In *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning (KR-89)*
Los Altos, CA: Morgan Kaufmann, 1989.

Towards a Theory of Access-Limited Logic for Knowledge Representation*

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

Abstract

One of the fundamental problems in the theory of knowledge representation is the difficulty of achieving both logical coherence and computational tractability. We present steps toward a theory of access-limited logic, in which access to assertions in the knowledge-base is constrained by semantic network style locality relations. 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') and is complete in some important special cases.

1 Introduction

Access-Limited Logic (ALL) is a logic for knowledge representation which utilizes semantic network style access limitations to guarantee computational tractability, even in very large knowledge-bases. Previous work has used the access limitations inherent in semantic networks for special purpose reasoning; in ALL these limitations form an integral part of the logic itself. A semantics for ALL has been defined by mapping queries, assertions and knowledge-bases to predicate calculus, and in terms of this mapping, consistency and weakened completeness results have been proven.

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. Thus a knowledge representation system, which does not give up expressive power, must use a weak inference system with an incomplete set of deduction rules or accept artificial resource limits (e.g. bounds on the number of applications of *modus ponens*). However, these approaches tend to be difficult to describe semantically and tend to place unnatural limits on an agent's reasoning ability [Levesque, 1986].

Our primary interest is the development of a system for the representation of commonsense knowledge. People seem to be able to reason efficiently with a very large commonsense knowledge-base. One reason for this is that when solving a given problem they only make use of the limited subset of their knowledge which is relevant to the problem.

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. Thus in a frame-based system it is natural to define the frames *directly accessible* from the frame a as those which appear in slots of a[1]. Extending this idea, one may define an *access path*, in a network of frames, as a series of frames each directly accessible from its predecessor. It proves useful to generalize this definition and allow access paths to branch on all values found in a given slot. 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. For example, "John's parent's sister" can be expressed in ALL as the path:

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

*This work has taken place in the Qualitative Reasoning Group at the Artificial Intelligence Laboratory, UT-Austin. Research of the Qualitative Reasoning Group is supported, in part, by NSF through grant IRI-8602665, and by NASA through grants NAG 2-507 and NAG 9-200.

†Supported in part by a fellowship from GTE.

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

This defines an access path from the frame for *John* to the frames for *John*'s parents (found by looking in the parent slot of the frame for *John*), 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 the 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 note that 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.

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.

However, 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, any concept in the knowledge-base is potentially reachable; A string of queries, while conveying no new information, can move the focus of attention around to invoke the rules of the system in any order. Thus, access-limited logic has a property we call *Socratic Completeness* [2] — 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 query succeeds. Further, ALL is *Partitionally Complete* — if the rules needed to derive a proposition are in the same partition as the proposition and the proposition can be proven using only backward-chaining rules then a query of the proposition succeeds.

The logical properties of ALL are stated more carefully in the next section. Section 3 examines the complexity of inference in ALL, section 4 presents a simple example from our implementation of ALL, section 5 discusses related work, and section 6 overviews our current plans for future work.

2 The Logical Coherence of ALL.

'Logical coherence' is an informally defined collection of desirable formal properties. We have proven that 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 in the last subsection of this section.

The rest of this section sketches the formal development of ALL. The full account can be found in [Crawford and Kuipers, 1989].

2.1 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:

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

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

If α is a list then:

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

[2] The idea of Socratic Completeness was invented independently in [Powers, 1987] where it is referred to as Socratic Adequacy.

2.2 Syntax of ALL

We now build up the syntax of ALL. First the alphabet of an ALL is defined and then terms, propositions, access paths, rules, knowledge-bases, and finally ALL formula are defined.

2.2.1 Alphabets, Terms and Propositions

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

2.2.2 Access Paths

An *access path* (or simply a *path*) is a pair $\langle V, \alpha \rangle$ such that: V is a set of variables, and α is a list of propositions in which the first term of each proposition is either a constant, a member of V, or has appeared previously in α (this can be made precise by a simple recursive definition). If $V = \{\}$ then we omit it and say α is an access path. A path of length one is a *primitive* path.

2.2.3 Rules

$Conseq \leftarrow Ant$ is an *if-needed rule* iff:

- $Key = r(t_1, \ldots, t_n)$[3] is a proposition,
- $Conseq = Key$,
- Ant is a list of propositions,
- Either t_1 is a constant and Ant is a path, or t_1 is a variable and $\langle \{t_1\}, Ant \rangle$ is a path, and
- $vars(Conseq) \subset vars(Ant)$.

$Ant \rightarrow Conseq$ is an *if-added rule* iff:

- Key and $Conseq$ are propositions.
- Ant is a list of propositions such that $head(Ant) = Key$,
- $\langle vars(Key), Ant \rangle$ is a path, and
- $vars(Conseq) \subset vars(Ant)$.

For any rule ρ: $Key(\rho)$, $Conseq(\rho)$, and $Ant(\rho)$ access its respective components.

[3] Intuitively, the Key of a rule is the proposition that the rule is indexed under in the knowledge-base.

2.2.4 Knowledge-Bases

A *Knowledge-Base*, K, is a seven-tuple

$$\langle C, R, Nr, Ar, F, P, A \rangle.$$

The definition of a knowledge-base is given in figure 1. 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

$$constants(\alpha) \subset C \wedge relations(\alpha) \subset R.$$

2.2.5 Operations and Formula

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*. Any query or assertion is an *operation*. Any primitive query or assertion is a *primitive* operation. If $\mathcal{O} = query(\alpha)$ or $\mathcal{O} = assert(\alpha)$ is an operations 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*.

2.3 Knowledge Theory

In this subsection we sketch the knowledge theory of ALL. The knowledge theory of ALL defines the value of ALL formula 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 and all applicable if-added rules are applied). A query of q, returns the substitutions needed to make q true in the knowledge-base, and a new knowledge-base (since processing the query may change the knowledge-base by invoking rules).

2.3.1 The Domain and Range of ALL Operations

Any given sets C, R, Nr, Ar, P and function A, define a finite set of possible knowledge-bases (differing only in facts) KB and an infinite set of ground substitutions Θ (binding variables in the alphabet to constants in C). For this subsection fix the sets C, R, Nr, Ar, P, and the function A. Then, 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):

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

We notate these returned values with pairs: $\langle <$ set of substitutions $>, <$ knowledge-base $> \rangle$. and use kb and sub as accessors on their first and second components respectively.

A *Knowledge-Base*, K, is a seven-tuple $\langle C, R, Nr, Ar, F, P, A\rangle$ 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 set of *partitions*, subsets of $C \times R$, $\{p_1, \ldots, p_n\}$, such that: $(\cup i : 1 \leq i \leq n : p_i) = C \times R$ (i.e. each element of $C \times R$ is in some p_i).
A	=	A rule association function mapping: $Nr \cup Ar \Longrightarrow C \cup R$, such that: $(\forall \rho : \rho \in Ar \cup Nr : A(\rho) \in R \rightarrow \{A(\rho)\} = relations(Key(\rho)))$ (i.e. if a rule ρ is associated with a relation then that relation must be the one appearing in $Key(\rho)$).

Figure 1: Definition of a Knowledge-Base.

2.3.2 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. 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 the partitions α is in. Intuitively, this union defines the rules which are available to the operation. Thus an operation on α has access to the rules of all partitions α is in.

More formally, if $K = \langle C, R, Nr, Ar, F, P, A\rangle$ 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 partition of K then $\alpha \in p$ iff $\langle c, r\rangle \in p$. If $P = \{p_1, \ldots, p_n\}$ and $\mathcal{O} = query(\alpha)$ or $\mathcal{O} = assert(\alpha)$ then union of partitions for $\mathcal{O}$ is:

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

2.3.3 The Values of ALL Operations

Defining the values of ALL operations is primarily a mater of formalizing the action of forward and backward chaining rules. We use the following basic notation for knowledge-bases and substitutions: If $K_1 = \langle C, R, Nr, Ar, F_1, P, A\rangle$ and $K_2 = \langle C, R, Nr, Ar, F_2, P, A\rangle$ are knowledge-bases, then:

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

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

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

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 operations $\mathcal{O}$, we define $\mathcal{O}_n(K, p)$ as 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 (the full formal definition of $\mathcal{O}_n$ is given in [Crawford and Kuipers, 1989]). We then define $\mathcal{O}$ in terms of $\mathcal{O}_n$ as shown in figure 2. Note that since $\mathcal{O}$ is defined as the union over all n of $\mathcal{O}_n$, recursive rules (e.g. rules of form $q \leftarrow q$) do not cause any problems in ALL (or its lisp implementation). Figure 3 shows an example of a query on a simple knowledge-base.

2.4 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

If $\mathcal{O}$ is a primitive operation allowed in a knowledge-base K then:

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

The result of a non-primitive operations is defined in terms of the results of its constituent primitive operations. Again assume that $\mathcal{O}$ is an operation allowed in K:

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

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

else

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

Figure 2: The definition of $\mathcal{O}$.

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

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

Further, $A(r_1(c,x) \leftarrow r_2(c,x)) = r_1$. Consider $query(r_1(c,x))(K)$ (where x is a variable). This is a primitive operation so we first compute $query_0(r_1(c,x))(K, par_K(r_1(c,x)))$. Rule back-chaining is cut off at depth 0 so no rules apply and $query_0(r_1(c,x))(K, par_K(r_1(c,x))) = \langle \{\}, K \rangle$ (an empty list of substitutions is returned since there is no known value of x such that the query succeeds). However when we calculate $query_1(r_1(c,x))(K, par_K(r_1(c,x)))$, the if-needed rule applies and $query_1(r_1(c,x))(K, par_K(r_1(c,x))) = \langle \{\{x/c\}\}, K + r_1(c,c) \rangle$ (where $\{x/c\}$ 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 3: A query on a simple knowledge-base.

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

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

Finally, $A(r_1(c,x) \leftarrow r_2(c,x)) = r_1$, $A(r_1(c,x) \rightarrow r_3(c,x)) = r_1$. Consider $query(r_3(c,c))(K)$. This query must fail since $r_3(c,c)$ is not a fact in K and there are no if-needed rules for $r_3(c,c)$. 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 4: A form of incompleteness in ALL.

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

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

Finally, $A(r_1(c,x) \leftarrow r_2(c,x)) = r_1$, $A(r_2(c,x) \leftarrow r_3(c,x)) = r_2$. Consider $query(r_1(c,c))(K)$. This query must fail since $r_2(c,c)$ is not a fact in K and is not in $par_K(r_1(c,c))$ (so no rules for $r_2(c,c)$ can 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 5: Another form of incompleteness in ALL.

necessary properties for any knowledge representation system.

Mapping ALL to predicate calculus is fairly straight forward. Propositions do not change at all. Paths become conjunctions. Rules become implications with all variables universally quantified (there are some complications in mapping rules associated with frames (as opposed to slots) — these are discussed in [Crawford and Kuipers, 1989]). Knowledge-bases become the conjunction of their rules and facts. We notate the Predicate Calculus equivalent of an ALL object, a, by $\mathcal{PC}(a)$.

2.5 Consistency

Consistency is often intuitively thought of as "You can't derive a contradiction." Thus consistency requires that the substitutions returned by a query must be semantic consequences of the old knowledge-base. 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) *For any knowledge-base K, any path α allowed in K, and any fact f allowed in K:*

1. $(\forall \theta \in \Theta \quad : \theta \in sub(query(\alpha)(K)) \quad : \mathcal{PC}(K) \models \mathcal{PC}(\alpha\theta))$
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 $\mathcal{O}$. We induct on n to show that $\mathcal{O}_n$ is consistent. We then induct on the length of α to show that $\mathcal{O}$ is consistent.

2.6 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 simply the requirements for consistency with their implications reversed:

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. $(\forall \theta \in \Theta_\alpha \quad : \mathcal{PC}(K) \models \mathcal{PC}(\alpha\theta) \quad : \theta \in sub(query(\alpha)(K)))$
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 examples in figures 4 and 5. Notice, however, that in the example in figure 4:

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

would succeed since $r_3(c,c)$ is added to

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

by the if-added rule $r_1(c,x) \rightarrow r_3(c,x)$. Similarly, in the example of in figure 5:

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

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

2.6.1 Socratic Completeness

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. $(\forall \theta \in \Theta_\alpha: \mathcal{PC}(K) \models \mathcal{PC}(\alpha\theta) : (\exists \beta: \beta \text{ a path allowed in } K : \theta \in sub(query(\alpha)(kb(query(\beta)(K))))))$
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 $\mathcal{LP}(K)$. We 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 and Kuipers, 1989, Apt, 1988, Lloyd, 1984]) there exists a path in ALL the query of which causes the rule to fire. The result then follows by a completeness result for the study of logic programming.

2.6.2 Partitional Completeness

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 can apply to a frame-slot in that partition. If p is a partition of a knowledge-base $K = \langle C, R, Nr, Ar, F, P, A\rangle$, and S is a set of rules from K then $S\backslash_p$ is the restriction of S to p (the set of rules from S which can apply to frame-slots in p — the formal definition is given in [Crawford and Kuipers, 1989]). If p is a union of several partitions then $S\backslash_p$ is just the union of S restricted to the partitions. The restriction of K to only the if-needed rules in p is:

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

where A' is A restricted to the domain $Nr\backslash_p$. Note that the restriction of K to some union of partitions p is never computed (in the definition of ALL formula or in our lisp implementation of ALL), but is only a formal object used to state the partitional completeness theorem.

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

$$\begin{aligned}(\forall\theta \in \Theta_\alpha \quad &: \quad \mathcal{PC}(K\backslash_{par_K(\alpha)}) \models \mathcal{PC}(\alpha\theta)\\ &: \quad \theta \in sub(query(\alpha)(K)))\end{aligned}$$

Proof (sketch): The proof of this theorem again relies on results from the study of logic programming. Let $ground(\alpha)$ be the set of all variable free instantiations of α. 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:

$$\begin{aligned}(\forall f \in ground(q) \quad &: \quad f \in T_{\mathcal{LP}(K\backslash_p)} \uparrow n(\emptyset)\\ &: \quad f \in kb(query_n(q)(K, par_K(q))))\end{aligned}$$

Which is shown by induction on n from the definition of $\mathcal{O}_n$ and which again implies the result by a completeness result from logic programming (for the induction to go through, this lemma must actually be strengthened somewhat — see [Crawford and Kuipers, 1989] for details).

2.7 About Partitions

An important part of the claim that ALL is logically coherent is the claim that knowledge can be divided into semantically distinct segments. Fortunately, partitions are not a new idea. Among other places, similar ideas can be found in Hayes' clusters [Hayes, 1985]. The related idea that reasoning can be done by separating rules into partitions is also not new. It is the idea behind, for example, blackboard architectures [Hayes-Roth, 1985] (the difference between partitions in ALL and the similar limitations in blackboard architectures is the idea of access paths, which allow us to use the entire knowledge-base as our 'blackboard').

3 The Computational Tractability of ALL.

In the worst case the time complexity of an ALL operation is a polynomial function of the size of the portion of the knowledge-base accessible from the local partition. We focus on primitive operations since non-primitive operations are defined as sequences of primitive operations (figure 2).

Assume $\mathcal{O}$ is a primitive operation allowed in a knowledge-base K. By examination of the rules in the partition of $\mathcal{O}$ in K we can determine:

- $reach(\mathcal{O}, K)$ — the set of all frame-slots which $\mathcal{O}$ can ever reference.
- $change(\mathcal{O}, K)$ — the set of frame-slots which $\mathcal{O}$ can ever change.
- $frames(\mathcal{O}, K)$ — the set of frames which $\mathcal{O}$ could possibly put into frame-slots in $change(\mathcal{O}, K)$.
- $operations(\mathcal{O}, K)$ — the set of all queries of frame-slots in $reach(\mathcal{O}, K)$ and assertions of frames in $frames(\mathcal{O}, K)$ into frame-slots in $change(\mathcal{O}, K)$.

(Formal definitions of these sets are given in [Crawford and Kuipers, 1989]). In a well partitioned knowledge-base these sets should be much smaller than the total size of the knowledge-base.

For a set S, let $\mid S \mid$ be the cardinality of S.

Theorem 4 (Complexity) *Assume $\mathcal{O}$ is a primitive operation allowed in a knowledge-base $K = \langle C, R, Nr, Ar, F, P, A\rangle$. Let*

- $o = \mid operations(\mathcal{O}, K) \mid$
- $c = \mid change(\mathcal{O}, K) \mid$
- $f = \mid frames(reach(\mathcal{O}, K)) \mid$
- $r =$ *the number of rules in* $par_K(\mathcal{O})$.
- $a =$ *the maximum arity of any relation in* R.
- $v =$ *the maximum number of variables in any rule in* $par_K(\mathcal{O})$.

The worst case time complexity of calculating $\mathcal{O}(K)$ is bounded by:

$$a^2 o^2 r (r+f)^2 c^{v+2}$$

Proof (sketch): Consider the vector of all operations $\mathcal{O}' \in operations(\mathcal{O}, K)$. For any n these operations produce a vector of knowledge-bases $\mathcal{O}'_n(K)$. We

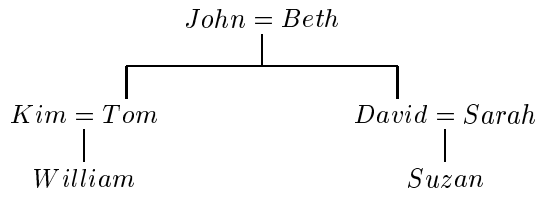

Figure 6: A genealogy.

show that if for some n and for all such $\mathcal{O}'$, $\mathcal{O}'_n(K) = \mathcal{O}'_{n+1}(K)$ then $\mathcal{O}(K) = \mathcal{O}_n(K)$. We then show that there must exist such an n which is less than or equal to $ao(r+f)c$ (by showing that knowledge-bases cannot shrink as n increases and showing a bound on how large they can grow). Finally, we show that the time to calculate any $\mathcal{O}'_n(K)$, from the values of all $\mathcal{O}'_{n-1}$, is bounded by $ar(r+f)c^{v+1}$.

4 Genealogy Example

An important part of our work with ALL has been our experience with the lisp implementation of ALL. We now present an introductory example from our implementation work. The knowledge-base consists of simple family relationships and the rules describe how to deduce more complex relationships. We emphasize that this example is one of the simplest we have implemented and is presented because it is relatively short and self-contained, yet gives a feel for the use of ALL and illustrates the use of access paths.

Figure 6 shows an example genealogy. To translate this into a knowledge-base assume that:

$$\begin{aligned} C &= \{ \; People, John, Beth, Kim, Tom, David, \\ &\quad\; Sarah, William, Suzan, Male, Female\} \\ R &= \{ \; isa, parent, child, son, daughter, brother, \\ &\quad\; sister, spouse, husband, wife, uncle, aunt, \\ &\quad\; cousin, gender\} \end{aligned}$$

Further, we need several if-added rules to enforce invariants in the knowledge-base. For example, we make sure that whenever there is a link $parent(x,y)$ there is also a link $child(y,x)$ (and vice-versa). Similarly, whenever there is a link $son(x,y)$ there are links $child(x,y)$ and $gender(x,Male)$, and so on. A important class of invariants are type restrictions — any frame put in a *parent*, *child*, or *spouse* slot '*isa*' '*People*'. The if-added rules are shown in figure 7. There are several other types of invariants which we

$$\begin{aligned} parent(x,y) &\rightarrow child(y,x) \\ child(x,y) &\rightarrow parent(y,x) \\ \\ son(x,y) &\rightarrow gender(y,Male) \\ son(x,y) &\rightarrow child(x,y) \\ daughter(x,y) &\rightarrow gender(y,Female) \\ daughter(x,y) &\rightarrow child(x,y) \\ \\ husband(x,y) &\rightarrow wife(x,y) \\ husband(x,y) &\rightarrow spouse(x,y) \\ husband(x,y) &\rightarrow gender(y,Male) \\ wife(x,y) &\rightarrow husband(x,y) \\ wife(x,y) &\rightarrow gender(y,Female) \\ spouse(x,y) &\rightarrow spouse(y,x) \\ \\ parent(x,y) &\rightarrow isa(y,People) \\ child(x,y) &\rightarrow isa(y,People) \\ spouse(x,y) &\rightarrow isa(y,People) \end{aligned}$$

Figure 7: If-added rules for genealogy example.

$$\begin{aligned} brother(x,z) &\leftarrow parent(x,y), son(y,z), x \neq z \\ sister(x,z) &\leftarrow parent(x,y), daughter(y,z), \\ &\qquad x \neq z \\ uncle(x,z) &\leftarrow parent(x,y), brother(y,z) \\ uncle(x,z) &\leftarrow aunt(x,y), husband(y,z) \\ aunt(x,z) &\leftarrow parent(x,y), sister(y,z) \\ aunt(x,z) &\leftarrow uncle(x,y), wife(y,z) \\ cousin(v,z) &\leftarrow parent(v,w), parent(w,x), \\ &\qquad child(x,y), y \neq w, \\ &\qquad child(y,z), v \neq z \end{aligned}$$

Figure 8: If-needed rules for genealogy example.

could add (e.g. whenever there are links $child(x, y)$ and $gender(y, Male)$ then there is a link $son(x, y)$) but which are not necessary for this example. Similarly, we could add additional type restricting if-added rules for some of the more complex relations (e.g. $uncle(x, y) \rightarrow isa(y, People)$). Note, however, that the type restricting rules together with the other invariants ensure that any frames in the relations son, $daughter$, $husband$, and $wife$ are '$People$'. We associate these if-added rules with the relations in their keys (e.g. $A(parent(x, y) \rightarrow child(y, x)) = parent$).

We use the if-needed rules shown in figure 8 to deduce the more complex relations. In these rules $x \neq y$ is true when x and y are bound to different frames. We associate these rules with the frame $People$ (thus they are available to fill slots in any frame known to be a $People$). Notice that the rules for $uncle$ and $aunt$ are mutually recursive, but this causes no problem in ALL (though it would cause an infinite loop in Prolog) since $query$ is defined as the union over all n of $query_n$ (see figure 2). Finally, we assume that the knowledge-base consists of a single partition, and initially contains no facts. We have thus defined an initial knowledge-base K_0.

Now we assert into K_0 the family relations in figure 6. This can be done by asserting the following path:

$$\begin{aligned} &wife(John, Beth), wife(Tom, Kim), \\ &wife(David, Sarah), son(John, Tom), \\ &son(Beth, Tom), son(John, David), \\ &son(Beth, David), son(Tom, William), \\ &son(Kim, William), daughter(David, Suzan), \\ &daughter(Sarah, Suzan) \end{aligned} \tag{1}$$

Asserting this path adds many more facts to the knowledge-base than just those mentioned in the path. For example, it adds $gender(Tom, Male)$, and that the frames $John$, $Beth$, Tom, Kim, $David$, $Sarah$, $William$, and $Suzan$ are all $People$. Let K_1 be the knowledge-base after the assertion of the path in 1.

Finally, we can make queries into K_1. Consider first,

$$query(uncle(William, x))(K_1).$$

Assume,

$$p = par_{K_1}(uncle(William, x)).$$

Clearly,

$$query_0(uncle(William, x))(K_1, p)$$

fails. Similarly,

$$query_1(uncle(William, x))(K_1, p)$$

fails since the facts $parent(William, John)$ and $parent(William, Beth)$ are known, but no brothers of $John$ or $Beth$ are known. However,

$$query_1(brother(John, y))(K_1, p)$$

succeeds with y bound to $David$ (by the if-needed rule for brother). Hence,

$$query_2(uncle(William, x))(K_1, p)$$

succeeds with x bound to $David$. Similarly,

$$query(cousin(Suzan, x))(K_1)$$

succeeds with x bound to $William$. One important advantage gained by the use of access paths is that the size of the knowledge-base could be increased with no effect on the time taken to compute these queries (unless we add frames which cause the access paths to branch — e.g. by adding more children of $John$ and $Beth$).

5 Related Work.

ALL draws from several diverse fields and we will not have space here to examine in detail its relationship to the large body of previous work. We simply sketch in general terms the fields from which it draws and a few particularly relevant past approaches.

ALL draws from semantic networks [Findler, 1979, Brachman *et al.*, 1983, Bobrow and Winograd, 1985, Vilain, 1985] the intuition that retrieval and reasoning can be guided 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, 1979]. ALL also draws from semantic networks its frame based data structures [Minsky, 1985].

ALL differs from past 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.*, 1983, Schubert *et al.*, 1983] or a weaker deduction system [Levesque, 1984, Patel-Schneider, 1985, Vilain, 1985] for general reasoning.

A notable exception to this rule is the recent work of Schubert [Schubert, 1979, Haan and Schubert, 1986]. 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 and Schubert, 1986]. The difference in ALL is that access is limited to known *access paths*, which may 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.

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 the proofs of the completeness of logic programming [Apt, 1988, Lloyd, 1984], have been used extensively in the completeness proofs for ALL.

6 Discussion and Future Work.

Ultimately we are working towards a formal theory which has the expressive power of predicate calculus, and is consistent and Socratically Complete, but still has polynomial time complexity. The current formalism of ALL (unlike our Lisp implementation) can express only implication — not general negation. It is straight forward 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. For example, from two rules of the form:

$$\begin{aligned} p &\leftarrow q \\ \neg p &\leftarrow q \end{aligned}$$

one should be able to conclude $\neg q$, but neither rule can apply since there are no facts. We are currently working to increase the deductive power of ALL by adding a *Reductio Ad Absurdum* mechanism. This involves adding the ability to make an assumption and then reason about its consequences. If the consequences include 'false' then we can conclude the negation of the assumption. In the above example we assume q and derive p and $\neg p$. Thus we can conclude $\neg q$. We believe that such a mechanism will allow Socratically Complete reasoning in the presence of classic negation.

There is also no obvious way to express full existential quantification in our current formalism or our implementation. We have incorporated definite descriptions (e.g. "The man with the wooden leg"), which define a type of existential quantification (a definite description should pick out a unique known frame or, if there is no frame meeting the description, create a new one) into the implementation of ALL, but we have not yet formalized them as they do not seem to translate naturally into predicate calculus.

However, we have observed that commonsense reasoning often involves reasoning about groups of similar objects, and that much of this reasoning can be done without full first order quantification. One may, for example, reason about a large class of objects by reasoning about a representative object having the properties common to all objects in the group (or reason about a 'skolem' object having properties that some (unknown) object of the group is known to have). In our implementation work we have been developing a "commonsense set theory" entirely within the quantifier limitations of ALL, and have applied it to several examples.

In general, our lisp implementation of ALL is considerably ahead of our formalism. Beyond definite descriptions and common-sense set theory we have implemented full negation (using *Reductio Ad Absurdum* as discussed above) and an ability to make default assumptions. We have built up a small knowledge-base of common-sense knowledge and have investigated several classes of problems:

- We have written a version of the inferential distance rule of Touretzky [Touretzky, 1986]), and have looked at some standard examples of multiple inheritance (e.g. royal elephants and birds that are penguins) and at a Nixon diamond.
- We have implemented a fairly standard solution to the Yale shooting problems [Hanks and McDermott, 1986].
- Using our common-sense set theory we have implemented a solution to McCarthy's sterilization problem [1987] and other more complex problems involving sets of similar objects.
- To demonstrate Socratic Completeness and the use of *Reductio Ad Absurdum*, we have implemented a solution to the following logical puzzle taken from [Wylie, 1957]:

  > 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 ?

  Our solution involves the use of rules which define notions of partial orders and one-to-one relations between sets. When we state the problem in ALL, our lisp implementation initially fails to solve it, but after a suitable sequence of preliminary queries (essentially the questions one would ask a person to step them through the puzzle) is able to do so.

References

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

[Apt, 1988] Apt, Krzysztof, R. Introduction to Logic Programming. To appear in *Handbook of Theoretical Computer Science*, ed. J. van Leeuwen., North Holland.

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

[Brachman *et al.*, 1983] Brachman, R.J., Fikes,R. E., and Levesque H. J. Krypton: a functional approach to knowledge representation, *Computer*, 16:67-73., 1983.

[Bobrow and Winograd, 1985] Bobrow, Daniel G., and Winograd, Terry. An Overview of KRL, a Knowledge Representation Language. In [Brachman and Levesque, 1985], pp. 263-285.

[Crawford and Kuipers, 1989] Crawford, J. M., and Kuipers, B. Access-Limited Logics. Forthcoming technical report.

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

[Haan and Schubert, 1986] Haan, J., and Schubert, L. K. Inference in a topically organized semantic net. In *Proc. Natl. Conf. Am. Assoc. Artif. Intell., Philadelphia, Pa.*, 1986, pp. 334-338.

[Hanks and McDermott, 1986] Hanks, Steve, and McDermott, Drew. Default Reasoning, Nonmonotonic Logics, and the Frame Problem. In *Proc. Natl. Conf. Am. Assoc. Artif. Intell., Philadelphia, Pa.*, 1986, pp. 328-333.

[Hayes, 1985] Hayes, Patrick J. The Second Naive Physics Manifesto. In Hobbs, Jerry R. and Moore, Robert C., *Formal Theories of the Commonsense World*, Ablex Publishing Co.: New Jersey, 1985, pp. 18-30.

[Hayes-Roth, 1985] Hayes-Roth, B. A Blackboard Architecture for Control. In *Artificial Intelligence Journal*, 26:251-321, 1985.

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

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

[Levesque, 1984] Levesque, H.J. A Logic of Implicit and Explicit Belief. In *Proc. Natl. Conf. Am. Assoc. Artif. Intell., Austin, Texas*, 1984, pp. 198-202.

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

[McCarthy, 1987] McCarthy, J. Generality in Artificial Intelligence. In *Communications of the ACM* 30:1030-1035, 1987.

[Minsky, 1985] Minsky, Marvin. A Framework for Representing Knowledge. In [Brachman and Levesque, 1985], pp. 246-262.

[Patel-Schneider, 1985] Patel-Schneider, P. A Decidable First-Order Logic for Knowledge Representation. In *Proc. Int. Jt. Conf. Artif. Intell., Los Angeles, California*, 1985, pp. 455-458.

[Powers, 1987] Powers, L. H. Knowledge by Deduction. In *The Philosophical Review*, LXXXVII, No. 3, 1978, pp. 337-371.

[Schubert, 1979] Schubert, L.K., R.G. Goebel, and N.J. Cercone, "The Structure and Organization of a Semantic Net for Comprehension and Inference," in [Findler, 1979], pp. 121-175.

[Schubert *et al.*, 1983] Schubert, L.K., Papalaskaris, M.A., and Taugher, J. Determining Type, Part, Color, and Time Relationships. In *IEEE Computer*, 16(10), 53-60, 1983.

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

[Vilain, 1985] Vilain, M. The restricted language architecture of a hybrid representation system. In *Proc. Int. Jt. Conf. Artif. Intell., Los Angeles, California*, 1985, pp. 547-551.

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

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