The ‘Ah Ha!’ Moment : When Possible, Answering the Currently Unanswerable using Focused Reasoning

Daniel R. Schlegel and **Stuart C. Shapiro**
Department of Computer Science and Engineering, and Center for Cognitive Science
University at Buffalo, Buffalo, NY 14260
<drschleg,shapiro>@buffalo.edu

Abstract

Focused reasoning is a method for continuing a specific inference task as soon as rules or facts which may assist in the reasoning are added to the knowledge base without repeating completed inference, re-posing queries, or performing unnecessary inference. Determining if focused reasoning should commence uses very few computational resources above those used normally to add a term to a knowledge base. We have developed three focused reasoning procedures – backward-in-forward, forward, and forward-in-backward – built upon Inference Graphs, a graph-based concurrent reasoning mechanism.

1 Introduction

When an inference process has stopped before its natural conclusion because of a lack of information, any new assertion which is relevant should immediately be used to continue that inference, but not derive anything unrelated to the inference task. Focused reasoning (FR) has been developed to do just this. Methods for performing focused reasoning have been developed for using forward, and bi-directional (Shapiro, Martins, & McKay, 1982) inference. To accomplish this, Inference Graphs (IGs) (Schlegel & Shapiro, 2013b, 2014), a concurrent graph-based method for natural deduction reasoning, are extended to recognize when new connections in the graph are relevant to incomplete inference procedures.

Inference Graphs are propositional graphs with message-passing channels added, allowing related terms to communicate about the results of inference, or to control inference. IGs implement $\mathcal{L}_A$, a Logic of Arbitrary and Indefinite Objects (Shapiro, 2004), which uses structured quantified terms.

One type of FR using IGs was briefly proposed in (Schlegel & Shapiro, 2013b), but unimplemented. Here we discuss the issue in detail, including two other types of FR, in an implemented system.

While our primary interest is in building a mind, any large knowledge base (KB) which allows for interleaved assertions and queries may benefit from FR. For example,[1] web services which perform frequent queries for users, such as eBay’s saved searches (eBay, Inc., 2013) which email new matching items to users daily, could use FR to send updates as soon as they are available, avoiding batch processing. Diagnostic tasks may also benefit: one may start with a problem they wish to solve, and then do diagnostic tests. The results could be added to the KB, where they are automatically applied to only the focused problem, without deriving other, unrelated, conclusions. Another application might be within event-driven programs where events interact in complex ways, requiring complex state management, such as spreadsheets.

In Section 2 we will discuss the issue of FR itself. In Section 3[2] we will discuss IGs in only the required amount of detail to further our discussion, and in Section 4 we will return to the issue at hand to discuss the different types of FR with examples, along with an algorithm to unify the three types of FR discussed. Finally, we will conclude with Section 5.

2 Focused Reasoning

Humans often consider problems they may not yet have answers for, and push those problems to the “back of their mind.” In this state, a human is still looking for a solution to a problem, but is doing so somewhat passively – allowing the environment and new information to influence the problem solving process, and hopefully eventually reaching some conclusion. That is, the examination of the problem persists beyond the time when it is actively being worked on.[3] We wish to add a similar capability to a reasoning system.

There are three types of FR possible within a reasoning system: forward FR, where all possible derivations are performed only from some specific new piece of knowledge, and continued upon the addition of relevant rules to the KB; forward-in-backward FR, in which backward inference occurs to try to answer a query, and as new facts or rules relevant to the query are added to the KB, they are used in attempting to answer the query; and backward-in-forward FR, which combines the previous two FR mechanisms.

Forward FR can be thought of as a kind of full-forward reasoning carried out only for a single asserted term. Full-forward reasoning is used most notably in production systems, and especially RETE networks (Forgy, 1979, 1982). In a RETE net, all new information is filtered through a graph generated from a set of production rules. Nodes in the graph perform comparisons against accepted values, and combine pieces of compatible information together. When a piece of information reaches some leaf node in the graph, the rule that leaf node represents is said to have matched. Full-forward inference produces the logical closure of a KB, but this is horribly wasteful in both time and space, so we favor doing

[1]We have not done significant study on these applications yet.

[2]Portions of the material in Sections 2 and 3 are adapted from (Schlegel & Shapiro, 2013a, 2014).

[3]Understanding this type of problem solving in humans is still active research, what has been discussed is only an intuitive explanation. It is distinct from the “Eureka effect” (Auble, Franks, & Soraci, 1979) which deals with insight, and limitations of memory recall in humans.

this only when it's explicitly asked for. RETE nets are also limited in that no new rules can be added once the system is started, which is not the case with IGs.

For reasoning systems capable of backward and bi-directional inference (BDI), the issue of FR is seldom tackled. SNePS 2's Active Connection Graph (ACG) (McKay & Shapiro, 1981) has a concept of "activating" a path of nodes when backward inference does not result in an answer. Later assertions meant to use this path must be asserted with forward inference (that is, the new term is asserted, and forward inference is carried out for that term only), and that forward inference process will use activated paths exclusively whenever they are available (Shapiro et al., 1982). The ACG is unable to later deactivate the path of nodes, so the conflation of the specialized forward inference using activated paths with the usual forward inference which ignores activated paths results in the need to occasionally throw the graph away as it could interfere with future inference tasks. In addition, activated paths are not extended backward when rules relevant to the reasoning task are added to the KB. John Pollock's OSCAR system uses a different type of BDI (Pollock, 1990),[4] and does not support FR.

The system we present here is capable of performing forward, forward-in-backward, and backward-in-forward FR. It also allows the addition of new rules once the system is running, extending the focused region of the graph. In addition, it does not limit the functionality of the graph in other ways – other inference tasks can be performed as usual. In effect, our system has none of the limitations of RETE nets, or ACGs, while being a more powerful FR tool.

3 Background: $\mathcal{L}_A$ and Inference Graphs

$\mathcal{L}_A$ is a first order logic designed for use as the logic of a KR system for natural language understanding, and for commonsense reasoning (Shapiro, 2004). The logic is sound and complete, using natural deduction and subsumption inference. Throughout this paper we will assume the deductive rules implemented are the standard rules of inference for FOL, though the actual implementation uses set-oriented connectives (Shapiro, 2010), which subsume the standard rules.

The logic makes use of arbitrary and indefinite terms (collectively, quantified terms) instead of the universally and existentially quantified formulas familiar in first order predicate logic (FOPL).[5] That is, instead of reasoning about *all* members of a class, $\mathcal{L}_A$ reasons about a *single* arbitrary member of a class. There are no two arbitrary terms representing the same arbitrary entity. For indefinite members, it need not be known *which* member is being reasoned about, the indefinite member itself can be reasoned about. Indefinite individuals are essentially Skolem functions, replacing FOPL's existential quantifier. We will only deal with arbitrary terms for the remainder of this paper.

Quantified terms are structured – they consist of a quantifier indicating whether they are arbitrary or indefinite, a syntactic variable, and a set of *restrictions*. The range of a quantified term is dictated by its set of restrictions, taken conjunctively. A quantified term q_i has a set of restrictions $R(q_i) = \{r_{i_1}, \ldots, r_{i_k}\}$, each of which make use of q_i's variable, v_i. The syntax used throughout this paper for $\mathcal{L}_A$ will be a version of CLIF (ISO/IEC, 2007). We will write an arbitrary term as `(every` v_i $R(q_i)$`)`. Quantified terms take wide scope, meaning within a logical expression v_i may be used instead of re-defining a quantified term. For example, the arbitrary Person, written `(every x (Isa x Person))`, can be referred to later within the same rule by using `x`, as in the following $\mathcal{L}_A$ expression, which is meant to mean that "if a person is arrested, then that person is detained."

```
(if (Arrested (every x (Isa x Person)))
    (Detained x))
```

Inference Graphs extend propositional graphs. In the tradition of the SNePS family (Shapiro & Rapaport, 1992), propositional graphs are graphs in which every well-formed expression in the knowledge base, including individual constants, functional terms, atomic formulas, or non-atomic formulas (which we will refer to as "rules"), is represented by a node in the graph. A rule is represented in the graph as a node for the rule itself (henceforth, 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 (*e.g.,* antecedent or consequent) the argument plays in the rule, itself. Every node is labeled with an identifier. Nodes representing individual constants, proposition symbols, function symbols, or relation symbols are labeled with the symbol itself. Nodes representing functional terms or non-atomic formulas are labeled `wft`*i*, for some integer, *i*. Every SNePS expression is a term, denoting a mental entity, hence `wft` instead of `wff`. An exclamation mark, "`!`", is appended to the label if it represents a proposition that is asserted in the KB. Nodes representing arbitrary terms are labeled `arb`*j*, for some integer, *j*. No two nodes represent syntactically identical expressions; rather, if there are multiple occurrences of one subexpression in one or more other expressions, the same node is used in all cases. Propositional graphs are built incrementally as terms are added to the knowledge base, which can happen at any time.[6]

To propositional graphs, IGs add *channels* within rules, within generic terms, and between terms which match each other (that is, unify, and satisfy certain subsumption and type relationships). Channels carry messages, and represent paths inference might take through the graph.

A channel contains a valve, a filter, and a switch. Valves

[4]Pollock's BDI is distinct from that of (Shapiro et al., 1982). The premise of Pollock's BDI is that there are inference rules useful in forward reasoning, and others for backward reasoning, and as such, to reach some meeting point between premises and goals, you must reason backward from the goals, and forward from the premises. The BDI of Shapiro, et al. adopted here, assumes some procedure which has linked related terms in a graph so that arbitrary forward reasoning from premises is never necessary in backward inference.

[5]See (Shapiro, 2004) for a translation between FOPL and $\mathcal{L}_A$.

[6]See (Schlegel & Shapiro, 2012) for the logic/graph mapping.

control inference by allowing or preventing message flow through the channels. When a valve is open, messages pass to the filter and switch unimpeded, otherwise they wait in a queue until the channel is opened. When a valve in a channel is open or closed, we call the channel open or closed accordingly. Messages carry substitutions. Filters discard messages with substitutions incompatible with the destination, and switches adjust the variable context of message substitutions which pass through them to ensure the substitutions are able to be understood by the destination of the channel.

Messages of several types are transmitted through the IG's channels, serving two purposes: to relay newly derived information, and to control the inference process. We'll concern ourselves only with four types here: `i-infer` and `u-infer` messages, which carry newly derived information – `i-infer` messages relay substitutions found for the originator of the message which the destination may be interested in, and `u-infer` messages relay substitutions found for the destination of the message; `backward-infer` messages, which pass backward through the graph opening valves; and `cancel-infer` messages, which pass backward through the graph closing valves.

Inference operations take place primarily in the rule nodes. When a message arrives at a rule node it is combined with messages which have previously arrived if the messages are compatible (*i.e.,* have arrived from different antecedents, and have compatible substitutions). By determining if a combined message has been produced from the proper number of antecedents, it can be determined if a rule node's inference rules can be applied.

An IG stores all results which it has derived, allowing later queries to be answered without repeating already completed derivations. What has been discussed is a simplified version of IGs only complex enough to present FR, we leave many of the details of IGs to other papers.

4 Types of Focused Reasoning

In the following three subsections, we will introduce three types of FR: forward-in-backward, forward, and backward-in-forward. For each type of FR, we will first describe what the common use case is for that type, and provide a small example which requires that type of FR. We will then describe a more concrete example inspired by the counter-insurgence (COIN) domain, complete with figures showing the associated IGs. These will each be described semi-formally for ease of understanding. In a later subsection, we will provide an algorithm which combines all the types of FR and makes their operation more formal.

4.1 Forward-In-Backward Focused Reasoning

The most common use of FR is when wondering about something which cannot yet be answered by the system using backward inference. For example, consider a knowledge base containing only `(if P R)` and `(if P Q)`. Then, the user asks about `R`. Backward inference is set up (*i.e.,* valves in the appropriate channels through `(if P R)` are opened) but no answer is forthcoming. Later, `P` is asserted (without forward inference). Since the appropriate valves are already open, `R` is derived immediately, without needing to pose the question again. Note that `Q` is not derived, since valves involving `(if P Q)` were not opened during backward inference.[7]

In a somewhat more complex example from the COIN domain, consider the following initial knowledge base:

```
;; Azam is a person
(Isa Azam Person)

;; If a person is arrested, they are detained.
(if (Arrested (every x (Isa x Person)))
    (Detained x))

;; A person is either detained or free.
(xor (Detained (every x (Isa x Person)))
     (Free x))
```

It is then asked by a user, "who are the detained persons?": `(Detained (?x (Isa ?x Person)))`. The top graph in Figure 1 shows the IG for this KB. The query is shown as `wft20`, using a `qvar` – a type of quantified term which acts much like an arbitrary, but is only for answering "wh-" style questions. The system recursively opens channels backward stemming from the query, but is unable to produce an answer, since none exists in the graph. The channels drawn with heavier weight are those which have been opened during backward inference. Notice that two routes are tried – A person might be detained if they are not free, or a person might be detained if they have been arrested. At some later time, it is added to the KB that Azam was arrested: `(Arrested Azam)`. The system knows that backward inference was in progress,[8] so upon the addition of the channel from `wft6!` to `wft1`, backward inference is continued back to `wft6!`, opening that channel. Since `wft6!` is asserted, this information immediately flows forward through the graph along open channels, and Azam is produced as an answer to the previously added query automatically. This is shown in the bottom half of Figure 1, where the heavier weight channels indicate the flow of messages from `wft6!` forward through the open channels. It's important to note that while this KB entails that Azam is not free, it does not derive this fact in this case since the channels from `wft2` to `wft5!` and `wft5!` to `wft4` were not opened by the backward inference process. So, derivations which are irrelevant to reaching the desired conclusion are not performed – we say inference is focused toward the query.

[7]Prolog with tabling can suspend paths of inference which cannot complete, and resume them if useful facts are found via the exploration of other paths within the same inference procedure (Swift & Warren, 2012). Focused reasoning is different, allowing automatic continuation of inference at the time when related terms are added to the KB, persisting beyond the run time of a single inference procedure.

[8]How does it know? In many cases, it is possible to tell by which channels are open. But, there are cases where this doesn't work (such as initiating backward inference on a term with no incoming channels) so it makes more sense to maintain a set of in-progress processes or a flag as detailed later.

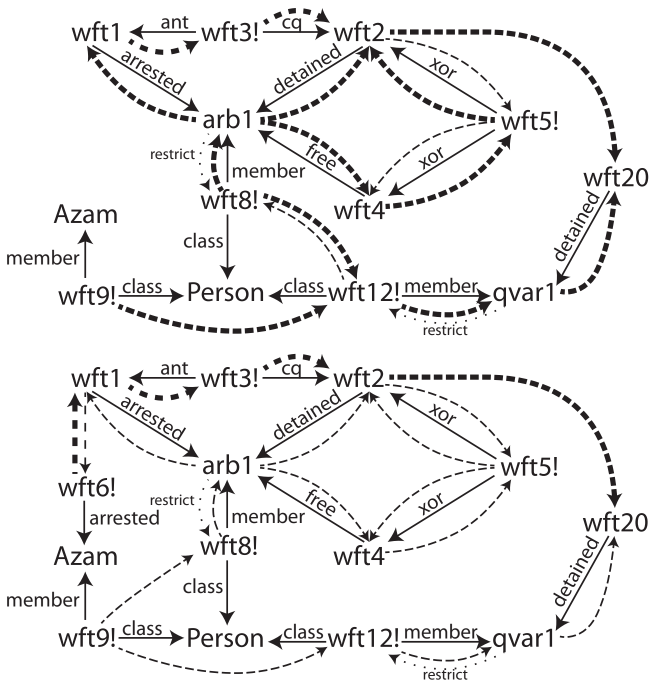

Figure 1: The IGs for the forward-in-backward FR example. Dashed lines represent channels. Restrictions have dotted arcs labeled "restrict". Channels drawn with a heavier weight are involved in the illustrated inference process. In the top graph, it has been asked "who are the detained persons?" (`wft20`), and backward inference has commenced. In the bottom graph, the fact that Azam has been arrested, `wft6!`, is added to the KB, and flows through the already open channels, deriving the result that Azam is detained.

4.2 Forward Focused Reasoning

A second type of FR can occur when a user wishes to perform forward inference on a term, but the knowledge base is not yet fully constructed. For example, consider an empty knowledge base where the user asserts `Q` with forward inference. Nothing new is derived, as the KB is otherwise empty. Later, `(if Q R)` is asserted. Since `Q` was asserted with forward inference, as soon as additional outgoing channels are connected to it, its assertional status flows forward through the graph, and `R` is derived. This derivation happens, again, without needing to reassert `Q`. One can think about this as a limited form of full-forward inference. Instead of adopting full-forward inference for all terms which are added to the KB, only `Q` has this property. Automatic inference in the graph is focused on what can be derived from `Q`, while unrelated terms (*e.g.,* `(if S T)` and `S`) may be added but without resulting in any automatic inference.

It's worth recognizing that all our inference mechanisms only follow existing channels, and do not create new terms which are possibly irrelevant to the knowledge base. For example, from the original KB with only `Q` asserted, there are an infinite number of true disjunctions which could be introduced, but are unhelpful for ongoing inference processes.

Consider our COIN example again with a slightly different set of terms initially asserted:

```
;; A person is either detained or free.
(xor (Detained (every x (Isa x Person)))
     (Free x))
```

It is then asserted with forward inference that Azam is a person, and has been arrested: `(Isa Azam Person)`, and `(Arrested Azam)`. The top of Figure 2 shows the resulting knowledge base. It is determined that `Azam` satisfies the restriction of `arb1` (that is, Azam is a Person, through the channel from `wft9!` to `wft8!`), but no new knowledge is derived. Later, as in the bottom of Figure 2, the rule that if a person is arrested then they have been detained is added:

```
(if (Arrested (every x (Isa x Person)))
    (Detained x))
```

Since `wft6!` was added with forward inference, when the new outgoing channel to `wft1` is added, forward inference continues. This allows the derivations that Azam is detained: `(Detained Azam)`, and Azam is not free: `(not (Free Azam))`.

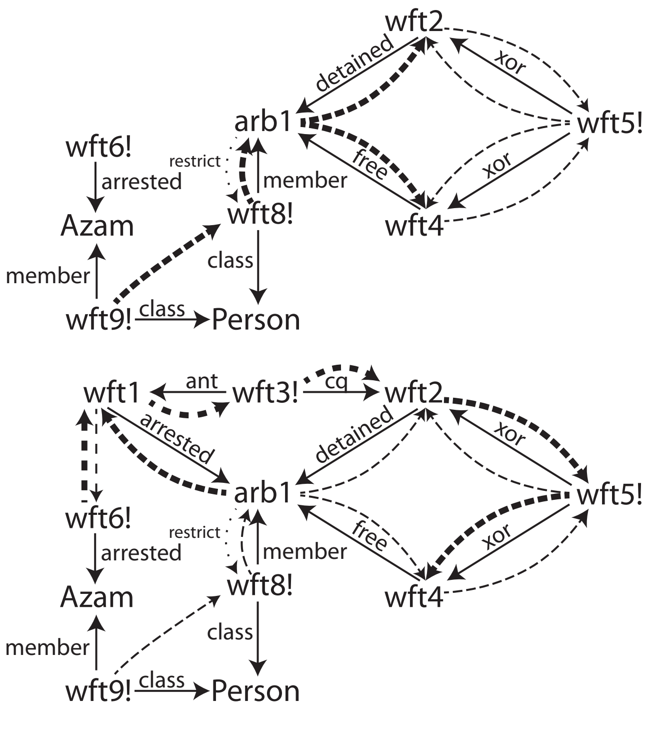

Figure 2: IGs for the forward FR example. In the top graph, Azam is a person, and has been arrested, `wft6!` and `wft9!`, are asserted with forward inference. In the bottom graph, the rule that if a person is arrested, they have been detained is added, allowing inference to continue, and for it to be derived that Azam is detained and not free.

4.3 Backward-In-Forward Focused Reasoning

A combination of the above two FR techniques is also possible. Consider a user who again asserts `Q` with forward inference into an empty knowledge base. Later, `(if P (if Q R))` is asserted. As with forward FR, `Q` recognizes that it has new outgoing channels, and sends its assertional status to `(if Q R)`. But, `(if Q R)` is not yet asserted, so backward inference attempts to derive `(if Q R)`, but fails. Later again, `P` is asserted (without forward inference). Inference then occurs as in forward-in-backward FR, `(if Q R)` is derived, then `R` is.

From the COIN domain again, consider the KB:

```
;; Ahmad is a person.
(Isa Ahmad Person)

;; If a person is a person of interest (POI),
;;  they are either under surveillance, or
;;   being sought out.
(if (POI (every x (Isa x Person)))
    (xor (UnderSurveillance x)
         (BeingSoughtOut x)))

;; If a person is a POI, they are of
;; interest to INSCOM
(if (POI (every x (Isa x Person)))
    (ofInterestTo x INSCOM))
```

Now, it is asserted with forward inference that Ahmad is not under surveillance: `(not (UnderSurveillance Ahmad))`, shown in the top of Figure 3. If the `xor` (`wft5`) were asserted, it could be derived that `(BeingSoughtOut Ahmad)` through forward inference, but it is not. So, the system initiates backward inference to attempt to derive the `xor`, by checking whether Ahmad is a POI. Since the system has no answer for that, inference halts. Sometime later, shown in the bottom half of Figure 3, `(POI Ahmad)` is added to the KB. The `xor` receives a message saying it is able to be used for the substitution of Ahmad for `arb1` (but not in general), and the initial forward inference task resumes, deriving `(BeingSoughtOut Ahmad)`. Here again it's worth noting that even though the IG entails that Ahmad is of interest to INSCOM, that was not derived since it was of no use to the backward inference task attempting to derive `wft5`, and it does not follow directly from the fact that Ahmad is not under surveillance, which was the assertion made with forward inference.

4.4 A Unifying Algorithm

In order to perform FR using IGs, two requirements must be fulfilled. Nodes must track whether they are part of a FR task (which one, and in which direction(s)), and whenever a channel is added to the graph it must be determined if forward or backward inference must continue along that channel.

We start by augmenting each node with two sets, initially empty, *fBR* and *fFwR* for focused inference tasks requiring future backward reasoning at that node, and focused inference tasks requiring future forward reasoning at that node,

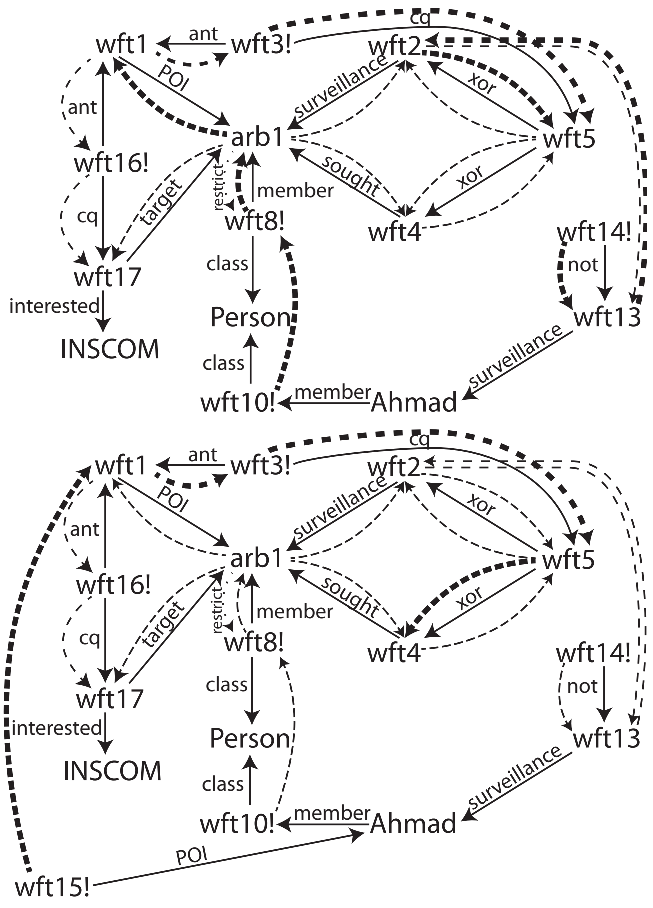

Figure 3: The IGs for the backward-in-forward FR example. In the top IG, it is asserted with forward inference that Azam is not under surveillance, `wft14!`. Forward inference proceeds to the unasserted `xor` rule of `wft5`, then backward inference tries to derive that rule, but is unable to at the present time. In the bottom IG, Ahmad is a POI (`wft15!`) is added to the KB, which allows the rule `wft5` to be used, and the fact that Ahmad is sought to be derived.

respectively. As `backward-infer` messages propagate backward through the graph opening channels, they add the goal of the backward reasoning task to the *fBR* set in each node. When forward inference is initiated, nodes reached have their *fFwR* set augmented with the origin of the forward inference task. Unlike IGs described in previous papers, we allow `backward-infer` messages to travel backward along already open channels if these sets need to be updated. Tracking which nodes are involved in each type of focused inference task allows one focused inference task to later be canceled without affecting the others.[9] To cancel these tasks, we use `cancel-infer` messages, which will only close a channel if

[9]If one wanted to simply cancel all or no FR tasks, these sets could be replaced with flags. Some book keeping is still required since it is impossible to tell whether forward or backward inference has been initiated from a node otherwise disconnected from the graph without some marker.

it's not needed for any more focused inference tasks, but can travel backward though the graph removing an entry from the nodes sets of future inference tasks.

When a new channel is added to the graph, the contents of its origin's *fFwR* or destination's *fBR* set determine whether or not to continue a FR task. If a new channel is created, and it's origin's *fFwR* set is non-empty, forward inference is continued along that new channel (and recursively forward), and the contents of the *fFwR* set is propagated forward. If a new channel has a destination with a non-empty *fBR* set, then backward inference starts at the new channel (and continues recursively), and the contents of the *fBR* set is propagated backward. These routines combine to allow for forward-in-backward FR, and forward FR.

The final aspect of the algorithm occurs when a rule is not asserted, but receives an `i-infer` message via forward inference, indicating an attempt to use that rule. In this case, that node attempts to have itself derived in general or for a specific substitution by beginning a backward reasoning task, adding itself to it's *fBR* set, and propagating that set backward. Once the rule has been derived, it recursively sends messages canceling the backward reasoning task backward through the graph, since it's purpose has been fulfilled. This allows for backward-in-forward FR.

5 Conclusion

Focused reasoning is a useful method for continuing inference tasks as new information is added to a knowledge base. We have presented three types of FR: forward-in-backward, forward, and backward-in-forward, building upon Inference Graphs – a concurrent graph-based inference mechanism already capable of forward, backward, and bi-directional inference. Each type of FR obviates the need to repeat completed inference, re-pose queries, or perform unnecessary inference. In addition, determining if FR should commence uses very few computational resources above those used normally to add a term to a knowledge base.

6 Acknowledgements

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.

References

Auble, P. M., Franks, J. J., & Soraci, S. A. (1979). Effort toward comprehension: Elaboration or "aha"? *Memory & Cognition*, *7*(6), 426–434.

eBay, Inc. (2013). *Saving your searches.* (http://pages.ebay.com/help/buy/searches-follow.html)

Forgy, C. (1979). *On the efficient implementation of production systems.* Unpublished doctoral dissertation, Carnegie-Mellon University, Department of Computer Science, Pittsburgh, PA, USA.

Forgy, C. (1982). Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence*, *19*, 17–37.

ISO/IEC. (2007, October). Information technology — Common Logic (CL): a framework for a family of logic-based languages, iso/iec 24707:2007(e) (First ed.) [Computer software manual]. Switzerland. (available from `http://standards.iso/ittf/license.html`)

McKay, D. P., & Shapiro, S. C. (1981). Using active connection graphs for reasoning with recursive rules. In *Proceedings of the seventh IJCAI* (pp. 368–374). Los Altos, CA: Morgan Kaufmann.

Pollock, J. L. (1990). Interest driven suppositional reasoning. *Journal of Automated Reasoning*, *6*(4), 419–461.

Schlegel, D. R., & Shapiro, S. C. (2012). Visually interacting with a knowledge base using frames, logic, and propositional graphs. In M. Croitoru, S. Rudolph, N. Wilson, J. Howse, & O. Corby (Eds.), *Graph structures for knowledge representation and reasoning, lecture notes in artificial intelligence 7205* (p. 188-207). Berlin: Springer-Verlag.

Schlegel, D. R., & Shapiro, S. C. (2013a). Concurrent reasoning with inference graphs (student abstract). In *Proceedings of AAAI-13* (p. 1637-1638). Menlo Park, CA: AAAI Press/The MIT Press.

Schlegel, D. R., & Shapiro, S. C. (2013b, December). Inference graphs: A roadmap. In *Poster collection of the second annual conference on advances in cognitive systems* (pp. 217–234).

Schlegel, D. R., & Shapiro, S. C. (2014). Concurrent reasoning with inference graphs. In M. Croitoru, S. Rudolph, S. Woltran, & C. Gonzales (Eds.), *Graph structures for knowledge representation and reasoning, lecture notes in artificial intelligence* (Vol. 8323, p. 138-164). Switzerland: Springer International Publishing.

Shapiro, S. C. (2004). A logic of arbitrary and indefinite objects. In D. Dubois, C. Welty, & M. Williams (Eds.), *Proceedings of KR2004* (pp. 565–575). Menlo Park, CA: AAAI Press.

Shapiro, S. C. (2010). Set-oriented logical connectives: Syntax and semantics. In F. Lin, U. Sattler, & M. Truszczynski (Eds.), *Proceedings of KR2010* (pp. 593–595). AAAI Press.

Shapiro, S. C., Martins, J. P., & McKay, D. P. (1982). Bi-directional inference. In *Proceedings of the fourth CogSci* (pp. 90–93). Ann Arbor, MI: the Program in Cognitive Science of The University of Chicago and The University of Michigan.

Shapiro, S. C., & Rapaport, W. J. (1992, January–March). The SNePS family. *Computers & Mathematics with Applications*, *23*(2–5), 243–275.

Swift, T., & Warren, D. S. (2012). XSB: Extending prolog with tabled logic programming. *Theory and Practice of Logic Programming*, *12*(1-2), 157–187.