How to Use Modalities and Sorts in Prolog

Andreas Nonnengart

Max-Planck-Institut für Informatik,
Im Stadtwald, 66123 Saarbrücken, Germany,
Tel.: (+49) 681 302 5369, Fax: (+49) 681 302 5401,
Email: Andreas.Nonnengart@mpi-sb.mpg.de

Abstract. Standard logic programming languages like Prolog lack the possibility of dealing with modalities and/or sorts. A first idea how to overcome this problem (and that without changing anything on Prolog itself) would be to apply the well-known relational translation approaches from modal and sorted logic into first-order predicate logic and to feed this translation result into Prolog. This, however, leads into other problems: firstly, the transformed problem is usually of much bigger size (number of clauses) than the original one and, secondly, very often it is not even in Horn form anymore.

In this paper a translation approach is proposed which avoids both of these problems, i.e. the number of clauses after translation is exactly as big as it would have been if we simply ignored the modal operators and sort restrictions and, also, the result is in Horn form provided it was already before (modulo modal operators and sorts).

1 Introduction

For many applications in the AI community (as for instance knowledge representation and taxonomic reasoning for several agents) there is a need for modalities and/or sort declarations. On the other hand, one of the most popular AI programming languages is Prolog and it is thus evident that a combination of Prolog with modalities and sorts apparently makes sense for AI implementations.

However, it is not at all obvious how modalities and sorts could be integrated into Prolog without changing Prolog itself. A first idea might be to use standard relational translation techniques into first-order predicate logic as they are perfectly well known from the literature. With these techniques it would be possible to translate modal and sorted logic programs into classical logic before being fed to the logic programming environment.

However, soon one will have to realize that there occur quite a lot of problems with this approach. First, in the extension by modalities, the translation results in an exponential explosion of the output formula. Also, certain properties of the underlying accessibility relation, as for example symmetry, result in clauses which inevitably lead to infinite loops under the Prolog control structure. And finally, for both modal logics and sorted logics, the resulting formula is not necessarily in Horn form even if the original formula was (if we ignored the modalities or the sort declarations respectively).
The translation approach proposed in this paper avoids these difficulties. Its theoretical basis has already been introduced in [4], although the possible application to logic programming has not been mentioned there (but in [3]) and the application to sorted logic has not been published before at all.

One of the most remarkable things about this approach is that it very often allows a significant simplification of the background theory given by the modal logic accessibility relation properties and sort declarations.

Note that this method is very closely related to the functional translation approach proposed by Ohlbach in [5], [6], [7] and others. At least the idea of replacing occurrences of certain binary predicates by suitable sets of functions is in essence identical to the approach presented there. Nevertheless, there are some quite important differences between the functional translation method and the one propose here. In the functional translation, theory clauses which stem from certain properties of the underlying accessibility relation get replaced by more or less complicated equations. These require either a pretty strong equality handling of the inference procedure or a theory unification algorithm which suits to the theory induced by the equations. Usually, however, neither of these features are available in standard logic programming languages like Prolog. Therefore, if we want to avoid changes in the logic programming language itself, we have to think of something else. Indeed, the translation approach presented in this paper does not require any changes in the programming language at all. To some extent it can be viewed as a mixed approach between relational and functional translation which tries to combine advantages from both and also tries to avoid their respective disadvantages as much as possible.

This paper is organized as follows: section 2 contains the basics of the relational translation for modal and sorted logics. Section 3 then explains how the functional simulation approach works and how it can be applied to modal and sorted logics. After that a certain saturation technique is introduced which allows to considerably simplify the background theories (for modal logics). Finally, the effect of the overall approach is demonstrated with two examples.

2 Relational Translation

Here and in the sequel we assume that the reader is already familiar with the basic principles of modal and sorted logics. Also, proofs are usually omitted in this section. In case the reader is interested in these s/he is referred to [4] and [3].

The idea of the relational translation is to make the implicit semantics of the extra features of the underlying logic explicit in the translation. I.e. in case of modal logics each predicate symbol, say $P$, gets replaced by a new predicate symbol, say $P'$, which accepts one more argument, namely (a symbol denoting) the current (or actual world) (similar for function symbols). In addition, two new predicate symbols are to be introduced, $R$ and $E$, which represent the accessibility relation and the domain element existence predicate respectively (in case of varying domains).