A Global Workspace Framework for Combining Reasoning Systems

John Charnley and Simon Colton

Combined Reasoning Group, Department of Computing, Imperial College, London.
jwc04@doc.ic.ac.uk sgc@doc.ic.ac.uk
http://www.doc.ic.ac.uk/crg

1 Introduction

Stand-alone Artificial Intelligence systems for performing specific types of reasoning – such as automated theorem proving and symbolic manipulation in computer algebra systems – are numerous, highly capable and constantly improving. Moreover, systems which combine various forms of reasoning have repeatedly been shown to be more effective than stand-alone systems. For example, the ICARUS system for reformulating constraint satisfaction problems [1] and the HOMER system for conjecture making in number theory [2]. However, in general, such combinations have been ad-hoc in nature and designed with a specific task in mind. With little general design consideration or a suitable framework for combining reasoning, in general every new combination has to be built from scratch and the resulting system is often inflexible and difficult to manage. We believe it is imperative that generic frameworks are developed if the field of combining reasoning systems is to progress. Such generic frameworks would provide standardised rule sets and toolkits to simplify the development of combined systems.

We describe here a generic framework based on the cognitive science theory of the Global Workspace Architecture [3]. In our framework, the individual reasoning techniques are each encapsulated within specialist processes attached to a blackboard-style global workspace, which is visible to all processes. We achieve relative simplicity in the framework by requiring fairly severe restrictions upon the behaviour of the attached processes. In particular, there is no inter-process communication other than what is broadcast on the global workspace. These restrictions help ensure that the resulting system is simple to understand. Furthermore, the encapsulation of reasoning techniques within discrete individual processes adds clarity and flexibility. We explain our framework, and how it is used, in §2. To demonstrate the capability of the framework, we have implemented combined systems incorporating Prover9 [4], Maple [5] and SICStus Prolog. In §3, we describe applications to mathematical theorem discovery and conjecture making which produce results comparable to the ICARUS and HOMER systems, respectively. This demonstrates that while the framework is easy to use, it is as powerful as the ad-hoc systems.

2 A Framework for Combining Reasoning Systems

The architecture defined by our framework is inspired by the Global Workspace Architecture [3]. 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.

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 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 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 by with the aid of libraries provided in the toolkit.

3 Applications and Results

Our first configuration demonstrates how the framework can combine machine learning, example construction and theorem proving processes to perform automated theory formation, similar to that performed by the HR system [6]. In overview, the configuration is required to invent new concepts (built from a set of user-supplied background concepts), make empirical conjectures which relate the concepts and then prove that some of the conjectures follow from a set of user-supplied axioms. We specified four types of broadcast artefacts, as follows:

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. 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; 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.

4. Explanation, in the form **exp**([D_1,D_2]|*K*|*P*), where D_1, D_2 and *K* represent a conjecture, as above, and *P* is a proof of that conjecture.

Our initial configuration uses the following processes:

1. DefinitionFormer processes propose new *Definitions*. They each encapsulate a different concept formation method, akin to production rules in HR. They react to *Concept* broadcasts, **conc**($D|E$). 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 *Concept* broadcast, **conc**(*D'*|*E'*|*C'*), by attempting to combine *D* and *D'*.

2. DefinitionReviewer, which reacts to *Definition* broadcasts, **def**(*D*), removes redundancy by checking whether *D* has been seen before. If not, it proposes for broadcast **conc**($D|\emptyset$), i.e. a concept with that definition 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. ConjectureMaker, compares the example sets of two *Concept* broadcasts. It reacts to the first *Concept* broadcast, **conc**($D_1|E_1$), (where $E_1 \neq \emptyset$), by spawning a clone process, *P*, which itself reacts to future *Concept* broadcasts **conc**($D_2|E_2$). In particular, if *P* finds that $E_1 = E_2$ it proposes **conj**([D_1, D_2],*eq*). Alternatively, if $E_1 \subset E_2$, it proposes **conj**([D_1, D_2],*im*) (or **conj**([D_2, D_1],*im*) if $E_2 \subset E_1$).

5. 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*)|*P*), whenever a proof, *P*, is found.

In addition for this configuration, we specified a process which proposes the background concepts at the start of the session. Moreover, we specified a simple rating scheme which assigns a rating of 1 to *Definitions*, 2 to *Concepts*, 3 to *Conjectures* and 4 to *Explanations*. We enhanced this configuration by preventing the dual development of empirically equivalent concepts (i.e. if **conj**([D_1, D_2],*eq*) is broadcast, then D_2 is no longer considered) which reduces duplication of effort. We implemented the configuration using GC and used it to find implied constraints about QG-quasigroups similar to those found by HR embedded in the ICARUS [1] combined system. Working with QG3, QG4 and QG5 quasigroups, the configuration generated the same theorems as ICARUS. For example, it found the same three theorems, $\forall\, a\, b\, (a*a = b \leftrightarrow b*b = a)$, $\forall\, a\, b\, ((a*b = b*a) \rightarrow a = b)$ and $\forall\, a\, b\, ((a*a = b*b) \rightarrow a = b)$; which were all used by ICARUS in reformulating QG3 constraint programs.

To demonstrate the flexibility of the framework, we extended this initial configuration to applications in number theory similar to those performed by the HOMER system [2]. We introduced new processes encapsulating Maple to provide background solutions to number theory functions and used several prover processes, each with different axiom sets, to perform conjecture filtering. We used the background functions $\sigma(n)$ (the sum of divisors of a number), $\tau(n)$ (the number of divisors) and $isprime(n)$ (a boolean predicate indicating whether a number is prime), together with the notion of equality. Our configuration achieved results very similar to those produced by HOMER, in terms of the concepts and conjectures discovered. Like HOMER, our system filtered out approximately 90% of all the conjectures it created, by showing them to be simple consequences of the definitions and hence uninteresting. Importantly, our system re-discovered the most interesting results from [2], including e.g., $isprime(\sigma(a)) \rightarrow isprime(\tau(a))$. Moreover, our system highlighted potential weaknesses in HR, by showing that concepts had been repeated due to variable ordering.

4 Conclusions and Future Work

Compared to building an ad-hoc combined system from scratch – which is currently the norm – it is relatively straightforward to construct systems using our GWA-based framework. Despite the framework's restrictions, it can be configured to achieve results equivalent to previous bespoke ad-hoc systems. We will continue to develop the GC toolkit, by adding additional reasoning processes for tasks such as model generation and SAT-solving. In addition, we aim to develop the user interface by providing graphical tools for selecting and tuning processes and for specifying new processes without having to write code explicitly. We will continue to improve our configurations in efforts to improve upon previous system results, for example, by introducing more sophisticated rating schemes. We intend to create new configurations for existing combined reasoning tasks, such as correcting false conjectures [7] and algebra classification [8], and to tackle new problems with the framework. Furthermore, the core-processing of the toolkit will be enhanced to take advantage of the distributed parallel nature of the underlying architecture, which should enhance performance. We hope to have demonstrated the potential of our GW-based generic framework for combining reasoning systems and we hope in future to add to the weight of evidence that combining reasoning systems is imperative for the advancement of Artificial Intelligence.

References

1. J. Charnley, S. Colton, and I. Miguel. Automatic generation of implied constraints. In *Proceedings of ECAI*, 2006.
2. S. Colton. Automated conjecture making in number theory using HR, Otter and Maple. *Journal of Symbolic Computation*, 39(5):593-615, 2004.
3. B. Baars. *A cognitive theory of consciousness*. Cambridge University Press, 1988.
4. W. McCune. Prover9. http://www.cs.unm.edu/ mccune/prover9/.
5. Waterloo Maple. *Maple Manual at http://www.maplesoft.on.ca*.
6. S. Colton. *Automated Theory Formation in Pure Mathematics*. Springer-Verlag, 2002.
7. S. Colton and A. Pease. The TM system for repairing non-theorems. In *Proceedings of the IJCAR'04 Disproving workshop*, 2004.
8. S. Colton, A. Meier, V. Sorge, and R. McCasland. Automatic generation of classification theorems for finite algebras. In *Proceedings of IJCAR*, 2004.