A Meta-Level Inference Architecture for Contexts

Pierre E. Bonzon
HEC,University of Lausanne,
1015 Lausanne, Switzerland

pbonzon@hec.unil.ch

Abstract

We present a simple meta-level inference architecture for implementing nested contexts. Based on an explicit non ground representation of both object and control knowledge, it treats contexts as constructs embodying a thight coupling between theories and inference rules. Processing is done via an extension of the traditional "vanilla" interpreter for logic programs allowing one to hop up and down the hierarchy of control clauses representing inference rules. An iterative deepening search of the corresponding state space prevents infinite recursion and ensures successful termination whenever possible. The resulting computational system resembles very much the tower architecture defined for functional programming, whereby each level represents a meta-level operating on the preceeding one.

The expressive power of this system is illustrated with a set of working Prolog examples, including McCarthy's model of theory lifting, the processing of nested beliefs, and the access of data and/or knowledge bases using generic procedural knowledge.

1. Introduction

The quest for the Grail in AI, i.e. computer equipped with common sense, has been on for quite some time, but has failed yet to come up with significant results. Numerous formal systems for performing the kind of reasoning presumably required for the task have been proposed. Although some of them have been implemented as computer programs, they are not generally available as user-friendly subsystems. Futhermore, as they now stand, they would not be able to cooperate with other subsystems. These shortcomings, we think, do represent one of the major reasons for the present disappointing state of affairs.

The requirement for effective system integration is particularly critical at a time when vast amounts of computerized common sense knowledge may soon be widely available (if we take for granted that projects like CYC will eventually mature to the point of becoming viable commercial ventures). Without matching common sense reasoning capabilities, we believe, computer systems will not be able to take full advantage of this new breed of data bases.

What is needed, we conclude, is a kind of a general architecture or facility (not another language, not another formal system) for implementing advanced knowledge representation and processing models. This facility should be provided on *existing software platforms* and be easily interfaced with available knowledge sources.

As a first step toward this goal, we have been looking for a prototyping system that could allow one to easily transcribe and experiment with models of beliefs, reflective or context dependent reasoning. The kind of systems we have in mind is best described by the following lines taken from A. Hutchinson's recent book on *Algorithmic Learning* [10]:

> " A clever learning program which can work at meta-levels might include a procedure for generating successive meta-levels. Whenever something within the program suggests that its present system is inadequate, the program could hop up a level, define an extension of its current object system, and then hop down again."

On the theoretical side, F. Gunchiglia & al. [7] , working on a possible foundation for meta-reasoning in the field of AI, introduced the concept of a *hierarchical meta-logic*. Given any axiomatized object theory expressed in propositional logic, this formal system produces theorems both in this base theory and in a related meta-theory. To quote from them, "the assumption of uniformity throughout the hierarchy leads to the possibility of capturing the whole infinite tower of theories within a single all-encompassing formal system schematic on the levels".

Introduced in the early 80's by B.C. Smith [14] in the context of functional programming, the concept of a tower architecture has not proved so far to lead to practical applications. As Hutchinson points out, "considered as a meta-system, it is not very expressive" , and, as such, does not allow to tackle significant problems. To overcome this limitation, Gunchiglia & al. first define a system where the property expressed at each meta level

is theoremhood. They allow non logical axioms to be added at appropriate levels, ending up with a natural deduction system for the whole hierarchy. Finally, they prove that the resulting formal system has a proof theory which is sound and complete with respect to a particular semantics capturing the desired relationship between object and meta theory.These developments bear strong similarities with the agenda put forward by J. McCarthy in his quest for generality in AI. Indeed, one the foremost characteristics of his view of contexts, as described in [12], lies in their ability to transcend themselves in order to relax or change some of the assumption they contain. Lifting theories from one context to another also provides facilities going in the same direction, i.e. that of extending ones original beliefs or capabilities in a non monotone way so as to achieve a true adaptative behavior.

Our own efforts have essentially consisted in trying and working out reasonably simple implementations of some the above theoretical ideas. Our emphasis will therefore be less on formalism than on computational requirements. We have decided at once that our investigation would be based on Prolog, which readily allows one to quickly develop and test various meta-level schemes using the non-ground representation [9]. At the same time, it provides a modern interface to leading commercial software applications (were these going to be needed for later developments that have not been attempted yet).

The good news are that we have been able to do this in a rather straightforward way, coming up with a compact meta-interpreter for a simple meta-level inference system implementing nested concepts. The bad news are that at the end, we have to rely on an *iterative deepening search* algorithm in order to escape from infinite recursion and ensure, when possible, a successful termination.

The rest of this paper is organized as follows. In the next section, we introduce a set of simple context models of increasing complexity, together with the corresponding interpreters for answering queries. These should allow the readers which are not familiar with McCarthy's view of contexts, including his rather intricate model of theory lifting, to get a hands on understanding of this notion. We then illustrate the expressive power of contexts with three examples. In the first one, we show how to process nested private beliefs. In the second one, we allow for the processing of beliefs as *common* knowledge, and illustrate this extended framework with a solution to the well known "three wise men problem". Finally, we show in a third exemple how to access data and/or knowledge bases using generic procedures implemented as contexts. We conclude with a (very) short discussion stressing the nature of concepts, as well as their current limitations, as they emerge from our preceeding developments.

2. Basic models of contexts and their corresponding meta-interpreter

2.1 A first model of nested explicit and implicit contexts

In our first model, an *explicit* outer context, say c0, is a named collection of

- modalities of the form

  `ist(c,p)`

  where p is a first order atomic sentence, and `ist(c,p)` is used to express that sentence p holds in an *implicit* inner context c

- simple implications involving such modalities.

example:

```
c0::[ist(c,on(a,b,s0)),
     ist(c,on(X,Y,S))
      ->ist(c,above(X,Y,S))].
```

N.B. - Following the Prolog convention, identifiers starting with an upper case letter represent variables

- "::" is a declared infix operator used as a ordinary predicate which simply associates c0 with an ordinary list.

A possible inference system based on modus ponens for answering queries in a given context can be embodied in a standard PROLOG meta-interpreter as follows:

```
interpret(Goal,C):- match(Goal,C);
                    lift(Goal,C).

lift(Goal,C):-match(Cond->Goal,C),
              interpret(Cond,C).
```

where `match(X,C)` retrieves the formulas in C which can be unified with X.

N.B. The standard Prolog operator ";" corresponds to a disjunction.

The top level calling the meta-interpreter with the proper outer context then becomes

```
Context:Goal :- Context::C,
                interpret(Goal,C).
```

A example of such a call, and its result, could be

```
?- c0:ist(C,above(X,Y,S)).

C=c,
X=a,
Y=b,
S=s0.
```

2.2 An extended model of nested explicit and implicit contexts

In our second model, an explicit outer context is a named collections of

- extended modalities involving both atomic sentences and simple implications on atomic sentences
- implications involving such extended modalities, possibly given under the form of axioms schemas, i.e. containing meta variables standing for arbitrary sentences.

```
c0::[ist(c,on(a,b,s0)),
     ist(c,(on(X,Y,S) -> above(X,Y,S))),

     ist(X,P->Q)
      -> ist(X,P)
          -> ist(X,Q)].
```

We must here carefully distinguish between extended modalities involving simple implications, such as

```
ist(c, (on(X,Y,S) -> above(X,Y,S)))
```

which are used to express that an implication holds in inner context c, and implications involving extended modalities (thereafer called meta-implications), such as

```
ist(X,P->Q) -> ist(X,P) -> ist(X,Q)
```

which are intended to represent meta-assertions, standing for inference rules valid in any inner context X.

N.B. Such meta-implications are routinely found, under various forms, in meta-programs written in Prolog. The theoretical logical issues raised by this practice have been adressed in a recent study [11], whose results imply that it is indeed quite safe to do so (i.e. there are corresponding sound and complete formal systems).

A possible PROLOG meta-interpreter for answering queries in contexts containing meta-assertions is as follows:

```
interpret(Goal,C):- match(Goal,C);
                    lift(Goal,C).

lift(Goal,C):-interpret(Cond->Goal,C),
              interpret(Cond,C).
```

It should be noticed that, according to the intended interpretation given above, `interpret` does not directly apply implications contained within modalities, but relies towards that goal on meta-implications, thus leading to a meta-interpretation process. More specifically, the recursive call to `interpret` which replaces in `lift` a call to `match` basically *unrolls* nested meta-implications associating to the right. Although our previous example still successfully terminates, this second recursive call to `interpret` opens the door to numerous cases of infinite recursion. A possible way out consists in implementing an iterative deepening search algorithm, with the top level now looking as follows:

```
Context:Goal :- Context::C,
                search(Goal,C,1).

search(Goal,C,N):-interpret(Goal,C,N),
                  N1 is N+1,
                  search(Goal,C,N1).
```

with the `lift` procedure itself completed as follows

```
lift(Goal,C,N):-N > 0,
                N1 is N-1,
                interpret(Cond->Goal,C,N1),
                interpret(Cond,C,N1).
```

where `interpret`, while carrying an extra argument, remains essentially as before.

2.3 A first model of nested explicit context

While our previous model clearly distinguishes *object* knowledge (asserted as extended modalities) and *control* knowledge (asserted as meta-implications), both kind of assertions are contained in the same context, i.e. c0. Our next model splits these two kinds of knowledge into distinct explicit contexts c0 and c.

We thus get

```
c::[on(a,b,s0),
    on(X,Y,S) -> above(X,Y,S) ].

c0::[ist(X,P->Q)
      -> ist(X,P)
          -> ist(X,Q)].
```

Our meta-interpreter, which works within c0, must now be instructed to *enter* c when needed. This can be expressed as follows:

```
interpret(Goal,C,N):- enter(Goal);
                      match(Goal,C);
                      lift(Goal,C,N).

enter(ist(Context,Goal)):-Context::C,
                          match(Goal,C).
```

The rest of the interpreter is unchanged.

2.4 A model of specialized contexts involving the lifting of theories

Our last model, while providing specialized contexts, still contains, within object contexts, both *specific* knowledge (e.g. on(a,b,s0) , a fact related to the particular context it belongs to), and *generic* knowledge (e.g. `on(X,Y,S) -> above(X,Y,S)`, akin to a rule which can be applied in various contexts). Our next model, which reproduces McCarthy's original proposal [12], allows for generic knowledge, or theories, to constitute independant contexts. These can be then accessed from any object context suitably equipped to initiate a process called *theory lifting*.

Our standard example now looks as follows:

```
above::[on(X,Y) -> above(X,Y)].

c::[on(a,b,s0),
    P(X,Y,S) -> ist(c(S),P(X,Y)),
    ist(c(S),P(X,Y)) -> P(X,Y,S),
    ist(above,P)  -> ist(c(S),P)].

c0::[ist(X,P->Q)
      -> ist(X,P)
          -> ist(X,Q),
     ist(X,P) -> ist(Y,ist(X,P)),
     ist(X,ist(Y,P->Q))
            -> ist(X,ist(Y,P))
                -> ist(X,ist(Y,Q))].
```

Generic object knowledge previously contained within c now constitutes a separate theory contained within context above. Additionally, both context c and c0 have been extended with new control knowledge. Context c has been equipped with the proper axioms required for lifting theories (see [12] for a thorough presentation of these axioms). c0 further contains two meta-implications, i.e.

- the backward direction of Buvac and Mason's Fl+ flatness axiom schemas[4]; this also corresponds to the lifting axiom introduced by Attardi and Simi [1] in their corrected version (carried out by hand in a natural deduction setting) of McCarthy's original proposal
- an additional axiom standing for an inference rule valid in any inner context X considered itself as an outer context with respect to an inner context Y, thus allowing one to carry inferences across *nested* contexts .

The previous version of our metainterpreter can be used unchanged in this extended framework.

2.5 A simplified model of theory lifting involving explicit parametrized contexts

In the above model of theory lifting, both specific object knowledge describing a particular situation s0, i.e. `on(a,b,s0)`, *and* control knowledge are contained in the same context c . Furthermore, it deals implicitly with parametrized contexts, such as c(S), which are not described explicitly. In our next model, which addresses both issues, we introduce explicit *parametrized* contexts.
As a result, situation `s0` can be described in a particular instance `c(s0)` of parametrized contexts `c(S)`. While our previous context `c` retains its specific axiom lifting `above` into any `c(S)`, c0 now further contains a meta-implication bringing any such specific axiom down to `c0`. There is no need any more for the meta-implications introduced in our previous `c0` to carry inferences across nested contexts :

```
above::[on(X,Y) -> above(X,Y)].

c(s0)::[on(a,b)].

c::[ist(above,P) -> ist(c(S),P)].

c0::[ist(C, ist(A,P)->ist(C(S),P))
      -> ist(A,P)
         -> ist(C(S),P),
     ist(X,P->Q)
      -> ist(X,P)
          -> ist(X,Q)].
```

example:

```
c0:ist(C(S),on(above(X,Y)).

C=c,
S=s0,
X=a,
Y=b.
```

3. Meta-level architectures and application models: a short review

3.1 Meta-level architectures

Meta-level architectures, as described extensively in [8], are by now well defined systems enjoying common properties (foremost, the explicit representation of control knowledge). At the same time, they can be further classified according to specific characteristics, such as their "locus of action", linguistic relation between levels, and so on.

By the obvious virtues of its associated interpreter, the system introduced in the previous section falls into the category of *non ground declarative, meta-level inference system*. As such, it does not require the use of a naming relation (as opposed to similar, *amalgated* systems like Alloy[2]). As argued in [8], and further confirmed in [11], nothing prevents the use of a single language, both at the object and meta-levels.

Being based on a *non-ground* [9] representation of both object and control knowledge, our system stands in strong contrast to the more versatile approach using the *ground* representation pioneered in [3]. This is not surprising, since the goals systems based on this later approach usually pursue, " namely the development of software tools (the metaprograms) that manipulate other programs (the object programs) as data, such as debuggers, compilers, program transformers, etc" [5] , are quite different from ours. When they aim, as we do, at representing and processing application knowledge, systems using the ground representation, such as Alloy [2] and Reflective Prolog [6], seem to be unnecessarily cluttered by naming relations.

3.2 Application models

As all architectures, our simple system represents nothing more than an empty shell, i.e. a tool that should possibly be used to model situations and solve practical problems. As a first assessment of the expressive power enjoyed by our computational model of contexts, we present a solution to three rather different problems from the current literature.

3.2.1 Encoding and processing of private beliefs

Potential applications which have already been extensively studied include, among others, the modelization of *agent beliefs* including self referential statements about truth or knowledge, which arise naturally in common sense reasoning. Our first example involves the modelization of an agent's *private* beliefs. The original formulation goes as follows [2]:

> "...Mary has the following five basic beliefs: anyone who she believes is thinking they like her, she finds charming; all men think all women are irresistible; everyone think that Mary is a woman; John is a man; everyone think that if they are irresistible then everyone likes them".

The solution given in [2] makes use of a special predicate Name " intended to mean that an atomic sentence Name(t,<t>) can be proved for any term t in any theory". A specialized proof system is then needed to carry the "interleaving of computations at different meta-levels".

As far as the object theory is concerned, our own formulation does not call for any such special predicate. It otherwise closely follows [2]'s original solution and includes their convention for modalities, i.e. `believe(X,P)` is substituted for `ist(X,P)`:

```
mary::[male(john),
       believe(X,female(mary)),
       believe(X,likes(X,mary))
       -> charming(X),
       believe(X,(irresistible(Z)
                  -> likes(Y,Z))),
       male(X)
       -> believe(X,(female(F)
                     -> irresistible(F)))].
```

N.B. As before, any assertion P contained in context `mary`, such as `male(john)`, has to be understood as `believe(mary,P)`, e.g. `believe(mary, male(john))`.

The inference system needed to account for the nesting of beliefs, on the other hand, simply consists of our previous outer context c0 (the Fl+ axiom is not needed here and has therefore been dropped), together with our usual metainterpreter, whose `enter` predicate must be adapted to reflect our new convention for modalities. The solution is as follows:

```
m0::[believe(X, P->Q)
      -> believe(X,P)
          -> believe(X,Q),

     believe(X, believe(Y, P->Q))
      -> believe(X,believe(Y,P))
         -> believe(X, believe(Y,Q))].

enter(believe(Person,Belief)):-
                      Person::C,
                      match(Belief,C).
```

example:

```
m0:believe(X,charming(Y)).

X=mary,
Y=john.
```

3.2.2 Encoding and processing of common beliefs

Our next example, i.e. the so-called "three wise men problem", is often regarded as a benchmark in the field of meta-reasoning, so we will refrain from restating it here. Among the numerous ways that have been proposed to solve this problem, we have chosen to reproduce the elegant proposal of Attardi and Simi. In their solution, "common knowledge is grouped in a single theory and lifting rules are provided for each agent to access it " [1]. The corresponding required proof is then carried out *by hand*, following a well defined process of natural deduction in and out of nested contexts.

In order to mechanize this proof, the disjunctive facts contained within Attardi and Simi's original formulation of common beliefs have been broken down into implications, leading to the object theory contained in the following inner context `wise`:

```
wise::
[(non white(2),non white(3))-> white(1),
 (non white(3),non white(1))-> white(2),
 (non white(1),non white(2))-> white(3),
 white(1) -> believe(2, white(1)),
 white(1) -> believe(3, white(1)),
 non white(1) -> believe(2,non white(1)),
 non white(1) -> believe(3,non white(1)),
 white(2) -> believe(3, white(2)),
 white(2) -> believe(1, white(2)),
 non white(2) -> believe(3,non white(2)),
 non white(2) -> believe(1,non white(2)),
 white(3) -> believe(1, white(3)),
 white(3) -> believe(2, white(3)),
 non white(3) -> believe(1,non white(3)),
 non white(3) -> believe(2,non white(3)),
 non believe(1,white(1)),
 non believe(2,white(2))].
```

In a rather straightforward way, the above three subsets of beliefs correspond respectively to the following common knowledge:

- at least one wise man spot is white
- each wise man can see his collegues spots
- the first and second wise men have not been able to find out their own spot's color.

A suitable outer control context tailored to process *any* inner context containing common knowledge is given in the following parametrized context `w0(W)`:

```
w0(W)::

[(X\=W,believe(W,P)) -> believe(X,P),
 believe(W,P) -> believe(W,believe(X,P)),
 believe(W,P->Q)
  -> believe(W,P)
      -> believe(W,Q),

 (believe(W,A),believe(W,B))
   -> believe(W,(A,B)),

 [believe(W,non P)]
  => (believe(W,non believe(X,Q)),
      believe(W,believe(X,Q)))
       -> believe(W,P)].
```

Notice first that parameter `W` in context `w0(W)` is intended to be unified with an inner object context name, e.g. `wise`. The first two meta-implications contained within `w0(W)` correspond to Attardi and Simi's axioms (1) and (2), and as such do " provide a proper account of common knowledge, allowing to derive the commonly known facts in any viewpoint, no matter how nested" [1]. Following the meta-implication for meta-interpretation, this context further contains two additional meta-implications, i.e.

- an axiom for processing conjunctive beliefs
- an axiom corresponding to the "reductio ad absurdum" inference rule of classical logic based on assumptions.

Accordingly, our meta-interpreter must be extended to carry out proofs under a given assumption. Towards this, we include an additional parameter as follows:

```
interpret((A=>B),C,D,N):-
                 append(A,D,AD),
                 interpret(B,C,AD,N).
```

The general interpreter clause must now be extended as follows:

```
interpret(Goal,C,D,N):-enter(Goal);
                       match(Goal,C);
                       unify(Goal,D);
                       lift(Goal,C,D,N).
```

N.B. While `match(X,Y)` and `unify(X,Y)` both retrieve formulas in Y which can be unified with X, `unify` further binds elements in Y in order to allow for uninstantiated assumptions contained in Y to get later bound.

Finally, the top level iterative deepening search becomes:

```
Context(Domain):Goal:-Context(Domain)::C,
                      search(Goal,C,[],1).

search(Goal,C,D,N):-interpret(Goal,C,D,N);
                    N1 is N+1,
                    search(Goal,C,[],N1).
```

example

```
w0(wise):believe(3,P(3)).

P = white.
```

3.2.3 *Accessing data and/or knowledge bases using generic procedural knowledge*

Our final example constitutes a first step towards systems relying on common sense reasoning capabilities to access and exploit data bases. It was first proposed in [13] to illustrate the concept of "heterogeneous knowledge compilation" in a system operating with a decision theoretic formulation. It does involve hypothetical data and rules about commodity trading, whose collection, in our framework, constitutes an *parametrized* object context appropriately named `trade(oil)`:

```
trade(oil)::
[cargo(venezuela,100000),
 price(venezuela,10),
 price(us,12),
 shipping(venezuela,us,0.5),

 (cargo(O,T),
  price(O,P))          -> contract(T,P,O),

 (contract(T,P,O),
  price(D,M),
  shipping(O,D,S),
  F is T*(M-P-S),
  F >0)                -> profit(F,D)].
```

The specific data contained in this context basically state that 100,000 tons of venezuelan oil could be bought at $10 a ton, brougth to the US for $0.5 a ton, and sold there for $12 a ton. Any business oriented mind faced with such knowledge would see an opportunity for making an immediate profit of $150,000. Whith the appropriate formulas for computing this profit given by two rules, how could an automated system discover this opportunity using generic knowledge *applicable in other contexts* ? In other word, how to instruct such a system to *learn* and *apply* from *first principles only* the following rule: " if the current US market price of crude oil is M then buying a cargo of T tons in Venezuela at price P will yield a net profit f(M,T,P)" for some known f ?

Toward this end, let us consider the *generic* procedural knowledge contained in a parametrized context carrying uninstantiated arguments. After retrieving rules for *actions* and *utilities*, it will compile and then apply conditions for relating actions and utilities into *opportunities*:

```
opportunity(Action,Utility)::

[(ActionCond -> Action(|X),
  UtilityCond -> Utility(|Y),
  compile(ActionCond -> Action(|X),
          UtilityCond -> Utility(|Y),
          OpportunityCond))
  -> OpportunityCond
     -> opportunity(Action(|X),
                    Utility(|Y))].
```

N.B. The Prolog syntax `(|X)` represents a variable length argument list.

The above `compile` procedure, which is meant to compute general opportunity conditions, simply involves the partial evaluation of possible actions (in the trade context, just the `contract` rule) with respect to given utilities (in the trade context, the `profit` rule).

In order to lift this `opportunity` theory into the trade context, let us now follow section 2.5:

```
trade::
[ist(opportunity(contract,profit),P)
 -> ist(trade(S),P)].
```

Notice how instantiated arguments, i.e. `(contract, profit)` are used to relate the `trade` and `opportunity` contexts. A suitable outer context c0 would look as follows:

```
c0::
[ist(C,ist(A,P) -> ist(C(S),P))
  -> (ist(A,P),not system(P),P\=(_,_))
       -> ist(C(S),P),
 ist(X,P->Q) -> ist(X,P) -> ist(X,Q),
 (ist(X,A),ist(X,B)) -> ist(X,(A,B)),
 (system(A),call(A)) -> ist(X,A)].
```

Included in this context are now two additional meta-implications dealing with conjunctive and system calls, which are reflected in the meta-interpreter as follows:

```
interpret((A,B),C,N):-interpret(A,C,N),
                      interpret(B,C,N).
interpret(A,_,_):- system(A),call(A).
```

example:

```
c0:ist(C,opportunity(X,Y).

C=trade(oil),
X=contract(100000,10,venezuela),
Y=profit(150000,us).
```

4. Conclusion

As they emerge from our preceedings developments, contexts cannot be simply identified with theories. They rather look like modular pieces of a multi-level construct embodying a tight interaction between object theories, on one hand, and control knowledge, such as generic procedures and/or inferences rules, on the other. The resulting computational system resembles very much the tower architecture defined for functional programming, whereby each level represents a meta-level operating on the preceeding one. As a result, as pointed out by McCarthy [12], "because relations among contexts are expressed as sentences in the language", they allow inferences that could only be done at the meta-level.

While we did, to a certain extend, fulfil our original goal of providing a general compact architecture for implementing nested contexts, the limitations of our current models are numerous, particularly when it comes to linking existing contexts. We view the development of adequate models of contexts linking as priority requirements for making contexts useful in practical applications such as those involving the access of CYC-like data and/or knowledge bases. In our last example, predefined instantiated context arguments were used as linking knowledge. This same linking knowledge could possibly follow from interactive dialogue sessions. In later extensions, such a knowledge could be discovered by the system itself, following appropriate *learning* and/or *training* sessions based on more elaborate models of context linking.

5. Acknowledgment

This work was done while the author was visiting at the International Computer Science Institute, Berkeley. Support was provided by the Swiss National Research Council, under grant no 21-33817.92.

6. References

[1] G. Attardi and M. Simi, Building Proof in Context, Pre-proceedings 4th Inter. Workshop on Meta Programming in Logic (META 94), 1994

[2] J. Barklund, K. Boberg and P. Dell'Acqua, A Basis for a MultiLevel Metalogic Programming Language, Pre-proceedings 4th Inter. Workshop on Meta Programming in Logic (META 94), 1994

[3] K. Bowen and R.Kowalski, Amalgating language and metalanguage in logic programming, in:Logic Programming, K. Clark and S. Tarnlund (eds), Academic Press, 1982

[4] S. Buvac and I.A. Mason, Propositional Logic of Context, Proc. 11th Natl. Conf. on Art. Intell. (AAAI93), 1993

[5] I. Cervesato and G. F. Rossi, Logic Meta-Programming Facilities in 'Log, in: A. Pettorossi (ed.), Meta-Programming in Logic, Proc. 3rd Int. Work. (META 92), Lecture Notes In Computer Science (vol 649), Springer Verlag, 1992

[6] S. Costantini and G.A. Lanzarone, A MetaLogic Programming Language, in: C. Levi and M. Martelli (eds.), Proceedings of the Sixth Inter. Conf. on Logic Programming, MIT Press, 1989

[7] F. Giunchiglia, L. Serafini and A. Simpson, Hierarchical Meta-Logics: Intuitions, Proof Theory and Semantics, in: A. Pettorossi (ed.), Meta-Programming in Logic, Proc. 3rd Int. Work. (META 92), Lecture Notes In Computer Science (vol 649), Springer Verlag, 1992

[8] F. van Harmelen, Meta-level Inference Systems, Pitman and Morgan Kaufman Publ. 1991

[9] P.M. Hill and J.W. Lloyd, Analysis of Metaprograms, in: H. Abramson and R Rogers (eds), Metaprogramming in Logic Programming, The MIT Press, 1989

[10] A. Hutchinson, Algorithmic Learning, Oxford University Press, 1994

[11] M. Kalsbeek, Correctness of the Vanilla meta-interpreter and ambivalent syntax, to appear in: K. Apt (ed.), Meta-Programming in Logic, MIT Press 1995

[12] J. McCarthy, Notes on Formalizing Context, Proc. 13th Int. Join Conf. on Art. Intell, (IJCAI93), 1993

[13] S. Russell, Execution Architectures and Compilation, Proc. 11th Int. Join Conf. on Art. Intell, (IJCAI89), 1989

[14] B.C. Smith, Reflection and Semantics in Lisp, Conf. Rec. 11th ACM Symp. on Princ. of Progr. Languages (POPL 84), 1984