AUTOMATIC GENERATION OF SEMANTIC ATTACHMENTS IN FOL

Luigia Aiello
Computer Science Department
Stanford University
Stanford, California 94305

ABSTRACT

Semantic attachment is provided by FOL as a means for associating model values (i.e. LISP code) to symbols of a first order language. This paper presents an algorithm that automatically generates semantic attachments in FOL and discusses the advantages deriving from its use.

I INTRODUCTION

In FOL (the mechanized reasoning system developed by R. Weyhrauch at the Stanford A.I. Laboratory [4,5,6], the knowledge about a given domain of discourse is represented in the form of an L/S structure.

An L/S structure is the FOL counterpart of the logician notion of a theory/model pair. It is a triple <L,S,F> where L is a sorted first order language with equality, S is a simulation structure (i.e. a computable part of a model for a first order theory), and F is a finite set of facts (i.e. axioms and theorems).

Semantic attachment is one of the characterizing features of FOL. It allows for the construction of a simulation structure S by attaching a "model value" (i.e. a LISP data structure) to (some of) the constant, function and predicate symbols of a first order language. Note that the intended semantics of a given theory can be specified only partially, i.e. not necessarily all the symbols of the language need to be given an attachment.

The FOL evaluator, when evaluating a term (or wff), uses both the semantic and the syntactic information provided within an L/S structure. It uses the semantic attachments by directly invoking the LISP evaluator for computing the value of ground sub-terms of the term (wff). It uses a simplification set, i.e. a user-defined set of rewrite rules to do symbolic evaluations on the term (wff). Semantic information and syntactic information are repeatedly used - in this order - until no further simplification is possible.

The research reported here has been carried out while the author was visiting with the Computer Science Department of Stanford University on leave from IEI of CNR, Pisa, Italy. Author's permanent address: IEI-CNR, via S. Maria 46, I-56100 Pisa, Italy

Semantic attachment has been vital in the generation of many FOL proofs, by significantly increasing the efficiency of evaluations. The idea of speeding up a theorem prover by directly invoking the evaluator of the underlying system to compute some functions (predicates) has been used in other proof generating systems. FOL is different from other systems in that it provides the user with the capability of explicitly telling FOL which semantic information he wants to state and use about a given theory. This approach has many advantages, mostly epistemological, that are too long to be discussed here.

II AUTOMATIC GENERATION OF SEMANTIC ATTACHMENTS

It is common experience among the FOL users that they tend to build L/S structures providing much more syntactic information (by specifying axioms and deriving theorems) than semantic information (by attaching LISP code to symbols). In recent applications of FOL, L/S structures are big, and (since the information is essentially syntactic)the dimension of the simplification sets is rather large. The unpleasant consequence is that the evaluations tend to be very slow, if feasible at all.

This has prompted us to devise and implement an extension of the FOL system, namely, a compiling algorithm from FOL into LISP, which allows for a direct evaluation in LISP of functions and predicates defined in First Order Logic. The compilation of systems of function (predicate) definitions from FOL into LISP allows FOL to transform syntactic information into semantic information. In other words, the compiling algorithm allows FOL to automatically build parts of a model for a theory, starting from a syntactic description.

Semantic attachment has often been criticised as error prone. In fact, the possibility of directly attaching LISP code to symbols of the language allows the FOL user to set up the semantic part of an L/S structure in a language different from that of first order logic. This forbids him to use FOL itself to check the relative consistency of the syntactic and semantic part of an L/S structure.

The automatic transformation of FOL axioms (or, in general, facts) into semantic attachments, besides the above mentioned advantage of substantially increasing the efficiency of the evaluator, has the advantage of guaranteeing the consistency between the syntactic and semantic specifications of an FOL domain of discourse, or at least, to keep to a minimum the user's freedom of introducing non-detectable inconsistencies.

The semantic attachment for a function (predicate) symbol can be automatically generated through a compilation if such a symbol appears in the syntactic part of an L/S structure as a definiendum in a system of (possibly mutually recursive) definitions of the following form:

$$\forall x_1 \ldots x_r . f_i x_1, \ldots x_r) = \sigma_i[\Phi_i, \Psi_i, \vec{c}_i, x_1, \ldots, x_r]$$

$$\forall y_1 \ldots y_s . (P_j(y_1, \ldots y_s) = \tau_j[\Phi_j, \Psi_j, \vec{c}_j, y_1, \ldots, y_s])$$

Here the f-s are function symbols and the P-s are predicate symbols. The σ are terms in Φ-s, Ψ-s, $\vec{c}$-s and x-s; the τ-s are wffs in the Φ-s, Ψ-s, $\vec{c}$-s and y-s. By $\vec{c}$ we denote a tuple of constant symbols. By Φ (resp. Ψ) we denote a tuple of function (resp. predicate) symbols. Φ (resp. Ψ) may contain some of f (resp. P), but it is not necessarily limited to them, i.e. other function and predicate symbols besides the definienda can appear in each definiens.

The compilation algorithm, when provided with a system of definitions, first performs a well-formedness check, then a compilability check.

The well-formedness check tests whether or not all the facts to be compiled are definitions, i.e. if they have one of the two following forms (note that here we use the word "definition" in a broader sense than logicians do):

$$\forall x_1 \ldots x_r . f_i(x_1, \ldots, x_r) = \ldots$$

$$\forall y_1 \ldots y_s . (P_j(y_1, \ldots, y_s) = \ldots$$

The compilability check consists in verifying that a) each definition is a closed wff, i.e. no free variable occurs in it; b) all the individual constants and the function (predicate) symbols appearing in the definiens either are one of the definienda or are attached to a model value (the first case allows for recursion or mutual recursion); c) the definiens can contain logical constants, conditionals and logical connectives but no quantifiers.

When the FOL evaluator invokes the LISP evaluator, it expects a model value to be returned; it does not know how to handle errors occurring at the LISP level. This, for various reasons too long to be reported here, justifies the three above restrictions. Actually, the second and the third restrictions can be weakened with and appropriate extension of the FOL evaluator and of the compiler (respectively) to cope with the new situation. More details are presented in [1].

To present a simple example of compilation, consider the following facts:

$$\forall y\ x . f(x,y) = \underline{\text{if}}\ P(x)\ \underline{\text{then}}\ g(x,x)\ \underline{\text{else}}\ f(y,x)$$

$$\forall y\ x . g(x,y) = x+y$$

If we tell FOL to compile them in an L/S structure where a semantic attachment exists both for the symbol P and for the symbol + (let them be two LISP functions named C-P and PLUS, respectively), it produces the following LISP code:

```
(DE C-f (x y)
    (COND ((C-P x) (C-g x x))
          (T (C-f y x))))

(DE C-g (x y) (PLUS x y))
```

and attaches it to the function symbols f and g, respectively.

III SOUNDNESS OF THE COMPILATION

The compiling algorithm is pretty straightforward, hence, its correctness should not constitute a problem. Conversely, a legitimate question is the following: Is the compilation process sound? In other words: Who guarantees that running the FOL evaluator syntactically on a system of definitions gives the same result as running the LISP evaluator on their (compiled) semantic attachments?

The answer is that the two evaluations are weakly equivalent, i.e. if both terminate, they produce the same result. This is because the FOL evaluator uses a leftmost outermost strategy of function invocation (which corresponds to call-by-name) while the mechanism used by the LISP evaluator is call-by-value. Hence, compiling a function can introduce some nonterminating computations that would not happen if the same function were evaluated symbolically.

This, however, does not constitute a serious problem and it will be overcome in the next version of FOL. In fact, it will be implemented in a purely applicative, call-by-need dialect of LISP (note that, call-by-need is strongly equivalent to call-by-name in purely applicative languages).

IV CONCLUSION

FOL is an experimental system and, as is often the case with such systems, it evolves through the experience of its designer and users. Particular attention is paid to extend FOL only with new features that either improve its proving power or allow for a more natural interaction between the user and the system (or both) in a uniform way. The addition of the compiling algorithm sketched in the previous sections is in this spirit. This extension of FOL has been very useful in recent applications (see, for instance [2]).

Experience has shown that the largest part of the syntactic information in an L/S structure can be compiled. This suggests a further improvement to be done on FOL evaluations. The use of the compiling algorithm leads to L/S structures where (almost) all the function (predicate) symbols of the language have an attachment. Hence, the strategy of the FOL evaluator to use semantic information first (that was the most reasonable one when semantic attachments were very few and symbolic evaluations could be rather long) is in our opinion no longer the best one. In fact, sometimes, properties of functions (stated as axioms or theorems in the syntactic part of the L/S structure) can be used to avoid long computations before invoking the LISP evaluator to compute that function.

Finally, a comment on related work. Recently (and independently), Boyer and Moore have added to their theorem prover the possibility of introducing meta-functions, proving them correct and using them to enhance the proving power of their system [3]. This is very much in the spirit of the use of META in FOL and of the compiling algorithm described here.

ACKNOWLEDGMENTS

The members of the Formal Reasoning Group of the Stanford A.I. Lab are acknowledged for useful discussions. Richard Weyhrauch deserves special thanks for interesting and stimulating conversations about FOL.

The financial support of both the Italian National Research Council and ARPA (through Grant No. MDA903-80-C-0102) are acknowledged.

REFERENCES

[1] Aiello, L., "Evaluating Functions Defined in First Order Logic." Proc. of the Logic Programming Workshop, Debrecen, Hungary, 1980.

[2] Aiello, L., and Weyhrauch, R. W., "Using Meta-theoretic Reasoning to do Algebra." Proc. of the 5th Automated Deduction Conf., Les Arcs, France, 1980.

[3] Boyer, R.S., and Moore, J.S., "Metafunctions: Proving them correct and using them efficiently as new proof procedures." C. S. Lab, SRI International, Menlo Park, California, 1979.

[4] Weyhrauch, R.W., "FOL: A Proof Checker for First-order Logic." Stanford A.I. Lab, Memo AIM-235.1, 1977.

[5] Weyhrauch, R. W., "The Uses of Logic in Artificial Intelligence." Lecture Notes of the Summer School on the Foundations of Artificial Intelligence and Computer Science (FAICS '78), Pisa, Italy, 1978.

[6] Weyhrauch, R.W., "Prolegomena to a Mechanized Theory of Formal Reasoning." Stanford A.I. Lab, Memo AIM-315, 1979; Artificial Intelligence Journal, to appear, 1980.