Abstract. Forward chaining is an inference algorithm based on modus ponens which is widely used but unfortunately incomplete with respect to the usual boolean logic. A solution to this incompleteness is to compile the knowledge base so that forward chaining becomes complete for any base of facts. This compilation is called achievement and was so far restricted to propositional calculus. In this paper, we extend this compilation method to the predicate calculus. To our knowledge, this is the first exact knowledge compilation for first order logic.

1 Introduction

Forward chaining is an algorithm which is widely used because it is very simple and quite efficient. It aims at producing the set of literals which are implied by the knowledge (a set of rules or clauses) and a set of facts. This is a production algorithm which differs from query algorithms because it produces a set of implied literals instead of just answering whether or not a literal is implied. We use the notation $Fwch(B \cup F)$ for the set of literals inferred by forward chaining from the knowledge base $B$ and the facts $F$. The knowledge base can be encoded as a set of rules or a set of clauses. On a set of rules, forward chaining keeps adding to the current base of facts the conclusion of a rule whose condition is satisfied by the facts, until nothing new is proved or a contradiction is detected. On a set of clauses, it is essentially a unit-propagation. These two kinds of forward chaining are connected together by the notion of variant which generalizes the notion of reciprocal. Forward chaining on a set of clauses $B$ makes the same computation as forward chaining on rules on the variants of $B$. Both algorithms are widely used. Most expert systems use forward chaining on rules while forward chaining on clauses is also known as BCP [FdK93] and used in ATMS as well as in most satisfiability tests. In logic programming, forward chaining refers to the bottom up computation of the solutions of a program.

* This work is supported by the Conseil Régional Nord-Pas-de-Calais and the CNRS as part of the GANYMEDE II project

1 A rule is a formula of the form $l_1 \land \ldots \land l_n \rightarrow l$ where each $l$ is a literal.

2 A fact is a literal (ground or not).

3 A variant of a clause $C$ is a rule which has $C$ for clausal form.
Unfortunately, forward chaining is not complete with respect to the usual boolean logic. For example, \( \neg a \) cannot be inferred from the knowledge \( \{ a \rightarrow b, a \rightarrow c, b \land c \rightarrow d \} \) and the fact \( \neg d \) because this deduction involves a disjunction \( (\neg b \lor \neg c) \) which forward chaining cannot handle. A solution to this problem is achievement. The main idea of achievement is quite simple and is due to Philippe Mathieu. It is detailed among others in [MD90], [Mat91] and [MD94]. Achieving a base consists in adding the rules or clauses so that forward chaining becomes complete without modifying the semantics of the base. The most important point is that the achieved base must be complete for any base of fact which will be used later. This provides us with a knowledge compilation method whose cost will be eventually amortized because forward chaining is faster than a complete inference method. Several other knowledge compilations have been defined since then [SK91], [KS92], [Dal95],[dV94],[Mar95], [SK96],[Sch96],[MS96],[dV96],[MM96] but only [Dal95] deals with a production algorithm. All methods but the approximate knowledge compilations of [SK91],[dV96],[SK96] are restricted to the propositional case.

This paper describes an exact knowledge compilation method for the first order case. This compilation is dedicated to forward chaining but one should keep in mind that completeness for forward chaining implies completeness for unit-refutation which is a goal directed (or top down) inference method which can be used to prove theorems. By lack of space, we only present the most important results with some simplification and restrict ourselves to partial achievement. Full details and proofs can be found in [Rou97a] and [Rou97b].

We first give precise definitions of achievement, after which we state a necessary and sufficient condition of achievement for the first order case. Then, we study the finiteness of the achieved base in the first order case. Next, we present a method based on SOL-resolution to compute a partial achievement of a base. Total achievement is obtained by extending the cycle search method [RM96] to predicate calculus. At last, we compare our approach to the approximate knowledge compilation of [dV96].

2 Achievement

Depending on the availability of information on the kind of base of facts that will be later used, we distinguish two kinds of achievement: total or partial achievement.

Definition 1. A total achievement is an operation which, from a base \( B \), produces a base denoted \( \text{Achvt}(B) \) such that \( \| B = \text{Achvt}(B) \) (semantic equivalence) and \( \| B \cup F \models l \iff l \in \text{Fwch}(\text{Achvt}(B) \cup F) \).

A partial achievement for a set of set of facts \( F \) and a set of literals \( L \) is an operation which, from a base \( B \), produces a base \( \text{Achvt}_{F,L}(B) \) such that \( \| B = \text{Achvt}_{F,L}(B) \) and \( \| B \cup F \models l \iff l \in \text{Fwch}(\text{Achvt}_{F,L}(B) \cup F) \).