[3] F. Giunchiglia and A. Cimatti. Introspective Metatheoretic Reasoning. In *Proceedings META-94, Fourth Intnl. Workshop on Meta-Programming in Logic*, Pisa, Italy, June 19-21, 1994. Also IRST-Technical Report 9211-21, IRST, Trento, Italy.

[4] F. Giunchiglia, P. Pecchiari, , and C. Talcott. An Analysis of the Reasoning Structures and Rules underlying the Integration of Linear Arithmetic in the Boyer-Moore Prover. Forthcoming Technical report.

[5] F. Giunchiglia, P. Pecchiari, , and C. Talcott. Reasoning Structures: An Architecture for Open Mechanized Reasoning Systems. Forthcoming Technical report.

[6] F. Giunchiglia and P. Traverso. Reflective Reasoning with and between a Declarative Metatheory and the Implementation Code. In *Proc. of the 12th International Joint Conference on Artificial Intelligence*, pages 111–117, Sydney, 1991. Also IRST-Technical Report 9012-03, IRST, Trento, Italy.

[7] F. Giunchiglia and P. Traverso. A Metatheory of a Mechanized Object Theory. Technical Report 9211-24, IRST, Trento, Italy, 1992.

[8] F. Giunchiglia and P. Traverso. Program Tactics and Logic Tactics. In *Proceedings 5th Intnl. Conference on Logic Programming and Automated Reasoning (LPAR'94)*, Kiev, Ukraine, July 16-21, 1994. Also IRST-Technical Report 9301-01, IRST, Trento, Italy.

[9] F. Giunchiglia and R.W. Weyhrauch. `FOL` User Manual - `FOL` version 2. Manual 9109-08, IRST, Trento, Italy, 1991. Also DIST Technical Report 91-0006, DIST, University of Genova.

[10] R.W. Weyhrauch. Prolegomena to a Theory of Mechanized Formal Reasoning. *Artif. Intell.*, 13(1):133–176, 1980.

about the structure of wffs. The project has turned out to be far more difficult than we had anticipated and has required multiple major recodings of `GETFOL` (which at the moment is more than one megabyte of source code) and redefinitions of MT. At the current state of the art we have succeeded in defining a metatheory MT with all the desired properties and, therefore, we have also envisaged a general schema to be followed when writing the code mechanizing OT. All of `GETFOL` has been recoded according to this general schema.

3 Conclusion

The next step ahead (already started, even if still with little success) is to use the system for synthesizing non trivial tactics. Our long term goal is to develop `GETFOL` into a system whose code is correct, whose inference steps are powerful enough to lessen the user from a lot of small easy steps, and which provides facilities for provably correct system development. We know that we are very far from it, and that many believe that this is impossible. We are hopeful. [6, 8, 3] are some published papers which report some of the work done so far. [1, 7] are some unpublished documents which describe some of this material more in detail. [5, 4] describe work which will become soon very useful. [5] is a preliminary still under revision report of the beginnings of a formal framework for the specification of interactive reasoning systems, called OMRSs (Open Mechanized Reasoning Systems). OMRSs are modeled as inference rules plus (tactic based) control. [4] is a preliminary still under revision report which applies the theory described in [5] to give a formal specification of a significant subset of NQTHM, more specifically the rewriter-simplifier- linear arithmetic theorem proving and interaction.

References

[1] A. Armando. *Architetture Riflessive per la Deduzione Automatica.* PhD thesis, DIST - University of Genoa, 1993.

[2] D. Basin, F. Giunchiglia, and P. Traverso. Automating Metatheory Creation and System Extension. In *AI*IA 1991, 2nd Conference of the Italian Association for Artificial intelligence*. Springer Verlag, 1991. Also IRST-Technical Report 9101-04.

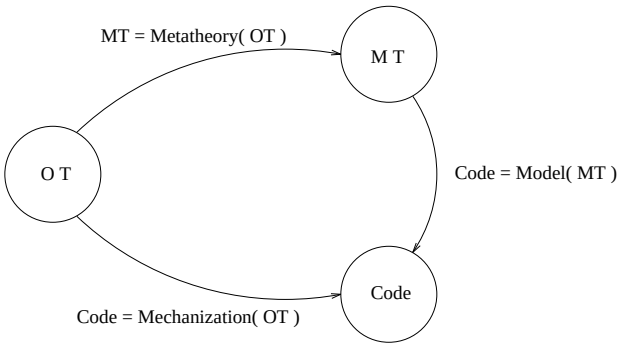

Figure 2: OT, MT and the code.

MT, this achieves the requirement that MT represent the computation which implements OT. In fact, we have the following two facts. First, the interpreted constants in MT denote objects of OT which are memorized in the data structures implementing OT. Second, the interpreted function and predicate symbols of MT correspond to the `HGKM` function symbols which compute their set-theoretic meaning. In other words, `HGKM` functions are finite presentations of the interpretations of the corresponding MT application symbols. Thus, for instance, the extension of $fandeltac$ is computed, via the `HGKM` evaluator, by `fandeltac`.

Starting from the complete re-implementation of the `FOL` system described in [9], this project has been developed as two subprojects which proceed in parallel, strongly influencing each other. The goal of the first subproject was the mechanization of OT, *i.e.* the production of code which implements OT and which, at the same time, constitutes a finite presentation of the model of MT. The second project aimed at the development of MT. These two projects have influenced each other in the sense that the mapping from computation to deduction and vice versa (and, as a consequence, *e.g.* the kind of tactics which can be written, how they can be executed and also the definition of the lifting and flattening functions) depends on the precise form of the axioms of MT and of the `HGKM` functions mechanizing OT. The problem and very interesting question rising any time we have found impossible to map MT into the code, or vice versa, was which between MT and the code had to be modified. The constraint coming from the code is that it must do what it is supposed to do, *e.g.*, test for theoremhood, assert a theorem, *efficiently*. It is our committment not to develop `GETFOL` into a toy implementation. The constraint coming from MT is that it must be usable *effectively* to, *e.g.* prove theoremhood of object level theorems, prove tactics, reason

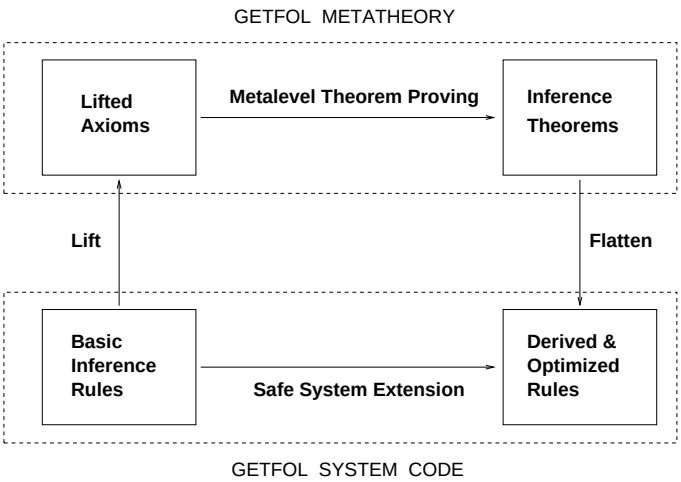

Figure 1: The lifting-reasoning-flattening cycle.

represented in figure 1 (this figure was first presented in [2]). Notice that it becomes possible to "cumulate" and compose inside a unified environment the output of code writing and metatheoretic theorem proving.

2 The development of the project

This project builds on and extends Richard Weyhrauch's work on the `FOL` system, in particular his work on meta, reflection principles, and simulation structures (where, using Weyhrauch's terminology, simulation structures are the mechanizable analogue of the notion of model) [10]. It can be described as an attempt to push the idea of linking computation in the code of a mechanized system and deduction in the system itself. From an implementational point of view, `GETFOL` has been developed on top of a reimplementation of the `FOL` system [10], described in [9]. `GETFOL` has, with minor variations, all the functionalities of `FOL` plus extensions, some of which described here, to allow for metatheoretic theorem proving. From a conceptual point of view, the close connection with Weyhrauch's work can be seen by analyzing the relation between OT, its mechanization, and MT, as shown in figure 2. MT is a metatheory of OT by construction. The code mechanizes OT by construction. The code of OT has been developed to be a finite (and partial) presentation of the model of

the metatheory represents the computation with implements deduction within the system itself. The intuition is that MT must have the following two properties. First, it must be possible to construct ground wffs and terms whose structure is in one-to-one correspondence with the computation tree (described at a certain level of abstraction) constructed by the computation carried out at the object level, *i.e.* in an object theory OT (which we suppose distinct from MT). In particular, there must be wffs, stating the provability of an object level theorem, whose structure can be put in one-to-one correspondence with the computation tree of the object level proof steps which prove the theorem itself. Second, the symbols occurring in such ground wffs and terms must have corresponding symbols in the underlying system code. In particular, this must be the case for function symbols representing inference rules and primitive tactics (*e.g.* $fandeltac$ corresponds to the `HGKM` function `fandeltac`, where `HGKM` is the implementation language and `fandeltac` performs $\wedge E_l$), for predicates representing preconditions to the applicability of inference rules (*e.g.* $Conj$ corresponds to the `HGKM` function `Conj`), and for constants denoting symbols of the language and theorems of OT (*e.g.* "A" corresponds to the data structure `A`).

The tight connection between the code and the metatheory gives the metatheory itself some "unusual" properties. Thus, for instance, the metatheory explicitly represents failure in the application of inference rules (and not only their partialness), and the fact that large amounts of the code implementing OT are partial, *i.e.* they work only for a limited class of inputs. However it also gives the metatheory some interesting properties. First, it is possible to express and theorem prove tactics, *i.e.* expressions which specify how to compose primitive tactics. Second, it is possible to give tactics a procedural content, *i.e.* to use them to assert object level theorems (possibly proofs). This can be done in two ways. Tactics can be interpreted, *i.e.* they can be given in input to an interpreter which then asserts in OT the proved theorem. Tactics can also be compiled into `HGKM` code which can then be executed to prove theorems in OT. This process of compilation is called *flattening*. Finally, it is possible to define a process, called *lifting*, which can be intuitively seen as the reverse of flattening, and which allows us to generate MT (its language and axioms) starting from the code implementing OT.

It becomes therefore possible, at least in principle, to define an iterative process where the metatheory is first lifted from the code, it is then used to prove a theorem representing an interesting tactic which is then compiled down, or possibly interpreted, in the code. As a result, derived rules can be executed like the rest of the system and used to shorten subsequent proofs. Logical manipulation at the theory level corresponds to program transformation at the system level. This reasoning cycle, which can be iterated, is schematically

First steps towards provably correct system synthesis of system code

Fausto Giunchiglia[1,2] Alessandro Armando[3]
Alessandro Cimatti[1] Paolo Traverso[1]

Mechanized Reasoning Group

[1]IRST, 38050 Povo, Trento, Italy
[2]DISA - University of Trento, Via Imana 5, Trento, Italy
[3]DIST - University of Genova, Via Opera Pia 11A Genova Italy

fausto@irst.it armando@dist.unige.it cx@irst.it leaf@irst.it

September 20, 1995

Abstract

The goal of this small paper is to give an overall informal description of a long term project, under development at the Mechanized Reasoning Group(s) at IRST and DIST, which aims at developing a system which can be used to extend its own code in a provably correct way.

1 The project

The goal of this small paper is to give an overall informal description of a long term project, under development at the Mechanized Reasoning Group(s) at IRST and DIST, which aims at developing a system which can be used to extend its own code in a provably correct way. Our approach is to build a system and a formal metatheory MT such that, informally put,

ISTITUTO PER LA RICERCA SCIENTIFICA E TECNOLOGICA
I 38100 TRENTO – LOC. PANTÉ DI POVO – TEL. 0461–814444
TELEX 400874 ITCRST – TELEFAX 0461–810851

FIRST STEPS TOWARDS CORRECT SYSTEM SYNTHESIS OF SYSTEM CODE

Fausto Giunchiglia
Alessandro Armando
Alessandro Cimatti
Paolo Traverso

May 1994
Technical Report # 9405-06

Publication Notes: Presented at the CADE-12 Workshop on Correctness and Metatheoretic Extensibility of Automated Reasoning Systems, Nancy, France, June 26, 1994.

ISTITUTO TRENTINO DI CULTURA