A Significant Extension of Logic Programming by Adapting Model Building Rules

Ricardo Caferra and Nicolas Peltier

LIFIA-IMAG
46, Avenue Félix Viallet 38031 Grenoble Cedex FRANCE
Ricardo.Caferra@imag.fr, Nicolas.Peltier@imag.fr

Abstract. A method by the authors for automated model building is extended and specialized in a natural way in order to increase the possibilities of logic programming. A rather complete, though reasonably short, description of the ideas and technicalities of the former method is given in order to make the paper self-contained. Specialization of several key rules permits to obtain three main theoretical results concerning extensions of logic programming: non-ground negative facts as well as inductive consequences can be deduced from programs. Goals containing negations, quantifications and logical connectives are allowed. It is proven that the proposed extension is strictly more powerful than SLDNF. Several non-trivial running examples show evidence of the interest of our approach. Last but not least, a nice side effect exploits the model building capabilities of the approach: it is shown on one representative example how the method can be used to detect (and to correct) errors in logic programs. Main lines of future research are given.

1 Introduction

Automated Deduction and Logic Programming have numerous and deep connections (see for example [ROB83, ROB92, WM91]). Although each of these two fields has its own specific aims, they often provides to each other useful ideas and techniques. The aim of this work is to show how a model building method (developed by the authors since 1990 in order to be used in Automated Deduction), can be profitably used in logic programming in order to extend its usefulness and increase its power. Model building is presently recognized as a very important topic in Automated Deduction. Although the usefulness of model building was recognized since the very beginning in Automated Deduction ([GHL83]), no systematic feasible method was proposed until the nineties (for more details see for example [BCP94]). In 1990 ([CZ90, CZ92]) we proposed a method — called RAMC\(^1\) — for simultaneous search for refutations and models for set of clauses. The method relies on the use of constrained clauses i.e. couples noted [\textit{clause} : \textit{constraints}], instead of standard clauses. The constraints codes the conditions either necessary to the application of the inference rule (resolution, factorization...) or sufficient to prevent their application. Roughly speaking, the method associates to each inference rule (resolution, factorization...)

\(^1\) standing for Refutation And Model Construction
its “dis-inference” counterpart (disresolution, disfactorization...). Dis-inference rules add to c-clauses the constraints preventing the application of the corresponding inference rule. The use of constraints can be seen as a way to perform meta-reasoning at the object level which allows to generate useful information that cannot be deduced by purely deductive approach (see [CZ92] for more details). To apply our approach to logic programming is a very natural idea: the capabilities of the method could obviously extend the capabilities of logic program interpreters, allowing for example to deduce facts that cannot be deduced by classical approaches. More precisely we show in this work that our method can be specialized in order to be profitably applied to logic programs. In particular our method allows to deduce non ground negative facts — feature not shared by negation as failure — or fact that are inductive consequences of the program. This can also in many cases prevent the non termination of programs. Moreover we propose a method in order to compute the solution of “complex” goals involving negation, connectives and quantifiers which cause trouble to most other approaches.

The rest of the paper is organised as follows: the second section is devoted to a brief — but complete — overview of our model building method. Section 3 shows how this method can be specialized in order to deal with logic programs and present a method to solve “complex” goals i.e. goals including quantification, connectives and negation. In section 4 we give several non trivial examples illustrating our approach. Section 5 is devoted to concluding remarks.

2 Simultaneous Search for Refutations and Models

The method presented here strongly relies on the use of equational constraints. Before presenting the rules of our method, we have to recall briefly some necessary results from [CL89] concerning equational problems (definition and decidability).

2.1 Equational Problems

Let \( \Sigma \) be a set of function symbols. We note \( \tau(\Sigma) \) the set of ground terms built on the signature \( \Sigma \) (i.e. the Herbrand universe). \( \bar{x}, \bar{y} \ldots \) (resp. \( \bar{t}, \bar{s} \)) denotes a vector of variables (resp. terms). \( \text{Var}(t) \) denotes the set of free variables of the term (resp. formula, clause...) \( t \). \( \mathcal{E}\{x \rightarrow t\} \) denotes the expression (term,formula...) \( \mathcal{E} \), obtained by replacing each free occurrence of \( x \) by \( t \). An equational formula \( \mathcal{P} \) is either an equation \( s = t \), where \( s \) and \( t \) are two terms, \( \perp \) (false), \( \top \) (true) or of the form: \( P \lor Q \), \( P \land Q \), \( \neg P \), \( \exists x.P \), \( \forall x.P \) where \( P \) and \( Q \) are two equational formula. The free variables \( \bar{e} \) of \( \mathcal{P} \) are called the unknowns. A substitution \( \sigma \) of variables \( \bar{e} \) is said to validate an equational formula \( \mathcal{P} \) iff the formula \( \mathcal{P}\sigma \) is true in the Herbrand universe. More precisely: A ground substitution \( \sigma \) validates an equational formula \( \mathcal{P} \) iff \( \mathcal{P}\sigma \) is valid. The set of substitutions \( \sigma \) that validates \( \mathcal{P} \) is called the set of solutions of \( \mathcal{P} \) and is noted \( \text{Sol}(\mathcal{P}) \).