Concurrent Inference Graphs

Daniel R. Schlegel
Department of Computer Science and Engineering
Univeristy at Buffalo, Buffalo NY, 14260
drschleg@buffalo.edu

Introduction

Since at least the early 1980s there has been an effort to utilize multiple processors for logical reasoning. Prior to the rise of the multi-core desktop computer, this often meant parallel algorithms on specialized hardware. Parallel logic programming systems designed during that same period were less attached to a particular parallel architecture, but parallelizing Prolog is a very complex problem (Shapiro 1989). There are more successful parallel implementations of Datalog, but it is a less expressive subset of Prolog. Recent work in statistical inference has returned to large scale parallelism using GPUs, but while GPUs are good at statistical calculations, they do not do logical inference well (Yan, Xu, and Qi 2009).

We present *inference graphs*, a graph-based natural deduction inference system which lives within a KR system and is capable of taking advantage of multiple cores and/or processors using concurrrent processing techniques rather than parallelism. We chose to use natural deduction inference, despite the existence of very well performing refutation based theorem provers, because our system is designed to be able to perform forward, bi-directional (Shapiro, Martins, and McKay 1982), and focused reasoning in addition to the backward inference used in resolution. Natural deduction also allows formulas generated during inference to be retained in the KB for later re-use, whereas refutation techniques always reason about the negation of the formula to be derived, making intermediate derivations invalid. In addition, our system is designed to allow formulas to be disbelieved, and to propogate that disbelief to dependent formulas. We believe inference graphs are the only concurrent inference system with all these capabilities. We have seen the need for these capabilities in the domains of cognitive robotics, and soft information fusion.

The capabilities of inference graphs come from the combination of three heretofore thought to be unrelated graph-based inference systems: Truth Maintenance Systems (TMS graphs) (Doyle 1979), RETE networks (RETE nets) (Forgy 1982), and Active Connection Graphs (ACGs) (McKay and Shapiro 1981). We have developed a common visualization for the three graphs which makes their structural commonalities more obvious, and suggests that their unique positive aspects may be combined. For example, from AGCs we take its ability to perform forward, backward, and bi-directional inference, RETE nets add the ability not only add assertions, but to remove them as well, and TMSes add the ability to perform truth maintenance operations within the same graph structure. Using the common visualization we have found that a mapping can be made to propositional graphs – a representation for knowledge – which we therefore use as the starting point upon which we add the features from the three inference systems. We believe inference graphs are the only concurrent inference system with all these capabilities.

My plan for developing and implementing inference graphs has three stages. All work has been (and will be) completed by me, with the advice of my advisor. In stage one we examine RETE nets, TMS graphs, and ACGs for similarities and differences in structure and functionality. In stage two, we develop the model for concurrency and implement inference graphs using propositional logic. In stage three, we will formulate more formally the relationships between RETE nets, TMS graphs, and ACGs, and extend the inference graph implementation to the Logic of Arbitrary and Indefinite Individuals (Shapiro 2004). Stage 1 was completed in 2011, stage 2 has just been completed (Schlegel and Shapiro 2013), and we will begin stage 3 by July.

Propositional Graphs

Propositional graphs in the tradition of the SNePS family (Shapiro and Rapaport 1992) are graphs in which every well-formed expression in the knowledge base, including individual constants, functional terms, atomic formulas, or non-atomic formulas ("rules"), is represented by a node in the graph. A rule is represented in the graph as a node for the rule itself (a *rule node*), nodes for the argument formulas, and arcs emanating from the rule node, terminating at the argument nodes. Arcs are labeled with an indication of the role the argument plays in the rule, itself. Every node is labeled with an identifier. No two nodes represent syntactically identical expressions; rather, a single node is used if there are multiple occurrences of one subexpression.

Inference Graphs

Inference graphs extend propositional graphs allowing messages about assertional status and inference control to flow through the graph. Certain arcs and certain reverse arcs already within the propositional graph are augmented with ACG-like *channels* through which information can flow. Channels flow from antecedents to rule nodes, from rule nodes to consequents, and from consequents to unifiable antecedents, where the latter act like the RETE alpha network – filtering incompatible messages. To control the flow of messages, channels have valves which can be either open or closed. When a valve is open, messages flow freely forward, but when it is closed they wait behind it. Rule nodes in inference graphs combine messages and carry out introduction and elimination rules to determine if itself or its consequents are true or negated – meaning the inference graph is both a representation of knowledge and the method for performing inference upon it.

Messages come in several types, each serving one of two purposes: relaying newly derived information, and controlling the inference process. A message can be sent to relay the information that its origin has been asserted or negated, that its destination should now be asserted or negated, or that its origin has either just become unasserted or is no longer sufficiently supported. Messages used to relay newly derived information carry a support set with them for truth maintenance similar to that of the ATMS (a type of TMS) (de Kleer 1986). Other messages travel backward along channels opening and closing valves to control the inference process. This second kind of message allows for backward and bi-directional inference, along with cancellation of unnecessary inference processes.

Inference operations take place in the rule nodes. When a message arrives at a rule node the message is translated into a RUI. RUIs contain information about how many (and specifically which) antecedents of a rule are known to be true or false, along with a set of support. All RUIs created at a node are cached. These RUIs are combined to determine if the rule node's inference rules can be applied – much like the RETE beta network. The RUI cache can also be used to prevent re-derivations and cut cycles. Our approach does make some rules difficult to implement, such as negation introduction, and proof by cases. For us, the advantages in capability outweigh the difficulties of implementation.

Concurrent Reasoning

To perform inference concurrently, the inference graph is divided into *inference segments* (henceforth, *segments*). A segment represents the inference operation – from receipt of a message to sending new ones – which occurs in a node. Valves delimit segments. When a message passes through an open valve a new *task* is created – the application of the segment's inference function to the message. When tasks are created they enter a global prioritized queue, where the priority of the task is the priority of the message. When the task is executed, inference is performed as described above, and any newly generated messages are sent toward its outgoing valves for the process to repeat.

If we arrange an inference graph so that a user's request (in backward inference) is on the right, and channels flow (where possible, cycles can exist) from left to right, we can see the goal of inference as trying to get messages from the left side of the graph to the right side of the graph. Every inference operation begins processing messages some number of levels to the left of the query node. Since there are a limited number of tasks which can be running at once due to hardware limitations, we must prioritize their execution, and remove tasks which are no longer necessary. Therefore,

1. tasks for relaying newly derived information using segments to the right execute before those to the left, and
2. once a node is known to be true or false, all tasks attempting to derive it (left of it in the graph) are canceled, as long as their results are not needed elsewhere.

Together, these two heuristics ensure that messages reach the query as quickly as possible, and time is not wasted deriving unnecessary formulas. The priorities of the messages (and hence, tasks) allow us to reach these goals.

To evaluate our progess, we have run tests to determine the speedup given by the concurrency model, and by the heuristics described in the propositional case. Tests show that these heuristics in the propositional case result in a 10-40x speed increase over various non-prioritized queues, and when ignoring the intrinsically sequential portions of inference, nearly linear speedup with the number of processors.

References

de Kleer, J. 1986. Problem solving with the ATMS. *Artif Intell* 28(2):197–224.

Doyle, J. 1979. A truth maintenance system. *Artif Intell* 19:231–272.

Forgy, C. 1982. Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artif Intell* 19:17–37.

McKay, D. P., and Shapiro, S. C. 1981. Using active connection graphs for reasoning with recursive rules. In *Proc. IJCAI-7*, 368–374. Los Altos, CA: Morgan Kaufmann.

Schlegel, D. R., and Shapiro, S. C. 2013. Concurrent reasoning with inference graphs (student abstract). In *Proc. AAAI-13 (this volume)*.

Shapiro, S. C., and Rapaport, W. J. 1992. The SNePS family. *Comp Math Appl* 23(2–5):243–275. Reprinted in Lehmann, F. 1992. Semantic Networks in Artificial Intelligence. Pergamon Press.

Shapiro, S. C.; Martins, J. P.; and McKay, D. P. 1982. Bi-directional inference. In *Proc. CogSci-4*, 90–93.

Shapiro, E. 1989. The family of concurrent logic programming languages. *ACM Comput. Surv.* 21(3):413–510.

Shapiro, S. C. 2004. A logic of arbitrary and indefinite objects. In Dubois, D.; Welty, C.; and Williams, M., eds., *Proc. KR2004*, 565–575. Menlo Park, CA: AAAI Press.

Yan, F.; Xu, N.; and Qi, Y. 2009. Parallel inference for latent dirichlet allocation on graphics processing units. In *Proc. NIPS*, 2134–2142.

This work has been supported by a Multidisciplinary University Research Initiative (MURI) grant (Number W911NF-09-1-0392) for Unified Research on Network-based Hard/Soft Information Fusion, issued by the US Army Research Office (ARO) under the program management of Dr. John Lavery.