Inference Graphs: Combining Natural Deduction and Subsumption Inference in a Concurrent Reasoner

Daniel R. Schlegel[a] and **Stuart C. Shapiro**[b]
[a]Department of Biomedical Informatics
[b]Department of Computer Science and Engineering
University at Buffalo, Buffalo NY, 14260
<drschleg,shapiro>@buffalo.edu

Abstract

There are very few reasoners which combine natural deduction and subsumption reasoning, and there are none which do so while supporting concurrency. Inference Graphs are a graph-based inference mechanism using an expressive first–order logic, capable of subsumption and natural deduction reasoning using concurrency. Evaluation of concurrency characteristics on a combination natural deduction and subsumption reasoning problem has shown linear speedup with the number of processors.

1 Introduction

Inference Graphs (IGs) are a graph-based inference mechanism using an expressive first–order logic (FOL), capable of subsumption and natural deduction (ND) reasoning using forward, backward, bi-directional (Shapiro, Martins, and McKay 1982), and focused reasoning (Schlegel and Shapiro 2014a), all using concurrency. In this paper we will focus our discussion on the ability of IGs to combine ND and subsumption reasoning using concurrency, a combination which no other system provides.[1]

ND is a proof–theoretic reasoning method which makes use of a set of introduction and elimination rules for each logical connective. Subsumption is a reasoning method which allows the derivation of knowledge about entire classes of entities from knowledge about other classes, without needing to introduce instances. Subsumption reasoning is most commonly associated with description logics (DLs) (Baader et al. 2010). It is often missing from first order reasoners because of the lack of non–atomic conceptual descriptions (Woods 1991).

There have been many ND reasoners over the years, and reasoning using subsumption is increasingly popular. The combination of these two techniques allows for more sophisticated reasoning about entire classes of entities. We know of only two reasoners which combine the two techniques, namely, ANALOG (Ali and Shapiro 1993; Ali 1994) — which is an ancestor of this work — and PowerLoom (USC Information Sciences Institute 2014). Unlike PowerLoom, IGs use a single graph structure to represent inference paths, whether they be for subsumption or ND. Neither ANALOG nor PowerLoom support concurrency.

IGs derive their ability to reason using both ND and subsumption largely from the logic they implement: a Logic of Arbitrary and Indefinite Objects ($\mathcal{L}_A$). $\mathcal{L}_A$ is a first–order logic which uses quantified terms instead of quantified formulas with universally and existentially bound variables. There are two types of quantified terms: arbitrary and indefinite. Arbitrary terms replace universally quantified formulas, and indefinite terms replace existentially quantified formulas. Both types of quantified terms are structured, having a set of *restrictions* which the quantified term satisfies. A quantified term subsumes another quantified term if their sets of restrictions are appropriately related (further detailed in Section 3.1). For example, an arbitrary term, a_1, subsumes another arbitrary term, a_2, if the restriction set of a_1 is a subset of that of a_2. Using this relationship, one can derive new information about a_2 from information about a_1, without introducing or knowing about any individuals which are instances of a_1 or a_2. This feature is the essence of subsumption reasoning.

$\mathcal{L}_A$ allows for forming expressions that use propositional connectives and quantified terms. Propositional connectives can be reasoned about using ND, while quantified terms can be reasoned about through subsumption reasoning. IGs allow the use of concurrency in both types of inference.

In Section 2 we will discuss $\mathcal{L}_A$ and some KR concerns. The IG formalism implementing $\mathcal{L}_A$ is presented in Section 3. Section 4 will discuss the IG concurrency model. Finally, Section 5 presents an evaluation of IG performance.

2 Background

2.1 A Logic of Arbitrary and Indefinite Objects

$\mathcal{L}_A$ is a FOL 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 ND 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),

[1]This paper is an extended and updated version of (Schlegel and Shapiro 2014c). Parts of it are adapted from (Schlegel and Shapiro 2013) and (Schlegel 2014). A conception of IGs using only ground predicate logic can be seen in (Schlegel and Shapiro 2014b).

which subsume the standard rules.

The logic makes use of arbitrary and indefinite terms (collectively, quantified terms). 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 defined by the conjunction of its set of restrictions. 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. Indefinite terms may be dependent on one or more arbitrary terms $D(q_i) = \{d_{i_1}, \ldots, d_{i_k}\}$. The structured nature of quantified terms allow them to satisfy Woods's requirement that a logic for subsumption have non–atomic conceptual descriptions. We write an arbitrary term as (every v_{q_i} $R(q_i)$) and an indefinite term as (some v_{q_i} $D(q_i)$ $R(q_i)$).[2] The structured nature of quantified terms syntactically differentiates them from universally and existentially quantified variables familiar in first–order predicate logic (FOPL).[3]

Semantically, arbitrary and indefinite terms are also different from FOPL's universally and existentially quantified variables. Instead of reasoning about *all* members of a class, $\mathcal{L}_A$ reasons about a *single* arbitrary member of a class.[4] 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.

To our knowledge, the only implemented systems which use a form of arbitrary term are ANALOG (Ali and Shapiro 1993) and Cyc (Lenat and Guha 1990), though arbitrary objects themselves were most famously defended by Fine (Fine 1983) after being attacked by Frege (Frege 1979). The logic of $\mathcal{L}_A$ is based on those developed by Ali and by Fine (Fine 1985a; 1985b), but is different — notably it is more expressive than ANALOG. It is designed with computation in mind, unlike Fine's work, which omits key algorithms.

Since an arbitrary term represents an arbitrary entity, there are no two arbitrary terms with the same set of restrictions. Occasionally it is useful to discuss two different arbitrary members with the same restrictions, so we provide the special restriction (notSame $q_{j_1} \ldots q_{j_k}$) for this purpose.

2.2 Knowledge Representation

In the tradition of the SNePS family (Shapiro and 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, hence `wft` instead of `wff`. An exclamation mark, "`!`", is appended to the label if it represents a proposition that is asserted in the KB. Arbitrary and indefinite terms are labeled `arb`*i* and `ind`*i*, respectively. 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.

Quantified terms are represented in the propositional graph just as other terms are. Arbitrary and indefinite terms also each have a set of restrictions, represented in the graph with special arcs labeled "restrict", and indefinite terms have a set of dependencies, represented in the graph with special arcs labeled "depend."

Each term in the knowledge base has a semantic type, itself existing within an ontology of semantic types which the user can add to. Only terms of type Proposition may be asserted (taken to be true) in the knowledge base. Generic terms are terms which contain either other generics, or arbitrary terms, and analytic generic terms are tautological generics built from the restrictions of an arbitrary term.

3 Combining Natural Deduction and Subsumption in IGs

IGs are an extension of propositional graphs. To propositional graphs, IGs add directed message passing *channels* wherever inference (ND or subsumption) is possible. Channels are built within rules and generic terms, and wherever terms *match* each other. Inference messages, containing substitutions, pass forward through channels. Nodes for rules collect and combine messages. When a received or combined message satisfies an inference rule it *fires*, sending more messages onward through the graph through its outgoing channels.

To motivate the discussion, we introduce an example inspired by the counter–insurgence domain, first only in the logical syntax of $\mathcal{L}_A$, and in a later subsection, using an IG.

```
;; A person is arrested if and only if
;;  they are held by a another person
;;   who is a corrections officer.
(iff
  (Arrested (every x (Isa x Person)))
  (heldBy x (some y (x)
              (Isa y Person)
              (Isa y CrctnsOfcr)
              (notSame x y))))

;; A person is detained if and only
;;  if they are held by another person.
```

[2] The syntax we use is a version of CLIF (ISO/IEC 2007).

[3] In many papers and books FOL and FOPL are used interchangeably. Here this is not the case — $\mathcal{L}_A$ and FOPL are both members of the class of FOLs, but $\mathcal{L}_A$ is not FOPL.

[4] This is in contrast to DLs, which reason about classes. DLs have two languages: one to discuss classes, another to discuss instances; $\mathcal{L}_A$ uses a single language to discuss instances and arbitrary individuals.

```
(iff
  (Detained (every x (Isa x Person)))
  (heldBy x
          (some y (x) (Isa y Person)
                      (notSame x y))))

;; A person is either detained,
;;  on supervised release, or free.
(xor
  (Detained (every x (Isa x Person)))
  (onSupervisedRelease x)
  (Free x))

;; A person who is not free
;;  has travel constraints.
(hasTravelConstraints
  (every x (Isa x Person)
           (not (Free x))))

;; A person who has been captured,
;; has been arrested.
(Arrested (every x (Isa x Person)
                   (Captured x)))
```

It will then be asked of the system if captured persons have travel constraints:

```
(hasTravelConstraints
  (every x (Isa x Person) (Captured x)))
```

This can be derived by recognizing that the arbitrary person who is captured, is a person (a subsumptive relationship), then applying rules in the KB which apply to persons. For example, since a person is arrested if and only if they are held by another person who is a corrections officer, then a *captured* person is arrested if and only if they are held by another person who is a corrections officer.

3.1 The Match Process

Whenever a term is added to the graph, it is matched with all other terms. A term, t_i, matches another term, t_j, if t_i and t_j unify,[5] if each substitution of one quantified term for another in the created substitutions follow appropriate subsumption rules, and if t_i is the same type, or a subtype of t_j. The match process is a way to identify more specific terms which may share their instances with more general ones. For example, consider that some person who is a corrections officer is still a person.

When t_i and t_j are unified, instead of producing a most general unifier, a factorization is produced which contains bindings for each of the terms being unified. While this is described in (McKay and Shapiro 1981), the main idea is that as unification is performed, instead of forming a single substitution, form two – σ_i and σ_j – such that all and only quantified terms in t_i are given bindings in σ_i, and all and only quantified terms in t_j are given bindings in σ_j. During this process, we treat quantified terms as simple variables without considering their restriction sets.

[5]We avoid pairwise unification using an approach similar to (Hoder and Voronkov 2009)

Once t_i and t_j have unified, we must determine in which direction(s) (if either) their substitutions are compatible in their subsumption relationship and in type. To define our notion of subsumption formally, we say:

1. an arbitrary, arb_i, subsumes another, arb_k, if $\forall r_{i_j} \in R(arb_i), \exists r_{k_l} \in R(arb_k)$ such that r_{i_j} matches r_{k_l},
2. an arbitrary, arb_i, subsumes an indefinite, ind_k, if $\forall r_{i_j} \in R(arb_i), \exists r_{k_l} \in R(ind_k)$ such that r_{i_j} matches r_{k_l}, and
3. an indefinite, ind_i, subsumes another, ind_k, if $\forall r_{k_j} \in R(ind_k), \exists r_{i_l} \in R(ind_i)$ such that r_{k_j} matches r_{i_l}.

If t_i and t_j unify, and for each substitution pair $t_l/v_l \in \sigma_i$, t_l has type equal or lower than v_l, and if t_l is a quantified term, v_l subsumes t_l, then we call t_i an *originator*, and t_j a *destination*, and add the 4-tuple $< t_i, t_j, \sigma_i, \sigma_j >$ to the set of matches to return. If t_i and t_j unify and for each substitution pair $t_l/v_l \in \sigma_j$, t_l has type equal or lower than v_l, and if t_l is a quantified term, v_l subsumes t_l, then we call t_j an *originator*, and t_i a *destination*, and add the 4-tuple $< t_j, t_i, \sigma_j, \sigma_i >$ to the set of matches to return. So, 0, 1, or 2 4-tuples are the result of the process.

3.2 Communication Within the Network

The results of the match process are used to create some of the *channels* in the graph. Channels are a pre-computation of every path that inference might take. Each channel starts at an originator node, and terminates at a destination. Each node has channels to every node that it can derive and to every node that can make use of inference results that the originator has derived. *Messages* are sent through the channels. Messages come in several types, and either communicate newly inferred knowledge (*inference messages*) or control inference operations (*control messages*).

Channels In addition to the originator and destination, each channel has a type and contains three structures — a valve, a filter, and a switch. Valves control the flow of inference; filters discard inference messages that are irrelevant to the destination; and switches adjust the variable context of the substitutions that inference messages carry from that of the originator to that of the destination.

There are three types of channels — i-channels, g-channels, and u-channels. I-channels are meant to carry messages that say "I have a new substitution of myself you might be interested in", and u-channels carry messages that say "you or your negation have been derived with the given substitution." G-channels are i-channels, but used only within generic terms.

Definition 3.1. A *channel* is a 6-tuple $< o, d, t, v, f, s >$, where o is the originator, d is the destination, t is the type, v is the valve, f is the filter, and s is the switch. ■

A filter serves to stop messages with irrelevant substitutions from flowing through a channel. The filter ensures that the incoming message's substitution is relevant to d by ensuring that, for every substitution pair t_f/y in the destination bindings, there is a substitution pair t_s/y in the passing message substitution such that either $t_f = t_s$ or t_s is a specialization of t_f, determinable through one-way pattern matching. If a message does not pass the filter, it is discarded.

Switches change the substitution's context to that of the destination term. The switch applies the originator binding substitution to the term of each pair in the passing message substitution. This adjusts the substitution to use quantified terms required by the destination. The updated substitution is stored in the passing message.

Valves control inference by allowing or preventing messages from passing to the destination.

Definition 3.2. A *valve* is a pair $< (open|closed), wq >$ where the first position indicates whether the valve is opened or closed, and wq is a waiting queue.[6] ■

When an inference message is submitted to a channel, it first is sent through the filter, then the switch. Should the message pass the filter, it will eventually reach the valve. The valve, depending on whether it is open or closed, allows the message to pass or prevents it from doing so. If a reached valve is closed, the message is added to that valve's waiting queue. When a valve is opened, the items in the waiting queue are sent on to the destination.

Channels are built between any two nodes which match each other. All channels from the match operation are i-channels, since they communicate new substitutions for the originator, which the destination may be interested in.

Channels are built in several other locations as well: within rules, generic terms, and quantified terms. Each of these channels is simpler than those created from the match process, as their filters and switches are no-ops.

Within rules, i-channels are built from each antecedent to the node for the rule itself, and u-channels are built from the rule node to each consequent. This allows the antecedents to inform the rule of newly satisfying substitutions, and it allows the rule node to send substitutions produced when the rule fires to its consequents.

A generic term, g, is defined recursively as a term that is a parent of one or more arbitrary terms $a_1, \ldots, a_n$, or one or more other generic terms, $g_1, \ldots, g_m$. Each a_i and g_k has an outgoing g-channel to g. This allows substitutions to begin at the arbitrary terms, and be built up successively as higher level generic terms are reached.

Arbitrary terms have i-channels from each restriction to the arbitrary itself. This allows arbitrary terms to find instances of themselves through the combination of substitutions from terms matching each restriction.

In Figure 1 the IG for the example is shown, with channels drawn as specified. For example, `wft2` has an i-channel to `wft3!` and `wft3!` has a u-channel to `wft2`, indicating that `wft2` may want to share a substitution it has with `wft3!`, and `wft3!` might derive `wft2`. Additionally, `wft19!` has an i-channel to `wft2`, indicating that `wft2` may be interested in `wft19!`'s assertional status and substitution.

Messages Messages of several types are transmitted through the IG's channels, serving two purposes: relaying derived information and controlling the inference process. A message can be used to relay the information that its origin has a new asserted or negated substitution instance (an `i-infer` or `g-infer` message), or that it has found a substitution for the destination to now be asserted or negated (`u-infer`). These messages flow forward through the channels, and are called inference messages. Control messages flow backward through channels, controlling inference: `backward-infer` messages open valves, and `cancel-infer` messages close them.

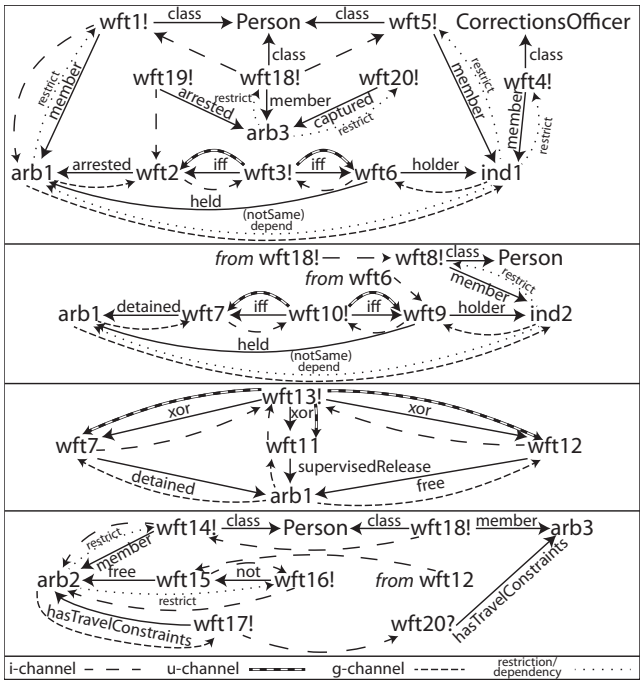

Figure 1: The IG for the example, split into four segments for easier reading. The top IG segment contains propositions meant to mean that a person is arrested if and only if they are held by a another person who is a corrections officer (`wft3`), and that the arbitrary captured person is arrested (`wft19`). The second segment contains the rule that a person is detained if and only if they are held by another person (`wft10`). The third segment contains the rule that a person is either detained, on supervised release, or free (`wft13`), and the final segment contains the generic proposition that a person who is not free has travel constraints (`wft17`), along with the question of whether the arbitrary person who is captured has travel constraints (`wft20`). Channels and restrictions/dependencies for quantified terms are drawn according to the key at the bottom of the figure.

Definition 3.3. A message is an 8-tuple:

$$< pri, subst, type, pos, neg, fNS, t?, fwd? >$$

where: pri is the priority of the message (discussed further in Section 4); $subst$ is a substitution; $type$ is the type of message; pos and neg are the are the number of known true ("positive") and negated ("negative") antecedents of a rule, respectively; the *fNS* is the *flagged node set*, which contains a mapping from each antecedent with a known truth value to its truth value; $t?$ indicates whether the message regards a

[6] This is a simplified conception of a valve which does not account for the dynamic nature of backward inference. See (Schlegel 2014) for more on this topic.

true or negated term; and $fwd?$ indicates whether this message is part of a forward inference process. ■

3.3 Inference by Combining Messages

Inference in the IG is essentially the combination of messages, and the determination of whether a resulting combination satisfies the requirements of a deductive rule, quantified term, or generic term.

Two messages, m_1 and m_2 may be combined if they are compatible – that is, if their substitutions and flagged node sets are compatible. We say that two substitutions, $\sigma = \{t_{\sigma_1}/v_{\sigma_1} \ldots t_{\sigma_n}/v_{\sigma_n}\}$ and $\tau = \{t_{\tau_1}/v_{\tau_1} \ldots t_{\tau_m}/v_{\tau_m}\}$, are compatible if whenever $v_{\sigma_i} = v_{\tau_j}$ then $t_{\sigma_i} = t_{\tau_j}$, and that two flagged node sets are compatible if they have no contradictory entries (that is, no antecedent of the rule is both true and false).

Two messages that are compatible are combined in the following way. Let

$$m_1 =< pri_1, subst_1, type_1, pos_1, neg_1, fNS_1, t?_1, fwd?_1 >$$

and

$$m_2 =< pri_2, subst_2, type_2, pos_2, neg_2, fNS_2, t?_2, fwd?_2 >$$

where m_2 is the most recently received message. The combined message, m_3, combines m_1 and m_2 as follows:

$$\begin{aligned} m_3 =< & max(pri_1, pri_2), merge(subst_1, subst_2), \\ & nil, |posEntries(fNS_3)|, |negEntries(fNS_3)|, \\ & fNS_3, nil, or(fwd?_1, fwd_2) > \end{aligned}$$

Some fields in m_3 are made nil, to be later filled in as necessary. The combined flagged node set, fNS_3, is the addition of all entries from fNS_2 to fNS_1.

Messages may be combined efficiently using several different data structures. Which structure is used depends on the logical operation a specific node performs. The approaches include a tree-based approach called a P-Tree, a hash-map based technique called an S-Index, and a default combinatorial algorithm (see (Choi and Shapiro 1992)). The result of the combination process is a set of new messages seen since just before the message arrived at the node. The newly created messages are added to the node's cache, and examined to determine if the conditions of the node are met to send out further messages. If the message arriving already exists in the cache, no work is done. This prevents re-derivations, and can cut cycles.

The pos and neg portions of the messages in the new combined set are used to determine if the conditions of the node are satisfied. For a rule node, this determines whether the rule may fire. For example, for a conjunctive rule to fire, pos must be equal to the number of rule arguments. A disadvantage of this reasoning approach is that some rules are difficult, but not impossible, to implement, such as negation introduction and proof by cases. For the non-rule-node combination nodes, this process determines if an instance has been found, by waiting until pos is equal to the number of required restrictions or subterms.

The restrictions of an arbitrary term arb, represented by analytic generic terms with i-channels to arb, are taken conjunctively. Therefore, messages from restrictions must be combined in arb as they are available. When a message m has been received or created via combination such that $m_{pos} = |R(q)|$, where m_{pos} is the number of positive antecedent instances in the message, it is considered to be an instance of the arbitrary, and is sent onward to any generic terms the arbitrary is part of. In the example this is evident in `arb2`, which has two restrictions: `wft14!` and `wft16!`. Only when both of these terms report to `arb2` with compatible substitutions can the combined substitution be sent to `wft17!` for instantiation of the generic.

As discussed earlier, a generic term, g, is defined recursively as a term that is a parent of one or more arbitrary terms $a_1, \ldots, a_n$, or one or more other generic terms, $g_1, \ldots, g_m$. As instances are discovered by a_i and g_k, substitutions for those instances are sent to g via the appropriate i-channel to g. g combines these substitutions should they be compatible, and sends out resulting messages. Unlike arbitrary terms where all restrictions must be satisfied for an instance to be made, generics require instances for only as many compatible subterms as are available.

In the example, we can now derive that the arbitrary captured person has travel constraints. Messages flow forward from `wft18!` (through `wft1!` and `arb1`), and from `wft19!` to `wft2`, which then satisfies `wft3!`, deriving a generic instance of `wft6` – the arbitrary captured person is held by a corrections officer. We derive the arbitrary captured person is detained by messages flowing from `wft6`, and `arb1` (through `ind2`) to `wft9`, which satisfies `wft10!` and derives a generic instance of `wft7`. The message from `wft7` satisfies the `xor` rule `wft13!`, allowing a negated instance of `wft12` to be derived – captured persons are not free. Finally we learn that the arbitrary captured person has travel restrictions since messages from `wft18!` (through `wft14!`), and from `wft12` (through `wft15` and `wft16!`) satisfy `arb2`, allowing a message to be sent asserting `wft20` (through `wft17!`).

4 Concurrency

The structure of IGs lends itself naturally to concurrent inference. Any number of nodes in the graph may process messages simultaneously without fear of interfering with any others. Only when a single node receives multiple messages must those messages be processed synchronously. This synchronous processing is necessary because the message caches are shared state. We need not concern ourselves with the actual order in which the messages are processed, since the operation is commutative, meaning there is no need to maintain a queue of changes to the message cache.

In order to perform inference concurrently, the IG 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, as seen in Figure 2. When a message passes through a 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. Tasks are removed from the queue and executed as processors become available. When a task is executed, inference is performed, and any newly generated messages are sent toward its outgoing valves for the process to repeat.

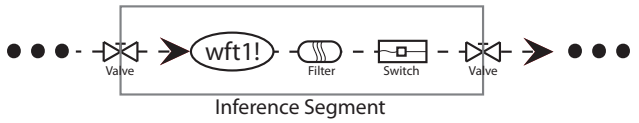

Figure 2: A single inference segment.

The goal of any inference system is to infer the knowledge requested by the user. If we arrange an IG so that a user's request (in backward inference) is on the right, and channels flow from left to right wherever possible (the graph may contain cycles), we can see this goal as trying to get messages from the left side of the graph to the right side of the graph. We, of course, want to do this as quickly as possible.

Every inference operation begins processing inference messages some number of levels to the left of the query node. Since there are a limited number of tasks that can be running at once due to hardware limitations, we must prioritize their execution, remove tasks that we know are no longer necessary, and prevent the creation of unnecessary tasks. Therefore,

1. Tasks relaying newly derived information using segments to the right are executed before those to the left.
2. Once a node is known to be true or false, all tasks still attempting to derive it are canceled, as long as their results are not needed elsewhere, and in all channels pointing to it that may still derive it, valves are closed.
3. Once a rule fires, all tasks for potential antecedents of that rule still attempting to satisfy it are canceled, as long as their results are not needed elsewhere. Valves are closed in channels from antecedents that may still satisfy it.

All `cancel-infer` messages have the highest priority. Then come `i-infer` and `u-infer` messages. `backward-infer` messages have the lowest priority, decreasing at each step backward through the graph. As `i-infer` and `u-infer` messages flow closer to the goal, they get higher priority, but their priorities remain lower than that of `cancel-infer` messages.

5 Evaluation

We have aimed to combine ND and subsumption reasoning within a system which supports concurrency. We have not implemented the fastest known algorithms for procedures such as subsumption, due only to research agenda. For this reason, the most interesting measure of the performance of IGs is the concurrency characteristics.

We evaluated the performance of IGs in backward inference, as it is the most resource intensive type of inference IGs can perform, and most fully utilizes the scheduling heuristics. To do so we used graphs of chaining and-entailments, meaning for each implication to derive its consequent, all its antecedents had to be true. Each entailment had bf antecedents, where bf is the branching factor, and a single consequent. Each antecedent and consequent made use of the same single arbitrary term, a, containing a single restriction.[7] Each consequent was the consequent of exactly one rule, and each antecedent was the consequent of another rule, up to a depth of d entailment rules. Exactly one consequent, cq, was not the antecedent of another rule. A single instance of each leaf node was asserted which made use of an arbitrary which was subsumed by a. We tested the performance of the system in backchaining on and deriving a term subsumed by cq.

Since we backchained on a term subsumed by cq, this meant terms subsumed by every rule consequent and antecedent in the graph would have to be derived. This is the worst case of entailment. The timings we observed are presented in Table 1.[8] This experiment showed that speedup grows linearly[9] as more CPUs are involved in inference.

Table 1: Inference times using 1, 2, 4, and 8 CPUs for 100 iterations in an IG with $bf = 2$ and $d = 7$.

CPUs	Inference Time (ms)	Speedup
1	229551	1.00
2	123015	1.87
4	65783	3.49
8	36013	6.38

Experiments testing the effect of depth and branching factor showed no statistically significant effect. All results were found to be very similar to those of (Schlegel 2014) which tested ND reasoning alone.

6 Conclusion

IGs are the only inference mechanism which combine ND and subsumption reasoning, and support concurrency. Inference is performed through a message passing architecture built upon propositional graphs. IGs derive their ability to perform both types of inference from a logic, $\mathcal{L}_A$, which uses structured quantified terms. Evaluation of concurrency characteristics on a combination ND and subsumption reasoning problem has shown linear speedup with the number of processors.

7 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.

[7] Adjusting the number of arbitrary terms or restrictions used has a very small impact, as calculation of instances is performed only once per arbitrary, and the arbitrary or arbitraries need to be shared by all nodes to perform meaningful inference.

[8] Tests were performed on a Dell Poweredge 1950 server with dual quad-core Intel Xeon X5365 processors and 32GB RAM. Each test was performed twice, with the second result being used.

[9] $y = 0.764x + 0.32$, $R^2 = 0.9986$

References

Ali, S. S., and Shapiro, S. C. 1993. Natural language processing using a propositional semantic network with structured variables. *Minds and Machines* 3(4):421–451.

Ali, S. S. 1994. *A "Natural Logic" for Natural Language Processing and Knowledge Representation*. Ph.D. Dissertation, Technical Report 94-01, Department of Computer Science, State University of New York at Buffalo, Buffalo, NY.

Baader, F.; Calvanese, D.; McGuinness, D. L.; Nardi, D.; and Patel-Schneider, P. F., eds. 2010. *The Description Logic Handbook: Theory, Implementation, and Applications*. New York, NY, USA: Cambridge University Press, 2nd edition.

Choi, J., and Shapiro, S. C. 1992. Efficient implementation of non-standard connectives and quantifiers in deductive reasoning systems. In *Proceedings of the Twenty-Fifth Hawaii International Conference on System Sciences*. Los Alamitos, CA: IEEE Computer Society Press. 381–390.

Fine, K. 1983. A defence of arbitrary objects. In *Proceedings of the Aristotelian Society*, volume Supp. Vol. 58, 55–77.

Fine, K. 1985a. Natural deduction and arbitrary objects. *Journal of Philosophical Logic*.

Fine, K. 1985b. *Reasoning with Arbitrary Objects*. New York: Blackwell.

Frege, G. 1979. Posthumous writings. In Hermes, H.; Kambartel, F.; Kaulbach, F.; Long, P.; White, R.; and Hargreaves, R., eds., *Posthumous Writings*. Blackwell: Oxford.

Hoder, K., and Voronkov, A. 2009. Comparing unification algorithms in first-order theorem proving. In *Proceedings of the 32nd annual German conference on Advances in artificial intelligence*, KI'09, 435–443. Berlin, Heidelberg: Springer-Verlag.

ISO/IEC. 2007. *Information technology — Common Logic (CL): a framework for a family of logic-based languages, ISO/IEC 24707:2007(E)*. ISO/IEC, Switzerland, First edition. available from `http://standards.iso/ittf/license.html`.

Lenat, D. B., and Guha, R. V. 1990. *Building Large Knowledge-Based Systems: Representation and Inference in the Cyc Project*. Reading, MA: Addison-Wesley.

McKay, D. P., and Shapiro, S. C. 1981. Using active connection graphs for reasoning with recursive rules. In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, 368–374. Los Altos, CA: Morgan Kaufmann.

Schlegel, D. R., and Shapiro, S. C. 2013. Inference graphs: A roadmap. In *Poster Collection of the Second Annual Conference on Advances in Cognitive Systems*, 217–234.

Schlegel, D. R., and Shapiro, S. C. 2014a. The 'ah ha!' moment : When possible, answering the currently unanswerable using focused reasoning. In *Proceedings of the 36th Annual Conference of the Cognitive Science Society*. Austin, TX: Cognitive Science Society. In Press.

Schlegel, D. R., and Shapiro, S. C. 2014b. Concurrent reasoning with inference graphs. In Croitoru, M.; Rudolph, S.; Woltran, S.; and Gonzales, C., eds., *Graph Structures for Knowledge Representation and Reasoning, Lecture Notes in Artificial Intelligence*, volume 8323 of *Lecture Notes in Artificial Intelligence*. Switzerland: Springer International Publishing. 138–164.

Schlegel, D. R., and Shapiro, S. C. 2014c. Inference graphs: A new kind of hybrid reasoning system. In *Proceedings of the Cognitive Computing for Augmented Human Intelligence Workshop at AAAI-14 (CCAHI@AAAI-14)*.

Schlegel, D. R. 2014. *Concurrent Inference Graphs*. Ph.D. Dissertation, State University of New York at Buffalo, Department of Computer Science, Buffalo, NY, USA.

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

Shapiro, S. C.; Martins, J. P.; and McKay, D. P. 1982. Bidirectional inference. In *Proceedings of the Fourth Annual Conference of the Cognitive Science Society*, 90–93. Ann Arbor, MI: the Program in Cognitive Science of The University of Chicago and The University of Michigan.

Shapiro, S. C. 2004. A logic of arbitrary and indefinite objects. In Dubois, D.; Welty, C.; and Williams, M., eds., *Principles of Knowledge Representation and Reasoning: Proceedings of the Ninth International Conference (KR2004)*, 565–575. Menlo Park, CA: AAAI Press.

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

USC Information Sciences Institute. 2014. PowerLoom knowledge representation and reasoning system. `www.isi.edu/isd/LOOM/PowerLoom`.

Woods, W. A. 1991. Understanding subsumption and taxonomy: A framework for progress. In Sowa, J. F., ed., *Principles of Semantic Networks*. San Mateo, CA: Morgan Kauffman. 45–94.