ALL: Formalizing Access Limited Reasoning[*]

J. M. Crawford[†]
Benjamin Kuipers
Department of Computer Sciences
The University of Texas At Austin
Austin, Texas 78712
jc@cs.utexas.edu
kuipers@cs.utexas.edu

25 May 1990

Abstract

Access-Limited Logic (ALL) is a language for knowledge representation which formalizes the access limitations inherent in a network-structured knowledge base. Where a classical deductive method or logic programming language would retrieve all assertions that satisfy a given pattern, an access-limited logic retrieves all assertions reachable by following an available access path. The complexity of inference is thus independent of the size of the knowledge-base and depends only on its local connectivity. Access-Limited Logic, though incomplete, still has a well defined semantics and a weakened form of completeness, *Socratic Completeness*, which guarantees that for any query which is a logical consequence of the knowledge-base, there exists a series of queries after which the original query will succeed. This chapter presents an overview of ALL, and sketches the proofs of its Socratic Completeness and polynomial time complexity.

1 Introduction

It has long been a guiding principle in work on semantic networks that by imposing a network structure on large knowledge bases one can increase the efficiency of reasoning. Intuitively, in a semantic network related concepts are located "close together" in the network and thus search and inference can be guided by the structure of the knowledge-base. However, formalisms for semantic networks have generally treated the semantic network notation as a variant of predicate calculus, and have regarded the access limitations inherent in a network to be an extra-logical indexing mechanisms. In Access-Limited Logic (ALL) we incorporate these access limitations directly into the logic. One benefit of this approach is that we can assess the impact of access limitations on the completeness and complexity of reasoning. ALL is, in fact, not complete, but it is Socratically Complete — that is, for any query which is a logical consequence of the knowledge-base, there exists a series of queries after which the original query will succeed. Further, the complexity of inference in ALL is independent of the size of a knowledge-base and depends only on the size of the accessible portion of the knowledge-base.

[*]This work has taken place in the Qualitative Reasoning Group at the Artificial Intelligence Laboratory, The University of Texas at Austin. Research of the Qualitative Reasoning Group is supported, in part, by NSF grants IRI-8602665, IRI-8905494, and IRI-8904454, by NASA grants NAG 2-507 and NAG 9-200, and by the Texas Advanced Research Program under grant no. 003658-175. This paper appeared in John Sowa (Ed.), *Principles of Semantic Networks*, San Mateo, CA: Morgan Kaufmann, 1991, pp. 299-330.

[†]Supported in part by a fellowship from GTE.

Reasoning is hard. If a knowledge representation language is as expressive as first-order predicate calculus, then the problem of deciding what an agent implicitly knows (i.e. what an agent could logically deduce from its knowledge) is unsolvable [Boolos & Jeffrey, 80]. Thus a sound and decidable knowledge representation and reasoning system must either give up expressive power, or use a weak inference system with an incomplete set of deduction rules or artificial resource limits (e.g. bounds on the number of applications of *modus ponens*). However, such inference systems tend to be difficult to describe semantically and tend to place unnatural limits on an agent's reasoning ability [Levesque, 86].

As an example of non-trivial inference, consider the following problem (from [Wylie, 57]):

> In a certain bank the positions of cashier, manager, and teller are held by Brown, Jones and Smith, though not necessarily respectively. The teller, who was an only child, earns the least. Smith, who married Brown's sister, earns more than the manager.
> What position does each man fill ?

A person looking at such a problem cannot come up with a solution immediately (though the positions follow from a fairly small amount of common-sense knowledge about families, partial orders, and co-reference). A certain amount of conscious thought is required; one has to ask oneself just the right questions. Similarly, we do not expect our knowledge-representation system to be able to solve such a problem immediately, since intuitively we expect it to be able to reason only about as well as a person could reason without conscious thought. We do, however, expect it to be able to solve such a problem after being given an appropriate set of leading questions. If we ask:

1. If Smith were the manager then how could he earn more than the manager ?
2. If Smith were the teller then how could he earn more than the manager ?
3. If Brown were the teller then how could he have a sister ?

then one can see immediately that Smith must be the cashier, Brown the manager, and Jones the teller. Similarly, a knowledge-representation system, after being ask by the user (or heuristically generating) such a series of questions should be able to determine which man holds which position. We have translated this problem into ALL and given it (along with appropriate common-sense knowledge) to our lisp implementation, Algernon. Inference in Algernon is incomplete, and Algernon fails initially to solve the problem. However, after we ask it the questions given above, it is able to determine which position each man fills.

More abstractly, ALL has an important property we call *Socratic Completeness*[1] — for any query of a proposition which is a consequence (in predicate calculus) of the knowledge-base, there exists a preliminary query after which the original query will succeed. ALL also has a more technical weakened completeness property which we call *Partitional Completeness*. Roughly, Partitional Completeness says that if all facts and rules needed to prove a query are located 'close enough' (see section 3) to the query, then it will succeed immediately.

The rest of this chapter is organized as follows. In section 2 we discuss our general approach to knowledge-representation. Section 3 gives an overview of the formalization of ALL, and the proofs of Socratic and Partitional Completeness. We outline the argument for the polynomial time complexity of ALL in section 4. In section 5 we briefly discuss related work, and section 6 is our conclusion.

2 Overview of Access-Limited Logic

In the broadest sense the study of knowledge-representation is the study of how to represent knowledge in such a way that the knowledge can be used by a machine. From this vague definition we can conclude that a knowledge-representation system must have the following properties:

[1]We have since discovered that the idea of Socratic Completeness is also used in [Powers, 78] where it is referred to as Socratic Adequacy.

1. It must have a reasonably compact syntax.
2. It must have a well defined semantics so that one can say precisely what is being represented.
3. It must have sufficient expressive power to represent human knowledge.
4. It must have an efficient, powerful, and understandable reasoning mechanism.
5. It must be usable to build large knowledge-bases.

It has proved difficult, however, to achieve the third and forth properties simultaneously.

Our approach in ALL begins with the well known mapping between atomic propositions in predicate calculus and slots in frames; the atomic proposition that the object a stands in relation r to the object b can be written logically as $r(a,b)$ or expressed, in frames, by including object b in the r slot of object a [Hayes, 79].

$$P(a,b) \quad \equiv \quad \boxed{\begin{array}{l} \texttt{a:} \\ \quad \texttt{P:} \\ \qquad \texttt{values: } \{\ldots\texttt{b}\ldots\} \end{array}}$$

We refer to the pair $\langle a, r\rangle$ as a *frame-slot*. Thus $r(a,b)$ is equivalent to saying that the value b is in the frame-slot $\langle a, r\rangle$. The frames *directly accessible* from a frame-slot are those which appear in the frame-slot.[2] Extending this idea, we define an *access path*, in a network of frames, as a sequence of frames such that each is directly accessible from a frame-slot of its predecessor. It is useful to generalize this definition and allow access paths to branch on all values in the frame-slots. A sequence of propositions defines an access path if any variable appearing as the first argument to a proposition has appeared previously in the sequence (operationally, this means that retrieval always accesses a known frame-slot). For example, "John's parent's sister" can be expressed in ALL as the path:

$$(parent(John, x), sister(x, y))$$

This defines an access path from the frame for *John* to the frames for *John*'s parents (found by looking in the frame-slot $\langle John, parent\rangle$), to *John*'s parents' sisters.

¿From access paths we build the inference rules of ALL. A rule is always associated with a particular slot in the network. Backward chaining *if-needed* rules are written in the form: $\beta \leftarrow \alpha$ (the structure of α and β is discussed below) and applied when a value for the slot is needed. Forward chaining *if-added* rules are written in the form: $\alpha \rightarrow \beta$ and applied when a new value for the slot is inserted. In either case the antecedent of a rule must define an access path (beginning with the slot the rule is associated with). For example, using the access path above we can write the if-needed rule:

$$aunt(John, y) \leftarrow parent(John, x), sister(x, y)$$

But we *cannot* write the (logically equivalent) rule:

$$aunt(John, y) \leftarrow sister(x, y), parent(John, x),$$

since the antecedent does not define an access path.[3]

[2] Slots in ALL contain only frames and rules (defined below).

[3] The restriction to access paths limits the syntax of ALL, but is not a fundamental limit on its expressive power since one could always add a new constant and make it the first argument to every predicate. This would amount to making the entire knowledge-base a single frame.

Where a classical deductive method or logic programming language would retrieve all known assertions that satisfy a given pattern, an access-limited logic retrieves all assertions reachable by following an available access path. The use of access paths alone, however, is insufficient to guarantee computational tractability in very large knowledge-bases. The evaluation of a path can cause an explosive back-chaining of rules which can spread throughout the knowledge-base. To prevent this, ALL introduces a second form of access limitation. The knowledge-base in ALL is divided up into partitions and back-chaining is not allowed across partitions — facts in other partitions are simply retrieved. When used together, these two kinds of access limitations can limit the complexity of inference to a polynomial function of the size of the portion of the knowledge-base accessible from the local partition.

3 The Logical Coherence of ALL.

A price must be paid for the efficiency of access limitations. Inference in ALL is weaker than inference in predicate calculus, since only locally accessible facts and rules can be used in deductions. However, logical coherence does not necessarily require completeness. Rather it is an informally defined collection of desirable formal properties. We have proven that a dialect of ALL has the following properties of a logically coherent knowledge representation system:

- ALL has a well defined syntax and proof theory.
- The semantics of ALL can be defined by a purely syntactic mapping of ALL knowledge-bases, queries and assertions to predicate calculus.
- In terms of this mapping, inference in ALL is consistent, Socratically Complete, and Partitionally Complete.

These properties are stated more precisely in theorems below.

We view these formal properties as necessary but not sufficient conditions for logical coherence. There remains, at least, the less formal claim that knowledge can be organized cleanly into partitions. This claim is discussed further in sections 3.4.2 and 6.2.

The rest of this section sketches the formal development of ALL. The full account can be found in [Crawford & Kuipers, 90]. Our formal work in ALL generally lags several months behind our implementation work and the results presented here only formalize a part of ALL. Specifically, our current formalism does not allow negation or quantification (though our lisp implementation supports both — see section 6).

3.1 Basic Notation

In the meta-theory of ALL we use the following notation. Quantified expressions are written in the form:

$$(\langle Quantifier\rangle\langle Variable\rangle : \langle Range\rangle : \langle Expression\rangle).$$

Thus, for example:

$$(\forall x : pred_1(x) : pred_2(x))$$

is read "For all x such that $pred_1(x)$, $pred_2(x)$". Similarly:

$$(\bigcup x : pred(x) : foo(x))$$

(where foo is a set valued function) denotes the union, over all x such that $pred(x)$, of $foo(x)$.

We delineate lists with the usual () and notate the empty list by nil. If α is a list then:

- $head(\alpha)$ is the first element in α.
- $rest(\alpha)$ is all but the first element in α.

We define $append(\alpha_1, \alpha_2)$ to be the result of appending the list α_1 to the beginning of the list α_2.

3.2 Syntax of ALL

The syntax of ALL is quite similar to the syntax of logic programming. Accordingly we develop the syntax of ALL generally following the notation in [Apt, 88].

3.2.1 Alphabets, Terms and Propositions

The *alphabet* of an Access-Limited Logic consists of countably infinite sets of variables, constants, and relations, the connectives $\leftarrow$ and $\rightarrow$, and the operators *query* and *assert*. A *term* is a constant or a variable. $r(t_1, \ldots, t_n)$ is a *proposition* iff r is an n-ary relation and all t_i are terms.[4] A *fact* is a proposition such that all t_i are constants. For a term, proposition, or list of propositions, α:

- $vars(\alpha)$ is the set of variables appearing in α.
- $relations(\alpha)$ is the set of relations appearing in α.
- $constants(\alpha)$ is the set of constants appearing in α.

If $vars(\alpha) = \emptyset$ then α is *ground.* (so a ground propositions is a fact).

3.2.2 Access Paths

An *access path* (or simply a *path*) is a pair $\langle V, \alpha \rangle$ where α is a list of propositions and V is a set of variables. In general the first argument to each proposition in α must have occurred previously in α. The variables in V are exceptions to this rule and may occur as the first argument to propositions without having occurred previously in α. The need for such exceptions will become apparent when rules are defined below. If $V = \{\}$ then we omit it and say α is a path. A path of length one is a *primitive* path.

3.2.3 Rules

Assume $r(t_1, \ldots, t_n)$ is a proposition and α is a non-empty list of propositions. $r(t_1, \ldots, t_n) \leftarrow \alpha$ is an *if-needed rule* iff both of the following hold:

1. Either t_1 is a constant and α is a path, or t_1 is a variable and $\langle \{t_1\}, \alpha \rangle$ is a path.
2. $vars(\{t_1, \ldots, t_n\}) \subset vars(\alpha)$.

Intuitively, the first restriction ensures that when the rule is "fired" (i.e. when the consequent of the rule has been unified against a primitive path), the antecedent is an access path. The second restriction ensures that any substitution which grounds the antecedent of a rule also grounds its consequent.

If $\rho = r(t_1, \ldots, t_n) \leftarrow \alpha$ is an if-needed rule, we use the accessor functions: $Key(\rho) = r(t_1, \ldots, t_n)$, $Conseq(\rho) = r(t_1, \ldots, t_n)$, and $Ant(\rho) = \alpha$. Intuitively, the Key of a rule is the proposition the rule is indexed under in the knowledge-base.

Assume $r(t_1, \ldots, t_n)$ is a proposition and α is a non-empty list of propositions. $\alpha \rightarrow r(t_1, \ldots, t_n)$ is an *if-added rule* iff both of the following hold:

1. $\langle vars(head(\alpha)), \alpha \rangle$ is a path.
2. $vars(\{t_1, \ldots, t_n\}) \subset vars(\alpha)$.

[4] n place relations cause no problems in ALL. Intuitively, $r(t_1, \ldots, t_n)$ corresponds to putting the value $(t_2, \ldots, t_n)$ in the r slot of t_1.

As for if-needed rules, the first restriction ensures that when the rule is "fired" (i.e. when the head of the antecedent of the rule has been unified against a fact being added to the knowledge-base), the antecedent is an access path. The second restriction again ensures that any substitution which grounds the antecedent of a rule, also grounds its consequent.

If $\rho = \alpha \rightarrow r(t_1, \ldots, t_n)$ is an if-added rule, we again use the accessor functions: $Key(\rho) = head(\alpha)$, $Conseq(\rho) = r(t_1, \ldots, t_n)$, and $Ant(\rho) = \alpha$.

3.2.4 Knowledge-Bases

If S is a set then $s_1, \ldots, s_n$ is a *partitioning* of S iff:

- $(\forall i : 0 \leq i \leq n : s_i \subset S)$, and
- $(\bigcup i : 1 \leq i \leq n : s_i) = S$

A *Knowledge-Base*, K, is a six-tuple $\langle C, R, Nr, Ar, F, P\rangle$ where:

C	$=$	A set of constants.
R	$=$	A set of relations.
Nr	$=$	A set of if-needed rules such that $(\forall \rho : \rho \in Nr$: $constants(\rho) \subset C \wedge relations(\rho) \subset R)$.
Ar	$=$	A set of if-added rules such that $(\forall \rho : \rho \in Ar$: $constants(\rho) \subset C \wedge relations(\rho) \subset R)$.
F	$=$	A set of facts such that $(\forall f : f \in F$: $constants(f) \subset C \wedge relations(f) \subset R)$.
P	$=$	A partitioning of $C \times R$.

If $K = \langle C, R, Nr, Ar, F, P, A\rangle$ is a knowledge-base and α is a proposition, list of propositions or a rule, then α is *allowed* in K iff $constants(\alpha) \subset C \wedge relations(\alpha) \subset R$. Finally, the members of P are referred to as the *partitions* of K.

Unless otherwise specified a knowledge-base K_i should be understood to have components $\langle C, R, Nr, Ar_i, F_i, P\rangle$ (we subscript Ar and F because, as will be seen, they are the two components which change when operations are performed).

3.2.5 Operations and Formulas

If α is a path then $query(\alpha)$ is a *query*. If α is a primitive path then $query(\alpha)$ is a *primitive* query. If f is a fact then $assert(f)$ is an *assertion* (assertions of paths are not currently allowed). Any query or assertion is an *operation*, and any assertion or primitive query is a *primitive* operation. If $\mathcal{O} = query(\alpha)$ or $\mathcal{O} = assert(\alpha)$ is an operation and α is allowed in a knowledge-base K, then $\mathcal{O}$ is *allowed* in K. If an operation $\mathcal{O}$ is allowed in a knowledge-base K, then $\mathcal{O}(K)$ is an ALL *formula*.

3.3 Mapping ALL to Predicate Calculus

We define the semantics of ALL by mapping ALL knowledge-bases, assertions, and queries to (first order) predicate calculus. An alternative approach would be to define a model theory for ALL, in terms of which ALL is complete. This could be done, but we believe that (since the model theory of predicate calculus is well understood), mapping to predicate calculus and appropriately weakening the notion of completeness gives a more perspicuous picture of the semantics of ALL. Further, we believe that consistency and Socratic Completeness relative to predicate calculus (or perhaps an appropriate non-monotonic logic) are necessary properties for any knowledge representation system.

Mapping ALL to predicate calculus is straightforward. Propositions do not change at all. Paths become conjunctions. Rules become implications with all variables universally quantified. Knowledge-bases become the conjunction of their rules and facts. We notate the Predicate Calculus equivalent of an ALL object, a, by $\mathcal{PC}(a)$.

3.4 Knowledge Theory

The knowledge theory of ALL defines the values of ALL formulas by defining the action of ALL operations (i.e. queries and assertions). Intuitively, the assertion of a fact f, adds f to a knowledge-base and returns the resultant knowledge-base (i.e. the knowledge-base after f is added, all applicable if-added rules are applied, and all if-added rules are closed (see section 3.4.5)). A query of q returns the substitutions needed to make q true in the knowledge-base. It also returns a new knowledge-base (since processing the query may change the knowledge-base by invoking rules).

3.4.1 Substitutions

A *substitution* is a finite mapping from variables to terms:

$$\theta = \{v_1/t_1, \ldots, v_n/t_n\}$$

where the v_i $(1 \leq i \leq n)$ are distinct variables. If all t_i $(1 \leq i \leq n)$ are constants then θ is *ground*. Let the variables in the alphabet (see section 3.2.1) be V. A substitution θ is a *renaming* iff it is a bijection (i.e. a $1:1$ onto mapping) from V to V.

If e is an expression and θ is a substitution then $e\theta$ is the result of applying θ to e (simultaneously replacing each occurrence in e of the variables in $\{v_1, \ldots, v_n\}$ with the corresponding term).

If there exists a substitution ζ such that $\eta = \theta \circ \zeta$ then θ is *more general* than η. Intuitively, if θ is more general than η then θ does strictly 'less work' than η. A *unifier* of two primitive propositions q_1 and q_2, is a substitution θ such that $q_1\theta = q_2\theta$. The *most general unifier* of two primitive propositions q_1 and q_2 is a unifier θ of q_1 and q_2 such that for any other unifier η of q_1 and q_2, θ is more general than η. A unifier θ of q_1 and q_2 is *relevant* iff it binds only variables in q_1 and q_2, and it maps to only variables in q_1 and q_2. We notate the most general relevant unifier of q_1 and q_2 as $mgru(q_1, q_2)$.[5] It has been shown in [Apt, 88] that any propositions which are unifiable, have a most general relevant unifier.

3.4.2 The Partitions of ALL Operations

Intuitively, a partition of K corresponds to a part of the knowledge-base which is somehow semantically cohesive, and distinct from the rest of the knowledge-base. Facts and rules are often thought of as being 'in' partitions and operations are thought of as 'taking place' in subsets of $C \times R$ (unions of partitions). The intuition behind this comes from the frame view of ALL knowledge-bases. Recall that ALL constants can be thought of as frames, and relations as slots in these frames (e.g. the fact $r(c_1, c_2)$ is equivalent to having the value c_2 in the r slot of the frame c_1). Thus a pair $\langle r, c\rangle$ can be thought of as a particular slot in a particular frame in the knowledge-base. Recall that we refer to such a pair as a *frame-slot*. Partitions are thus sets of frame-slots. Further, note that any primitive path α (by the definition of a path) must reference exactly one frame-slot and thus can be said to be 'in' a partition. In fact, since partitions can overlap, it can be in several partitions and any operation on α is performed 'in' the subset of $C \times R$ formed by taking the union of these partitions. Intuitively, this union defines the rules which are available to the operation.

More formally, if K is a knowledge-base and $\alpha = r(c, t_1, \ldots, t_n)$ is a primitive path (i.e. c a constant and all t_i, $1 \leq i \leq n$, are terms) and p is a subset of $C \times R$ (e.g. a partition or the union of several partitions) then $\alpha \in p$ iff $\langle c, r\rangle \in p$. If $P = \{p_1, \ldots, p_n\}$[6] and $\mathcal{O} = query(\alpha)$ or $\mathcal{O} = assert(\alpha)$ then the union of partitions for $\mathcal{O}$ is:

$$par_K(\mathcal{O}) = (\cup i : 1 \leq i \leq n \wedge \alpha \in p_i : p_i)$$

[5] The use of most general relevant unifiers (instead of just most general unifiers) is necessary for technical reasons. The basic problem with most general unifiers is that one can compose a most general unifier with an arbitrary renaming and the result is still a most general unifier.

[6] Recall that a knowledge-base K is understood to have components $\langle C, R, Nr, Ar, F, P\rangle$. Thus P is the set of partitions of K.

3.4.3 The Domain and Range of All Operations

Any given sets C, R, Nr, P define a finite set of possible knowledge-bases KB (differing only in facts and if-added rules), and an infinite set of ground substitutions Θ (binding variables in the alphabet to constants in C).[7] For any operation $\mathcal{O}$ allowed in the knowledge-bases in KB (note that an operation allowed in any knowledge-base in KB is allowed in all knowledge-bases in KB):

$$\mathcal{O} : KB \longrightarrow 2^{\Theta} \times KB. \tag{1}$$

We notate these returned values with pairs: $\langle$ 'Set of Substitutions', 'Knowledge Base' $\rangle$. and use *sub* and *kb* as accessors on the first and second components respectively.

3.4.4 The Values of ALL Operations

Defining the values of ALL operations is primarily a matter of formalizing the action of forward and backward chaining rules. We use the following basic notation for knowledge-bases and substitutions:

If K_1 and K_2 are knowledge-bases (differing only in their facts and if-added rules), then:

$$K_1 \cup K_2 = \langle C, R, Nr, Ar_1 \ \cup \ Ar_2, F_1 \ \cup \ F_2, P\rangle.$$

If further, f is a fact allowed in K_1 then:

$$K_1 + f = \langle C, R, Nr, Ar_1, F_1 \ \cup \ \{f\}, P\rangle,$$

and $f \in K_1$ iff $f \in F_1$. If θ and η are substitutions then $\theta \circ \eta$ notates θ followed by η. If further, Θ_1 is a set of substitutions then $\eta \circ \Theta_1 = \{\eta \circ \theta_1 \mid \theta_1 \in \Theta_1\}$.

For a primitive operation $\mathcal{O}$, we define $\mathcal{O}_n(K, p)$ to be the result of the operation $\mathcal{O}$ on the knowledge-base K, in some subset of $C \times R$, p, with rule chaining cut off at depth n. A certain amount of technical care is required to formalize $\mathcal{O}_n$ (as it requires carefully defining forward and backward rule chaining). Details of the definition are given in the appendix.

We define $\mathcal{O}$ to be the union over all n of $\mathcal{O}_n$ (in the partition of $\mathcal{O}$). The idea here is to cause $\mathcal{O}$ to be a fixed-point of rule applications. We will eventually show (in section 4) that there is always an n after which increasing the depth of rule chaining does not affect the result. Thus, intuitively, $\mathcal{O}$ chains just far enough so that chaining any farther would have no affect (one advantage of this approach is that recursive rules (e.g. rules of form $q \leftarrow q$) do not cause problems in ALL (or its lisp implementation)).

It will be important that the knowledge-bases resulting from ALL operations are 'closed'. We guarantee this by applying the function *closure* to the knowledge-base which is to be returned. Closed knowledge-bases and *closure* are discussed in the next subsection (3.4.5).

We thus define:

$$\mathcal{O}(K) = closure((\bigcup n : n > 0 : \mathcal{O}_n(K, par_K(\mathcal{O}))))$$

The result of a non-primitive operation can then be defined in terms of the results of its constituent primitive operations. If $\mathcal{O}$ is non-primitive then it must be a query (see section 3.2.5). Assume $\mathcal{O} = query(\alpha)$ for some path α. Further, assume $q = head(\alpha)$, and $\alpha' = rest(\alpha)$, then:

If $sub(query(q)(K)) = \{\}$ (i.e. $query(q)$ 'failed'):

$$\mathcal{O}(K) \quad = \quad \langle\{\}, K\rangle$$

else ($query(q)$ succeeded so we branch on all resultant substitutions and union the results):

$$\begin{aligned}\mathcal{O}(K) = closure((\bigcup \theta \in sub(query(q)(K)) :: \langle\ &\theta \circ sub(query(\alpha'\theta)(K)),\\ &kb(query(q)(K)) \cup kb(query(\alpha'\theta)(K))\rangle\ ))\end{aligned} \tag{2}$$

Figure 1 shows a query on a simple knowledge-base.

[7] Technically, we should write $KB_{C,R,Nr,P}$ and $\Theta_{C,R,Nr,P}$, but we generally omit the subscripts since they are clear from context.

Assume K is a knowledge-base such that:

$$\begin{array}{rcl} C & = & \{c\} \\ R & = & \{r_1, r_2\} \\ Nr & = & \{r_1(c,x) \leftarrow r_2(c,x)\} \\ Ar & = & \{\} \\ F & = & \{r_2(c,c)\} \\ P & = & \{\{\langle c, r_1\rangle, \langle c, r_2\rangle\}\} \end{array}$$

Consider $query(r_1(c,x))(K)$ (where x is a variable). This is a primitive operation and $par_K(r_1(c,x)) = \{\langle c, r_1\rangle, \langle c, r_2\rangle\}$, so we must first compute:

$$query_0(r_1(c,x))(K, \{\langle c, r_1\rangle, \langle c, r_2\rangle\})$$

Rule back-chaining is cut off at depth zero so no rules apply and

$$query_0(r_1(c,x))(K, \{\langle c, r_1\rangle, \langle c, r_2\rangle\}) = \langle\{\}, K\rangle$$

(an empty list of substitutions is returned since there is no known value of x for which the query succeeds). However, when we calculate

$$query_1(r_1(c,x))(K, \{\langle c, r_1\rangle, \langle c, r_2\rangle\}),$$

the if-needed rule applies and

$$query_1(r_1(c,x))(K, \{\langle c, r_1\rangle, \langle c, r_2\rangle\}) = \langle\{\{x/c\}\}, K + r_1(c,c)\rangle$$

($\{x/c\}$ binds x to c). As n is increased further there are no other rules to apply so

$$query(r_1(c,x))(K) = \langle\{\{x/c\}\}, K + r_1(c,c)\rangle.$$

Figure 1: A query on a simple knowledge-base.

Assume K is a knowledge-base such that:

$$\begin{aligned} C &= \{c\} \\ R &= \{r_1, r_2, r_3\} \\ Nr &= \{\} \\ Ar &= \{r_1(x,x), r_2(x,x) \rightarrow r_3(x,x)\} \\ F &= \{r_1(c,c)\} \\ P &= \{\{\langle c, r_1\rangle, \langle c, r_2\rangle, \langle c, r_3\rangle\}\} \end{aligned}$$

Assume that we define $closure(K) = K$. Consider $assert(r_2(c,c))(K)$. After this operation both $r_1(c,c)$ and $r_2(c,c)$ will be facts in the knowledge-base. However, $query(r_3(c,c))(K)$ will fail. The rule $r_1(x,x), r_2(x,x) \rightarrow r_3(x,x)$ never applies because $r_1(c,c)$ was added before $r_2(c,c)$.

Figure 2: An example of if-added incompleteness.

3.4.5 The Problem of If-Added Incompleteness

In ALL, the application of an if-added rule is triggered by the assertion of a fact which matches the *Key* of the rule. One problem with this approach is that, if one is not careful, it can be the case that rules, whose antecedents are entailed by the knowledge-base, may never fire. Such a case is shown in the figure 2.

Our solution to this problem is to 'close' the if-added rules in the knowledge-base. Intuitively this means that for any fact f and any if-added rule $\alpha \rightarrow q$, if there is some substitution $\theta = mgru(f, head(\alpha))$, then we add to the knowledge-base the rule $rest(\alpha)\theta \rightarrow q\theta$. For a knowledge-base K, we notate by $closure(K)$, the knowledge-base formed by closing all the if-added rules in K. We also use the shorthand $closure(\langle K, \Lambda\rangle)$ for $\langle closure(K), \Lambda\rangle$.

Consider the example in figure 2. Note that the initial knowledge-base is not closed, since it includes the rule $r_1(x,x), r_2(x,x) \rightarrow r_3(x,x)$ and the fact $r_1(c,c)$, but not the rule $r_2(c,c) \rightarrow r_3(c,c)$. Further, if we consider $assert(r_2(c,c))(closure(K))$, then the resultant knowledge-base does include $r_3(c,c)$.

3.4.6 Implementation Note

There are three important differences between the formal definitions of ALL operations given here and our lisp implementation. First, in the formalism, when an operation branches (e.g. when several rules are applied or when the evaluation of a path branches on several possible instantiations of its variables) the branches are computed separately ('in parallel') and the results are unioned together. In our implementation the branches are computed serially (i.e. one rule is applied and then the next rule is applied in the resultant knowledge-base).[8] There are two advantages of the formalization presented here over a 'serial' formalization. First, the complexity analysis is considerably simplified, and second, the formalism given here would also apply to a parallel implementation of ALL.

The second difference is that our implementation supports limits on the accessibility of rules, which have been omitted (for simplicity) from the current formalization. In our implementation, a rule can be 'associated' with a frame f_0, and only accessed from frames known to be in an *isa* relation with f_0. Intuitively, such rules apply only to members of the set f_0.[9]

[8] Since ALL operations are monotonic, the serial implementation returns knowledge-bases which are supersets of those given by the formalism. Hence, our completeness results carry over to the serial case.

[9] Such access limitations can be formalized. The key idea is that when a rule associated with a set is translated to predicate calculus (see section 3.3) one must prepend an appropriate *isa* relation to its antecedent. It should be possible to show that the completeness results carry over (some care must be taken, however, when defining the *closure* of a knowledge-base with

Our implementation of *closure* demonstrates a useful application of rules associated with sets. One might worry that closing a knowledge-base might add a large number of if-added rules and thus slow the system (since we have to try to unify against all of them). However, we associate these if-added rules with very small sets (sets of size one) and they are thus ignored except when they are needed. Consider a rule added in the closure of a knowledge-base. It must be of form $rest(\alpha)\theta \rightarrow q\theta$. It follows from the definition of if-added rules that $rest(\alpha)\theta$ must be a path. This implies that $head(rest(\alpha))$ must be of form $r(c, t_1, \ldots, t_n)$. Thus (in our implementation) we simply associate this rule with a set consisting of the single element (frame) c (creating such a set if it does not exist).

The third difference is also related to our implementation of *closure*. Consider an assertion of a fact f. This assertion may trigger an if-added rule which asserts a fact f' into a partition p' which is disjoint from the partitions of f. In our implementation, f' is queued (in p') and any if-added rules for f' are not applied until 'attention' is drawn to p' (by some operation in p'). This ensures that the complexity of an operation is a function only of the rules in its partitions. We have not yet incorporated the notion of 'queuing' assertions into our formalism. Thus facts are closed with respect to all if-added rules in the knowledge-base, and the complexity of an operation (as will be seen in section 4) is a function of the set of all if-added rules in the knowledge-base.

3.5 Consistency

Consistency is often intuitively thought of as "You can't derive a contradiction." Consistency requires that the substitutions returned by a query must be semantic consequences of the old knowledge-base.[10] The requirements on the new knowledge base are more subtle. Consistency intuitively requires that propositions do not suddenly become true, or, in model theoretic terms, that models are not suddenly lost. Thus any model of the new knowledge-base must also be a model of the old knowledge base (and in an assertion a model of the formula being asserted):

Theorem 1 (Consistency of ALL) *For any knowledge-base K, any path α allowed in K, and any fact f allowed in K:*

1. $(\forall\theta \in \Theta : \theta \in sub(query(\alpha)(K)) : \mathcal{PC}(K) \models \mathcal{PC}(\alpha\theta))$
2. $\mathcal{PC}(K) \models \mathcal{PC}(kb(query(\alpha)(K)))$
3. $(\mathcal{PC}(K) \wedge \mathcal{PC}(f)) \models \mathcal{PC}(kb(assert(f)(K)))$

Proof (sketch): The proof of consistency is primarily a matter of carefully working through the definition of $\mathcal{O}$. One inducts on n to show, for all n, that $\mathcal{O}_n$ is consistent. One can then induct on the length of α to show that $\mathcal{O}$ is consistent.

3.6 Completeness

Completeness can be thought of as "Any true fact is derivable." Thus completeness requires that all substitutions which are semantic consequences of the old knowledge-base are returned by query. Completeness also requires that true facts do not suddenly become false. In model theoretic terms this means that we do not gain models. Thus any model of the old knowledge-base must also be a model of the new knowledge-base. Note that the requirements for completeness are essentially the requirements for consistency with their implications reversed:

respect to an *isa* relation).

[10] More precisely, for any query of a path α, if θ is returned then $\mathcal{PC}(\alpha\theta)$ must be a consequence of the knowledge-base.

Assume $K = \langle C, R, Nr, Ar, F, P\rangle$ is a knowledge-base such that:

$$\begin{aligned} C &= \{c\} \\ R &= \{r_1, r_2, r_3\} \\ Nr &= \{r_1(c,x) \leftarrow r_2(c,x)\} \\ Ar &= \{r_1(c,x) \rightarrow r_3(c,x)\} \\ F &= \{r_2(c,c)\} \\ P &= \{\{\langle c, r_1\rangle, \langle c, r_2\rangle, \langle c, r_3\rangle\}\} \end{aligned}$$

Consider $query(r_3(c,x))(K)$. This query must fail since it matches no facts in F and there are no *if-needed* rules for $r_3(c,x)$. But, any model of $\mathcal{PC}(K)$ must be a model of $\mathcal{PC}(r_3(c,c))$ (by the two rules and the fact that $r_2(c,c)$ is in F). Hence, inference in ALL is not complete.

Figure 3: A form of incompleteness in ALL.

Conjecture 1 (Completeness of ALL) *For any knowledge-base K, any path α allowed in K, and any fact f allowed in K, let Θ_α be the set of all ground substitutions binding all and only variables in α. Then:*

1. $(\forall \theta \in \Theta_\alpha : \mathcal{PC}(K) \models \mathcal{PC}(\alpha\theta) : \theta \in sub(query(\alpha)(K)))$
2. $\mathcal{PC}(kb(query(\alpha)(K)) \models \mathcal{PC}(K)$
3. $\mathcal{PC}(kb(assert(f)(K))) \models (\mathcal{PC}(K) \wedge \mathcal{PC}(f))$

Unfortunately, part one of this conjecture is false. In some cases, rules necessary for a query to succeed cannot be accessed. Two such cases are shown in the figures 3 and 4.

Notice, however, that in the example in figure 3:

$$query(r_3(c,x))(kb(query(r_1(c,x))(K)))$$

would succeed since $r_3(c,c)$ is added to $kb(query(r_1(c,x))(K))$ by the if-added rule $r_1(c,x) \rightarrow r_3(c,x)$. Similarly, in figure 4,

$$query(r_1(c,x))(kb(query(r_2(c,x))(K)))$$

succeeds. This suggests the idea behind *Socratic Completeness.* Informally, the Socratic Completeness Theorem says that for any query of α which 'should' succeed in a knowledge-base, there exists a series of preliminary queries, ?, after which a query of α will succeed. We also show a second type of weakened completeness result, *Partitional Completeness.* Partitional Completeness says that if all information needed to process a query can be located by the if-needed rules in the partitions of the query, then the query will succeed.

3.6.1 Socratic Completeness

To state the Socratic Completeness theorem we need a shorthand for the result of a series of queries:

If ? is a series of paths allowed in a knowledge-base K then let:

$$query(nil)(K) = K \quad (3)$$
$$query(?)(K) = query(head(?))(kb(query(rest(?)(K)))) \quad (4)$$

Assume $K = \langle C, R, Nr, Ar, F, P\rangle$ is a knowledge-base such that:

$$\begin{aligned}
C &= \{c\} \\
R &= \{r_1, r_2, r_3\} \\
Nr &= \{r_1(c,x) \leftarrow r_2(c,x), r_2(c,x) \leftarrow r_3(c,x)\} \\
Ar &= \{\} \\
F &= \{r_3(c,c)\} \\
P &= \{\{\langle c, r_1\rangle\}, \\
&\quad\ \{\langle c, r_2\rangle, \langle c, r_3\rangle\}\}
\end{aligned}$$

Consider $query(r_1(c,x))(K)$. This query must fail since the only rule for $r_1(c,x)$ depends on $r_2(c,x)$, which matches no facts in F and is not in $par_K(r_1(c,x))$ (so no rules for $r_2(c,x)$ fire). But, any model of $\mathcal{PC}(K)$ must be a model of $\mathcal{PC}(r_1(c,c))$ (by the two rules and the fact that $r_3(c,c)$ is in F).

Figure 4: Another form of incompleteness in ALL.

Theorem 2 (Socratic Completeness) *For any knowledge-base K, any path α allowed in K, and any fact f allowed in K, let Θ_α be the set of all ground substitutions binding all and only variables in α. Then:*

1. $(\forall \theta \in \Theta_\alpha : \mathcal{PC}(K) \models \mathcal{PC}(\alpha\theta)$

$$: (\exists ? : ?\ a\ series\ of\ paths\ allowed\ in\ K : \theta \in sub(query(\alpha)(kb(query(?)(K))))))$$

2. $\mathcal{PC}(kb(query(\alpha)(K))) \models \mathcal{PC}(K)$
3. $\mathcal{PC}(kb(assert(f)(K))) \models (\mathcal{PC}(K) \wedge \mathcal{PC}(f))$

Proof (sketch): Parts 2 and 3 follow relatively easily from the definitions of $\mathcal{O}$, and $\mathcal{PC}$. Part 1 is shown by induction on the length of α. The tricky part is the base case. We map K to an equivalent logic program $\mathcal{LP}(K)$. One can then show that for any rule in K which would apply on the next iteration of $T_{\mathcal{LP}(K)}$ (where T is the immediate consequence operator in logic programming — see [Crawford & Kuipers, 90] or [Apt, 88]) there exists a path in ALL the query of which will cause the rule to fire (this result would not hold for if-added rules if we did not close our knowledge-bases — see section 3.4.5). We know from the study of logic programming (see [Apt, 88]) that 'completeness' with respect to the immediate consequence operator is sufficient to guarantee completeness.

3.6.2 Partitional Completeness

Intuitively, Partitional Completeness says that if all information needed to prove a query is located 'close enough' to the query then the query will succeed. Formally 'close enough' will mean that the information is reachable using only if-needed rules in the partitions of the query. In order to state the Partitional Completeness Theorem, we first have to define which rules in the knowledge-base are considered 'part' of which partitions. A rule is considered a part of a partition if it could be triggered by queries to, and assertions into, the frame-slots in that partition. Counterintuitively, partitions limit access to *rules* not facts. When we speak of a fact as being in a partition, we mean that the fact is queried (or asserted) using the rules in that partition. Theoretically a rule could access facts in every partition of the knowledge-base; it is the use of access paths, not partitions, which limits access to facts.

If K is a knowledge-base, $p \subset C \times R$, and S is a set of rules from K, then $S\backslash_p$, the restriction of S to p, is given by:

$$\begin{aligned} S\backslash_p = \{\rho\theta \mid\ & \rho \in S \wedge Key(\rho) = r(t_1, \ldots, t_n) \wedge \\ & ((t_1\ a\ constant \wedge \theta = \emptyset \wedge \langle t_1, r\rangle \in p) \vee \\ & (t_1\ a\ variable \wedge (\exists c \in C : \langle c, r\rangle \in p : \theta = \{t_1/c\})))\}. \end{aligned}$$

Intuitively, $S\backslash_p$ is the restriction of S to only the rules in p. The knowledge-base K with its rules restricted to only the if-needed rules in $p \subset C \times R$ is given by:

$$K\backslash_p = \langle C, R, Nr\backslash_p, \emptyset, F, \{p\}\rangle. \tag{5}$$

Note that $K\backslash_p$ is never computed (in the definition of ALL formula or in our lisp implementation of ALL); It is only a formal object used to state the partitional completeness theorem.

Theorem 3 (Partitional Completeness) *For any knowledge-base K and any primitive path q allowed in K, let Θ_q be the set of all ground substitutions binding all and only variables in q, then:*

$$(\forall \theta \in \Theta_q : \mathcal{PC}(K\backslash_{par_K(q)}) \models \mathcal{PC}(q\theta) : \theta \in sub(query(q)(K)))$$

Proof (sketch): The proof of this theorem again relies on results from the study of logic programming. Let $ground(q)$ be the set of all variable free instantiations of q. Further, for any logic program pg, and any set of facts I, let:

$$\begin{aligned} T_{pg} \uparrow 0(I) &= I \\ T_{pg} \uparrow (n+1)(I) &= T_{pg}(T_{pg} \uparrow n(I)) \end{aligned}$$

The key lemma is that for any $p \subset C \times R$, and for all $n > 0$:

$$ground(q) \cap T_{\mathcal{LP}(K\backslash_p)} \uparrow n(\emptyset) \subset kb(query_n(q)(K, p))$$

Intuitively, this says that if a fact is an instantiation of q, and is in the knowledge-base produced by n iterations of the immediate consequence operator, then it is in the knowledge-base produced by $query_n(q)$. From this lemma the result again follows by the observation (from the study of logic programming, see [Apt, 88]) that 'completeness' with respect to the immediate consequence operator is sufficient to guarantee completeness.

4 Complexity

In this subsection we show that a primitive ALL operation can be computed in time polynomial in the size of the portion of the knowledge-base accessible to it. We focus on primitive operations since non-primitive operations are defined as sequences of primitive operations (equation 2).

To discuss the complexity of ALL it is useful to return to the view of an ALL knowledge-base as a collection of frames and slots. This view was introduced in section 1 and discussed further in section 3.4.2. The basic idea is that constants are thought of as *frames*, and relations are thought of as *slots*. The fact $r(c_1, c_2)$ is equivalent to putting c_2 in the r slot of the frame c_1. Recall that if c is a frame and r is a slot then we refer to the pair $\langle c, r\rangle$ as a *frame-slot*. Recall also that partitions are simply collections of frame-slots.

The definition of the accessible portion of a knowledge-base is fairly technical and is given in [Crawford & Kuipers, 90]. In this section we simply give the intuitions behind the definitions.

For a knowledge-base K, and $p \subset C \times R$ we first define $rules(p) = Ar \cup Nr\backslash_p$. Now, consider a primitive operation $\mathcal{O}$ allowed in a knowledge-base K, and assume $\mathcal{O} = query(q)$ or $\mathcal{O} = assert(q)$. We define the $reach_n(q, p)$ to include all frame-slots which can be accessed in the calculation of $\mathcal{O}_n$ in p, with rule backchaining cut off at depth n, and $change_n(q,p)$ to include all frame-slots which can be changed in the calculation of $\mathcal{O}_n$ in p (with rule backchaining cut off at depth n). We define $frames_n(q,p)$ to include all *frames* which $\mathcal{O}_n$ can access (with rule backchaining cut off at depth n). $frames_n(q,p)$ includes the frames appearing in frame-slots in $reach_n(q,p)$, plus the frames appearing explicitly in rules in $rules(p)$ or in q itself. Finally, we define $closure_{f\&r}(\mathcal{O})$ to include all facts and rules which could potentially be added to the knowledge-base by $\mathcal{O}$.

We define $ops_n(\mathcal{O})$ to include all operations that $\mathcal{O}_n$ 'potentially depends on'. These include all queries of frame-slots in $reach_n(q, par_K(\mathcal{O}))$ and all assertions (of frames in $frames_n(q, par_K(\mathcal{O}))$) into frame-slots in $change_n(q, par_K(\mathcal{O}))$ (some amount of care is required to prove that *ops* includes all the assertions and queries which $\mathcal{O}_n$ depends on — for details (and a formal definition of 'the set of all operations which $\mathcal{O}$ depends on') see [Crawford & Kuipers, 90]).

We then define:

$$reach(q,p) = (\bigcup n : n > 0 : reach_n(q,p)) \tag{6}$$

$$change(q,p) = (\bigcup n : n > 0 : change_n(q,p)) \tag{7}$$

$$frames(q,p) = (\bigcup n : n > 0 : frames_n(q,p)) \tag{8}$$

$$ops(\mathcal{O}) = (\bigcup n : n > 0 : ops_n(\mathcal{O})). \tag{9}$$

Figures 5 and 6 show *reach*, *change*, and *ops* for two simple queries.

Theorem 4 *If $\mathcal{O} = query(q)$ or $\mathcal{O} = assert(q)$ is a primitive operation allowed in a closed knowledge-base K then let:*

$$nops(\mathcal{O}) = \mid ops(\mathcal{O}) \mid \tag{10}$$

$$nfr(\mathcal{O}) = \mid frames(q, par_K(\mathcal{O})) \mid \tag{11}$$

$$nrules(\mathcal{O}) = \mid rules(par_K(\mathcal{O})) \mid \tag{12}$$

Finally, let ma be the maximum arity of any relation in R, $mvars(\mathcal{O})$ be the maximum number of variables in any rule in $rules(par_k(\mathcal{O}))$, and len be the maximum length of the antecedent of any rule in $rules(par_K(\mathcal{O}))$. $\mathcal{O}(K)$ can be computed in time of order:

$$len^5 \times nops(\mathcal{O})^2 \times nrules(\mathcal{O})^5 \times nfr(\mathcal{O})^{5mvars(\mathcal{O})+ma}.$$

Proof (sketch): $\mathcal{O}$ is defined as an infinite union of $\mathcal{O}_n$'s thus it is not obvious how it can be computed. However, one can show that if there is an n at which all the operations which $\mathcal{O}$ 'depends on' (i.e. all operations in $ops(\mathcal{O})$) return the same value they returned at $n-1$, then $\mathcal{O}(K) = \mathcal{O}_n(K, par_K(\mathcal{O}))$.

One can further show that such an n exists and can compute a bound for it. Consider the vector of all operations in $ops(\mathcal{O})$. There are:

$$nops(\mathcal{O})$$

such operations. As we increase n the knowledge-bases returned by these operations may not shrink. Further, if we ever reach a point where none of them grow then we can quit. Any ALL operation can only add a fact or complete an if-added rule (there are no operations, for example, which create an entirely new frame). The facts and rules which can be added must be in the set $closure_{f\&r}(\mathcal{O})$, and one can show that the size of this set is bounded by $len \times nrules(\mathcal{O}) \times nfr(\mathcal{O})^{mvars(\mathcal{O})}$. Each iteration may increase at worst one knowledge-base in the set of knowledge-bases returned by the operations in $ops(\mathcal{O})$. Thus if n is greater than or equal to

$$nops(\mathcal{O}) \times len \times nrules(\mathcal{O}) \times nfr(\mathcal{O})^{mvars(\mathcal{O})}$$

Recall the knowledge-base, K, from figure 1:

$$\begin{aligned}
C &= \{c\} \\
R &= \{r_1, r_2\} \\
Nr &= \{r_1(c,x) \leftarrow r_2(c,x)\} \\
Ar &= \{\} \\
F &= \{r_2(c,c)\} \\
P &= \{\{\langle c, r_1\rangle, \langle c, r_2\rangle\}\}
\end{aligned}$$

Consider $\mathcal{O} = query(r_1(c,x))$ (where x is a variable). Let $q = r_1(c,x)$ and $p = par_K(\mathcal{O})$, then:

$$\begin{aligned}
reach_0(q,p) &= \{\langle c, r_1\rangle\} \\
reach_1(q,p) &= \{\langle c, r_1\rangle, \langle c, r_2\rangle\} \\
reach_2(q,p) &= \{\langle c, r_1\rangle, \langle c, r_2\rangle\} \\
reach_n(q,p) &= \{\langle c, r_1\rangle, \langle c, r_2\rangle\} \\
reach(q,p) &= \{\langle c, r_1\rangle, \langle c, r_2\rangle\}
\end{aligned}$$

Thus:

$$\begin{aligned}
change(q,p) &= \{\langle c, r_1\rangle\} \\
ops(\mathcal{O}) &= \{query(r_1(c,x)), query(r_2(c,x)), query(r_1(c,c)), \\
&\quad query(r_2(c,c)), assert(r_1(c,c))\}
\end{aligned}$$

Figure 5: The accessible frame-slots and dependent operations for a simple query.

Recall the knowledge-base, K, from figure 4:

$$\begin{aligned}
C &= \{c\} \\
R &= \{r_1, r_2, r_3\} \\
Nr &= \{r_1(c,x) \leftarrow r_2(c,x), r_2(c,x) \leftarrow r_3(c,x)\} \\
Ar &= \{\} \\
F &= \{r_3(c,c)\} \\
P &= \{\{\langle c, r_1\rangle\}, \\
&\quad \{\langle c, r_2\rangle, \langle c, r_3\rangle\}\}
\end{aligned}$$

Consider $\mathcal{O} = query(r_1(c,x))$ (where x is a variable). Let $q = r_1(c,x)$ and $p = par_K(\mathcal{O})$, then:

$$\begin{aligned}
reach_0(q,p) &= \{\langle c, r_1\rangle\} \\
reach_1(q,p) &= \{\langle c, r_1\rangle, \langle c, r_2\rangle\} \\
reach_2(q,p) &= \{\langle c, r_1\rangle, \langle c, r_2\rangle\} \\
reach_n(q,p) &= \{\langle c, r_1\rangle, \langle c, r_2\rangle\} \\
reach(q,p) &= \{\langle c, r_1\rangle, \langle c, r_2\rangle\}
\end{aligned}$$

Thus:

$$\begin{aligned}
change(q,p) &= \{\langle c, r_1\rangle\} \\
ops(\mathcal{O}) &= \{query(r_1(c,x)), query(r_2(c,x)), query(r_1(c,c)), \\
&\quad query(r_2(c,c)), assert(r_1(c,c))\}
\end{aligned}$$

Figure 6: The accessible frame-slots and dependent operations for a query in a knowledge-base with two partitions.

then all knowledge-bases must be 'full'.

Thus it only remains to find the time to calculate the result of a primitive operation $\mathcal{O}_n$ given the results of all operations $\mathcal{O}'_{n-1}$. We may have to apply at most

$$nrules(\mathcal{O})$$

rules. Each rule may branch on all values in $frames(q, par_K(\mathcal{O}))$ for all variables. Thus we may have

$$nfr(\mathcal{O})^{mvars(\mathcal{O})}$$

branches (the results of which must be unioned together). Finding the result of each branch involves taking at most len unions and closures. There are thus order:

$$len$$

unions and closures per branch. Each union is done on a knowledge-base of form $K + S$ where S is of size $\mid closure_{f\&r}(\mathcal{O}) \mid$ or smaller. Thus each union can be done in time of order:

$$len \times nrules(\mathcal{O}) \times nfr(\mathcal{O})^{mvars(\mathcal{O})}$$

Finally, one can show that each closure can be computed in time:

$$len^2 \times nrules(\mathcal{O})^2 \times nfr(\mathcal{O})^{2mvars(\mathcal{O})+ma}$$

Multiplying these bounds together gives the time bound in the theorem.

5 Related Work

ALL draws from several diverse fields. We attempt only to sketch in general terms the fields from which it draws and discuss a few particularly relevant past approaches.

ALL draws from semantic networks [Brachman et al., 83, Bobrow & Winograd, 77, Findler, 79, Quillian, 67, Shapiro, 89, Vilain, 85] the intuition that retrieval and reasoning can be guided and limited by the structure of the network. This has long been a key intuition behind semantic networks: "...the knowledge required to perform an intellectual task generally lies in the semantic vicinity of the concepts involved in the task." [Schubert, 79]. In particular, ALL draws from semantic networks its frame based data structures [Minsky, 75] and the idea of access paths. Our use of access paths is closely related to previous work on path based inference. Path based inference can be traced back (at least) to [Raphael, 68] and later to [Schwarcz et al., 70] and [Shapiro & Woodmansee, 69]. A good discussion of path based and node based inference (both of which are partially subsumed by inference in ALL and would be totally subsumed if ALL supported full quantification — see section 6.4) is given in [Shapiro, 78].

One difference between ALL and much recent careful work on knowledge representation is that ALL (along with first-order logic and the original work on semantic networks) allows the knowledge-base designer to name the relations used in the knowledge-base. After Woods' influential "What's in a Link" paper [Woods, 75], many knowledge representation languages restricted the allowable relations to a small set which were given a precise syntax and semantics [Brachman 79, Shapiro, 89]. Our approach in ALL is to define our semantics by "borrowing" the model theory of predicate calculus (by mapping ALL knowledge-bases to statements in predicate calculus and proving consistency and weakened completeness results) and to allow relations to be given any names. The *meanings* of the relations are thus restricted only by the contents of the knowledge-base (as in predicate calculus).

ALL also differs from past formal work on semantic networks in that it uses a single general purpose retrieval/reasoning mechanism which is guided by the structure of the network. Past work has generally used the structure of the network only for special purpose reasoning (spreading activation, classification etc.), and has relied on a first-order logic theorem prover [Brachman et al., 83, Schubert et al., 83] or a weaker deduction system [Levesque, 84, Patel-Schneider, 85, Vilain, 85] for general reasoning.

A notable exception to this generalization is the recent work of Haan and Schubert [Schubert, 79, Haan & Schubert, 86]. ALL and the networks of Schubert share several features including the use of access limitations to guide reasoning. The most obvious way to use the structure of a semantic network to limit access would be to perform deduction with facts not more than a few (say maybe two) nodes away in the network. The problem with this strategy is that some nodes (e.g. the node for your spouse) may have a large number of links, many of which are irrelevant to the problem at hand. The solution used in ECOSYSTEM is to maintain a taxonomy of knowledge and use this taxonomy to guide reasoning [Haan & Schubert, 86]. The difference in ALL is that access is limited to known *access paths*, which access facts many nodes away in the network, but do so in a controlled fashion. Thus in ALL it is the structure of the knowledge itself (or more specifically the structure of the access paths in the rules) which controls access and reasoning.

Another relevant line of research is the work on *vivid* knowledge-bases [Etherington et al., 89]. A vivid knowledge-base "... trades accuracy for speed ..." [Etherington et al., 89] by constructing a complete database of ground facts, from facts that may be presented in a more expressive language. This approach has some of the same goals as ALL — particularly in the area of efficiency — but takes a much different approach and makes different trade-offs. At a very high level, the principal differences are:

- ALL represents all the knowledge that has been asserted (though not all of it may be accessible at a given time) while a vivid knowledge-base is an approximation of the asserted knowledge (thus weakened completeness results such as Socratic Completeness do not hold for vivid reasoning).
- To obtain increased efficiency, ALL trades completeness for speed while a vivid approach trades both consistency and completeness for speed.

The design of the inference mechanism in ALL has been heavily influenced by logic programming. In fact any function-free logic program (without negation) can be written in ALL. Further, the notation and results from the proof of the completeness of logic programming [Apt, 88, Lloyd, 84] have been used extensively in the completeness proofs for ALL. In a sense our use of access-paths is a strategy for ordering conjunctive queries and as such is related to the more elaborate approach given in [Smith & Genesereth, 85]. In fact, if one follows a discipline of avoiding frame-slots containing a large number of frames[11] then the use of access-paths enforces an ordering on conjunctive queries much like that discussed in [Smith & Genesereth, 85].

6 Conclusion

Given a knowledge representation system with a model theory and a knowledge-base, one may divide the set of all possible queries in several ways. For example, one can distinguish the queries which succeed from those which fail. If this set is exactly equal to the set of all queries which are model theoretic consequences of the knowledge-base then the knowledge representation system is consistent and complete. In ALL we divide the set of all queries into three sets:

- Those which succeed immediately.
- Those which will succeed after some appropriate series of preliminary queries.
- Those which will never succeed (without additional assertions).

[11] E.g. if the set *things* is very large, then one would like to avoid filling the slot *members* with all the members of *things* — rather, the preferred representation in ALL for "*f* is a *thing*" would be to put *things* in the *isa* slot of *f*.

Socratic Completeness then gives a precise characterization of the second and third sets — the second set is equal to the set of all queries which are model theoretic consequences of the knowledge-base, and the third set is equal to the set of all queries which are not model theoretic consequences. Partitional Completeness gives a partial characterization of the first set — a query will succeed immediately if all the information needed to prove it is located 'close enough' to the query in the knowledge-base.

6.1 About Socratic Completeness

One of the questions asked about our work[12] was "What good is Socratic Completeness when Socrates is dead?" Meaning that the hard part of reasoning has simply been pushed off to the problem of posing the right questions. The first answer to this question is that in a system with the expressive power of first-order logic, the incomputability never goes away; our approach decomposes the problem of reasoning into two parts: the (tractable) problem of computing the results of queries and the (intractable) problem of deciding what queries to ask. Past work has made other divisions — e.g. the T-box and A-box of [Brachman et al., 83]. The second answer is that our goal is to develop a knowledge representation system with *understandable* inferential power. To this end, Socratic Completeness is a necessary (but not yet sufficient) property. Socratic Completeness guarantees that there is some hope of eventually finding the right questions (by guaranteeing that the questions exist).

These answers suggest two directions for future work: first, encoding (in the knowledge-base) commonsense knowledge about what general types of queries are useful for solving common types of complex reasoning problems; and second, the identification of other weakened completeness properties which, like Partitional Completeness, define what queries should immediately succeed.

Socratic Completeness is also a step toward a formal specification of what inferential power a knowledge representation system should provide. Due to its intractability, full logical completeness is too strong a specification, but provides an upper bound in the search for an appropriate specification. Socratic Completeness is a fairly weak specification but is arguably a necessary property. Thus it provides a lower bound on the space of appropriate specifications.

6.2 About Partitional Completeness

Partitioning the knowledge-base is not a new idea. Minsky's original proposal ([Minsky, 75]) for frames envisioned a structure on the knowledge-base consisting of groups of related frames. Hayes' Naive Physics Manifesto ([Hayes, 85]) also viewed commonsense knowledge as consisting of clusters of closely related concepts, loosely related to each other. Closer to the implementation level, blackboard architectures ([Hayes-Roth, 85]) also group inference methods into weakly interacting partitions. While these intuitions about the modularity of knowledge are persuasive, it must be admitted that it has not yet been empirically demonstrated that the contents of large-scale commonsense knowledge-bases divide naturally into partitions.

If we accept the intuition that knowledge can be meaningfully divided into partitions (or perhaps before we commit to accepting this intuition), we would like to know what effect partitions have on reasoning. Intuitively, one would expect that the rules in the partition of a query would somehow be more easily accessible to the query. The Partitional Completeness Theorem gives a partial formalization of this intuition, by saying that if a query is a logical consequence of the if-needed rules in its partitions then the query will succeed immediately. The theorem also gives us an empirical way to test a partitioning of a large knowledge-base — if simple queries depend on many rules in other partitions (and thus require many preliminary queries before they succeed) then the partitions are not well chosen.

[12]By Rich Thomason at the 1989 Workshop on Formal Aspects of Semantic Networks.

6.3 About the Complexity Results

The expression given in theorem 4 for the complexity of inference in ALL involves too many variables to be easily comprehended. In a 'typical' knowledge-base one might expect that:

$$nrules(\mathcal{O}) \approx nfr(\mathcal{O}) \tag{13}$$
$$nops(\mathcal{O}) \approx nfr(\mathcal{O})^{ma} \tag{14}$$

Let $n = nfr(\mathcal{O})$. If we further assume that len is small, then ALL operations can be computed in time of order:

$$n^{3ma+5+5mvars(\mathcal{O})}$$

Certainly a tighter bound could be computed (with somewhat more work). In general we have found that our examples run much faster than the worst case bound. However, the complexity analysis is still a useful exercise. Our implementation of ALL originally used an algorithm which was exponential in the worst case. Replacing it with an algorithm similar to the one given here greatly improved its run time. Further, the complexity result gives some guidance in the design of knowledge-bases. For example, it suggests that while the length of rules makes little difference, rules with many variables should be avoided.

6.4 Implementing ALL

Our lisp implementation of ALL is considerably more powerful than the formalism presented in this paper and can express definite descriptions, full negation, some types of defaults, and quantification. Beyond simple examples of forward and backward-chaining we have investigated some standard examples of default inheritance (essentially implementing the inferential distance rule of Touretzky [1986]), the Yale shooting problem [Hanks & McDermott, 86], several examples which involve reasoning about sets of similar objects, and some examples of reasoning from quantified information (e.g. From "Every man loves a woman" conclude that there must be some woman that *George* loves).. We have also solved the bank problem mentioned in the introduction. Our most recent work involves the use of ALL to express the ideas of Qualitative Process Theory [Forbus, 84]. The result [Crawford, Farquhar, & Kuipers, 90] is a system which compiles qualitative descriptions of physical situations into qualitative differential equations which can be simulated by QSIM [Kuipers, 86].[13]

Ultimately we are working towards a formal theory which has the expressive power of predicate calculus, and is consistent and Socratically Complete, but is still computationally tractable. It is straightforward to add to ALL the ability to express full classic negation (i.e. not negation by failure), but then inference in ALL (using rules alone) is no longer Socratically Complete. We are currently working to formalize in ALL the notion of reasoning by *Reductio Ad Absurdum* (used in our implementation). Reasoning by *Reductio Ad Absurdum* involves adding assumptions to the knowledge-base and then reasoning about their consequences (and if the consequences of an assumption include 'false' concluding the negation of the assumption). We believe that such a mechanism will allow Socratically Complete reasoning in the presence of classic negation. Further, the queries determine what assumptions are made, so the complexity of ALL should still be polynomial (though some hard problems may require an exponential number of preliminary queries).

There is also no way to express existential quantification in our current formalism. We have incorporated definite descriptions, which define a type of existential quantification, into the implementation of ALL, but their formalization is not straight-forward (as they do not seem to translate naturally into predicate calculus). Our most recent work has been on adding, to the implementation, the ability to represent, and do some kinds of reasoning with, arbitrarily nested quantified expressions. Our approach to nested quantification is based on the idea of arbitrary objects [Fine, 85]. One may, for example, reason about a large class of objects by reasoning about an 'arbitrary object' having the properties common to all objects in the group. Future papers will discuss this work in more detail.

[13]This is joint work with Adam Farquhar.

7 Acknowledgments

The authors would like to thank Len Schubert, Rich Thomason, and Charles Petrie for careful readings and useful comments on various versions and drafts of this paper.

A Appendix – Formal Definition of $\mathcal{O}_n$.

In this appendix we give the formal definition of $\mathcal{O}_n$. This amounts to defining with great care the familiar behavior of forward and backward chaining rules in a knowledge-base. This is a non-trivial exercise, but it is necessary in order to carefully prove the theorems. Further, a careful formulation of forward and backward chaining reveals at least one interesting and unexpected problem — the problem of if-added incompleteness discussed in section 3.4.5.

Formally, for any n, and any operation $\mathcal{O}$ allowed in the knowledge-bases of KB:

$$\mathcal{O}_n : KB \times 2^{C \times R} \Longrightarrow 2^{\Theta} \times KB. \tag{15}$$

We define $\mathcal{O}_n$ in the following 3 cases. In all cases assume K is a knowledge-base, α a (non-empty) path allowed in K, q a primitive path allowed in K, f a fact allowed in K, and p a subset of $C \times R$. We use the shorthand:

$$lookup(q)(K) = \{\theta \in \Theta | (\exists f \in K :: \theta = mgru(q, f))\}. \tag{16}$$

Case 1: Base case: $\mathcal{O}$ is a primitive operation, and $\mathcal{O} \notin p$ or $n = 0$.

If $\mathcal{O} = query(q)$ then

$$\mathcal{O}_n(K, p) = \langle lookup(q)(K), K \rangle \tag{17}$$

else $\mathcal{O} = assert(f)$ and

$$\mathcal{O}_n(K, p) = \langle \{\{\}\}, closure(K + f) \rangle. \tag{18}$$

Case 2: $\mathcal{O}$ is a primitive operation, $n > 0$, and $\mathcal{O} \in p$.

First we find the rules which apply. Let η be a renaming which maps variables in rules in Nr to variables not used in $\mathcal{O}$ or K (this must be possible since the alphabet contains a countably infinite number of variables).

If $\mathcal{O} = query(q)$ then:

$$R = \{\rho\eta\theta \mid \rho \in Nr \wedge \theta = mgru(Key(\rho)\eta, q)\} \tag{19}$$

Else, $\mathcal{O} = assert(f)$ and:

$$R = \{\rho\theta \mid \rho \in Ar \wedge \theta = mgru(Key(\rho), f)\} \tag{20}$$

If $R = \emptyset$ then let:

$$K' = kb(\mathcal{O}_{n-1}(K, p)) \tag{21}$$

Otherwise, we apply the rules and union the results. Applying a rule consists of querying its antecedent and then asserting its consequent. The consequent is asserted with all substitutions under which the antecedent succeeds.

If $\mathcal{O} = query(q)$ then:

$$\begin{aligned} K' = closure(\bigcup \rho \in R :: kb(query_{n-1}(Ant(\rho))(K, p)) \cup \\ (\bigcup \theta \in sub(query_{n-1}(Ant(\rho))(K, p)) :: \\ kb(assert_{n-1}(Conseq(\rho\theta))(K, p)))) \end{aligned} \tag{22}$$

$$\mathcal{O}_n(K,p) = \langle lookup(q)(K'), K' \rangle \tag{23}$$

Else, $\mathcal{O} = assert(f)$ and:

$$\begin{aligned} K' = closure(\bigcup \rho \in R :: kb(query_{n-1}(Ant(\rho))(K+f,p)) \cup \\ (\bigcup \theta \in sub(query_{n-1}(Ant(\rho))(K+f,p)) :: \\ kb(assert_{n-1}(Conseq(\rho\theta))(K+f,p)))) \end{aligned} \tag{24}$$

$$\mathcal{O}_n(K,p) = \langle \{\{\}\}, K' \rangle \tag{25}$$

Case 3: Non-primitive queries. $\mathcal{O} = query(\alpha)$ where α a path of length greater than 1. Assume $head(\alpha) = q$, and $rest(\alpha) = \alpha'$. If $sub(query_n(q)(K,p)) = \{\}$ (i.e. $query_n(q)$ 'failed'),

$$\mathcal{O}_n(K,p) = \langle \{\}, K \rangle \tag{26}$$

else ($query_n(q)$ succeeded so we branch on all resultant substitutions and union the results):

$$\begin{aligned} \mathcal{O}_n(K,p) = closure((\bigcup \theta \in sub(query_n(q)(K,p)) \\ :: \langle \theta \circ sub(query_n(\alpha'\theta)(K,p)), \\ kb(query_n(q)(K,p)) \cup kb(query_n(\alpha'\theta)(K,p)) \rangle)) \end{aligned} \tag{27}$$

References

[Allen, 84] Allen, J. F., Giuliano, M., and Frisch A. M. (1984). *The HORNE Reasoning System*, TR 126, Computer Science Department, University of Rochester, Rochester NY.

[Apt, 88] Apt, Krzysztof, R. (1988). Introduction to logic programming (revised and extended version), Technical Report TR-87-35, Department of Computer Science, University of Texas at Austin, Austin Texas. To appear in *Handbook of Theoretical Computer Science*, (J. van Leeuwen, Managing Editor), North-Holland.

[Brachman, 77] Brachman, R. J. (1977). What's in a concept: structural foundations for semantic networks. *Int. J. Man-Machine Studies 9*: 127-152.

[Brachman 79] Brachman, R. J. (1979). On the epistemological status of semantic networks. In [Findler, 79]. (Reprinted in [Brachman & Levesque, 85], pp. 191-215.)

[Brachman et al., 83] Brachman, R.J., Fikes,R. E., and Levesque H. J. (1983). Krypton: A functional approach to knowledge representation. FLAIR Technical Report No. 16, Fairchild Laboratory for Artificial Intelligence Research, Palo Alto, CA, May, 1983 (revised version in IEEE Computer **16**(10), 1983, 67-73). (Reprinted in [Brachman & Levesque, 85], pp. 412-429.)

[Brachman & Levesque, 85] Brachman, Ronald J. and Levesque, Hector, J. *Readings in Knowledge Representation*, Morgan Kaufmann, Los Altos, Cal., 1985.

[Bobrow & Winograd, 77] Bobrow, Daniel G., and Winograd, Terry (1977). An overview of KRL, a knowledge representation language. *Cognitive Science* **1**(1), 3-46. (Reprinted in [Brachman & Levesque, 85], pp. 263-285.)

[Boolos & Jeffrey, 80] Boolos, George S., and Jeffrey, Richard C., *Computability and Logic*, Cambridge University Press, New York, 1980.

[Crawford & Kuipers, 89] Crawford, J. M., and Kuipers, B. (1989). Towards a theory of access-limited logic for knowledge representation. *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*, Morgan Kaufmann, Los Altos, California.

[Crawford & Kuipers, 90] Crawford, J. M., and Kuipers, B. Towards a Formalization of Access Limited Logic. Technical Report AI TR-90-133, University of Texas at Austin, 1990.

[Crawford, Farquhar, & Kuipers, 90] Crawford, J., Farquhar, A., and Kuipers, B. (1990). QPC: A Compiler from Physical Models into Qualitative Differential Equations. *AAAI-90.*

[Etherington et al., 89] Etherington, David W., Borgida, Alex, Brachman, Ronald J., and Kautz, Henry (1989). Vivid knowledge and tractable reasoning: preliminary report. *IJCAI-89* pp. 1146-1152.

[Fagin & Halpern, 88] Ronald Fagin and Joseph Y. Halpern. (1988). Belief, awareness, and limited reasoning. *Artificial Intelligence*: **34**: 39-76.

[Findler, 79] Findler, N.V., *Associative Networks: Representation and Use of Knowledge by Computer*, Academic Press, New York, 1979.

[Fine, 85] Fine, Kit, *Reasoning With Arbitrary Objects*, Aristotelian Society Series, Volume 3, Basil Blackwell, Oxford, 1985.

[Forbus, 84] K. D. Forbus. (1984). Qualitative process theory. *Artificial Intelligence*, 24:85–168.

[Haan & Schubert, 86] Haan, J., and Schubert, L. K. (1986). Inference in a topically organized semantic net. *AAAI-86*, pp. 334-338.

[Hanks & McDermott, 86] Hanks, Steve, and McDermott, Drew (1986). Default reasoning, non-monotonic logics, and the frame problem. *AAAI-86*, pp. 328-333.

[Hayes, 85] Hayes, Patrick J. (1985). The second naive physics manifesto. In *Formal Theories of the Commonsense World*, ed. Hobbs, Jerry R. and Moore, Robert C., Ablex Publishing Co., New Jersey, pp. 18-30.

[Hayes, 79] Hayes, Patrick J. (1979). The logic of frames. In *Frame Conceptions and Text Understanding*, ed. D. Metzing, Walter de Gruyter and Co., Berlin, pp. 46-61. (Reprinted in [Brachman & Levesque, 85], pp. 288-295.)

[Hayes-Roth, 85] Hayes-Roth, B. (1985). A blackboard architecture for control. *Artificial Intelligence Journal*, 26:251-321.

[Hintikka, 62] Hintikka, J. *Knowledge and Belief.* Cornell University Press, Ithaca, NY, 1962.

[Kay, 73] Kay, M. (1973). The MIND system. In *Natural Language Processing*, ed. R. Rustin. Algorithmics Press, New York.

[Kuipers, 78] Kuipers, B. J. (1978). Modeling spatial knowledge. *Cognitive Science* **2**: 129-153.

[Kuipers, 79] Kuipers, B. J. (1979). On representing commonsense knowledge. In [Findler, 79].

[Kuipers, 86] B. J. Kuipers. (1986). Qualitative simulation. *Artificial Intelligence*, 29:289–338.

[Lenat et al., 86] Doug Lenat, Mayank Prakash, and Mary Shepard. (1986). CYC: Using common sense knowledge to overcome brittleness and knowledge acquisition bottlenecks. *AI Magazine* **VI**(4), Winter, 1986.

[Levesque, 86] Levesque, H. J. (1986). Knowledge representation and reasoning. In *Ann. Rev. Comput. Sci.* 1:255-87. Annual Reviews Inc, Palo Alto, California.

[Levesque, 84] Levesque, H.J. (1984). A logic of implicit and explicit belief, *AAAI-84*, pp. 198-202.

[Lloyd, 84] Lloyd, J.W. *Foundations of Logic Programming*, Springer-Verlag, New York, 1984.

[McCarthy, 87] McCarthy, J. (1987). Generality in artificial intelligence. *CACM* **30:**1030-1035.

[Minsky, 75] Minsky, Marvin. (1975). A framework for representing knowledge. In *The Psychology of Computer Vision*, ed. P.H. Winston, McGraw-Hill, New York. (A later version reprinted in [Brachman & Levesque, 85], pp. 246-262.)

[Moore, 79] Moore, R. (1979). Reasoning about Knowledge and Action, MIT PhD thesis.

[Patel-Schneider, 85] Patel-Schneider, P. (1985). A decidable first-order logic for knowledge representation. *IJCAI-85.*

[Palamidessi, 89] Palamidessi, Catuscia. (1989). Algebraic Properties of Idempotent Substitutions. University of Pisa, manuscript.

[Powers, 78] Powers, L. H. (1978). Knowledge by deduction. The Philosophical Review, LXXXVII, No. 3, pp. 337-371.

[Quillian, 67] Quillian, M. Ross. (1967). Word concepts: A theory and simulation of some basic semantic capabilities. *Behavioral Science* **12**, 410-430. (Reprinted in [Brachman & Levesque, 85], pp. 98-118.)

[Raphael, 68] Raphael, B. (1968). SIR: semantic information retrieval. In *Semantic Information Processing*, ed. M. Minsky, MIT Press, Cambridge, MA., pp. 33-145.

[Schubert, 79] Schubert, L.K., R.G. Goebel, and N.J. Cercone, (1979). The structure and organization of a semantic net for comprehension and inference. In [Findler, 79], pp. 121-175.

[Schubert et al., 83] Schubert, L.K., Papalaskaris, M.A., and Taugher, J. (1983). Determining type, part, color, and time relationships. *IEEE Computer*, **16**(10), 53-60.

[Schubert & Hwang] Schubert, L.K., and Hwang C.H. (1989). An Episodic Knowledge Representation for Narrative Texts. *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*, Morgan Kaufmann, Los Altos, California.

[Schwarcz et al., 70] Schwarcz, R.M., Burger, J.F., and Simmons, R.F. (1970). A deductive question-answerer for natural language inference. *CACM 13*, **3** (March 1970), 167-183.

[Shapiro, 78] Shapiro, S.C. (1978). Path-based and node-based inference in semantic networks. *Tinlap-2: Theoretical Issues in Natural Languages Processing*, ed. D. Waltz, ACM, New York, 219-225.

[Shapiro, 89] Shapiro, S.C. and the SNePS Implementation Group (1989). SNePS-2 User's Manual. Department of Computer Science, SUNY at Buffalo, 31pp.

[Shapiro & Woodmansee, 69] Shapiro, S.C. and Woodmansee, G.H. (1969). A net structure based relational question answerer: description and examples. *IJCAI-69*, The MITRE Corp., Bedford, MA., pp. 325-346.

[Smith & Genesereth, 85] Smith, David E., and Genesereth, Michael R. (1985). Ordering conjunctive queries. *Artificial Intelligence*, **26:** 171 - 215.

[Touretzky, 86] Touretzky, David S. *The Mathematics of Inheritance Systems*, Morgan Kaufmann, Los Altos, California, 1986.

[Treitel & Genesereth, 87] Treitel, Richard, and Genesereth, Michael R. (1987). Choosing directions for rules. *Journal of Automated Reasoning* **3:** 395-431.

[Vilain, 85] Vilain, M. (1985). The restricted language architecture of a hybrid representation system. *IJCAI-85*, pp. 547-551.

[Woods, 75] Woods, W. (1975). What's in a link: foundations for semantic networks. In *Representation and Understanding: Studies in Cognitive Science* , ed. Bobrow, D. and Collins, A., Academic, New York, pp. 35-82.

[Wylie, 57] Wylie, C.R., Jr. *101 Puzzles in Thought & Logic*, Dover Publications Inc, Mineola, New York, 1957.