CASE INFERENCE IN RESOLUTION-BASED LANGUAGES

T. Wakayama
School of Computer and Information Science
313 Link Hall, Syracuse University, Syracuse, NY 13244

T.H. Payne
Department of Mathematics and Computer Science
University of California, Riverside, CA 92521

Abstract. Informally, case inference is a type of inference that inherently involves disjunctions in deriving definite consequences. We show that a difficulty with efficient implementation of case inference in resolution-based languages stems from the fact that case inference always requires derived clauses to be reused as side clauses: in general, the number of derived clauses is quite large, and storing all of them seems unacceptably inefficient in programming language settings. However, our results also show that in retrieving definite information, this use of derived clauses is necessary only when case inference is required. This in turn leads to our next finding that storing a relatively small class of derived clauses, which is characterized in terms of certain properties of case inference, is sufficient for proving all definite consequences. We then show that a conservative approximation of this class can be, in effect, precomputed for clause sets not containing purely negative clauses.

§1. Introduction

In logic programming and deductive databases, there has been a strong interest in developing languages that can reason properly in the presence of indefinite information, i.e., information expressed as a disjunction of atomic facts [2,3,4,8,9,10,14]. It has been known that the presence of this type of information in databases causes serious computational difficulties: the efficiency of SLD-resolution is no longer available, yet more general resolution schemes are not quite efficient enough to be the basis of programming languages.

One reason, among others, that it is still desirable to keep indefinite information in databases is that it could play, in certain circumstances as illustrated below, an indispensable role in deriving definite consequences. In this study, we refer to this type of inference as case inference and identify its basic yet sufficient computational mechanism in the context of linear resolution. We then briefly discuss the feasibility of adapting it into an efficient Prolog-like environment.

To see how case inference arises naturally in application, consider the following:

(1.1) Example.

\begin{align*}
\text{symptom}(S). \\
\text{cause}(C_1) \lor \text{cause}(C_2) & \rightarrow \text{symptom}(S). \\
\text{treatment}(T_0) & \leftarrow \text{cause}(C_1).
\end{align*}
treatment(T₁) ← cause(C₁).
treatment(T₀) ← cause(C₂).
treatment(T₂) ← cause(C₂).

In addition, suppose that the presence of the symptom S requires some immediate treatment, and that the doctors cannot afford to wait for further test results which may identify the exact cause of the symptom S and allow them finer treatments (such as T₁ or T₂). The point of this example is that the present state of the knowledge base, although incomplete, is already mature enough to establish the validity of the treatment T₀, and that any inference procedure intended for such applications should be able to detect this situation.

However, this seemingly simple situation is by no means an easy one for resolution-based programming languages, e.g., try SLDNF-resolution on the above example with query ∃x treatment(x). The major reason for this is that, as we will see in §3, case inference always requires the use of derived clauses (i.e., center clauses in linear resolution including the top goal clause) as side clauses: in general, the number of derived clauses is quite large and their management is a major efficiency problem in linear resolution. However, our results also show (§3) that in retrieving definite information, the use of derived clauses is necessary only when case inference is required. This in turn leads to the finding (§4) that storing a relatively small class of derived clauses, characterized in terms of certain properties of case inference, is sufficient for proving all definite consequences. In §5, we briefly discuss how a conservative approximation of this class can be, in effect, precomputed for a set of definite and indefinite clauses.

Although any implementation of case inference in the context of linear resolution must have some way of reintroducing at least some derived clauses back to the refutation, storing them is not the only way. Loveland has shown, in his recent work on near Horn Prolog[8], that desired derived clauses can be, in effect, regenerated by reentering once-resolved subgoals. We call his approach the reentry approach, and ours the reduction approach, where reduction is, as we will see in the next section, a way of storing and utilizing derived clauses. The two approaches are quite different, and it is interesting to compare them. But a full comparison is beyond the scope of this presentation, and we will mention only a few points on this in later sections.

Throughout, we employ the following conventions:

- All upper case letters denote literals, either positive or negative.
- An A-clause is a clause that contains the literal A.
- If $\mathcal{F}$ is a formula then $\exists(\mathcal{F})$ and $\forall(\mathcal{F})$ denote the existential closure and the universal closure of $\mathcal{F}$, respectively.
- $\Gamma \models \mathcal{F}$ means that the formula $\mathcal{F}$ is a logical consequence of the set of axioms $\Gamma$ in the sense of classical first order logic.

§2. Ordered Linear Resolution and Definite Answers

The form of linear resolution we study is OL-resolution[1], which is essentially the same as SL-resolution[5] and the model elimination method [7]. In this section, we introduce basic definitions on OL-resolution and establish a preliminary result on question-answering aspects of OL-resolution.