PROLOG EXTENSIONS BASED ON TABLEAU CALCULUS

Wolfgang Schonfeld
Heidelberg Science Center
IBM Germany
Tiergartenstr. 15
D-6900 Heidelberg

Abstract

The intention of this paper is to help bridging the gap between logic programming and theorem proving. It presents the design of a Gentzen type proof search procedure, based on classical tableau calculus, for knowledge bases consisting of arbitrary first order formulas. At each proof search step, when a new formula is to be chosen from the knowledge base, the procedure chooses in such a way that the search space is small. When applied to a Horn clause knowledge base and an atomic goal, it performs the same proof search steps as any PROLOG interpreter would do. Hence, PROLOG can be viewed as a special Gentzen type procedure just as it is a special (namely, linear input) resolution procedure.

1. Introduction

Problem description. Expert systems are able to draw conclusions from data they contain. This deductive process can adopt different forms. There is the tradition of automated theorem proving, there are the many expert systems working with a production rule formalism, and there is PROLOG, a programming language which, at the same time, is a logical language.

PROLOG has proven very successful though it has some limitations. One of them is that it only accepts Horn clauses as rules and atomic formulas as goals. This restriction makes proof search simple and effective. But some properties cannot be expressed by Horn clauses, e.g. the linearity of an order relation. There are application domains where this handicap is not severe. In the LEX (linguistics and logic based Legal EXpert system) project at Heidelberg Science Center where the paper in hand was worked out, we consider the expressiveness of PROLOG to be too low. Only about 80 % of the knowledge in legal applications can be formulated. For the rest, we need negation, disjunction etc.

Existing solutions. Resolution calculus ([Robinson65]) is able to accept arbitrary first order formulas, even in non-clausal form if the extensions described in [Murray82] are used. Some of its defects when applied in its pure form to proving mathematical theorems are discussed in [Bledsoc77]. Bledsoe points out that heuristics should play an important role. Another defect is (|Clocksin81], p.221): 'Resolution tells us how to derive a consequence from two clauses, it does not tell us either how to decide which clauses to look at next or which literals to match.'

As compared with resolution based systems, Gentzen type systems did not yet attract much attention. An important theoretical article is [Beth59] where the notion of a *tableau* is introduced to present a Gentzen type system in a more natural way. This tableau method has been mainly used to prove completeness of various logic calculi such as modal logic [Rautcnberg79]. It has been used to test equivalence of relational expressions [SagivSl]. And it has drawn some attention in the legal domain where it is regarded as a natural formalization of legal reasoning [Hcrbcrgcr80].

To implement *tableau calculus* as described in [Beth59] is impossible since it is not *invertible* [Richter78]. This is because any Herbrand term can be substituted for the universal variables. It is the idea of Bowen [Bowen82] and (probably independently) of Wrightson [Wrightson84] to use unification in order to find the necessary substitutions. (TABLOG [Malachi84] is *not* based on tableau calculus though its name might suggest. It uses non-clausal resolution as in [Murray82|.)

Solution proposed in this paper. Regardless which calculus we use, one problem still remains: In which way should possible applications of rules be ordered ? This is important in case of a very large knowledge base. In each state of proof search, the inference engine can choose among a huge set of formulas. PROLOG uses a rather simple and efficient strategy to make this choice: It tries to unify the current predicate with the head of some rule. The purpose of the paper in hand is to extend the PROLOG idea to non-Horn formulas. More precisely, it presents a strategy with the following properties:

- It accepts arbitrary formulas of first order predicate logic.
- When given a Horn clause knowledge base and a provable atomic goal, it finds the same proof in the same way as any PROLOG interpreter would do.

The paper starts with a (slightly non-traditional) description of tableau calculus for single formulas. The next chapter contains the description of a goal oriented, depth-first strategy with backtracking, as mentioned above, for propositional formulas. How this extends to predicate logic and some further extensions are described in the final chapter.

An implementation of this procedure is under work. Instead of mere first order formulas, it will accept so called discourse representation structures, a modification of predicate logic to cope with linguistic phenomena.

2. Tableaux for propositional formulas

Tableau calculus as developed in [Beth59] is a formalization of the search for counterexamples or, equivalently, for models. Assume e.g. that the formula y =

$$((-R\&-S)|S)\&((R\&-S)|(-R\&S))\&-R$$

is satisfiable. ('&', 'l', '-' mean 'and', 'or', 'not' resp.) Satisfiability can be visualized by drawing a *logic diagram.*

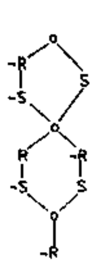

Figure 1

y is satisfiablc if and only if there is a *non-contradictory path* through its diagram, i.e. a path containing no complementary pair of literals.

This method to represent formulas is used in [Bibcl83]. It should be compared with the usual and-or-graphs. Bibel describes a procedure which directly checks for the existence of non-contradictory paths. A better way to get an overview of all paths is to arrange them into a tree.

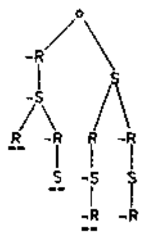

Figure 2

More precisely, we call the smallest tree containing the same paths as the diagram of y the *tree* of y. A *tableau* for y is a tree for y where any contradictory branch is cut off as soon as a complementary pair occurs. (In [Beth59], a tableau docs not only contain literals but also the initial formula as well as all intermediate steps. Since these steps are 'straightforward', it suffices to represent only the final result.) A *proof for* the unsatisfiability of y is a tableau for it in which all branches are contradictory, y is satisfiablc if and only if its tree is no proof. Hence, any procedure which, given a formula, generates a tableau and checks whether it is a proof is *correct* and *complete.*

3. Tableaux for sets of propositional formulas

Let Γ be a set of formulas which is to be checked for satisfiability. One way to do that is to proceed as above as if Γ were given by the conjunction of its elements in a fixed, but arbitrarily chosen order. For example, suppose that $\Gamma = \{\gamma_i | i = 1, 2, 3\}$ where $\gamma_1 = (\neg R \ \& \ \neg S)|S$, $\gamma_2 = (R \ \& \ \neg S)|(\neg R \ \& \ S)$, $\gamma_3 = \neg R$. Then Figure 2 may be called a tableau for Γ. If Γ is infinite, its tableau may be infinite. Again, Γ is not satisfiable if and only if the tableau is a proof, i.e. all its branches are closed by contradiction. Note that a proof is necessarily finite.

If the order of Γ is not fixed, in which way can a tableau for Γ be composed by the tableaux of its elements so that completeness still holds ? Suppose that a tree T is composed of the tableaux of the elements of Γ. A branch of T is said to *cross* (a tableau of) a formula $\gamma \in \Gamma$ if it contains as subpath a branch of the tableau for γ. T is called a *tableau for T* if each non-contradictory branch crosses each $\gamma \in \Gamma$ at least once. If T is no proof, then it contains at least one non-contradictory branch. The interpretation determined by such a branch satisfies each $\gamma \in \Gamma$ (since it crosses it). If T is a proof, then no interpretation can satisfy *T.* Hence, *T* is unsatisfiable if and only if there is a tableau for *T* which is a proof.

Choosing the next formula. To make search effective, it is necessary to keep the tableau as small as possible. First, no path should cross a formula twice. Second, branches should be cut off by a complementary pair as soon as possible. For example, a better tableau than Figure 2 on page 3 is

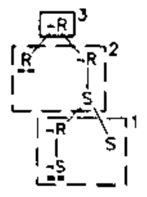

Figure 3

Two literals are *linked* if they form a complementary pair. Furthermore, a literal is *linked to a formula* γ if it is complementary to some literal in γ. Suppose that a set T of formulas contains an element *a* singled out for some reason, *a* is called the *goal.* Furthermore, suppose that a tableau for *T* is being generated, starting with a, and suppose that this process is now in a state where the tableau of another $\gamma \in \Gamma$ has to be appended to a certain non-contradictory branch B.

The idea is to choose a $\gamma \in \Gamma$ so that at least one of the resulting branches contains a complementary pair. To this end, let *8* be the last formula crossed by B, and let $B|\delta$ be the restriction of B to δ, i.e. that subpath of B which is part of the tableau for $\underline{\delta}$. Furthermore, let C (for 'choices') be the set of all $\gamma \in \Gamma$ with the following properties.

1 . γȝ linked to some literal on $B|\delta$.
2. γ is not crossed by B.

C is ordered in a certain way, e.g. by respecting a given order of r. This is used to organize backtracking. If backtracking occurs, the subtableau starting with the actually chosen $\gamma \in C$ is removed, and a new subtableau is generated starting with the next $\gamma \in C$. We say that the *choice at δ is altered.* Backtracking means that we go upwards on B up to the next δ where such an altering of a choice is possible.

The following gives the whole algorithm, called SEARCH.

1. Start with the.tableau for α
2. Mark contradictory paths.
3. If all paths are contradictory, stop with 'succeed'.
4. Otherwise, let B be the leftmost non-contradictory path.
5. Let C be as above.
6. If C is empty, then check whether any choice can be altered along B.
   a. If yes, then backtrack.
   b. Otherwise, choose a $\gamma \in \Gamma$ which is not yet crossed by B and go to 8. If there is none, then stop with 'fail'.
7. Otherwise, choose a $\gamma \in C$.

8. Append its tableau to B.
9. Goto 2.

This strategy is again correct and complete. Correctness follows from the fact that any tableau (found by a strategy whatsoever) with all branches closed by contradictions is a correct proof. To see completeness, note that 6.b. guarantees that all formulas in *T* are crossed exactly once by each non-contradictory branch. If it is applied to the above *T* with γ3 as a goal, then Figure 3 on page 4 results.

Comparison with PROLOG. It is claimed that algorithm SEARCH, when applied to propositional Horn clauses, is very close to PROLOG'S search strategy. Some remarks will help to understand the claim.

The *8* as considered above corresponds to the current predicate. C is the set of rules with which this predicate matches (with the above mentioned exception that matches leading to loops are avoided). Step 4 means that the first predicate on the right hand side of a rule is chosen.

One important difference is that SEARCH does never get into a loop. This comes from the fact that C as defmed above does not contain any formula crossed previously by branch B. Such loop checking is essential when definitions of the form *P(x)* s *a(x)* are evaluated. Note what PROLOG interpreters do when given the data base *a*<- *b,b* +~ *a.*

The main difference comes in with 6.b. PROLOG stops at this point with 'fail'. SEARCH will in general go on and eventually generate a satisfying interpretation. This is necessary since it may e.g. happen that *T* contains a contradiction which cannot be reached from the goal by chaining through linked formulas. Note that no PROLOG knowledge base can be inconsistent. Furthermore, note that resolution calculus captures this situation by working breadth-first.

Predicate logic Suppose *T* consists of arbitrary predicate logic formulas. A naive implementation of the tableau calculus would be to substitute all Herbrand terms for all universal variables in a fixed order and then to generate a tableau from the resulting set of variable-free formulas *(level saturation).* It is one of the key observations in [Bowen82] and in [Wrightson84] that only those substitutions need to be performed which produce complementary pairs. The combination of their idea (they do not worry about the propositional case, e.g. backtracking) with those of this paper is not difficult and yields a strategy for full first order logic extending PROLOG.

4. Extensions

Our strategy may be refined in many ways. E.g. we could extend the definition of the choice set C in such a way that more than one link is taken into account. This would lead far beyond PROLOG.

Another point to be mentioned is the following. SEARCH composes tableaux starting with the root. But we might equally well try to compose them starting from the leaves. This is nothing else than resolution calculus. Using this unified view of different proof procedures, the following can be said.

- Resolution calculus generates a *theory,* i.e. a deductively closed set of formulas, and stops when it derives the empty clause.
- Tableau calculus generates *models* and stops when all cases lead to contradiction.
- In the PROLOG case, both calculi coincide since the theory to be generated consists of atomic formulas only, describing the 'minimal' model.
- Resolution may make multiple use of the same inference, of lemmas, and so reduce *proof length.*
- Tableau calculus may be extended to recognize loops in the generation of models and so reduce *proof search length* [Schonfeld83].

Both proof search approaches formalize different reasoning principles, and it seems useful to combine them. Mathematicians work this way: When given an open problem, they start by deriving some relevant facts from known theorems (resolution!). But then, they disbelieve their hypothesis and try to construct counterexamples (tableau!). They may switch back to the theorems and so on. I do not claim that their reasoning principles arc the ultima ratio. But this shows that a combination of different principles can be useful. (In addition, we should not forget Bledsoe's claim for heuristics.)

To draw a final conclusion from all these considerations - I believe that tableau calculus is not a substitute for other calculi, but a useful complement.

References

[Beth59] E.W. Beth, The foundations of mathematics, North-Holland Pub. Co., Amsterdam 1959,

[Bibel83] W. Bibel, Matings in Matrices, Communications of the ACM 26(1983), 844-852,

[Bledsoe77] W.W. Bledsoe, Non-resolution theorem proving, Artificial Intelligence 9(1977), 1-35,

[Bowcn82] K.A. Bowen, Programming with full first order logic, Machine Intelligence 10(1982), 421-440,

[Clocksin81] W.F. Clocksin, C.S. Mellish, Programming in Prolog, Springer, New York 1981,

[Herberger 80] M. Herberger, D. Simon, Wissenschaftstheorie fiir Juristen, Alfred Metzner Verlag, Frankfurt 1980,

[Malachi84] Y. Malachi, Z. Manna, R. Waldingcr, TABLOG: The deductive-tableau programming language, SRI Technical Note 328 (1984),

|Murray82] N.V. Murray, Completely non-clausal theorem proving, Artificial Intelligence 18(1982), 67-85,

[Rautcnberg79] W. Rautenberg, Klassische und nichtklassische Aussagcnlogik, Vieweg 1979,

[Richter78] M.M. Richter, LogikkalkUle, Teubner, Stuttgart 1978,

[Robinson65] J.A. Robinson, A machine-oriented logic based on the resolution principle, J. ACM 12(1965), 23-41,

[Sagiv81] Y.C. Sagiv, Optimization of queries in relational databases, UMI Research Press, Ann Arbor 1981,

[Sch6nfeld83] W. Schonfeld, Proof search for unprovable formulas, 7th German Workshop on Artif. Int. GWAI-83, Springer, 1983, 207-215,

[Wrightson84] G. Wrightson, Semantic tableaux, unification, and links, Technical Report CSD-ANZARP-84-001, University of Wellington, 1984,