[Bun88] A. Bundy. The Use of Explicit Plans to Guide Inductive Proofs. In R. Luck and R. Overbeek, editors, *CADE9*. Springer-Verlag, 1988. Longer version available as DAI Research Paper No. 349, Dept. of Artificial Intelligence, Edinburgh.

[BvHH+89] A. Bundy, F. van Harmelen, J. Hesketh, A. Smail, and A. Stevens. A Rational Reconstruction and Extension of Recursion Analysis. In *Proc. 11th IJCAI conference*. International Joint Conference on Artificial Intelligence, 1989.

[BW81] A. Bundy and B. Welham. Using meta-level inference for selective application of multiple rewrite rules in algebraic manipulation. *Artificial Intelligence*, 16(2):189–212, 1981. Also available as DAI Research Paper 121, Dept. Artificial Intelligence, Edinburgh.

[CAB+86] R.L. Constable, S.F. Allen, H.M. Bromley, et al. *Implementing Mathematics with the NuPRL Proof Development System*. Prentice Hall, 1986.

[DS79] M. Davis and T. Schwartz. Metamathematical extensibility for theorem verifiers and proof-checkers. *Computer and Matemathics with Applications*, 5:217–230, 1979.

[GMW79] M.J. Gordon, A.J. Milner, and C.P. Wadsworth. *Edinburgh LCF - A mechanised logic of computation*, volume 78 of *Lecture Notes in Computer Science*. Springer Verlag, 1979.

[GSnt] F. Giunchiglia and A. Smaill. Reflection in constructive and non-constructive automated reasoning. In J. Lloyd, editor, *Proc. Workshop on Meta-Programming in Logic Programming*. MIT Press, In print. Also available as DAI Research Paper 375, Dept. of Artificial Intelligence, Edinburgh.

[GT90] F. Giunchiglia and P. Traverso. GETFOL Manual - version 1 release 1. Technical report, IRST, Forthcoming 90.

[How88] D. J. Howe. Computational metatheory in Nuprl. In R. Lusk and R. Overbeek, editors, *CADE9*, 1988.

[KC86] T.B. Knoblock and R.L. Constable. Formalized Metatheory in Type Theory. Technical Report TR 86-742, Dept. Computer Science, Cornell University, 1986.

[SS86] L. Sterling and E. Shapiro. *The Art of Prolog*. MIT Press, 1986.

[Tar36] A. Tarski. Der Wahrheitsbegriff in den formalisierten Sprachen. *Studia Philosophica*, 1:261–405, 1936. English translation in [Tarski 56].

[Tar56] A. Tarski. *Logic, Semantics, Metamathematics*. Oxford University Press, 1956.

[Wey80] R.W. Weyhrauch. Prolegomena to a theory of Mechanized Formal Reasoning. *Artificial Intelligence. Special Issue on Non-monotonic Logic*, 13(1), 1980.

[Wey82] R.W. Weyhrauch. An example of FOL using Metatheory. Formalizing reasoning systems and introducing derived inference rules. In *Proc. 6th Conference on Automatic Deduction*, 1982.

7 Conclusion

In this paper we have shown how a metatheory of a mechanized logic can be defined and used to reason about proof plans. In particular proof plans can be

- built by theorem proving. Doing so we are guaranteed that, provided that the metatheory is correct and truthful proof plans *are built correct.*
- executed *without adding or modifying the underlying code.* The resulting system is thus able to acquire new theorem proving capabilities without any addition and/ or modification of the underlying code.

The resulting architecture is *uniform* in the sense that a tower of metatheories can be defined, *each using the same code and the same commands* (natural deduction and derived inference rules), each such that plan formation at one level can be obtained by plan execution one level up. Note that the commands are used to reason about the code that implements them.

Currently, work is in progress to prove the correctness (we have a proof of this) and other properties of the system using the reflection principle, but without considering the issue of mechanization. A truthful metatheory of a mechanized logic is currently being studied.

Possible extensions of the work described in this paper are to study how proof plans can be compiled down into the code; how to optimize the planning part (a related issue is to consider proof plans representing possibly failing tactics) and how to take advantage (if possible and feasible) of the tower of metatheories by reasoning at an arbitrary level.

Acknowledgements

This work started when the first author was at the AI Department of Edinburgh University. The research described in this paper owes a lot to the openess and sharing of ideas which exists in the Mathematical Reasoning group in Edinburgh and Mechanized Reasoning group in Trento. Alan Bundy and Richard Weyhrauch have provided, in different moments, invaluable support; without them the research described in this paper could have never been done. The authors also thank Alessandro Armando, Alessandro Cimatti, Frank van Harmelen, Luciano Serafini, Alex Simpson, Alan Smaill, Andrew Stevens, Caroline Talcott and Toby Walsh.

References

[BM81] R.S. Boyer and J.S. Moore. Metafunctions: proving them correct and using them efficiently as new proof procedures. In R.S. Boyer and J.S. Moore, editors, *The correctness problem in computer science*, pages 103–184. Academic Press, 1981.

[BSG+88] A. Bundy, D. Sannella, F. Giunchiglia, F. Van Harmelen, J. Hesketh, P. Madden, A. Smaill, A. Stevens, and L. Wallen. Proving properties of logic programs: A progress report. In *1988 Alvey Conference*, pages 131–133, 1988. Also DAI research paper No. 361, Dept. Artificial Intelligence, Edinburgh.

predicates (such as $Prov$) and terms (such as $\lceil\phi\rceil$) [GSnt]; the explicit link in the code does not exist.

As Boyer and Moore do [BM81], we use functions defined in the underlying code. On the other hand, in our approach the underlying code is not modified and/ or extended [16]. In Boyer and Moore's approach, the search strategies are *proved correct* by theorem proving. In our approach, provided that the metatheory is correct and truthful, proof plans are *built correct* by theorem proving.

The work most closely related to ours is Alan Bundy's [Bun88] and Richard Weyhrauch's [Wey82]. In the Mathematical Reasoning Group in Edinburgh a metatheoretic proof planner (called CLAM) has been built on top of OYSTER, the Edinburgh re- implementation of the NuPrl system [CAB+86]. One way of seeing this work is as follows [Bun88]: given a basic set of metatheoretic tactics, described declaratively with "*methods*", how can they be composed at run time (namely with the knowledge of the goal to be proved) to build a global strategy (the proof plan)? Before performing any object level theorem proving activity, a phase of *proof planning* in the metatheory is performed. The motivations underlying our work are very similar to those described in [Bun88]. In both cases the goal is to understand (mathematical) reasoning. On the other hand, so far, the work in Edinburgh has been more concentrated on (and been successful in) increasing the library of proved theorems, (see also [BvHH+89]) while we have focused more on representational issues. We have focused on obtaining uniformity and using the same logic at all levels (while the Clam's metalogic is different from the object logic) and have studied how plan formation can be done by theorem proving. From the point of view of (proof) planning, even if the information carried by PP-wffs is similar to that carried by the methods used in Clam, there are some differences. In our approach the structure of the object level deduction is explicitly represented in the metalogic (by the proof plan term) thus possibly allowing bottom up reasoning. Clam's methods have a tactic slot. Clam's proof plans have weakened preconditions (to achieve efficiency at planning time) and may fail (at running time). PP-wffs are metalevel representations of derived inference rules. This maybe a problem as, in certain applications (mathematical reasoning is one of these), top down proof planning with derived inference rules seems as costly as performing the search directly in the object space. That this is always the case, for instance with bottom up proof planning, is not obvious. We are currently investigating the problem.

The approach used in this paper is clearly strictly connected to the fact that we are using (a re-implementation of) the FOL system [Wey80] and builds on the work described in [Wey82]. The major improvements we see with respect to the work described in [Wey82] are as follows. We have explained why (something very similar to) the old FOL evaluator is the correct machinery to "execute" a proof plan (this topic is much more general and is not totally faced here). We have introduced a metatheory for deducibility in a mechanized logic and provided a general methodology for using it to plan (effectively and efficiently) object level proofs by theorem proving. Notice that in [Wey82] the metatheoretic functions were "attached" [Wey80] to expression manipulation routines and **not** to the inference rule functions. This shift is substantial since, because of that, we have been able to make effective use of the notion of mechanization and to argue for the correctness of the proof plans. This has required an almost total re-thinking of the old FOL expression manipulation routines and inference rules' code.

[16] Not considering the possibility, not exploited yet, of compiling proof plans into code.

```
BASIC_PLAN2:
forall f1 f2 tb1 tb2 succ t1 t2 t3 v t. (T(f1) and T(f2) and
 FORALL(wof(f1)) and BASEEQ(wof(f1),ZERO) and
 FORALL(wof(f2)) and STEPEQ(wof(f2),succ) and
 EQWFF(allewff
        (mkforall(v,hypot(matrix(mk-ripple-wff(wof(f2),succ,t1,t2,t3)))),
        ZERO),
       mk-base-wff(wof(f1),tb1,tb2))                    and
 EQWFF(allewff
        (mkforall(v,hypot(matrix(mk-ripple-wff(wof(f2),succ,t1,t2,t3)))),
        t),
       concl(matrix(mk-ripple-wff(wof(f2),succ,t1,t2,t3))))
  imp T(basic-plan(f1,f2,tb1,tb2,succ,t1,t2,t3,v,t)))
```

We can now execute `BASIC_PLAN2` (or its less efficient version `BASIC_PLAN1`) to derive the object level theorem:

```
GETFOL:: reflect BASIC_PLAN2,plus-base,plus-step,y,y + z,s,x,y,z,x,s(x);
  1   forall x y z. (x + (y + z) = (x + y) + z)
```

6 Related work

One of the key issues faced in the literature of metatheoretic theorem proving has been that of extensibility, namely of how, by using metatheoretic capabilities, the inference mechanism could be improved without endangering the soundness of the system. Following a distinction first made in [BM81] all the attempts can be divided in two broad classes. In the first class (containing [BM81, DS79, KC86, Wey82]), to preserve soundness, the procedures should be proved correct in a formalized metatheory, in the second (containing [CAB+86, GMW79]) a mechanism is provided which guarantees that any added metatheoretic tactic, written in terms of primitive inference rules, will preserve correctness.

From this perspective our approach can be seen as part of class two above: provided that the metatheory is correct and correctly defines the object theory (from [GSnt] we call this property **truthfulness**) the newly built metalevel strategies, which are derived inference rules and extend the theorem proving capabilities of the system, are guaranteed to be correct.

Our metatheory is fully declarative and based on a logical (first order) formalism. This is where we differ from all the work based on the use of tactics and ML [CAB+86, GMW79]. In our approach we are able to reason about the explicit representation of search strategies by metatheoretic theorem proving.

The major difference with the other work on formalized logical metatheories (such as [How88, KC86]) is that in our axiomatization the issue of mechanization is central. In the work described in [How88, KC86] the metatheory is independent of how the underlying code is structured. It is important to remember that it is because we have defined a metatheory of the mechanized logic that we are able to see the implementation code as its natural procedural interpretation. This has been achieved (among other things) by having the link *name-object* explicit (in GETFOL's code). In classical approaches everything works correctly (*ie.* the reflection principle) because of the properties carried by the various

```
GETFOL:: reflect INDUCTION 1 2 x s(x);
  3   forall x y z. (x + (y + z) = (x+ y) + z)
```

We first execute the base and the ripple out PP-wff on the base equation and on the step equation. The induction step is executed on the facts created at the object level by the two previous reflections (the facts labeled 1 and 2).

In the example above the control was still (partially) at the object level. The third (and last) approach is to plan by metalevel theorem proving (following the methodology described previously). The proof of the composition of the three PP-wffs BASE, RIPPLE and INDUCTION has the general structure described in section 3: two $\forall E$s are performed on INDUCTION, the first substituting `f1` with `base(f1,tb1,tb2)`, the second substituting `f2` with `ripple(f2,succ,t1,t2,t3)`. A $\forall E$ is then applied to BASE and RIPPLE to propagate back the INDUCTION preconditions (this is done by a sequence of applications of $\supset E$s and then of $\supset I$s). The final step is to apply $\forall I$s to obtain the first version of the basic plan:

```
BASIC_PLAN1:
forall f1 f2 tb1 tb2 succ t1 t2 t3 v t. (T(f1) and T(f2) and
 FORALL(wof(f1)) and BASEEQ(wof(f1),ZERO) and
  FORALL(wof(f2)) and STEPEQ(wof(f2),succ)  and
   FORALL(wof(ripple(f2,succ,t1,t2,t3)))      and
    IMP(matrix(wof(ripple(f2,succ,t1,t2,t3)))) and
     EQWFF(allewff
            (mkforall(v,hypot(matrix(wof(ripple(f2,succ,t1,t2,t3))))),
            ZERO),
           wof(base(f1,tb1,tb2)))                    and
       EQWFF(allewff
              (mkforall(v,hypot(matrix(wof(ripple(f2,succ,t1,t2,t3))))),
              t),
             concl(matrix(wof(ripple(f2,succ,t1,t2,t3))))
                 imp
   T(induction(base(f1,tb1,tb2),ripple(f2,succ,t1,t2,t3))))
```

So far we have done theorem proving without postconditions. Using appropriate postconditions for the ripple out deduction function the two conjuncts

```
FORALL(wof(ripple(f2,succ,t1,t2,t3)))
IMP(matrix(wof(ripple(f2,succ,t1,t2,t3))))
```

can be matched and eliminated. Then, by using further postconditions, `ripple` and `base` can be replaced with alternative functions on their wffs (`mk-ripple-wff`, `mk-base-wff`). At this point it is possible to define the deduction function `basic-plan` as the composition of `base`, `ripple` and `induction`.

```
GETFOL:: forall f1 f2 tb1 tb2 succ t1 t2 t3 v t.
     (basic-plan(f1,f2,tb1,tb2,succ,t1,t2,t3,v,t) =
           induction(base(f1,tb1,tb2),ripple(f2,succ,t1,t2,t3)))
```

The result of this phase of metalevel theorem proving with postconditions is the following new version of the basic plan:

$$\frac{\forall v \forall w(k(s(v),w) = s(k(v,w)))}{\forall x(\forall y \forall z(k(x,k(y,z)) = k(k(x,y),z)) \supset (\forall y \forall z(k(s(x),k(y,z)) = k(k(s(x),y),z)))} \quad (20)$$

$$\frac{A[x,0] \qquad \forall x(A \supset A[x,succ(x)])}{\forall x A} \quad (21)$$

The base and ripple out rules are defined to generate the base and step cases of the induction tactic. In the induction rule, the notation $A[x,t]$ means that $A[x,t]$ is the formula obtained from A substituting the variable x with the term t.

We have then represented these inference rules as PP-wffs in the metatheory as follows:

```
GETFOL:: AXIOM BASE: forall f t1 t2.
    (T(f) and FORALL(wof(f)) and BASEEQ(wof(f),ZERO)
                imp T(base(f,t1,t2)));

GETFOL:: AXIOM RIPPLE: forall f succ t1 t2 t3.
    (T(f) and FORALL(wof(f)) and  STEPEQ(wof(f),succ)
                imp T(ripple(f,succ,t1,t2,t3)));

GETFOL:: AXIOM INDUCTION: forall f1 f2 v t. (T(f1) and T(f2) and
  FORALL(wof(f2)) and IMP(matrix(wof(f2))) and
       EQWFF(allewff(mkforall(v,hypot(matrix(wof(f2)))), ZERO),
             wof(f1)) and
         EQWFF(allewff(mkforall(v,hypot(matrix(wof(f2)))),t),
               concl(matrix(wof(f2))))
                imp T(induction(f1,f2)));
```

In the inductiom axiom, the fifth conjunct (`EQWFF (.)`) states that, if the first premise is $A[x,0]$, then the hypothesis of the matrix of the second fact is A. The sixth conjunct (`EQWFF(.)`) states that $A[x,succ(x)]$ is the conclusion of the matrix of the second fact.

$+$ is then recursively defined as follows:

```
GETFOL:: axiom plus-base : forall v. (zero + v = v);
GETFOL:: axiom plus-step : forall v w. (s(v) + w = s(v + w));
```

To prove the goal we can use different strategies. The first obvious method is to work completely at the object level and apply its inference rules. We do not consider this approach. The second, more interesting strategy, is to forget about the object level rules and, sitting in the object theory, build the proof by reflecting down the metatheoretic PP-wffs described above [15]. In GETFOL this can be done as follows:

```
GETFOL:: reflect BASE plus-base y  y + z;
  1   forall y z. (0 + (y + z) = (0 + y) + z)

GETFOL:: reflect RIPPLE plus-step s x y z;
  2   forall x. ((forall y z. (x + (y + z) = (x + y) + z)) imp
                 (forall y z. (s(x) + (y + z) = (s(x)+ y) + z)))
```

[15] It can be proved that, defining the metatheory as described in this paper all the object level commands can be simulated by reflecting down certain metalevel PP-wffs.

2. compute the object level fact whose name is the argument of T. Note that this can be performed by an interpreter I such that:

$$I(\text{“}n\text{”}) = n$$
$$I(g(n_1, ..., n_p)) = I(g)(I(n_1), ..., I(n_p))$$
$$I((h_m \circ ... \circ h_1)(n_1 ... n_p)) = (I(h_m) \circ ... \circ I(h_1))(I(n_1), ..., I(n_p))$$

   Note that I performs exactly what is defined to be the interpretation of function symbols in a first order model;

3. from $\vdash T(\text{“}A\text{”})$ in the metatheory infer $\vdash A$ in the object theory (in other words, apply a reflection principle).

The point is thus how to perform the basic interpretations, in other words $I(\text{“}n_i\text{”})$ and $I(\text{“}h_i\text{”})$. In a metatheory of a mechanized theory the object level theory is seen intensionally, in how it is implemented. **Thus the elements of the object level theory are the data structures and code implementing it**. If n_i is a fact then it will be the data structure implementing it, while $I(h_i)$ will be the code implementing the inference rule function denoted by h_i.

All this discussion can be generalized to give a procedural interpretation of proof plans in their most general form (as in (6)). This is not done here for lack of space. One point is worth noting here. We have spoken of using a reflection principle. Reflection principles, it is well known, are dangerous and can make a theory inconsistent (provided it contains enough arithmetics) [GSnt]. We do not have here a full reflection principle, in particular the metatheory uses a language which is distinct from that of the object theory and it is impossible to create dangerous self-references (a paper on the topic is forthcoming).

Notice that, in order to implement I, the system must have in its code an explicit way to remember the pairs < quotation-mark name, denoted object >. The code performing the above steps is implemented in GETFOL and can be run with the command `REFLECT`. In [GSnt] the details of the implementation in GETFOL of `REFLECT` are reported (but see also [Wey80]).

5 An example

Let us consider a simplified version of the BMTP (as described in [Bun88] [13]) as composed of the *base tactic*, the *ripple out tactic* and the *induction tactic*. The goal is to see how the BMTP can be used to prove the associativity of +. To implement the BMTP in GETFOL we have developed three object level inference rules corresponding to the three tactics (GETFOL has no built-in theoretic axiom/inference rules). The *base*, *ripple* and *induction* inference rules have been respectively implemented to perform as follows [14]:

$$\frac{\forall v(k(0, v) = v)}{\forall x \forall y(k(0, k(x, y)) = k(k(0, x), y))} \tag{19}$$

[13] With the difference that here inference rules are defined to work from the axioms to the goal.

[14] For lack of space and sake of simplicity, the code of the inference rules and how dependencies are handled is not described here. Moreover *base* and *ripple* are described for how they work with + (not considering the general case).

the signs of which each single word is composed and the order in which these signs and words follow one another. Thus if "A", "B" and "C" are names respectively of A, B and C then the structural- descriptive name of $((A \supset B) \wedge (B \supset C)) \supset (A \supset C)$ is $mkimp(mkand(\ mkimp(\text{“}A\text{”}, \text{“}B\text{”}), mkimp(\text{“}B\text{”}, \text{“}C\text{”})), mkimp(\text{“}A\text{”}, \text{“}C\text{”}))$, where, for instance, $mkimp$ is a function symbol such that $mkimp(\text{“}A\text{”}, \text{“}B\text{”})$ is understood in the metalogic as saying that we have A followed by $\supset$ and then by B (in other words $mkimp(\text{“}A\text{”}, \text{“}B\text{”})$ represents $A \supset B$).

Notice that for the basic symbols of the language (*ie.* predicate symbols) we can have only quotation-mark names while for composite ones (*ie.* wffs) we can have both kinds of names.

Rewording what said above, structural-descriptive names describe how to obtain the denoted object by "constructing" its name starting from quotation-mark names. **All the components of the name are structurally linked in a way which is isomorphic to the way the objects denoted by the components are composed to build the object denoted by the name.**

The metatheory we are here interested in is concerned with deductions, namely with facts and with how to obtain new facts from old ones. Thus we will have quotation-mark and structural-descriptive names of facts. It is easy to think of the quotation-mark name of a fact, it can be for instance a string containing the wff, the assumptions, some notation for giving it a position in the deduction tree, all of this between quotation marks. But what is a structural-descriptive name of a fact? We have to find the operation denoted by the metatheoretic structural link. When building the name of an expression, the metatheoretic structural links denote string concatenation (in fact, by string concatenation a new string is obtained from old ones). When building a theorem, the metatheoretic structural links denote derivability (in fact, by applying an inference rule a new fact is obtained from old ones). Thus the structural- descriptive name of a fact is a function which says how to deduce it from some basic facts, **it is the proof plan term**, occurring as an argument of the predicate symbol T. Thus, for instance, the name of the theorem $\vdash ((A \supset B) \wedge (B \supset C)) \supset (A \supset C)$ is $impi(\text{“}(A \supset B) \wedge (B \supset C)\text{”}, impi(\text{“}A\text{”}, third_r\ (third_l(\text{“}A\text{”}, \text{“}(A \supset B) \wedge (B \supset C)\text{”}), \text{“}(A \supset B) \wedge (B \supset C)\text{”})))$ (where $impi$ is the inference rule function of $\supset I$) namely $PP\_TRANSITIVE$'s deduction function applied to A and $(A \supset B) \wedge (B \supset C)$.

$$T(impi(\text{“}(A \supset B) \wedge (B \supset C)\text{”}, impi(\text{“}A\text{”}, third_r(third_l(\text{“}A\text{”}, \text{“}(A \supset B) \wedge (B \supset C)\text{”}), \text{“}(A \supset B) \wedge (B \supset C)\text{”}))))$$

is thus an alternative notation for:

$$T(\text{“}((A \supset B) \wedge (B \supset C)) \supset (A \supset C)\text{”})$$

More in general, the wff in equation 18 is an alternative notation for:

$$T(\text{“}(K_m \circ ... \circ K_1)(A_1, ..., A_n)\text{”})$$

We can thus use a metatheoretic proof plan to assert an object level theorem performing the following steps:

1. perform a $\forall E$ on the proof plan and substitute the bound variables with names of object level facts;

PP_THIRD_L from the postconditions of the IR-wffs that compose it (PP_IMPE and PP_ANDE_L):

$$IMPEWFF: \quad \forall f_1 \forall f_2 (wof(impe(f_1, f_2)) = concl(wof(f_2))) \tag{15}$$

$$ANDEWFF: \quad \forall f (wof(ande_l(f)) = lfand(wof(f))) \tag{16}$$

A $\forall E$ can be applied on $IMPEWFF$ (15) substituting f_2 with $ande_l(f_2)$. The consequence wff contains $impe(f_1, ande_l(f_2))$. It can be replaced by $third_l(f_1, f_2)$ for (7). The term $wof(ande_l(f_2))$ can be replaced by $lfand(wof(f_2))$ using ANDEWFF in (16). The proved postcondition is thus:

$$THIRDWFF: \quad \forall f_1 \forall f_2 (wof(third_l(f_1, f_2)) = concl(lfand(wof(f_2)))) \tag{17}$$

4 Executing proof plans

To simplify matters let us consider a simple proof plan with no preconditions and no requirements on the premises:

$$\forall f_1 ... \forall f_n T((k_m \circ ... \circ k_1)(f_1, ..., f_n))$$

where $k_m, ..., k_1$ are metatheoretic representations of m deduction functions $K_m, ..., K_1$ such that $(K_m \circ ... \circ K_1)(f_1, ..., f_n)$ is a proof tree (all these hypotheses can be dropped and the treatment generalized). Let us then suppose that a sequence of $\forall E$s has been performed to obtain:

$$T((k_m \circ ... \circ k_1)(a_1, ..., a_n)) \tag{18}$$

where $a_1, ..., a_n$ are metalevel individual constants which are representations of n facts $A_1, ..., A_n$. The term $(k_m \circ ... \circ k_1)(a_1, ..., a_n)$ says that the result of the application of the composed function $K_m \circ ... \circ K_1$ to the facts $A_1, ...A_n$ is an object level theorem.

Now, let us concentrate on a (only apparently) different topic. Tarski, in order to study the defineability of the truth predicate in (certain) formalized languages, introduced the notion of formal metatheory [Tar36] . In order to state some properties of the object theory he gave himself the ability to mention object level syntactic objects, namely he gave himself **names for the elements of the lexicon**. In doing so he distinguished between two particular kinds of names:

- **quotation mark names**. By quotation-mark name he meant "*... every name of an expression which consists of quotation marks and such that the named expression lies between them*". Thus the quotation mark name of the individual constant a is "a" [11], of the wff $A \wedge B$ is "$A \wedge B$", that of the wff $((A \supset B) \wedge (B \supset C)) \supset (A \supset C)$ (proved by $PP\_TRANSITIVE$) is "$((A \supset B) \wedge (B \supset C)) \supset (A \supset C)$" [12].

- **structural-descriptive names**. By structural- descriptive name he meant every name *... which describes the words which compose the expression denoted, as well as*

[11] To be correct, throughout the paper we should have put an extra pair of quotes around connotative uses of (occurrences of) names.

[12] The fact that a quotation- mark name has quotation marks in it is, of course, not relevant. What is relevant is that it is uniquely defined (there cannot be two objects with the same quotation- mark name) and indivisible (in a logical language, it must be an individual constant).

- Independently of the level of metatheory and of the wff to be deduced, proof plans have always the same shape. As a particular case, the representation of the metatheoretic PP-wff's proof is a PP-wff, too. The formula in (13) is exactly of the form given in (6) (the definition of a generic PP-wff). $PP\_PLANNING$ has the same *syntactical* form as PP_THIRD_L, the deduction it represents. This argument can be iterated: *independently of the level, all deductions can be represented as PP-wffs. The process of building proof plans is uniform over the levels.*

- We noticed above that the process of "appending" a deduction to another has some general characteristics. The sequence of inference rule applications does not depend on the particular predicates in the formulae, but on their main symbols. Thus we noticed that we have always a "symmetrical" sequence of $\forall E$s, $\supset E$s, $\supset I$s and finally $\forall I$s. The same code, implementing the appending of two proof plans can be used uniformly anywhere in the system. This fact, together with the fact that proof plans have always the same form at all levels, means that, *not only can we use the same deductive machinery at all levels, but also the same strategy.* The difference is that at the meta level the search space is composed of inference rules, one level up it is composed of inference rules on proof plans (namely on strategies for composing inference rules), two levels up of inference rules on strategies for composing strategies for composing inference rules etc. etc.

We have preconditions about leaves, but nothing is explicitly stated about the properties of the endformula , the **postconditions**. In the planning literature (and here too) two (proof) plans are composed, inside a more complex (proof) plan, by (partially) "matching" the preconditions of one with the postconditions of the other. The result of this process (when successful) is the construction of a (proof) plan whose preconditions and postconditions are satisfied by the goal and the theory. In this framework **proof plans' postconditions are metatheoretic wffs**. In order to show briefly how this "matching" is performed, let us consider a simple deduction, a sequence of $\wedge I$ and $\wedge E$. The composition without postconditions would deduce:

$$\forall f_1 \forall f_2 (T(f_1) \wedge T(f_2) \wedge CONJ(wof(andi(f_1, f_2))) \supset T(ande_l(andi(f_1 f_2))))$$

One of $\wedge I$'s postcondition wffs is:

$$ANDIWFF: \quad \forall f_1 \forall f_2 (CONJ(wof(andi(f_1, f_2)))) \tag{14}$$

$ANDIWFF$ can be "matched" with PP_ANDE_L (8) preconditions in order to obtain:

$$\forall f_1 \forall f_2 (T(f_1) \wedge T(f_2) \supset T(ande_l(andi(f_1, f_2))))$$

Notice that this is exactly how redundant preconditions can be dropped in a proof plan thus making execution more efficient. The precondition $CONJ(wof(andi(f_1 f_2)))$ is redundant as, forall f_1 and forall f_2 it is always true that the main symbol of the consequence of a $\wedge I$ rule is a $\wedge$. By theorem proving on the postconditions, part of the process of *plan execution* is *simulated* and "pre-computed" (thus making execution more efficient) at plan formation.

The postconditions of newly formed proof plans can be derived by metalevel theorem proving from the postconditions of the given proof plans (at the bottom level, the IR-wffs). Let us briefly describe this process by an example. Let us derive some postconditions of

hypotheses are preconditions on leaf nodes of the former. These deductions are in normal form and "symmetrical" (as it should be, since all the PP-wffs have the same shape).

To build the new PP-wff (PP_THIRD_L) we can use different strategies. One method, explained above, is to derive it by applying inference rules step by step at the metalevel. Another approach is to represent the proof of the PP-wff PP_THIRD_L as we did for object theory proofs. This proof will be represented by a new PP-wff, let us call it $PP\_PLANNING$. Thus, as we briefly said in the introduction (but see also section 4), PP_THIRD_L will be proved simply by "executing" $PP\_PLANNING$.

In order to keep the explanation simpler, let us suppose that the hypotheses of the PP-wffs are written as a sequence of implications rather than as a conjunction [10]. To make the reading easier we define the following deduction functions:

$$\forall f_1 \forall f_2 \forall f_3 \forall x_1 \forall x_2 \forall x_3 \forall p_1 \forall p_2 \forall p_3 \\ (imp\_alle(f_1, f_2, f_3, x_1, x_2, x_3) = \\ impi(f_3, impe(impe(f_3, alle(f_1, x_1, p_1)), alle(alle(f_2, x_2, p_2), x_3, p_3))))$$

$$\forall f_1 \forall f_2 \forall f_3 \forall x_1 \forall x_2 \forall x_3 \forall p_1 \forall p_2 \forall p_3 \\ (alli\_imp\_alle(f_1, f_2, f_3, x_1, x_2, x_3) = \\ alli(imp\_alle(f_1, f_2, f_3, x_1, x_2, x_3), x_2, p_2))$$

The proof of PP_THIRD_L, can thus be represented as follows:

$$\begin{array}{l} PP\_PLANNING: \\ \forall f_1 \forall f_2 \forall f_3 \forall x_1 \forall x_2 \forall x_3 \forall p_1 \forall p_2 \forall p_3 \quad (T(f_1) \wedge T(f_2) \wedge T(f_3) \wedge \\ \quad FORALL(wof(f_1)) \wedge FORALL(wof(f_2)) \wedge \\ \quad IMP(wof(alle(f_1, x_1, p_1))) \wedge \quad HP(wof(f_3), wof(alle(f_1, x_1, p_1))) \wedge \\ \quad IMP(wof(alle(alle(f_2, x_2, p_2), x_3, p_3))) \wedge \\ \quad HP(wof(impe(f_3, alle(f_1, x_1, p_1))), wof(alle(alle(f_2, x_2, p_2), x_3, p_3))) \wedge \\ \quad FREE(p_2, wof(imp\_alle(f_1, f_2, f_3, x_1, x_2, x_3))) \wedge \\ \quad \neg FREEIN(p_2, wffsof(deps(imp\_alle(f_1, f_2, f_3, x_1, x_2, x_3)))) \wedge \\ \quad FREE(p_1, wof(alli\_imp\_alle(f_1, f_2, f_3, x_1, x_2, x_3))) \wedge \\ \quad \neg FREEIN(p_1, wffsof(deps(alli\_imp\_alle(f_1, f_2, f_3, x_1, x_2, x_3)))) \\ \supset T(alli(alli\_imp\_alle(f_1, f_2, f_3, x_1, x_2, x_3), x3, p1))) \end{array} \tag{13}$$

Of course the process could be iterated. Thus, for instance, to prove $PP\_PLANNING$ we can use the inference rules or use a proof plan. **Plan execution at one level results in the formation of a plan one level below.**

Three observations are worth making. All of them point out (with increasing strength) the **uniformity** of the approach.

- The first point is the most obvious and it holds for most of the work previously done with formal metatheories. Deductions in the object theory are represented in the metatheory as (metatheoretic) wffs. The proof theory is represented with objects which can be manipulated by the proof theory itself. From the point of view of theorem proving an interesting consequence is, on the other hand, *that the same deduction/ decision procedures* (*i.e* inference rules and derived inference rules) *can be used at any level.*

[10] To complete this proof we need the PP-wff in the former form. This hypothesis avoids carrying around the deduction function of the proof that $A \wedge B \supset C$ yields $A \supset (B \supset C)$.

(some of which may be PP-wffs). Different theorem proving strategies can be adopted. For instance deduction functions can be (functionally) composed (in a sort of *top down approach*). This approach is similar to that used in [Bun88]. The idea is to do reasoning on the preconditions and (when existing) on the postconditions (see later). Another possible strategy is (in a sort of *bottom up approach*) to look at proof plan terms, to reason on their structure and to derive (by some kind of reasoning by analogy) "similar" proof plans.

We hint here only how the top down approach can be realized. In the final part of this section we will also (briefly) explain how postconditions can be used.

Let us consider how PP_THIRD_L can be obtained from the composition of PP_ANDE_L and PP_IMPE. a_1, a_2 are parameters, PP_ANDE_L's proof plan is defined as:

$$PP\_ANDE_L : \forall f(T(f) \wedge CONJ(wof(f)) \supset T(ande_l(f))) \tag{8}$$

We apply first a $\forall E$ to PP_IMPE (described in (1)) by substituting f_1 with a_1 and f_2 with $ande_l(a_2)$, thus obtaining (9). We apply then a $\forall E$ on (8) replacing f with a_2 (thus obtaining (10).

$$\begin{array}{c} T(a_1) \wedge T(ande_l(a_2)) \wedge IMP(wof(ande_l(a_2))) \wedge HP(wof(a_1), wof(ande_l(a_2))) \\ \supset T(impe(a_1, ande_l(a_2))) \end{array} \tag{9}$$

$$T(a_2) \wedge CONJ(wof(a_2)) \supset T(ande_l(a_2)) \tag{10}$$

Notice we have $T(ande_l(a_2))$ as the conclusion of (10) and as a conjunct of the hypotheses of (9). From (10) we can derive $T(ande_l(a_2))$ depending on $T(a_2) \wedge CONJ(wof(a_2))$. (9) can be rewritten as a sequence of implications, thus $\supset E$ can be applied to $T(ande_l(a_2))$ and (9) to obtain:

$$\begin{array}{c} T(a_1) \wedge IMP(wof(ande_l(a_2))) \wedge HP(wof(a_1), wof(ande_l(a_2))) \\ \supset T(impe(a_1, ande_l(a_2))) \end{array} \tag{11}$$

depending on $T(a_2) \wedge CONJ(wof(a_2))$. With an $\supset I$ with arguments (11) and $T(a_2) \wedge CONJ(wof(a_2))$ (discharging $T(a_2) \wedge CONJ(wof(a_2))$) and a subsequent $\forall I$ we obtain:

$$\begin{array}{c} \forall f_1 \forall f_2 (T(f_1) \wedge T(f_2) \wedge CONJ(wof(f_2)) \wedge \\ IMP(wof(ande_l(f_2))) \wedge HP(wof(f_1), wof(ande_l(f_2))) \\ \supset T(impe(f_1, ande_l(f_2)))) \end{array} \tag{12}$$

which is what we wanted. The proved theorem is a new PP-wff. It is an equivalent version of PP_THIRD_L (5) (in our metatheory, if f is a fact then $wof(ande_l(f)) = lfand(wof(f))$ holds).

The shown proof has some characteristics which are valid in general. You usually have to start with a sequence of $\forall E$s and replace variables representing leaf nodes with terms representing intermediate nodes (deduction functions applied to leaf nodes). This is how preconditions on intermediate nodes "propagate back". The intermediate nodes appear in the hypotheses of some PP-wffs and in the conclusions of others. In the next step a sequence of $\supset E$s (possibly interleaved with other operations) must be performed. At this point the basic parts have been extracted and the wanted PP-wffs can be constructed by a sequence of $\supset I$s and then of $\forall I$s. Notice that a deduction can be "appended" to another by assuming the hypotheses of the former, applying $\supset E$ twice and introducing an implication whose

where: $k_m, ..., k_1$ are metatheoretic function symbols representing the rule functions of the rules applied in the deduction and "$\circ$" means function composition [9]. $(k_m \circ ... \circ k_1)(f_1, ..., f_n)$ is called the **proof plan term**. $PREC(f_1, ..., f_n)$ is the conjunction of the preconditions of the rules applied in the deduction. The preconditions of a rule are applied to a term whose functional part takes into account the inference rule functions which map the leaves into the intermediate nodes. For instance, PP_THIRD_L's preconditions (see equation 5) are the conjunction of $\wedge E$'s and $\supset E$'s preconditions. $\supset E$ is applied to an intermediate node, thus its preconditions are applied to $lfand(wof(f_2))$. Note finally that IR-wffs are particular cases of PP-wffs (this is why we called PP_IMPE the equation in (1)); generalizing IR-wffs, PP-wffs say that, if the preconditions hold, then the endformula is a node of the tree.

Even if it is always possible to represent a deduction as a composition of rule functions, we want to avoid to describe deductions any time from "first principles". Inference rules are too fine grained to be effectively used in global strategies. This can be avoided by producing a conservative extension of the metatheory by defining a new function (called the **deduction function**) as follows:

$$\forall f_1 ... f_n (g(f_1, ..., f_n) = (k_m \circ ... \circ k_1)(f_1, ..., f_n))$$

The deduction function g is defined to behave as the composition of the inference rule and/or deduction functions involved in the deduction it represents. Notice that deduction functions can be used in the representation of more complex deductions and thus in the definition of their deduction functions. Thus, for instance, the deduction function $third_l$ for PP_THIRD_L can then be defined as:

$$\forall f_1 \forall f_2 (third_l(f_1, f_2) = impe(f_1, ande_l(f_2))) \tag{7}$$

and then used (together with the dual function $third_r$) to define the deduction function $transitive$ of $PP\_TRANSITIVE$:

$$\forall f_1 \forall f_2 (transitive(f_1, f_2) = impi(f_2, impi(f_1, third_r(third_l(f_1, f_2), f_2))))$$

The metalevel representation of the deduction becomes:

$$\begin{array}{l} PP\_TRANSITIVE: \\ \forall f_1 \forall f_2 ((T(f_1) \wedge T(f_2) \wedge \\ \quad CONJ(wof(f_2)) \wedge IMP(lfand(wof(f_2)) \wedge HP(wof(f_1), lfand(wof(f_2))) \wedge \\ \quad IMP(rtand(wof(f_2))) \wedge HP(concl(lfand(wof(f_2))), rtand(wof(f_2)))) \\ \qquad\qquad \supset T(transitive(f_1, f_2)))) \end{array}$$

Notice that the use of deduction functions allows reasoning at the desired level of detail. The "correct" level of detail can be chosen independently of the particular theorem proving strategy.

3 Proof planning as theorem proving

As proofs are represented as metalevel wffs, we can perform proof planning by metalevel theorem proving. The goal is to deduce new PP-wffs from a set of generic metalevel formulae

[9] The notation should be made precise explaining how to denote function composition with functions with more than one argument. Since not relevant in this context, this issue is not faced.

More generally it can be shown inference rules can be represented as (we call wffs of this kind "**IR-wffs**"):

$$\forall f_1 ... \forall f_n (T(f_1) \wedge .... \wedge T(f_n) \wedge PREC(f_1, ..., f_n) \supset T(k(f_1, ..., f_n)))$$

where: f_1 ... f_n are variables ranging over facts of the object theory, T is an unary predicate such that $T(x)$ holds when the metatheoretic term x represents a node in a deduction, $PREC(x_1, ..., x_n)$ represents the preconditions for applicability of an inference rule (notice that in some cases, *eg.* $\supset I$, there are no preconditions), k is a function symbol representing the rule function.

A deduction can be seen as *a tree of inference rule applications*, where the premises of the rule applications are intermediate nodes or leaves of the tree. Thus the representation of a deduction contains a mapping from the leaves into the endformula, the mapping being performed by a *composition of inference rule functions*. For instance, the representation of $\{A, (A \supset B) \wedge C)\} \vdash B$, with deduction tree[8]:

$$\supset E \; \frac{A \qquad \wedge E \; \dfrac{(A \supset B) \wedge C}{A \supset B}}{B}$$

is:

$$\begin{array}{l} PP\_THIRD_L : \; \forall f_1 \forall f_2 (T(f_1) \wedge T(f_2) \wedge \\ \quad CONJ(wof(f_2)) \wedge IMP(lfand(wof(f_2))) \wedge HP(wof(f_1), lfand(wof(f_2))) \\ \qquad \supset T(impe(f_1, ande_l(f_2)))) \end{array} \tag{5}$$

while the representation of $\vdash ((A \supset B) \wedge (B \supset C)) \supset (A \supset C)$ with proof tree:

$$\supset I \; \frac{\supset E \; \dfrac{A \quad \wedge E \; \dfrac{(A \supset B) \wedge (B \supset C)}{A \supset B}}{B} \qquad \wedge E \; \dfrac{(A \supset B) \wedge (B \supset C)}{B \supset C}}{\dfrac{C}{A \supset C}}$$
$$((A \supset B) \wedge (B \supset C)) \supset (A \supset C)$$

is:

$$\begin{array}{l} PP\_TRANSITIVE : \; \forall f_1 \forall f_2 (T(f_1) \wedge T(f_2) \wedge \\ \quad CONJ(wof(f_2)) \wedge IMP(lfand(wof(f_2))) \wedge HP(wof(f_1), lfand(wof(f_2))) \wedge \\ \quad IMP(rtand(wof(f_2))) \wedge HP(concl(lfand(wof(f_2)))), rtand(wof(f_2))) \\ \qquad \supset T(impi(f_2, impi(f_1, impe(impe(f_1, ande_l(f_2)), ande_r(f_2)))))) \end{array}$$

Thus the generic representation of an object level deduction is (we call wffs of this kind "**PP-wffs**" or **proof plans**):

$$\forall f_1 ... \forall f_n (T(f_1) \wedge .... \wedge T(f_n) \wedge PREC(f_1, ..., f_n) \supset T((k_m \circ ... \circ k_1)(f_1, ..., f_n))) \tag{6}$$

[8]Note that, for simplicity, we write object level proof trees using a Natural Deduction notation rather than a sequent calculus style. The resulting formula pairs can be easily obtained by prefixing each formula with the set of assumptions it depends on.

```
1. (DEFLAM impe-rule (f1 f2)
2.    (IF (AND (T f1) (T f2))
3.        (IF (AND (IMP (wof f2)) (HP (wof f1) (wof f2)) )
4.            (proof-add-node (impe f1 f2))
5.            (`` .. rule not applicable .."))
6.        (`` .. premises not correct .")))
```

Figure 1: The GETFOL implementation of Modus Ponens

A first component is the code implementing the requirement on the **premises** (in figure 1, line 2, the two calls to the function "`T`"): a rule can be applied in a deduction if (at least one of) the premises are nodes of the deduction tree. Nodes are pairs of the form $< ?, w >$, where w is a wff depending on the set $?$ of assumptions. We call these pairs *formula pairs* or *facts* (or simply wffs when no confusion arises). **Preconditions** are the next component. A rule can be applied subject to certain constraints on w and $?$. For instance $\supset E$ can be applied only if one of the two premises is an implication and such that its hypothesis is equal to the other wff (in figure 1, line 3, the calls to `IMP` and `HP`. `wof` is a function which, given a fact, returns its wff). The **(inference) rule function** ("`impe`" in figure 1, line 4) is the third component. It maps the premises into a fact whose wff is their logical consequence. Thus for instance the $\supset E$ rule function is defined as:

$$k_{MP} : (< ?_1, w_1 >, < ?_2, w_1 \supset w_2 >) \longrightarrow < ?_1 \cup ?_2, w_2 >$$

The fourth component is the **proof tree operator** which makes a logical consequence into a node of the proof ("`proof-add-node`" in figure 1, line 4). The proof tree operator always performs the same operation, independently of the inference rule (*ie.* it adds a new fact to the proof tree). This operation is such that `T` holds of its argument (`(impe f1 f2)` in figure 1). Thus the (declarative) metatheoretic representation of $\supset E$ is [6] [7]:

$$\begin{aligned} PP\_IMPE: \quad & \forall f_1 \forall f_2 (T(f_1) \wedge T(f_2) \wedge \\ & IMP(wof(f_2)) \wedge HP(wof(f_1), wof(f_2)) \supset T(impe(f_1, f_2))) \end{aligned} \tag{1}$$

In the same way, the representations of $\supset I$, $\forall E$ and $\forall I$ are:

$$PP\_IMPI: \quad \forall f_1 \forall f_2 (T(f_1) \wedge T(f_2) \supset T(impi(f_1, f_2))) \tag{2}$$

$$PP\_ALLE: \quad \forall f \forall x \forall a (T(f) \wedge FORALL(wof(f)) \supset T(alle(f, x, a))) \tag{3}$$

$$\begin{aligned} PP\_ALLI: \quad & \forall f \forall x \forall a (T(f) \wedge \\ & FREE(a, wof(f)) \wedge \neg FREEIN(a, wffsof(deps(f))) \\ & \supset T(alli(f, x, a))) \end{aligned} \tag{4}$$

[6] The intuitive meaning of the predicate and function symbols used (*IMP, HP, impe*) should be obvious. Their interpretation (as in all the examples listed in the rest of the paper) is left to the reader. For instance *impe* is the metalevel function symbol representing the object level inference rule function which performs the implication elimination. IMP is a metalevel predicate true when its argument represents a object level formula whose main symbol is the implication.

[7] In this paper we do not consider errors. A complete metatheoretic description of inference rules (and thus of `impe-rule`) should also describe how the various errors (*eg.* "`rule not applicable`") are dealt with.

(just like tactics) to produce an object level theorem [2]. A consequence of this last fact and of the possibility of memorizing previously built proof plans is that the proving ability of a system can be incrementally increased with no modification/ extension of the underlying code [3]. Technically, proof plans are used to assert an object level theorem by using a reflection principle [GSnt] and the fact that a subexpression of **a proof plan is a name of an object level theorem**.

The architecture implemented using the two ideas above is **uniform** in that a tower of metatheories can be defined, each using the same logic (and thus the same code), each such that proof plan execution at one level results in proof plan formation one level below. The only exception is the bottom layer where proof plans are usually not defined. In general each theory has a distinct language.

The paper follows this path. Section 2 introduces the notion of "*mechanized logic*" and shows how its declarative metatheory can be built; in this section the notion of proof plan is introduced. In section 3 it is hinted how an object level proof can be planned by metatheoretic theorem proving on proof plans and other metalevel formulae. In this section the **uniformity** of the architecture is pointed out. Section 4 shows how proof plans can be executed to prove the object level theorem. Finally section 5 presents an implementation in GETFOL [GT90] [4] of a simplified version of the "Boyer and Moore Theorem Prover (BMTP from now on) based on the above ideas. Section 6 discusses the related work.

2 A metatheory of a mechanized theory

We informally speak of the **mechanization of logic** [5] meaning that we think of logic not as something used to *describe* reasoning but rather as something used to *automate* reasoning *on a machine*. Thus, for instance, we see a wff not as a string but, rather, as a data structure. As a consequence, not only does a metatheory of a "mechanized logic" describe the usual properties (*eg.* being a wff, being a theorem), but also those (intensional) aspects which are relevant to its implementation on a computer (*eg.* a wff is a list, an inference rule code calls certain functions). Note that this is substantially different from logicians' approach but also, for instance, from the work done in the NuPrl project [CAB+86, KC86, How88] in which the metatheory is completely independent of the structure of the underlying code.

Since we are here interested in a metatheory describing (mechanized) deducibility, one of the first problems to be faced is how to represent inference rules applications. Figure 1 presents (a part of) the GETFOL implementation of modus ponens ($\supset E$). In the code reported in figure 1 we can identify four basic components which exist for any inference rule (note that these components must exist, in some form, in any code implementing inference rules).

[2] This is different from what happens, for instance, in Prolog where meta-interpreters [SS86] are written in terms of the object level logic (by exploiting its procedural interpretation). In our approach the object level logic has **no** procedural interpretation; proof plans are interpreted in terms of the code implementing it.

[3] An interesting extension of this work is then to study how to compile the proof plans back into the code.

[4] GETFOL is a complete re-implementation of the FOL system [Wey80].

[5] In the following we use standard natural deduction conventions and terminology. This is done simply because the deductive machinery implemented in GETFOL is (a variation of) natural deduction (GETFOL's logic is similar to sequent calculus in that assumptions are explicitly carried through at each deduction step). The ideas described in this paper are of course independent of the used logic.

Plan formation and execution in a uniform architecture of declarative metatheories

Fausto Giunchiglia
Paolo Traverso
Mechanized Reasoning Group
IRST
Povo, 38100 Trento, Italy
fausto%irst@uunet.uu.net
leaf%irst@uunet.uu.net

Abstract

We show how explicit control strategies can be *represented* in a declarative (classical) metatheory as first order formulae (proof plans). Proof plans can be *reasoned about* (by metatheoretic theorem proving) to modify the search strategy and *"executed"* (by suitably "interpreting" them in terms of the deductive machinery implementation code) to prove a theorem in the object theory. The resulting architecture is *uniform* as it becomes possible to define a tower of metatheories, each using the same deductive machinery, each (but the lowest) being able to represent proof plans with formulae of the same shape. Plan formation at one level can be obtained by plan execution one level up. The realization of these ideas in the GETFOL system is briefly described via the implementation of a simplified version of the Boyer and Moore theorem prover.

1 Introduction

The idea of using metatheories in theorem proving has been extensively studied in the past, a not exhaustive list is [DS79, Wey82, BM81, BW81, Bun88, BSG+88, KC86, How88, GMW79]. The goal of the research described in this paper is to get a better understanding of (mathematical) reasoning and to automate it on a machine (partially) by the use of metatheoretic theorem proving. The work is based on two main ideas:

(1): instead of directly performing search by object level theorem proving, it seems better to perform a step of planning in the metatheory (in other words "to do **proof planning**" [1]). Doing so, one should be able to get some sense of direction, a "proof schema" and thus (partially) avoid the combinatorial explosion. In our approach **proof plans are first order wffs, proof planning is performed by metalevel theorem proving.**

(2): the metatheory should describe not only the object level logic but also how it is implemented, the intensional properties of the code implementing it. If this result is achieved then it is possible to link the (declarative) metatheory to the object level code. The code can thus be used, under certain conditions, to give a procedural interpretation to some classes of metatheoretic wffs (the proof plans). Proof plans can therefore be "executed"

[1] The meanings of the terms "proof planning" and "proof plan" are very similar to those used in [Bun88], see section 6 on related work.

ISTITUTO PER LA RICERCA SCIENTIFICA E TECNOLOGICA
I 38100 TRENTO – LOC. PANTÉ DI POVO – TEL. 0461–314444
TELEX 400874 ITCRST – TELEFAX 0461–302040

PLAN FORMATION AND EXECUTION IN AN UNIFORM ARCHITECTURE OF DECLARATIVE METATHEORIES

F. Giunchiglia, P. Traverso

March 1990
Technical Report # 9003-12

Published in Bruynooghe (ed.) *Proceedings of the 2nd Workshop on Meta-Programming in Logic (META-90)*, MIT Press, pp. 306–322.

ISTITUTO TRENTINO DI CULTURA