) solutions the better.

The most signficant part of the speedup was that due to the extra symmetry breaking constraints added by hand to the encoding. These reduced the domains of possible values for the cells in the second row of the table of the loop operation—the first row is fixed as it lists the elements of the form $e * x$, which of course is x in every case. The canonical representative of each isomorphism class is first in the lexicographic order in which this second row is most significant, so clearly we lose nothing by constraining the numbers early in the row to be as low as possible. Since e is the first (lowest-numbered) element, we can conveniently represent all elements as $(e+x)$ where x is an integer in the range $0 \ldots N-1$.[4] We are concerned to add constraints limiting the values of elements of the form $(e+1)*(e+x)$ where $0 < x < N$.

Where N is odd, this is simple. There are no fixed points for the inverse operation, so $(e+1)^{-1} = (e+2)$, so there are two possibilities for the value of $(e+1)*(e+1)$: it could be $(e+2)$ or it could be something else, where "something else" might as well be $(e+3)$ since all choices are symmetric. By similar reasoning, for $x > 1$, the canonical member of each isomorphism class has $(e+1)*(e+x) < (e+2x)$.

Where N is even, there is an additional complication. It is possible that some elements other than e are fixed points for inverse, and it can happen that for all of these fixed points a, the element $(e+1)*a$ is not a fixed point. In that case, the usual upper bound does not apply, but instead the values of such $(e+1)*a$ can be assigned arbitrarily. We choose to assign them in ascending order. We introduce a boolean flag (another decision variable) which will be set just in case the first 6 elements are all fixed points for inverse and $(e+1)*(e+2)$ is not a fixed point. Provided the flag is not set, a constraint similar to that for odd values of N applies. That is, using $\mathbf{f}(\varphi)$ to abbreviate $(e+1)*(e+\varphi)$:

Basic symmetry breakers:

$$e \leq x$$
$$x^{-1} < (x+2)$$
$$(x^{-1} = x \wedge y < x) \Rightarrow y^{-1} = y$$

For odd values of N:

$$x^{-1} < x+2$$
$$x^{-1} = x \Leftrightarrow x = e$$
$$\mathbf{f}(1) < (e+4)$$
$$(x > 1 \wedge 2x < N) \Rightarrow \mathbf{f}(x) < (e+2x)$$

[4]Naturally, we could let the elements *be* the integers $0 \ldots N-1$ or $1 \ldots N$, as in the Zinc encoding suggested in Figure 1, in which case the notation would be simplified. FINDER, however, is picky about types and complains if we confuse "element" with "int", so we keep the long-winded version for present purposes.

size	*quasigroups*	*loops*	*IPloops*	*groups*
1	1	1	1	1
2	1	1	1	1
3	5	1	1	1
4	35	2	2	2
5	1411	6	1	1
6	1130531	109	2	2
7	1.21×10^{10}	23746	2	1
8	2.70×10^{15}	1.06×10^{8}	8	5
9	1.52×10^{22}	9.37×10^{12}	7	2
10	2.75×10^{30}	2.09×10^{19}	47	2
11	— ? —	— ? —	49	1
12	— ? —	— ? —	2684	5
13	— ? —	— ? —	10342	1

Table 1: Numbers of algebras of given order

For even values of N:

$$\mathbf{f}(1) = e$$
$$(\neg\text{FLAG} \wedge 0 < x < N/2) \Rightarrow \mathbf{f}(x) < (e + 2x + 1)$$
$$\text{FLAG} \Rightarrow (e+5)^{-1} = (e+5)$$
$$(\text{FLAG} \wedge x > 1 \wedge (e+x)^{-1} = (e+x)) \Rightarrow (\mathbf{f}(x))^{-1} \neq \mathbf{f}(x)$$
$$(\text{FLAG} \wedge 1 < x < y \wedge (e+y)^{-1} = (e+y)) \Rightarrow \mathbf{f}(x) < \mathbf{f}(y)$$

The enhanced symmetry breaking pays handsomely, as shown in Figure 2 where the runtimes and numbers of solutions (before postprocessing) with and without the extra symmetry breakers are compared. It is worth noting that in generating 626,888 solutions to the order 12 problem, for example, FINDER backtracks only 202,549 times. That means that three quarters of the branches in the search tree end in solutions, so it is unlikely that significant improvement to the efficiency of the search is possible. Any future advance will need to involve better symmetry removal, to cut down still further the number of solutions generated.

2.4 The numbers

Table 1 shows the count of all IP loops of small orders. For comparison, the number of these which are associative (i.e. groups) is also shown, as are the numbers of quasigroups and loops.[5] Clearly, at very small sizes, associativity has no room to fail given the loop and inverse postulates. Up to order 4, indeed, the existence of an identity element alone is enough to force a quasigroup to be an abelian group. It seems from the table, however, that IP loops will resemble loops rather than groups in that, once clear of the initial noise, the numbers of such algebras will show a monotonic exponential increase with size. At the same time, it can be seen that the existence of a two-sided inverse is in some intuitive sense a "strong" property: of the 10^{30} quasigroups of order 10, one in every hundred billion is a loop, but of these loops only one in every 40 million trillion has an inverse.

An important subvariety of any variety of groupoid is that obtained by imposing a postulate of commutativity. The numbers of commutative IP loops are shown in Table 2. It was something of a surprise

[5]The numbers of quasigroups and loops are taken from the paper of McKay *et al* [MMM07] which also contains an account of the many errors making up the history of counting these objects.

size	*groups*	*non − groups*	*total*
1	1		1
2	1		1
3	1		1
4	2		2
5	1		1
6	1		1
7	1		1
8	3		3
9	2		2
10	1	5	6
11	1	1	2
12	2	12	14
13	1	7	8
14	1	179	180

Table 2: Numbers of Commutative IP loops of given order

to observe that the smallest such loop which is not a group is of order 10. It seems that commutativity tends to enforce associativity, at least at small sizes: only one of the 48 non-associative IP loops of order 11, for instance, is commutative.

3 New results

In abstract algebra, the effect of generating the structures of small sizes is often to provide a supply of data rather than a supply of theorems. This means that model searches function more like experiments in an empirical science than like proof searches in mathematics as standardly conceived. The rôle of diagrams in traditional geometry is somewhat similar: a diagram is not a proof, but it can supply a disproof, and inspection of diagrams can suggest conjectures to the mathematician with an eye for regularities. In the same way, identifying patterns in the numbers or distribution of small structures is a good way of formulating conjectures in abstract algebra.

In the present case, we have been able to use the "data" provided by FINDER to arrive at several new results concerning IP loops. These are not necessarily very deep mathematics, and their proofs, once the regularities have been observed, are not especially hard. The trick is to formulate the conjecture in the first place, and for this purpose access to the quasi-empirical data is invaluable.

3.1 Order of subloops

Steiner loops satisfy the condition $\forall x(x^2 = e)$ or equivalently $\forall x(x^{-1} = x)$. We wondered whether there was anything to say about the distribution of elements satisfying the self-inverse condition in IP loops which are not Steiner loops in general. Fortunately, in generating the algebras, as explained in §2 above, part of our technique was to set the inverse operation before generating the loop operation. Thus we were presented immediately with the numbers of IP loops of each order with each possible choice of inverse, where the difference between two inverse operations is just in the number of self-inverse elements. Hence our generation method itself resulted in a study of the distribution of self-inverse elements among IP loops of each size. To our initial surprise, there appeared to be no such elements at all (other than the

size	$k=2$	$k=3$	$k=4$	$k=5$
2	1		1	
3		1		
4	1		2	
5				1
6				
7		1		
8	1		4	
9		2		
10	1		10	
11				
12				
13		64		10

Table 3: Numbers of IP loops satisfying $x^k = e$

identity) in IP loops of odd order. We knew, of course, that Steiner loops are always of even order, but expected that IP loops of any cardinality would typically contain at least *some* fixed points for inverse.

They do not, however, as can be shown by a simple counting argument:

Theorem 1. *Let $L = \langle S, * \rangle$ be a finite IP loop. Then the cardinality of S is even iff L has an element of order 2—that is, an element a such that $a \neq e$ but $a^2 = e$.*

Proof. Left to right, the result is trivial: since the inverse operation is of period 2, the set of elements of L which are *not* fixed points for it must be of even cardinality. If the order of L is even, therefore, there must also be an even number of self-inverse elements, so e cannot be the only such element.

For the converse, suppose a is self-inverse and distinct from e. Let the operation L_a be defined on S by the equation $L_a(x) = a * x$. Then L_a is of period 2, as $L_a(L_a(x)) = a * (a * x) = a^{-1} * (a * x) = x$. Moreover, L_a has no fixed point, as if $L_a(x) = x$ then $a * x = x$ so $a = e$ contrary to the supposition of the theorem. Therefore L_a partitions S into pairs, so $|S|$ is even. □

Corollary 2. *No IP loop of odd order has a subloop of even order.*

3.2 IP loops of exponent k

Following on from this theorem, we examined the spectra of the equations $x^k = e$ for small values of k. The observations up to order 13 are summarised in Table 3.[6]

The spectrum of Steiner loops (the column $k = 2$ in the table) is well known to consist of 1 and all integers congruent to 2 or 4 (mod 6). The argument that all Steiner loops fall into that spectrum is nice enough to be worth rehearsing here. First note that Steiner loops are commutative, because for any elements x and y, $(x.xy)(yx) = y(yx) = x = (x.xy)(xy)$ so $xy = yx$. Next, if $xy = z$ then $xz = x(xy) = y$, so such loops are "fully commutative" in that for any triple x, y and z, the six equations obtained by permuting the variables in "$xy = z$" are all equivalent. Steiner loops thus correspond directly to Steiner triple systems, or sets of triples of elements from a set such that every pair of elements from the set occurs in exactly one triple. The identity of the loop is added to allow for the case where x and y are the same. Evidently, there are three pairs in every triple, so the number of triples in a Steiner triple system is one

[6] As usual, we assume association to the left, defining $x^0 = e$ and $x^{k+1} = x^k * x$.

third of the number of pairs of elements in the set. Thus, where the set has n elements, $n^2 - n$ must be divisible by 6. Expressing n as $6k+i$ for some i in 0..5, we see immediately that $i^2 - i$ must be divisible by 6, requiring i to be either 0, 1, 3 or 4. Hence $n+1$, the order of the Steiner loop, must be congruent to 1, 2, 4 or 5 (mod 6). But odd orders greater than 1 are impossible by Theorem 1, so except for the degenerate case $n = 1$ the order of the loop must be congruent to 2 or 4 (mod 6).

The core of this argument, divisibility by 6, can be generalised.

Theorem 3. *Let L be an IP loop of order $3n$. Then L contains an element x distinct from e such that $x^2 = x^{-1}$.*

Proof. Consider any elements a, b and c, all distinct from e, such that $ab = c$ in L. Then the following all hold:

$ab = c$
$cb^{-1} = a$
$a^{-1}c = b$
$b^{-1}a^{-1} = c^{-1}$
$bc^{-1} = a^{-1}$
$c^{-1}a = b^{-1}$

Moreover, the six table entries represented by these equations are all distinct unless one of them is of the form $xx = x^{-1}$. If L contains no such x, therefore, the table entries not involving e are partitioned into blocks of 6. There are $(3n-1)^2 - (3n-1)$ such entries, so $(3n-1)^2 - (3n-1)$ is a multiple of 6. That is, $9n^2 - 9n + 2$ is a multiple of 6. Let $3n = 6k+i$ where $0 \leq i < 6$. Then $9(6k+i)^2 - 9(6k+i) + 2$ is divisible by 6, so $9(36k + 12ki + i^2) - 56k - 9i + 2$ is divisible by 6, so $9(i^2 - i) + 2$ is divisible by 6. But it is not. □

Theorem 4. *Let L be an IP loop of exponent 5. Let n be the order of L. Then either $n \equiv 1$ (mod 12) or $n \equiv 5$ (mod 12).*

Proof. For any element x of L, $x^4 = x^{-1}$ and it is not hard to show that $x^3 = (x^2)^{-1}$. It is left as a satisfying exercise to show that for all x, i and j, $x^i * x^j = x^{i+j}$ (mod 5). It follows that L is composed of a number of subloops of order 5, each of course of the form $\{e, x, x^2, x^3, x^4\}$. That is, they are disjoint except for e. Therefore $n \equiv 1$ (mod 4).

Clearly, L contains no element x distinct from e such that $x^2 = x^{-1}$, so by Theorem 3 its order is not a multiple of 3 and therefore $n \not\equiv 9$ (mod 12). □

Theorem 5. *Let L be an IP loop of exponent 3. Let n be the order of L. Then either $n \equiv 1$ (mod 6) or $n \equiv 3$ (mod 6).*

Proof. For every element x of L, $x^3 = e$ or equivalently $x^2 = x^{-1}$. It follows that n is odd, and also that Theorem 3 is not directly useful. We can adapt the argument, however. If we ignore the first row and column of the table, we are left with $(n-1)^2$ entries. Each row contains two "anomalous" entries: e in the x^{-1} column and x^{-1} on the diagonal. Removing those two entries from each row, that leaves $(n-1)^2 - 2(n-1)$ to be filled with blocks of 6 as before. Thus $(n-1)^2 - 2(n-1)$ is divisible by 6. Expressing n as $6k+i$, we find that $i^2 - 4i + 3$ is divisible by 6, which is to say $i = 1$ or $i = 3$. □

3.3 The square property

A groupoid has the *square property* iff $(xy)^2 = x^2y^2$ for all x and y. It is well known that a group is commutative iff it has the square property. This is not true of IP loops, however. The smallest counterexample

is of order 10:

$*$	1	2	3	4	5	6	7	8	9	10
$e=1$	1	2	3	4	5	6	7	8	9	10
2	2	1	4	3	6	5	9	10	7	8
3	3	4	1	2	7	8	5	6	10	9
4	4	3	2	1	9	10	8	7	5	6
5	5	6	7	9	2	1	10	3	8	4
6	6	5	8	10	1	2	3	9	4	7
7	7	9	5	8	10	3	4	1	6	2
8	8	10	6	7	3	9	1	4	2	5
9	9	7	10	5	8	4	6	2	3	1
10	10	8	9	6	4	7	2	5	1	3

x	x^{-1}
1	1
2	2
3	3
4	4
5	6
6	5
7	8
8	7
9	10
10	9

This IP loop is commutative, but lacks the square property as $(3*5)^2 = 4$ but $3^2 * 5^2 = 2$. The converse is also not valid for IP loops: there are 3 non-commutative IP loops of order 12 with the square property, and 2 more of order 13.

Another property which suffices for a group to be abelian is that it is of order p^2 where p is a prime. IP loops of such orders are not in general commutative, as for example there are 5 non-commutative ones of order 9. However, both commutative IP loops of order 9 are groups, leading us to wonder whether all commutative IP loops of order p^2 are groups. The answer is negative: there is a commutative non-associative IP loop of order 11, so its direct product with any IP loop also of order 11 is a non-commutative IP loop of order 121 which is not a group.

3.4 Some rare IP loops

A loop is said to be *flexible* iff it satisfies

$$x(yx) = (xy)x$$

and *alternative* iff

$$x(xy) = (xx)y$$
$$(xy)y = (xy)y$$

Steiner loops and groups are flexible and alternative. It turns out that the smallest IP loop which is flexible and alternative but neither a group nor a Steiner loop is of order 12, and there are, up to isomorphism, only two such loops of order 12 and none of order 13.

Steiner loops and groups are also *C-loops*, meaning they satisfy

$$x(y(yz)) = ((xy)y)z$$

All C-loops are known to be IP loops and alternative. Up to order 13, there is only one non-associative, non-Steiner C-loop. Again it is of order 12.

4 Future directions

This paper has added to the store of known "small" examples of core algebraic structures. IP loops inhabit the space between very tightly constrained varieties (groups, Steiner loops) and very loose ones (quasigroups). They are closely related to an interesting generalisation of relation algebras. We have detailed the IP loops up to the orders at which the number becomes too big for a mathematician to know them all. The most obvious extensions of our work are:

1. Complete the account of the spectrum of IP loops of exponent k, for all k. We have the impression that it is not very difficult, but settling this issue properly would be satisfying.

2. Extend the investigation to particular classes of IP loops. For example, enumerate the small C-loops. Since these are comparatively rare, it will be necessary to go to larger sizes before enumeration ceases to be worthwhile. Phillips and Vojtěchovský [PV06] report very small numbers of C-loops up to order 14, for which they used MACE-4. It is possible that significantly extending the search may raise different challenges for automated reasoning.

3. Investigate the use of GAP-ECLiPSe or a similar hybrid which brings computational group theory to bear on the problem of symmetries in search spaces. Since this detects many symmetries and avoids them early, it is potentially an important tool for getting further with the enumeration of IP loops or species of them.

4. Experiment with more systematic symmetry breakers such as the "least number" heuristic of Jian Zhang and its extensions. Dealing with symmetry, rather than with search inefficiency, is the main bottleneck in the algebra generation process at present.

5. Pick out some new benchmark problems from our work, for finite domain constraint solvers, for SAT solvers or for SMT systems.[7]

References

[AH01] Gilles Audemard and Laurent Henocque. The extended least number heuristic. In *Proceedings of the International Joint Conference on Automated Reasoning (IJCAR)*, pages 427–442, 2001.

[APSS05] Anbulagan, Duc Nghia Pham, John K. Slaney, and Abdul Sattar. Old resolution meets modern SLS. In *Proceedings of the National Conference of the American Association for Artificial Intelligence (AAAI)*, pages 354–359, 2005.

[AS08] Asif Ali and John Slaney. Counting loops with the inverse property. *Quasigroups and Related Structures*, 16:13–16, 2008.

[DD01] Gilles Dequen and Olivier Dubois. The non-existence of a (3,1,2)-conjugate orthogonal Latin square of order 10. In *Principles and Practice of Constraint Programming (CP)*, pages 108–120, 2001.

[dlBMRW06] Maria Garcia de la Banda, Kim Marriott, Reza Rafeh, and Mark Wallace. The modelling language Zinc. In *Principles and Practice of Constraint Programming (CP)*, pages 700–705, 2006.

[FSB93] Masayuki Fujita, John Slaney, and Frank Bennett. Automatic generation of some results in finite algebra. In *Proceedings of the thirteenth International Joint Conference on Artificial Intelligence (IJCAI-13)*, pages 52–57, 1993.

[GHKL03] Ian Gent, Warwick Harvey, Tom Kelsey, and Steve Linton. Generic SBDD using computational group theory. In *Principles and Practice of Constraint Programming (CP)*, pages 333–347, 2003.

[GS97] Carla Gomes and Bart Selman. Problem structure in the presence of perturbation. In *Proceedings of the National Conference of the American Association for Artificial Intelligence (AAAI)*, pages 221–226, 1997.

[Mad82] Roger Maddux. Some varieties containing relation algebras. *Transaction of the American Math Society*, 272:501–526, 1982.

[MB88] Rainer Manthey and François Bry. SATCHMO: A theorem prover implemented in Prolog. In *Proceedings of the ninth Conference on Automated Deduction (CADE-12)*, pages 415–434, 1988.

[7]This research was supported by NICTA (National ICT Australia) and by the Australian National University. NICTA is funded through the Australian Government's *Backing Australia's Ability* initiative, in part through the Australian Research Council. The authors wish to acknowledge useful discussions with colleagues Tomasz Kowalski and Brendan McKay, and the constructive comments of the anonymous referees.

[McC] William McCune. Prover9 and MACE 4. http://www.cs.unm.edu/∼mccune/mace4/.

[MMM07] Brendan McKay, Alison Meynert, and Wendy Myrvold. Small latin squares, quasigroups and loops. *Journal of Combinatorial Designs*, 15:98–119, 2007.

[PV06] J. D. Phillips and Petr Vojtěchovský. C-loops: An introduction. *Publicationes Mathematicae Debrecen*, 68:115–137, 2006.

[SA07] John Slaney and Asif Ali. IP loops of small order, 2007. http://users.rsise.anu.edu.au/∼jks/IPloops/.

[Sla94] John Slaney. FINDER, finite domain enumerator: System description. In *Proceedings of the twelfth Conference on Automated Deduction (CADE-12)*, pages 798–801, 1994.

[ZBH96] Hantao Zhang, Maria Paola Bonacina, and Jieh Hsiang. PSATO: a distributed propositional prover and its application to quasigroup problems. *Journal of Symbolic Computation*, 11:1–18, 1996.

[Zha96] Jian Zhang. Constructing finite algebras with FALCON. *Journal of Automated Reasoning*, 17:1–22, 1996.

[ZS00] Hantao Zhang and Mark Stickel. Implementing the Davis-Putnam method. *Journal of Automated Reasoning*, 24:277–296, 2000.