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.

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 sub-groups 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 $A * A = B \land B * B = A$. 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
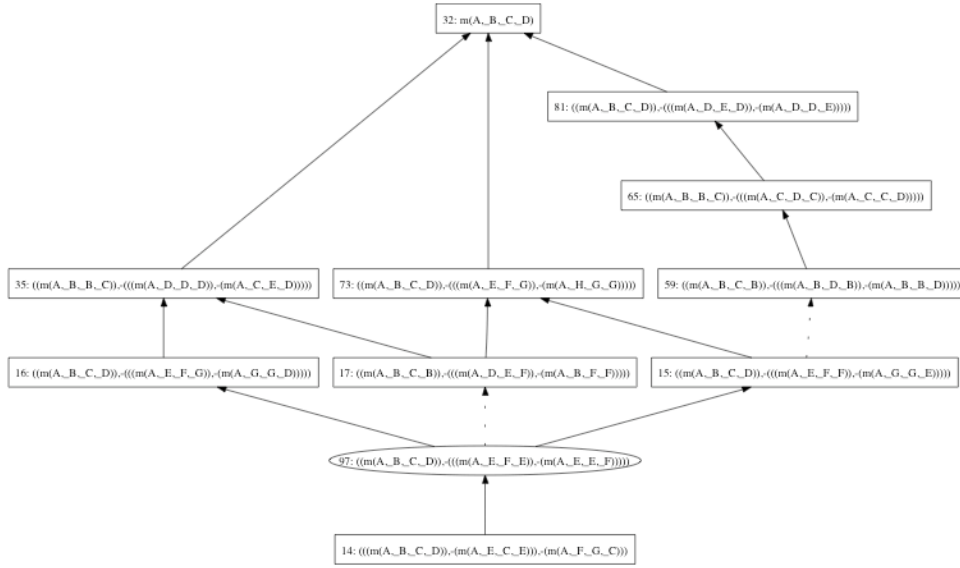
Figure 1: Graph showing the implication relationships between the specialisation of interest (circled) and other specialisations of star algebras.

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

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*.