INFERENCE WITH RECURSIVE RULES

Stuart C. Shapiro and Donald P. McKay

Department of Computer Science
State University of New York at Buffalo
Amherst, New York 14226

ABSTRACT

Recursive rules, such as "Your parents' ancestors are your ancestors", although very useful for theorem proving, natural language understanding, questions-answering and information retrieval systems, present problems for many such systems, either causing infinite loops or requiring that arbitrarily many copies of them be made. We have written an inference system that can use recursive rules without either of these problems. The solution appeared automatically from a technique designed to avoid redundant work. A recursive rule causes a cycle to be built in an AND/OR graph of active processes. Each pass of data through the cycle resulting in another answer. Cycling stops as soon as either the desired answer is produced, no more answers can be produced, or resource bounds are exceeded.

Introduction

Recursive rules, such as "your parents' ancestors are your ancestors", occur naturally in inference systems used for theorem proving, question answering, natural language understanding, and information retrieval. Transitive relations, e.g. ∀(x,y,z)[ANCESTOR(x,y) & ANCESTOR(y,z)→ ANCESTOR(x,z)], inheritance rules, e.g. ∀(x,y,p) [ISA(x,y) & HAS(y,p) → HAS(x,p)], circular definitions and equivalences are all occurrences of recursive rules. Yet, recursive rules present problems for system implementors. Inference systems which use a "naive chaining" algorithm can go into an infinite loop, like a left-to-right top-down parser given a left recursive grammar [4]. Some systems will fail to use a recursive rule more than once, i.e. are incomplete [6,12]. Other systems build tree-like data structures (connection graphs) containing branches the length of which depend on the number of times the recursive rule is to be applied [2,13]. Since some of these build the structure before using it, the correct length of these branches is problematic. Some systems eliminate recursive rules by deriving and adding to the data base all implications of the recursive rules in a special pass before normal inference is done [9].

The inference system of SNePS [11] was designed to use rules stored in a fully indexed data base. When a question is asked, the system retrieves relevant rules and builds a data structure of processes which attempt to derive the answer from the rules and other information stored in the data base. Since we are using a semantic network to represent all declarative information available in the system, we do not make a distinction between "extensional" and "intensional" data bases, i.e. non-rules and rules are stored in the same data base. More significantly, we do not **distinguish** "base" from "defined" relations. Specific instances of ANCESTOR may be stored as well as a rule defining ANCESTOR. This point of view contrasts with the basic assumption of several data base question answering systems [3,8,9]. In addition, the inference system described here does not restrict the left hand side of rules to contain only one literal which is a derived relation [3], does not need to recognize cycles in a graph [3,8] and does not require that there be at least one exit from a cycle [8].

The structure of processes may be viewed as an AND/OR problem reduction graph in which the process working on the original question is the root, and rules are problem reduction operators. Partly influenced by Kaplan's producer-consumer model [5], we designed the system so that if a process working on some problem is about to create a process for a subproblem, and there is another process already working on that subproblem, the parent process can make use of the extant process and so avoid solving the same problem again. The method we employ handles recursive rules with no additional mechanism. The structure of processes may be viewed as an active connection graph, but, as will be seen below, the size of the resulting structure need not depend on the number of times a recursive rule will be used.

This paper describes how our system handles recursive rules. Aspects of the system not directly relevant to this issue will be abbreviated or omitted. In particular, details of the match routine which retrieves formulas unifiable with a given formula will not be discussed (but see [10]).

The Inference System

The SNePS inference system builds a graph of processes [7,11] to answer a question (derive instances of a given formula) based on a data base of assertions (ground atomic formulas) and rules (non-atomic formulas). Each process has a set of

The work was supported in part by the National Science Foundation under Grant No. MCS78-02274.

registers which contain data, and each process may send messages to other processes. Since, in this system, the messages are all answers to some question, we will call a process P2 a boss of a process P1 if P1 sends messages to P2. Some processes, called data collectors, are distinguished by two features: 1) they can have more than one boss; 2) they store all messages they have sent to their bosses. The stored messages are used for two purposes: a) it allows the data collector to avoid sending the same message twice; b) it allows the data collector to be given a new boss, which can immediately be brought up to date by being given all the messages already sent to the other bosses. Four types of processes are important to the discussion of recursive rules. They are called INFER, CHAIN, SWITCH and FILTER. INFER and CHAIN are data collectors, SWITCH and FILTER are not.

Four Processes

An INFER process is created to derive instances of a formula, Q. It first matches Q against the data base to find all formulas unifiable with Q. The result of this match is a list of triples, $<T,\tau,\sigma>$, where T is a retrieved formula called the target, and τ and σ are substitutions called the target binding and source binding respectively. Essentially τ and σ are factored versions of the most general unifier (mgu) of Q and T. Pairs of the mgu whose variables are in Q appear in σ, while those whose variables are in T appear in τ. Any variable in term position is taken from T. Factoring the mgu obviates the need for renaming variables. For example if Q=P(x,a,y) and T=P(b,y,x), we would have σ={b/x,x/y} and τ={a/y,x/x} (the pair x/x is included to make our algorithms easier to describe). Note that $Q\sigma = T\tau$ = P(b,a,x), the variables in the variable position of the substitution pairs of σ are all and only the variables in Q, the variables in the variable position of τ are all and only the variables in T, all terms of σ came from T, and the non-variables in τ came from Q.

For each match $<T,\tau,\sigma>$ that an INFER finds for Q, there are two possiblities we shall consider. First, T might be an assertion in the data base. In this case, σ is an answer ($Q\sigma$ has been derived). If the INFER has already stored σ, it is ignored. Otherwise, σ is stored by the INFER and the pair $<Q,\sigma>$ is sent to all the INFER's bosses. For σ to be a reasonable answer, it is crucial that all its variables occur in Q. The other case we shall consider is the one in which T is the consequent of some rule of the form $A1\&...\&An\supset T$. (Our system allows other forms of rules, but consideration of this one will suffice for explaining how we handle recursive rules). In this case, the INFER creates two other processes, a SWITCH and a CHAIN to derive instances of $T\tau$. The SWITCH is made the CHAIN's boss, and the INFER the SWITCH's boss. It may be the case that an already extant CHAIN may be used instead of a new one. This will be discussed below.

The SWITCH process has a register which is set to the source binding, σ. The answers it receives from the CHAIN are substitutions β, signifying that $T\tau\beta$ has been derived. SWITCH sends to its boss the application $\sigma\backslash\beta$, the substitution derived from σ by replacing each term **t** in σ by $t\beta$. The effect of the SWITCH is to change the answer from the context of the variables of T to the context of the variables of Q. In our example, the CHAIN might send the answer β={c/x}. SWITCH would then send $\sigma\backslash\beta$ = {b/x,x/y}\ {c/x}= {b/x,c/y} to the INFER, indicating that $Q\sigma\backslash\beta$ = P(x,a,y){b/x,c/y} = P(b,a,c) has been derived. The importance of the factoring of the mgu of Q and T into the source binding σ and the target binding τ -- a separation which the SWITCH repairs -- is that the CHAIN can work on T in the context of its original variables and report to many bosses, each through its own SWITCH.

A CHAIN process is created to use a particular substitution instance, τ, of a particular formula, $A1\&...\&Ak\supset T$ to deduce instances of $T\tau$. Its answers, which will be sent to a SWITCH, will be substitutions β such that $T\tau\beta$ has been deduced using the rule. For each Ai, $1\le i\le k$, the CHAIN tries to discover if $Ai\tau$ is deducible by creating an INFER process for it. However, an INFER process might already be working on $Ai\alpha$. If $\alpha=\tau$, the already extant INFER is just what the CHAIN wants. It takes all the data the INFER has already collected, and adds itself to the INFER's bosses so that it will also get future answers. If α is more general than τ, the INFER will produce all the data the CHAIN wants, but unwanted data as well. In this case the CHAIN creates a FILTER process to stand between it and the INFER. The FILTER stores a substitution consisting of those pairs of τ for which the term is a constant, and when it receives an answer substitution from the INFER, it passes it along to its CHAIN only if the stored substitution is a subset of the answer. For example, if τ were {a/x,y/z,b/w} and α were {u/x,v/z,v/w}, a FILTER would be created with a sutstitution of {a/x,b/w}, insuring that unwanted answers such as {c/x,d/z,b/w} produced by the more general INFER were filtered out. If α is not compatible with τ, or is less general than τ, a new INFER must be created. However, if α is less general than τ, the old INFER might already have collected answers that the new one can use. These are taken by the new INFER and sent to its bosses. Also, since the new INFER will produce all the additional answers that the old one would (plus others), the old INFER is eliminated and its bosses given to the new INFER with intervening FILTERs. The net result is that the same structure of processes is created regardless of whether the more general or less general question was asked first.

A CHAIN receives answers from INFERs (possibly filtered) in the form of pairs $<Ai,\beta i>$ indicating that $Ai\beta i$, an instance of the antecedent Ai, has been deduced. Whenever the CHAIN collects a set of consistent substitutions {βi,...,βn}, one for each antecedent, it sends an answer to its bosses consisting of the combination of β1,...,βk (where the combination of β1 = {t11/v11,...,t1n,/v1n1} ,...,βk = {tk1/vk1,...,tknk/vknk} is the mgu of the expressions (v11,...,v1n1,...,vk1,...,vknk) and (t11,...,t1n1,...,tk1,...,tknk) [1, p.187]).

Recursive Rules Cause Cycles

Just as a CHAIN can make use of already existing INFERs, an INFER can make use of already existing CHAINs, filtered if necessary. A recursive rule is a chain of the form $A1 \& \ldots \& Ak \supset B1, B1 \& \ldots \& Bn \supset \ldots \supset C$, with C unifiable with at least one of the antecedents, A1 say. When an INFER operates on A1, it will find that C matches A1, and it may find that it can use the CHAIN already created for C. Since this CHAIN is in fact the INFER's boss, this will result in a cycle of processes. The cycle will produce more answers as new data is passed around the cycle, but no infinite loop will be created since no data collector sends any answer more than once. (If an infinite set of Skolem constants is generated, the process will still terminate if the root goal had a finite number of desired answers specified [11, p.194]).

Figure 1 shows a structure of processes which we consider an active connection graph. It is built to derive instances of ANCESTOR(William,w) form the rules $\forall(x,y)$[PARENT(x,y)$\supset$ANCESTOR(x,y)] and $\forall(x,y,z)$[ANCESTOR(x,y) & PARENT(y,z)$\supset$ ANCESTOR(x,z)]. The notation for the rule instances is similar to that presented in [3]. Note particularly the SWITCH in the cycle which allows newly derived instances of the goal ANCESTOR(William,w) to be treated as additional instances of the antecedent ANCESTOR(William,y). A similar structure would be built regardless of the order of asserting the two rules, the order of antecedents in the two antecedent rule, the order of execution of the processes, whether the query had zero, either one, or both variables ground, or if the two antecedent rule used ANCESTOR for both antecedents.

Summary

In the SNePS inference system, recursive rules cause cycles to be built in a graph structure of processes. The key features of the inference system which allow recursive rules to be handled are: 1) the processes that produce derivations (INFER and CHAIN) are data collectors; 2) data collectors never send the same answer more than once; 3) a data collector may report to more than one boss; 4) a new boss may be assigned to a data collector at any time -- it will immediately be given all previously collected data; 5) variable contexts are localized, SWITCH changing contexts dynamically as data flows around the graph; 6) FILTERs allow more general producers to be used by less general consumers.

REFERENCES

1. Chang, C.-L., and Lee, R.C.-T., Symbolic Logic and Mechanical Theorem Proving, Academic Press, New York, 1973.
2. Chang, C.-L., and Slagle, J.R., Using rewriting rules for connection graphs to prove theorems, Artificial Intelligence 12, 2 (August 1979), 159-180.
3. Chang, C.-L., On evaluation of queries containing derived relations in a relational data base. In Formal Bases for Data Bases, Gallaire, H., Minker, J. and Nicolas, J. (eds.), Plenum, New York, 1980.
4. Fikes, R.E., and Hendrix G.G., The deduction component. In Understanding Spoken Language, Walker, D.E., ed., Elsevier North-Holland, 1978, 355-374.
5. Kaplan, R.M., A multi-processing approach to natural language understanding. Proc. National Computer Conference, AFIPS Press, Montvale, NJ, 1973, 435-440.
6. Klahr, P., Planning techniques for rule selection in deductive question-answering. In Pattern-Directed Inference Systems, Waterman, D.A., and Hayes-Roth, R., eds., Academic Press, New York, 1978, 223-239.
7. McKay, D.P. and Shapiro, S.C., MULTI--A LISP based multiprocessing system. Proc. 1980 LISP Conference, Stanford University, 1980.
8. Naqvi, S.A., and Henschen, L.J., Performing inferences over recursive data bases. Proc. First AAAI Conference, Stanford University, 1980.
9. Reiter, R., On structuring a first order data base, Proc. Second National Conference, Canadian Society for Computational Studies of Intelligence, 1978, 90-99.
10. Shapiro, S.C., Representing and locating deduction rules in a semantic network. Proc. Workshop on Pattern-Directed Inference Systems. In SIGART Newsletter, 63 (June 1977), 14-18.
11. Shapiro, S.C., The SNePS semantic network processing system. In Associative Networks: The Representation and Use of Knowledge by Computers, Findler, N.V., ed., Academic Press, New York, 1979, 179-203.
12. Shortliffe, E.H., Computer Based Medical Consultations: MYCIN, American Elsevier, New York, 1976.
13. Sickel, S., A search technique for clause interconnectivity graphs, IEEE Transactions on Computers Vol. C-25, 8 (August 1976), 823-835.

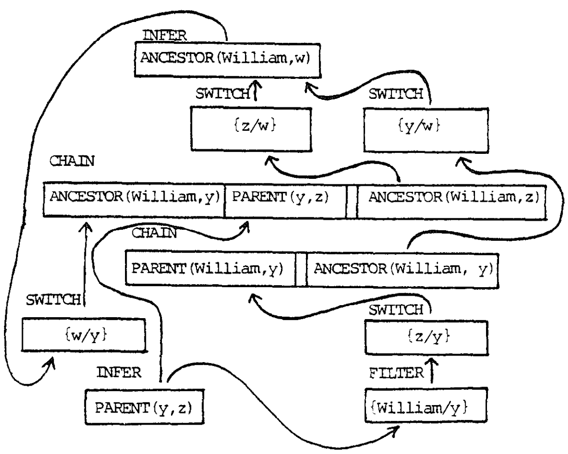

Figure 1.