SIGART Bulletin 2(3): 35-44, June 1991.

Algernon — A Tractable System for Knowledge-Representation

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

Abstract

Access-Limited Logic (ALL) is a theory of knowledge representation which formalizes the access limitations inherent in a network structured knowledge-base. Where a deductive method such as resolution would retrieve all assertions that satisfy a given pattern, an access-limited logic retrieves only those assertions reachable by following an available access path. The time complexity of inference in ALL is a polynomial function of the size of the *accessible portion* of the knowledge-base, rather than an exponential function of the size of the entire knowledge-base (as in much past work). Access-Limited Logic, though incomplete, still has a well defined semantics and a weakened form of completeness, Socratic Completeness, which guarantees that for any fact which is a logical consequence of the knowledge-base, there is a series of preliminary queries and assumptions after which a query of the fact will succeed.

Algernon implements Access-Limited Logic. Algernon is important in testing the claims that common-sense knowledge can be encoded cleanly using access paths, and that in common-sense reasoning the preliminary queries and assumptions can generally be determined from domain knowledge. In this paper we overview the principles of ALL and discuss the application of Algernon to three domains: expert systems, qualitative model building, and logic puzzles.

1 Introduction

Access-Limited Logic (ALL) is a language for knowledge representation which provides both logical coherence and computational tractability. ALL embeds the access-limitations of a network-structured knowledge base in the logic. Operations on ALL knowledge-bases are guaranteed to terminate in time polynomial in the size of the *locally accessible* portion of the knowledge base. Furthermore, ALL is Socratically Complete: for any fact which is a semantic consequence of the knowledge base, there is a sequence of preliminary queries and assumptions, after which a query of the fact will succeed [10, 9]. While these formal properties are important, they do not necessarily guarantee that ALL can actually be used to represent knowledge. There are two further claims which can only be shown empirically: (1) that the syntactic restrictions on ALL still allow one to express common-sense knowledge cleanly and (2) that in practice the preliminary queries and assertions can generally be determined from domain knowledge.

It has been apparent at least since the work of [24] (and others such as [15]) that a knowledge representation language sufficient to support the building of large knowledge bases must have a clear semantics. Without a clear semantics one can never be sure exactly *what* a given expression represents, which deductions should follow from it, or how it compares to an expression in a different knowledge representation language. Experience with formally specified knowledge representation systems has revealed a trade-off between the expressive power of such systems and their computational complexity [18, 19]. Further, to ensure tractable and complete reasoning, one must restrict the expressiveness of a knowledge representation language almost to the point of unuseability (except within restricted domains [1, 6]). The most common solution to this problem has been to combine fast, special purpose inference with a first-order logic theorem prover [4, 21] or a weaker deduction system [Levesque, 84, Patel-Schneider, 85, Vilain, 85]. The main problem with such systems is that while they answer some queries efficiently they mysteriously fail on others (or never answer at all).

Our approach to knowledge-representation in ALL is based on two claims: first, that there exists a set of inference mechanisms, efficiently computable using a network structured representation, which are sufficient for a large class of common-sense reasoning problems, and second, that these inference mechanisms are Socratically Complete (thus guaranteeing that any logical consequence of a knowledge-base can be inferred after some series of 'leading' questions). Time complexity and Socratic Completeness are formal properties which have been shown elsewhere [9]. The claim that the representation and inference mechanisms in ALL are sufficient for a large class of common-sense reasoning problems can necessarily only be proven or disproven empirically. In this paper we overview results from the application of the lisp implementation of ALL to three fields: simple expert systems, qualitative model building, and logic puzzles.

2 Tractable Reasoning and Experimentation

Reasoning is hard. If a knowledge representation language is as expressive as first-order predicate calculus, then the problem of deciding what an agent could logically deduce from its knowledge is unsolvable [3]. Thus a knowledge-representation system must either give up expressive power or completeness[1]. If a system is not capable of complete inference under a given semantics (*e.g.*, the semantics of predicate calculus) then arguably this implies that the semantics is not appropriate for the system (*e.g.*, if according to the semantics $Th \models f$, but the system cannot derive f from Th, then the meaning of Th to the system is clearly not the same as the meaning given it by the semantics). If, however, a system is incomplete, but is Socratically Complete, then the semantics accurately reflects the potential reasoning ability of the system (and the actual reasoning ability of the system given appropriate queries and assumptions).

There are thus three solutions to the undecidability of predicate calculus: restrict the expressive power of the system, replace the model theoretic semantics of predicate calculus (with perhaps a more operational semantics), or give up completeness but guarantee Socratic Completeness. In all cases experimentation is critical. If a system has weak expressive power, then one must demonstrate that there are a significant number of interesting problems which can be solved within the limits of the system. If one uses an operational semantics then one must demonstrate that it gives users a sufficiently clear picture of what deductions the system is able to make. Finally, with Socratically Complete systems one must demonstrate empirically that (at least in some domains) the preliminary operations can be derived.

3 Introduction to Algernon

Algernon is a frame based knowledge-representation system which supports both forward and backward chaining rules of inference. This section describes Algernon in sufficient depth that the example below can be understood. For a more detailed discussion see [9].

The expressive power of Algernon greatly exceeds that of ALL. Algernon supports full first order quantification (which we are in the process of adding to ALL), and a general mechanism for escaping to lisp. Our general methodology has been to test constructs in Algernon before attempting the more difficult job of formalizing them (and then modifying the implementation based on what we learn from the formalization).

[1] One could also give up tractability, however, if a system is intractable then in practice this means that it does not return on some inputs (or returns only after some unacceptable period of time), and thus the system is effectively incomplete

3.1 Access Paths

Our approach in Algernon 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 b in the r slot of the frame for a [16]:

$r(a,b) \quad \equiv$

```
a:
    r:
        values: { ...b... }
```

The main advantage of this frame based representation is that the (conceptual) objects related to a frame can be easily accessed by looking in a slot of the frame. We define an *access path*, in a network of frames, as a sequence of frames each directly accessible from (*i.e.*, appearing in a slot of) its predecessor. A sequence of predicates defines an access path iff any variable appearing as the first argument to a predicate has appeared previously in the sequence. For example, "John's parent's sister" can be expressed in Algernon as the path: `((parent John ?x) (sister ?x ?y))` (variables are always denoted in Algernon by Lisp atoms whose print names begin with '?'). The Algernon path `((parent John ?x) (sister ?x ?y))` is equivalent to the predicate calculus statement: $parent(John, ?x) \wedge sister(?x, ?y)$. In predicate calculus this statement is equivalent to: $sister(?x, ?y) \wedge parent(John, ?x)$. However, the corresponding sequence of predicates, `((sister ?x ?y) (parent John ?x))`, is *not* an access path because a query of `(sister ?x ?y)` would require a search of the knowledge-base.

3.2 Rules Are Implications

We represent logical implications as rules. For example, the logical assertion that

$$\forall x, y.[spouse(x, y) \rightarrow spouse(y, x)]$$

can be represented by the *forward-chaining rule*:

```
((spouse ?x ?y) -> (spouse ?y ?x)).
```

Intuitively, such a rule says that whenever we learn a relation `(spouse f g)` we should immediately conclude `(spouse g f)`.

Algernon also allows *backward-chaining* ("if-needed") rules. For example:

```
((aunt John ?y) <- (parent John ?x)
                   (sister ?x ?y))
```

Intuitively, this rule says that if you need to find an aunt of `John` then you should look for a sister of a parent of `John`. Notice that the antecedent of this rule is an access path. All rules in Algernon must define access paths.

3.3 The Syntax of Algernon

The interface to Algernon is through the lisp functions `a-assert` and `a-query`. Both functions take two arguments. The first argument is a comment string and the second is the path to be asserted or queried. Thus the simplest type of assertion in Algernon would be something like:

```
(a-assert "The father of Charles is Adam"
          '((father Charles Adam)))
```

Where `Charles` and `Adam` are names of frames in the knowledge-base, and `father` is a slot in the knowledge-base. This assertion would add the value `Adam` to the `father` slot of the frame with name `Charles` (and apply any if-added rules for this slot). The corresponding query:

```
(a-query "Is Adam the father of Charles ?"
         '((father Charles Adam)))
```

would succeed iff `Adam` is in (or could be proven to be in) the `father` slot of `Charles`. Queries (and assertions) return all known (or provable) values of variables. Thus the query:

```
(a-query "Who are sisters of fathers of Charles ?"
         '((father Charles ?x) (sister ?x ?s)))
```

would find all sisters of fathers (*i.e.*, aunts) of `Charles`. Access paths are queried by querying the first predicate and then, branching on all sets of variable bindings, querying the rest of the path.

Algernon slots can hold 'non-values' as well as values. Non-values are used to express negation; they say that an object does not stand in some relation to some other object. Non-values are denoted by `(not p)`, where `p` is a proposition. Thus an assertion of `(not (likes suzan chocolate))` says that suzan does not like chocolate.[2]

Algernon also supports several special forms. Short descriptions of the most important special forms are given below. The full syntax of all special forms is given in [9].

`:taxonomy` Adds to the basic taxonomic structure in the knowledge-base. The taxonomic structure of sets forms the "backbone" of an Algernon knowledge-base. Sets are described by lists whose elements are (names of) the members of the sets and whose sublists are subsets of the set. For example, `(:taxonomy (objects (companies Bank)))` defines the set `companies` which is a subset of the set `objects` and which includes the individual `Bank`.

`:slot` Declares a new slot. For example: `(:slot sister (people people))` declares the slot `sister` to be a relation between two members of the set `people`.

`:rules` and `:srules` Add deduction rules to the knowledge-base. `:rules` is followed by the name of the (frame for the) set the rules are to be associated with, and then the rules. `:srules` is similar except that it associates rules with a slot instead of a set. For example:

```
(:srules (:slot spouse)
         ((spouse ?x ?y) ->
          (spouse ?y ?x)))
```

associates the forward-chaining rule with the slot `spouse` (the construction `(:slot spouse)` is needed to refer to the *slot* `spouse` instead of the *frame* with name `spouse`).

`:create` Creates a new frame. The form `(:create ?x name)` would create a frame with name `name` and bind the variable `?x` to it.

`:forc` Find OR Create. `:forc` finds a frame satisfying a description, or creates a new frame and asserts the description. For example, `(:forc ?x (father Tom ?x))` binds `?x` to the father of Tom, if the father of Tom is known, otherwise it creates a new frame, binds `?x` to it, and asserts that it is the father of Tom.

`:assume` Adds an assumption to the knowledge-base. Algernon includes a simple truth maintenance system to support dependency directed backtracking.

3.4 The Incompleteness of Algernon

There are three sources of incompleteness in Algernon.[3] First, if-added and if-needed rules can be combined in such a way that logically entailed deductions cannot be drawn (without preliminary queries). We refer to this as *if-added/if-needed* incompleteness. An example of if-added/if-needed incompleteness is shown in figure 1. Second, Algernon allows the knowledge-base to be partitioned. Partitions limit rule backchaining — facts in other partitions are simply retrieved (for details see [8]). Finally, in the presence of negation, there may be logical consequences of the knowledge-base which cannot be derived by simple rule application. Such deductions can be drawn in Algernon by making assumptions and then reasoning by contradiction. Reasoning by contradiction is demonstrated in the annotated example.

---

[2] Technically, it adds `chocolate` to the non-value facet of the `likes` slot of the frame with name `suzan`.

[3] To be more precise, there are three sources of incompleteness in the portion of Algernon formalized by ALL — see the introduction to section 3.

Consider a knowledge-base containing the rules:

```
((r1 c ?x) <- (r2 c ?x))
((r1 c ?x) -> (r3 c ?x))
```

and the fact `(r2 c c)`. Logically this entails `(r3 c c)`, but this deduction cannot be drawn in Algernon without first querying `(r1 c c)`.

Figure 1: An example of if-added/if-needed incompleteness.

4 Applications of Algernon

Algernon has been used by a graduate level expert systems class and several simple expert systems have been built using it [9]. In general, expert systems are rule based and do not depend on complex reasoning (*e.g.*, proofs by contradiction). Thus the incompleteness of Algernon was not a problem in these applications. The access path restrictions also seemed to pose no difficulties. Algernon's use by the class did, however, pressure us to increase the usability of Algernon by working on its user interface and documentation. We were also forced to modify Algernon's algorithms in order to make it easier for our users to understand the order in which rules were applied (this was particularly important in applications which queried their users and needed to do so in a particular order).

Algernon has also been used to support our work on qualitative model building [7]. In general, the reasoning required for model building is 'straight line' reasoning — it involves no proofs by contradiction or reasoning by cases. Partitional boundaries occasionally had to be adjusted in order to make sure that model building did not terminate prematurely. Some cases of if-added/if-needed incompleteness also arose, but these were debugged fairly easily, and fixed by modifying the rules involved. The access path restriction forced the addition of inverses for some slots, but otherwise was not a problem.

The natural language group at MCC has recently begun using Algernon, and has developed functions which translate from an abstract knowledge-base interface based on Ontolingua [14] down to Algernon queries and assertions [2].

Logic puzzles are an interesting domain for Algernon for two reasons: first, they generally require a limited amount of general knowledge and thus can be studied without first having to build a large knowledge-base of common-sense facts, and second they generally do require complex reasoning and thus test our ability to generate preliminary operations. The annotated example demonstrates the application of Algernon to a simple logic puzzle.

5 Annotated Example

Consider the following problem (from [25]):

> 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 ?

These facts can be asserted into the Algernon knowledge-base as shown in figure 2. A small amount of commonsense knowledge is also needed to solve this problem (*e.g.*, anyone with a sister is not an only child), and is asserted into Algernon as shown in the figure. Declarations for the slots unique to the bank problem are also shown. Several generally useful slots (*e.g.*, **coreferent**, **less**, **least**, **one-to-one**) are defined in Algernon's 'background' knowledge-base (see [9]). Notice that we explicitly create frames for 'the cashier', 'the manager', and 'the teller'. Solving the problem then becomes a matter of deriving appropriate coreference links between these frames and the frames for Brown, Jones, and Smith.

Initially Algernon is unable to derive these coreference relations. We lead Algernon to the solution by asking a series of questions which are similar to the questions a person might ask themselves (we have also implemented heuristics capable of generating preliminary operations for this problem — see section 6).

For the queries, we show the output produced by Algernon. The first line in the output is only a comment. Next is either a message that the operation failed, or a list of the variable bindings produced. Variable bindings are shown in the form: `?u --- frame [name]`, where, '`frame`' is frame itself, and '`name`' is the value in the `name` slot of the frame.[4] Finally, Algernon produces the following performance data:

> Insertions: The number of new facts added to the knowledge-base.
> Rule applications: The number of rule applications.
> Unifications: The number of unifications performed.
> Matches: The number of matches performed.
> Frame insertions: The number of values added to slots of frames.[5]
> Frame accesses: The number of retrievals of values from frames.

For the operations which result in contradictions we include some tracing output to show the contradiction and how it is resolved.

Note that the last query (as well as two others) returns two sets of variable bindings. This is because the path

[4] This is provided to increase readability in cases in which the frame is something like `frame26`.

[5] This is distinct from the number of insertions since Algernon uses the frames for various kinds of internal bookkeeping.

```
(a-assert "New sets." '((:taxonomy (objects (companies Bank)
                                            (positions cashier manager teller)
                                            (people)))))
(a-assert "New slots."
   '((:slot sister (people people)
            :comment "(sister a b) = The sister of a is b.")
     (:slot only-child (people booleans)
            :cardinality 1
            :comment "(only-child a true) = a is an only child.")
     (:slot holds (people companies positions)
            :comment "(holds ?p ?c ?pos) = ?p holds position ?pos in company ?c.")
     (:slot position (companies positions people)
            :comment "(position ?c ?pos ?p) = In company ?c, ?pos is held by ?p.")))

(a-assert "Sisters and only children."
          '((:RULES people
                    ((sister ?p1 ?p2) -> (not (only-child ?p1 true))))))

(a-assert "Holds and position."
          '((:SRULES (:slot position)
                     ((position ?c ?pos ?p) -> (holds ?p ?c ?pos)))
            (:SRULES (:slot holds)
                     ((holds ?p ?c ?pos) -> (position ?c ?pos ?p)))))

;; One extra rule to enhance Algernon's ability to reason about one-to-one relationships.
(a-assert "Forward chaining rule for cf-member."
          '((:RULES sets
                    ((one-to-one-into ?s1 ?s2) (member ?s1 ?x) -> (cf-member ?x ?s2)))))

(a-assert "In a certain bank the positions of cashier, manager, and teller are held by
           Brown, Jones and Smith, though not necessarily respectively."
   '((:forc ?cp (position Bank cashier ?cp))  ; This finds or creates "the cashier".
     (:forc ?mp (position Bank manager ?mp))  ; "The manager."
     (:forc ?tp (position Bank teller ?tp))   ; "The teller."
     (:create ?b Brown) (:create ?s Smith) (:create ?j Jones)
     (isa ?b people) (isa ?j people) (isa ?s people)
     ;;
     (:create ?pos posts) (member ?pos ?cp) (member ?pos ?mp) (member ?pos ?tp)
     (complete ?pos true)          ; ?pos = {"the cashier", "the manager", "the teller"}.
     (:create ?emp employees) (member ?emp ?b) (member ?emp ?j) (member ?emp ?s)
     (complete ?emp true)          ; ?emp = {Brown, Smith, Jones}.
     ;;
     (one-to-one ?pos ?emp)
     ;;
     ;; Finally, the implicit assumption that Brown, Jones and Smith are different people:
     (:assume (not (coreferent ?b ?j))) (:assume (not (coreferent ?j ?s)))
     (:assume (not (coreferent ?s ?b)))))

(a-assert "The teller, who was an only child, earns the least."
   '((position Bank teller ?tp) (only-child ?tp true) (least ?tp posts)))

(a-assert "Smith, who married Brown's sister, earns more than the manager."
   '((:forc ?sis (sister Brown ?sis))
     (spouse Smith ?sis) (position Bank manager ?man) (greater Smith ?man)))
```

Figure 2: Algernon assertions to set up the Bank problem.

```
  (a-query "What positions do Brown, Smith and Jones hold ?"
           '((coreferent Brown ?be) (holds ?be Bank ?post)
             (coreferent Smith ?se) (holds ?se Bank ?post)
             (coreferent Jones ?je) (holds ?je Bank ?post)))

QUERYING:   What positions do Brown, Smith and Jones hold ?

 *Query failed.*

 Insertions: 3           Rule applications: 7    Unifications: 7        Matches: 26.
 Frame insertions: 25    Frame accesses: 182
```

Figure 3: Query initially fails.

```
  (a-query "If Smith were the manager then could he earn more than the manager ?"
           '((position Bank manager ?man) (:assume (coreferent Smith ?man))
             (not (greater Smith ?man))))

QUERYING:   If Smith were the manager then could he earn more than the manager ?

  ** Beginning raa Trace **
 Contradiction:        (not (greater smith frame2))
                       (greater smith frame2)

   (not (greater smith frame2)) supported by assumptions: ((coreferent smith frame2))
   (greater smith frame2) supported by assumptions: nil

 Dropping assumption: (coreferent smith frame2).
 Asserting its negation: (not (coreferent smith frame2)).
  ** End raa Trace **

 *Query failed.*

 Insertions: 27          Rule applications: 96    Unifications: 35       Matches: 258.
 Frame insertions: 166   Frame accesses: 1789
```

Figure 4: Proof by contradiction that Smith is not the manager.

```
  (a-query "If Smith were the teller then could he earn more than the manager?"
          '((position Bank teller ?t) (:assume (coreferent Smith ?t))
            (position Bank manager ?man) (not (greater Smith ?man))))

QUERYING:   If Smith were the teller then could he earn more than the manager?

  ** Beginning raa Trace **
 Contradiction:       (not (greater smith frame2))
                      (greater smith frame2)

   (not (greater smith frame2)) supported by assumptions: ((coreferent smith frame3))
   (greater smith frame2) supported by assumptions: nil

 Dropping assumption: (coreferent smith frame3).
 Asserting its negation: (not (coreferent smith frame3)).
  ** End raa Trace **

 *Query failed.*

 Insertions: 28         Rule applications: 90    Unifications: 35       Matches: 243.
 Frame insertions: 164  Frame accesses: 1837
```

Figure 5: Proof by contradiction that Smith is not the teller.

```
  (a-query "Hence, Smith holds which job ?" '((coreferent Smith ?se) (holds ?se Bank ?post)))

QUERYING:   Hence, Smith holds which job ?

 Result 1:
   Bindings:            ?post   --- cashier [cashier]
                        ?se    --- frame1 [nil]

 Result 2:
   Bindings:            ?post   --- cashier [cashier]
                        ?se    --- smith [smith]

 Created frame:        frame1-selfset

 Insertions: 15         Rule applications: 33    Unifications: 6        Matches: 88.
 Frame insertions: 50   Frame accesses: 536
```

Figure 6: Hence Smith is the cashier.

```
  (a-query "If Brown were the teller then would he be an only child ?"
           '((position Bank teller ?t) (:assume (coreferent Brown ?t)) (only-child Brown true)))

QUERYING:   If Brown were the teller then would he be an only child ?

  ** Beginning raa Trace **
 Contradiction:        (only-child brown true)
                       (not (only-child brown true))

   (only-child brown true) supported by assumptions: ((coreferent brown frame3))
   (not (only-child brown true)) supported by assumptions: nil

 Dropping assumption: (coreferent brown frame3).
 Asserting its negation: (not (coreferent brown frame3)).
  ** End raa Trace **


 *Query failed.*

 Insertions: 19         Rule applications: 40    Unifications: 14       Matches: 122.
 Frame insertions: 72   Frame accesses: 789
```

Figure 7: Proof by contradiction that Brown is not the teller.

```
  (a-query "Hence Brown holds which job ?" '((coreferent Brown ?be) (holds ?be Bank ?post)))

QUERYING:   Hence Brown holds which job ?

 Result 1:
   Bindings:            ?post   --- manager [manager]
                        ?be    --- frame2 [nil]
   Assumption:         ((not (coreferent smith brown))
                        (not (coreferent jones smith))
                        (not (coreferent brown jones)))

 Result 2:
   Bindings:            ?post   --- manager [manager]
                        ?be    --- brown [brown]
   Assumption:         ((not (coreferent smith brown))
                        (not (coreferent jones smith))
                        (not (coreferent brown jones)))

 Deleted value:        (not (coreferent jones frame3))

 Insertions: 13         Rule applications: 40    Unifications: 17       Matches: 150.
 Frame insertions: 101  Frame accesses: 1206
```

Figure 8: Hence Brown is the manager.

```
  (a-query "Hence Jones holds which job ?" '((coreferent Jones ?je) (holds ?je Bank ?post)))

QUERYING:   Hence Jones holds which job ?

 Result 1:
   Bindings:            ?post   --- teller [teller]
                        ?je    --- frame3 [nil]

 Result 2:
   Bindings:            ?post   --- teller [teller]
                        ?je    --- jones [jones]

 Created frame:        jones-selfset

 Insertions: 21         Rule applications: 63    Unifications: 14       Matches: 213.
 Frame insertions: 110   Frame accesses: 1134
```

Figure 9: Hence Jones is the teller.

succeeds with `?je` bound to either `frame3` (the frame for "the teller") or `jones`. Since these two frames are now known to be coreferent, any fact true of either frame is inherited by the other.

The entire example (assertions and queries) runs in about four seconds on a Sparc Station One.

6 Conclusion

As discussed above, there are three sources of incompleteness in ALL. In theory, one could eliminate partitional incompleteness and if-added/if-needed incompleteness (by making the entire knowledge-base a single partition and duplicating all if-added rules as if-needed rules) without sacrificing tractability (though this would clearly increase the average-case time-complexity of ALL).

If we thus ignore partitional incompleteness and if-added/if-needed incompleteness, we can informally characterize the deeper source of incompleteness in ALL. ALL is incomplete for problems which require reasoning by cases (or, equivalently, reasoning by contradiction). If one can reason in a 'straight line' (by applying *Modus Ponens*) from known facts to a conclusion then ALL should also be able to derive the conclusion immediately. If, however, one must consider several cases and show that in each case the conclusion follows, then ALL will generally require preliminary operations.

We are currently investigating the problem of generating the series of queries and assumptions necessary to draw non-obvious deductions. The hard part of this problem is generating the assumptions; one can show that any result deducible using only queries is deduced by a fixed polynomial series of queries. This suggests that one might want to combine Algernon with assumption maintenance techniques to efficiently search the space of possible assumptions.

In the domain of logic puzzles, we have implemented a simple heuristics for generating the assumptions and queries. If a puzzle contains two sets, $S1$ and $S2$, in a one-to-one relationship, then for each member of $S1$ and each member of $S2$, one can assume a coreference link between the two members and query the *negation* of the facts given in the puzzle. Though this corresponds to a fairly dumb search for contradictions, the time complexity of the procedure is just the size of $S1$, times the size of $S2$, times the number of facts in the puzzle. This method is sufficiently powerful to solve the bank problem presented above, but it is not sufficient to solve all logic puzzles as some puzzles require nested assumptions (and the complexity of complete reasoning with nested assumptions is exponential in the depth of the nesting). However, one can bound the depth of assumption nesting (thus guaranteeing time complexity polynomial in the size of the knowledge base but exponential in the depth bound) and asymptotically approach complete reasoning [10]. Such an approach might be useful for deriving the best solution possible within some time bound.

Space limitations prevent a fair review of the large body of related work. The interested reader is referred to [9].

Algernon has been an important part of our work in ALL. It has been critical, both in testing new constructs for knowledge-representation, and in empirically investigating the problem of query and assumption generation. More recently it has been used to support our work in qualitative model building. We plan to make it generally available for research purposes in the near future, and expect it to be useful for building large common-sense knowledge-bases.

Acknowledgments

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 the Texas Advanced Research Program under grant no. 003658-175, NSF grants IRI-8905494 and IRI-8904454, and by NASA grant NAG 2-507.

References

[1] Allen, J. (1983). Maintaining knowledge about temporal intervals. *CACM* 26(11):832-43.

[2] Barnett, J., Rich, E., and Wroblewski, D. (1991). A functional interface to a knowledge base for use by a natural language processing system. Manuscript. Knowledge-Based Natural Language Project, MCC, 3500 West Balcones Center Dr., Austin, Texas.

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

[4] 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 [5], pp. 412-429.)

[5] Brachman, R.J. and Levesque, H.J. (1985). *Readings in Knowledge Representation*, Morgan Kaufmann, Los Altos, Cal.

[6] Bundy, A., Byrd, L., Mellish, C. (1985). Special-purpose, but domain-independent, inference mechanisms. In *Progress in Artificial Intelligence*, ed. L. Steels, J. Campbell, pp. 93-111. London: Ellis Horwood.

[7] Crawford, J., Farquhar, A., and Kuipers, B. (1990). QPC: A compiler from physical models into Qualitative Differential Equations. *AAAI-90*.

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

[9] Crawford, J. M. (1990). Access-Limited Logic — A language for knowledge-representation. University of Texas at Austin dissertation. Available as Technical Report number AI90-141, Artificial Intelligence Laboratory, The University of Texas at Austin.

[10] J. M. Crawford and B. J. Kuipers. (1991). Negation and Proof by Contradiction in Access-Limited Logic. To appear in *AAAI-91*.

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

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

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

[14] Gruber, T., Pang, D., and Rice, J. *Ontolingua: A Language to Support Shared Ontologies.* Technical Report, Stanford Knowledge Systems Lab, Palo Alto, California.

[15] Hayes, Patrick J. (1974). Some problems and non-problems in representation theory. In *Proc. AISB Summer Conference*, University of Sussex. (Reprinted in [5], pp. 3-22.)

[16] 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 [5], pp. 288-295.)

[17] D. B. Lenat and R. V. Guha. (1990). *Building Large Knowledge-Based Systems.* Addison-Wesley, Reading, MA.

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

[19] Levesque, H.J., Brachman R.J. (1985). A fundamental tradeoff in knowledge representation and reasoning (revised version). In [5], pp. 41-70.

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

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

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

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

[24] 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.

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