Languages, Meta-languages and MetateM, A Discussion Paper

MICHAEL FISHER, *Department of Computing, Manchester Metropolitan University, Chester Street, Manchester M1 5GD, United Kingdom, E-Mail: M.Fisher@doc.mmu.ac.uk*

ANTHONY HUNTER, RICHARD OWENS, *Department of Computing, Imperial College of Science, Technology and Medicine, 180 Queen's Gate, London SW7 2BZ, UK. E-mail: {abh,rpo}@doc.ic.ac.uk*

HOWARD BARRINGER, *Department of Computer Science, University of Manchester, Oxford Road, Manchester M13 9PL, UK. E-mail: howard@cs.man.ac.uk*

DOV GABBAY, *Imperial College, E-mail: dg@doc.ic.ac.uk*

GRAHAM GOUGH, *University of Manchester, E-mail: graham@cs.man.ac.uk*

IAN HODKINSON, *Imperial College, E-mail: imh@doc.ic.ac.uk*

PETER MCBRIEN, MARK REYNOLDS, *King's College, Dept of Computer Science, The Strand, London WC2R 2LS, E-mail: {markr,pjm}@dcs.kcl.ac.uk*

DEREK BROUGH, *Imperial College, E-mail: drb@doc.ic.ac.uk*

Abstract

Meta-languages are vital to the development and usage of formal systems, and yet the nature of meta-languages and associated notions require clarification. Here we attempt to provide a clear definition of the requirements for a language to be a meta-language, together with consideration of issues of proof theory, model theory and interpreters for such a language.

1 Introduction

1.1 Why bother with Meta-level Reasoning?

There are two main types of occasion where we need meta-level reasoning. One is where we have one system A wanting to talk about another system B, either because A wants to describe B and reason about it, or because A wants to simulate B and use B as part of itself. It might want to have only a partial description of the other system, in order to interact with it. For example, a number of systems may be in conflict, and require a master meta-system to resolve that conflict. The ability to describe the reasoning and inference processes by use of a meta-language is also useful if we wish to formalize the 'process' of inference within logical languages and reason about the truth of statements in a particular language. In these cases, the meta-level comes from the need to describe, reason and specify other systems.

Note that this kind of meta-relationship between the systems is static. Both exist unchanged, with one describing the other. We may even have the case where one system is computing and developing in time and the other system describes the changes of state of the first system. This is also a case of static description, since it is non-interacting and purely descriptive.

Systems, however are dynamic, i.e. they change with time. Once we permit a system to change, we conceptually have a progression of snapshots of the state of the system. Many applications give rise to systems whose evolution depends on their past. An example of particular interest is the executable temporal logic METATEM [1].

These systems are usually needed when we want to cope with and respond to an open future, and be able to handle unexpected interactions. Such systems would need internal meta-level capabilities. The current part of the system (the "now" part) may need to be able to describe the past part of itself, acting as a static meta-system to its own past. The next step in the future may depend on the current meta-description of the past. We have a need for a language which can be partitioned into layers of sub-languages, each higher layer acting as a meta-language for the lower layer. Without going into details, a self-meta language would be a fixed-point of such a layering process. If this is done for the temporal logic METATEM we will get a language capable of the above dynamic evolution.

In this report we will consider the (possibly bidirectional) link between a language and its meta-language. In order to support this we provide a discussion of views on the definition of a language and a meta-language within this section, and we provide our own view of meta-languages in Section 2. In Section 3, we discuss particular issues of relating such languages.

1.2 What are Meta-languages?

A *language* can be seen abstractly as a set of strings; alternatively it can be seen constructively as a grammar definition. We use the latter for this paper. For a language L, some existing language has to be used to formulate and reason about the syntax and semantics of L. A language that is used to reason about another language is called a *meta-language* and the language reasoned about is called the *object-language*.

Malpas [14] provides a perspective on the function of meta-languages for logical languages and makes the connection between meta-languages and definitions of truth.

> "A meta-language performs two functions, one analytical, and one prescriptive. The analytic function of the meta-language is to characterise object-language statements by properties such as *tautology* or *contradiction*. It also allows the characterisation of object-language language theories, and the object-language itself, by properties such as *complete* or *consistent*.
>
> "The prescriptive function of the meta-language is to establish the truth value of an object-language statement that is neither a tautology nor a contradiction.

> "In predicate logic, the simplest construct that can denote a truth value is an atomic formula. An atomic formula is composed of a predicate symbol and some number of terms, none of which is capable of denoting a truth value by itself. When a meta-language statement describes a property of an object-language statement, there is an important syntactic relation between the two: the object language statement acts as a term (i.e., a syntactic object that cannot denote a truth value) in the meta-language statement."

The object-language may be, but need not be, different from the meta-language. Thus, a language can (under certain restrictions—see later) be used to reason about itself. This commonly occurs in English. For example, the sentence

John is a name with four letters.

is an example of the use of English as its own meta-language. Note that ambiguities can arise with this type of usage (for example, John is also a red-headed man) and such ambiguities can often be avoided by using *quoting*, such as,

"John" is a name with four letters.

As we will see later, the use of quoting, though enabling the resolution of such ambiguities, causes practical problems in the area of executable logics.

Alternatively, the meta-language may be different from the object-language. For example, if we use English as a meta-language for Propositional Calculus, *well-formed formulae* (wff) can be defined as follows.

If α and β are wff, then $\alpha \wedge \beta$, $\alpha \vee \beta$, and $\alpha \Rightarrow \beta$ are wff.

Here, we can see that English is used to discuss features of the object-language, but furthermore, we see that the Greek letters α and β play an interesting rôle; they are *meta-variables*. It is intended that when using the meta-language these variables are instantiated with object-level formulae. In this way, it can be seen that a meta-language can be used to discuss the object-language to quantify over elements of the object-language. Note that α and β are instantiated with meta-language *representations* of object-language formulae, i.e., formulae of Propositional Calculus, described in English.

In this paper, we are particularly concerned with the use of logic languages and meta-languages and their relevance to computing. In computer science, a *meta-program* is a program that manipulates other programs, e.g., compilers, debuggers, etc. An interpreter provides a syntactic and semantic description of a programming language.

Before considering specific methods of representing object-level statements at the meta-level, we will look in more detail at the connection between languages, in particular the connection between object-languages and meta-languages. But first, we will review a little of the history of the use of meta-languages in logical and linguistic studies.

1.3 Some History

Theories of Truth

Tarski's main motivation for developing a theory of meta-languages was to define the semantical theory of truth [21], [11]. He developed the famous 'T schema' which is

$$(\mathbf{T}) \qquad S\ is\ true \text{ if and only if } p$$

where S is the name of the sentence p. For example, one instantiation of this scheme could be

'snow is white' is true if and only if *snow is white.*

Tarksi suggests that all definitions of truth must be instantiations of the T schema and shows that the statement S must be a sentence of the meta-language. Tarski also showed how such an approach could avoid the problems associated with logical paradoxes such as the Liar Paradox. He claimed that the Liar Paradox showed the inconsistency of 'semantically closed languages' and rejects these for this reason (see [11], [7] for more on semantically closed languages and the T schema).

The T schema is a statement in the meta-language, yet the p in the schema is a statement in the object-language. Thus, every sentence of the object-language must occur in the meta-language; also the name of every object-language sentence must occur in the meta-language.

Unfortunately, in his treatment of theories of truth [21], Tarski does not give a formal definition of the conditions for one language to be considered as the meta-language of another. He only states informally that

> "...we assume that the usual rules of definition are observed in the meta-language."

and says little about the hierarchy of languages that the use of a meta-language creates. The formation of a meta-language depends on the vague notion of one language being "essentially richer" than another. In particular, Haack [11] requires that the meta-language be essentially richer than the object-language. In fact, Tarski states that

> "For the condition of 'essential richness' of the meta-language proves to be, not only necessary, but sufficient for the construction of a satisfactory definition of truth, i.e., if the meta-language satisfies this condition, the notion of truth can be defined in it."

Though Tarski does not give any formal definition of a meta-language in his paper, he does state some interesting points regarding meta-languages. He shows that the notion of *truth-in-O* (where O is the object-language) must be given in M (the meta-language). He also states that M must contain O or translations of all sentences of O as part, plus the means to refer to expressions of O. So from this perspective, the meta-language must also contain the same expressions that O has, *with the same interpretations*.

Though not giving a general definition of constraints that must be placed on a meta-language, Tarski gives a procedure for producing a meta-language, M, for a specific object-language, O, and for producing the notion of 'true-in-O'. The steps he gives are as follows.

1. specify the syntactic structure of O.
2. specify the syntactic structure of M.
   M is to contain
   (a) either the expressions of O, or translations of the expressions of O.
   (b) syntactical vocabulary, including names of the primitive symbols of O, a concatenation symbol (for forming 'structural descriptions' of compound expressions of O), and variables ranging over expressions of O.
   (c) the usual apparatus available in a language such as M.
3. define 'satisfies-in-O'.
4. define 'true-in-O' in terms of 'satisfies-in-O'.

Axioms in the Meta-Language

In the collection of papers published as [22], Tarski again gives descriptions of the constraints that he argues must be satisfied in a language for it to be considered as a meta-language. He states that

> "The names of the expressions of the first language (O) and of the relations between them, belong to the second language, the meta-language (M, which may contain O as a part). The description of these expressions, definition of the complicated concepts, especially of those connected with a deductive theory ..... is the task of the second theory, which we shall call the *meta-theory*."

Thus, he suggests that not only should a meta-language contain every name in the object-language and a translation of every expression of the object-language, but it must also contain a theory representing the "structural-descriptive concepts consistent with our intuitions". Tarski devises a set of axioms that can be added on to a general (logical) language to enable that language to be a meta-language for another general language. Thus, given a semantics for the two languages, the set of 'true' statements of the meta-language is restricted semantically through the addition of this theory. The axioms he gives basically define a unique naming function so that all operators, terms and expressions in the object-language have a unique representation in the meta-language. Thus, the syntactic structure of the elements of the object-language are preserved as are the syntactic categories that such expressions belong to.

An example of one of Tarski's axioms is

> If x,y, z and t are expressions, then we have $x\hat{\ }y = z\hat{\ }t$ if and only if either
> 1. $x = z$ and $y = t$, or,
> 2. there is an expression, u, such that $x = z\hat{\ }u$ and $t = u\hat{\ }y$, or,
> 3. there is an expression, u, such that $z = x\hat{\ }u$ and $y = u\hat{\ }t$.

Here, x, y, z and t represent objects in the meta-language that are names of expressions in the object-language. Note also that the concatenation operator, '$\hat{\ }$', is used for constructing expressions.

Quantification and Quoting

The obvious replacement for the T schema is the axiom

$$\textbf{(D)} \qquad \forall p.\ \text{“}p\text{” is true}_O \ \textbf{iff}\ p$$

where true_O is the predicate *true-in-O* [11]. Tarksi rejects such a form as he believes that quantifying into quotation marks is meaningless. For example, he (and later, Quine [18]) believes that

Snow is white

is no more a part of

"Snow is white"

than "rat" is a part of "Socrates". Such quantification might be acceptable if the quoting mechanism was to be represented as a function, but Tarski has fundamental objections to such a scheme. Quine introduced such a quotation function and called the mechanism 'quasi-quotation', as the quantification allowed inside such quotes is restricted.

Belnap and Grover [3] develop such quoting further and give a comprehensive treatment of the meaning of quantifying within quotes. They also consider the structure of a meta-language and, during their description of quoting mechanisms, look at the cases where

1. the meta-language (M_1) is disjoint from the object language (O_1)
2. the meta-language (M_2) is overlaps the object language (O_2)
3. the meta-language (M_3) contains the object language (O_3)

Unfortunately, they do not give a formal definition of a meta-language, they just give examples and statements suggesting what should be in a meta-language. For example, they suggest that

> "M_2 has at least
> - truth functions (ordinary English phrases interpreted in the 'classical' way)
> - some grammatical and semantic predicates (which ones will be obvious from the illustration)
> - identity ('='). We also let M_2 have a modicum of set theory."

They then give examples of object-languages and meta-languages and state the following.

> "How to handle additional grammatical and semantic features should be clear even to the meanest intellect."

Curry [6] gives a further description of meta-languages in logical systems and also motivates the use of Quine's quasi-quotes [18] in that he says that we require statements such as

"$a \wedge b$" is the name of "a" and "b"

yet, without being able to quantify into quotes, this statement might just as well be taken as

"$a \wedge b$" is the name of "p" and "q"

Curry also develops definitions for 'syntactical' metatheories. Suppose we have a theory T concerning a language L, then T is said to be syntactical (relative to L) just when the statements of T refer only to the 'syntax' of L, that is, the structure of its expressions as strings of symbols. Curry also describes what elements he thinks a meta-language must contain, but again gives no formal definition. If M is a meta-language for L, then, he says, M must contain

- names of the expressions of L and ways of stating the syntactical relationships between them,
- names of the categories of objects in L.

Note that the syntactical relationships between names of expressions in L are given in a meta-theory. Thus, syntactical constraints on the object-language, L, are given by semantic constraints on the meta-language, M.

2 A Formal View of Languages and Meta-languages

2.1 Some Definitions

Languages can be defined in many different ways. For example, we could define a language as the set of strings that are the elements of that language. A more constructive approach is to define a language by using a grammar, i.e. a basic set of atomic symbols together with rules for generating syntactic objects from other syntactic categories. This is the style of definition for languages that we will use, though we will, at times, refer to the 'set of strings' representation.

DEFINITION 2.1
A *Language* is a 4-tuple of the form

$$\langle S, T, C, S_0 \rangle$$

where
S is a set of *sortnames*,
T is an S-indexed family of sets of terms
(this can be seen as defining the terms that are in each sort),
C is an $(S^* \times S)$-indexed family of sets of combinators (the combinators are effectively 'rules of production' for the sorts, showing how each term in any set in T has been constructed),
S_0 is a set of *distinguished* sortnames representing the 'identified' sorts of the grammar.

Note that later we may want to extend this definition to be a 5-tuple incorporating another element, E, which is a family of equations over T. This family effectively generates equivalence classes over the terms of the language and can be seen as extending the language definition to incorporate an equational theory.

We now move on to the concept of one language containing *names* for all the statements in another language. We first require the definition for a *naming* function.

DEFINITION 2.2
A *naming function* (or *quoting function*) for a language L_1 in another language L_2 is a one-to-one mapping from terms in L_1 to terms in L_2.

Examples of naming relations include Gödel Numbers, which are used to produce a natural number representation of statements in First-Order Logic, and the application of quotes in English, where a quoted statement can be manipulated as if it were a simple term.

DEFINITION 2.3
We say that a language L is *named* in another language M if and only if there is a naming function for L in M, i.e., there is a function, ν, of the form

$$\nu : L \rightarrow M$$

Again, First-Order Logic can be *named* in the language of natural numbers (through Gödel Numbering).

DEFINITION 2.4
A language, M is a *candidate* for a meta-language for a language L, if and only if:

1. L is named in M, and,
2. M contains within its syntactic definition
   (a) S_x, a sort containing all sortnames from L,
   (b) S_t, a sort containing the names of all terms in L,
   (c) S_c, a sort containing the names of all constructors from L, and,
   (d) S_{s_0}, a sort containing the distinguished sortnames from L.

For example, if we consider the object-language (L) as Propositional Logic, the meta-language (M) as First-Order Logic, and the naming function as a function, ν, then in M we require

1. S_x, a sort containing sortnames from L, for example Proposition, Formula, Binary-Operator, etc.
2. S_t, a sort containing all the names of terms in L, for example, if p, q, r, $p \wedge r$ are all elements of the object-language, then $\nu(p)$, $\nu(q)$, $\nu(r)$, and $\nu(p \wedge r)$ are all terms in the meta-language.
3. S_c, a sort containing all the names of constructors from L, for example, if one of the constructors in L is

   $$C_b\text{: Formula Binary-Operator Formula } \rightarrow \text{ Formula}$$

   then $\nu(C_b) \in S_c$.
4. S_{s_0}, a sort containing the distinguished sortnames from L, thus, as the set of distinguished sorts in L is simply {Formula}, then S_{s_0} contains just that sortname.

DEFINITION 2.5
A *meta-language*, M, for an object-language, L, is a language such that

1. M is a candidate for a meta-language for L, and,
2. the naming function in M is compositional with respect to the elements of L, i.e. there is a function α such that

   $$\forall f, a_1, \ldots, a_n \in L.\ \alpha(\nu(f), \nu(a_1), \ldots, \nu(a_n)) \ = \ \nu(f(a_1, \ldots, a_n))$$

Note that the α function is a meta-language mechanism for applying function names to arguments. The use of such a function could be avoided if ν was forced to map functions in the object-language to functions in the meta-language, i.e., $\nu(f)$ would be a true function rather than the name of a function. In this case, the above condition would reduce to

$$\forall f, a_1, \ldots, a_n \in L.\ \nu(f)(\nu(a_1), \ldots, \nu(a_n)) = \nu(f(a_1, \ldots, a_n))$$

2.2 Proof Theory and Model Theory for Object Languages

The definition of a language, given above, does not include issues of proof theory for that language. In this framework, such issues are supported at the meta-level. For example to obtain classical logic for a propositional object-language, we can represent in the meta-language some form of classical proof mechanism, such as Hilbert axioms, by axiomatizing the required proof theory in the meta-language.

In this way, we can support different logics. Using a propositional language composed of atoms and the set of logical symbols $\{\wedge, \Rightarrow\}$ in the usual way, we can encode intuitionistic logic proof theory by the following meta-language axioms, where quantification is over formulae, which is an object-language distinguished sort:

$$\begin{array}{rl}
\forall x, y & \text{formula}(\nu(x \Rightarrow (y \Rightarrow x))) \\
\forall x, y, z & \text{formula}(\nu((x \Rightarrow (y \Rightarrow z)) \Rightarrow ((x \Rightarrow y) \Rightarrow (x \Rightarrow z)))) \\
\forall x & \text{formula}(\nu(f \Rightarrow x)) \\
\forall x, y & \text{formula}(\nu(x)) \wedge \text{formula}(\nu(x \Rightarrow y)) \Rightarrow \text{formula}(\nu(y))
\end{array}$$

and, we can encode classical propositional logic by adding the following extra meta-language axiom:

$$\forall x, y \ \ \text{formula}(\nu(((x \Rightarrow y) \Rightarrow x) \Rightarrow x))$$

To obtain model theory for a language, we require a separate language with which to represent models. For example from the symbols, $T_{\text{bool}} = \{\top, \bot\}$, $T_{\text{binop}} = \{\vee, \wedge\}$, and $T_{\text{not}} = \{\neg\}$, we can have the language defined by the following:

if $x \in T_{\text{bool}}$ then $x \in T_{\text{lang}}$
if $x, y \in T_{\text{lang}}$ and $z \in T_{\text{bool}}$ then $xzy \in T_{\text{lang}}$
if $x \in T_{\text{lang}}$ and z is in T_{not}, then $zx \in T_{\text{lang}}$

Using this language, we can encode within the meta-language a proof theory for Boolean algebra. Furthermore, we can use the encoding of Boolean algebra as a semantic language for the encoding of classical propositional logic. A model is then based on an appropriate assignment of semantic language terms to object language terms. Correctness results for classical propositional logic are then statements in the meta-language that express certain equivalences between the proof theory of the object language and the proof theory of the semantic language.

In this way, we can use the meta-language to encode a variety of semantics for any particular encoded logic. For example for intuitionistic propositional logic, we can define semantics in terms of Kripke trees, and in terms of Heyting algebra.

We can also collapse a semantic language into the meta-language. For example, we can collapse a fragment of the above semantics for classical propositional logic that comprises of the set $T_{\text{bool}} = \{\top, \bot\}$. This fragment can then be used to make meta-statements such as the following:

$$\text{truth-value}(\nu(p \wedge q)) = \top \text{ if truth-value}(\nu(p)) = \top \text{ and truth-value}(\nu(q)) = \top$$

Such a collapsing allows the use of the object-language statement as a meta-statement without the need for quoting.

2.3 Summary

In Section 2, we have provided a definition of a language, and of a meta-language for a language. We have also argued that the proof theory for a formal system is essentially an axiomatisation of that proof theory in the meta-language. Finally we have argued that the semantics for a formal system can be captured by relating the statements of the object language to statements of another language, the semantic language, via the meta-language.

3 Relating Languages

Given two languages, L and M, the minimal requirement for M to be considered as a meta-language for L is that the sentences of L can be named in M. This minimal requirement allows us to specify, for example, both an interpreter for L in M, and a semantics for L in M.

As there seems to be some confusion on this point, we will first look at the following questions before giving more formal definitions.

- What is an interpreter?
- What is a semantics?

3.1 What is an Interpreter?

Given a language, L, an interpreter is simply some function[1] (or set of functions) that is specified in the meta-language M of L, and captures the required proof theory of some formal system. Hence given a subset of the set of strings of the language L, further strings of L can be generated according to the equivalence classes of the strings in the language.

For example, if we take the language Lisp as a meta-language for some executable logic, then we can define some function, say `exec` as follows

```
(define (exec f)
  (let ( (g (convert-to-dnf f)) )
     (cond
       ( (is-prop g)        (print "proposition is true: " f) )
       ( (is-disjunction g) (or (exec (get-left g))
                                (exec (get-right g))) )
         ....and so on.....
```

[1]What we call a function could just as easily be a predicate, a procedure, etc.

Here the 'formula' `f` is a meta-language representation of a particular formula in the object language.

Similarly, an `exec` predicate can be defined in some executable logic if we use this as a meta-language for L:

```
is_prop(p) => (exec(p) <=> is_true(p))

exec(f v g) <=> (exec(f) v exec(g))

exec(f & g) <=> (exec(f) & exec(g))
```

Note that there is no direct link between the meta-language description of the interpreter and the actual sentences of the object-language other than the fact that representations of such sentences are used as arguments to the meta-level predicate `exec`. In the first example, the effect of the interpreter is simply to output various strings when interpreting statements in the object language, whereas in the second example the truth of the predicates `exec` and `is_true` is all that is effected by the interpretation process.

As we will see later, it is often desirable to link the operations and statements carried out at the meta-level to some statements in the object-level. This involves the use of linking or *comprehension* statements, for example

```
is_true(p) <=> p
```

This is a statement in the meta-language that directly links a sentence in the meta-language to one in the object-language. Such statements usually only make sense when the object-language and meta-language are closely related. If they are both logics, then we can say that the truth of a statement at the object-level can be directly affected by the truth of a different statement at the meta-level. Such statements are often termed *reflection principles* [8], [24], [4], [20].

Finally, note that though we might have defined some interpreter function, such as `exec`, it may not even have any effect in the meta-language unless we call the interpreter explicitly, i.e., by evaluating

```
exec(Q)
```

where `Q` is some representation of an object-level statement.

3.2 What is a Semantics?

For a language, L, a semantics is simply a description of the meaning of sentences in L given in a particular meta-language, and so constitutes part of the theory of the language. Such a semantics is equivalent to the definition of an interpreter given above and, in fact, an interpreter can be seen as a version of the operational semantics of the object language.

Thus, when we give semantics of logical formula, such as

$$\mathcal{M} \models \varphi \wedge \psi \text{ if and only if } \mathcal{M} \models \varphi \text{ and } \mathcal{M} \models \psi$$

we are actually (recursively) defining some function, say `sat`, representing the satisfaction relation. For example, if the meta-language used was English (as above), we could define `sat` by

```
sat(M,phi and psi) if and only if sat(M,phi) and sat(M,psi)
```

Again, '`phi and psi`' is a meta-language representation of the object-language statement $\varphi \wedge \psi$.

As in the interpreter examples given above, the effect of this semantic function is confined to the meta-language and will remain so unless we link the meta-language definition of `sat` explicitly to some sentences in the object-language, again by using comprehension statements.

3.3 Languages for Comprehension Statements

Note that in both the interpreter and semantics cases, the comprehension statements have to be specified in some common language. If the languages L and M intersect, then it may be possible to specify such statements in the intersection, otherwise the statements must be made in some language that is a superset of both L and M.

As an example of the use of comprehension (or reflection) axioms, consider the amalgamated system described in [4]. Here, the naming of terms in the object-language L is carried out by quoting. Thus, if t is a term in L, then $\overline{t}$ is a term in M, the meta-language. They define a predicate *demo* that represents the interpreter for logical formulae and give comprehension rules such as

$$\frac{\vdash_M \; demo(\overline{A}, \overline{B})}{A \vdash_L B}$$

Note that the rules are given using a logical meta-language that is a superset of both M and L.

If L and M are the same language, then many levels of meta-language interaction can be linked. For example,

$$\frac{\vdash_L \; demo(\overline{C}, \overline{demo(\overline{A}, \overline{B})})}{C \vdash_L demo(\overline{A}, \overline{B})}$$

Thus, we can easily generate an arbitrary number of meta-levels in a system where $L = M$.

Thus, we have stated that though the minimal requirement for one language to be considered as a meta-language of another is the ability of the meta-language to name the terms of the object language, an ability to relate the two languages together is also useful. That the meta-language, M, must describe (consistently) the semantics of the object-language L is not essential, though it is obviously desirable. As long as M can describe at least *some* of the features of the object-language then M can be described as a meta-language.

3.4 Summary

We looked at languages, L, which had a naming in a meta-language M. We found that we needed some statements in the meta-language to make sure that the syntactic-structure of the object-language is preserved in the named terms in the meta-language.

We found that such statements in the meta-language effectively gave a semantics for terms in L. Such a semantics is minimal in that it says that two terms are semantically equivalent if they are syntactically equivalent. However, we may (will, probably) want to make 'better' semantics that refine semantic equivalence to encompass more than syntactic equivalence. For example, consider the two object-language terms '$1+2$' and '3' and consider a naming function on object-language terms, ν, that maps these terms as follows

object-language term	mapping via 'ν'	meta-language term
$1+2$	$\nu(1+2)$	one-plus-two
3	$\nu(3)$	three

Thus, we can induce a more refined semantics for the object-language by giving the meta-language statement

$$\text{one-plus-two} = \text{three}.$$

This can alternatively be seen as an extra equational theory added to the definition of the object-language, i.e., where the term '$1+2$' is equated with '3'.

Thus, we can either see a semantics as a statement in the meta-language referring to the names of object-language terms, or we can interpret a semantics as an equational theory added on to the object-language definition.

4 The Representation of Terms

To be able to explicitly manipulate the elements of the object language, we need to be able to use the *names* of object-level entities in sentences of the meta-language. From the work of Frege, and more recently Perlis [16], a method for achieving this is to *quote* object-level expressions to generate a name for the sequence of symbols representing the expression. Thus, the name of the object a is the symbol "a". So, if we wish to say something about the symbol representing a, rather than about a itself, we would use "a". For example,

$$\begin{array}{ll} & Q(a) \\ \text{but} & single\_char(\text{“}a\text{”}) \end{array}$$

The example used by Genesereth and Nilsson [10] is

$$\begin{array}{ll} & Large(John) \\ \text{but} & Small(\text{“}John\text{”}) \end{array}$$

meaning that John is large, but John's name is small.

When developing a meta-language of any kind (whether for a logical description, or as a meta-interpreter[2]) there are a variety of ways of representing terms of the object language in the meta-language.

The simplest (conceptually) method for achieving this is to represent all object language terms as ground terms in the meta-language. This is called the *Ground Representation* [12].

[2] We will often use the term *meta-interpreter* rather than *meta-circular interpreter*, even though any interpreter is a meta-interpreter [15].

4.1 Ground Representation

In this approach, all terms (whether ground or not) in the object-language are represented as ground terms in the meta-language, i.e.,

- each constant or variable of the object-language is represented by a unique constant of the meta-language,
- each n-ary function, n-ary predicate or n-ary connective of the object language is represented by a unique n-ary function of the meta-language.

Though this approach is simple, it is problematic when an interpreter for the object-language is to be constructed using the meta-language. As object-level variables are represented as meta-level constants, we cannot utilize the mechanisms for coping with variables provided at the meta-level for resolving object-level terms. For example, meta-level unification cannot be used to unify object-level terms as these have been represented as meta-level constants.

Using an example from Prolog, if we have ground representation for object-level terms, then we might have Prolog clauses of the form:

```
clause("P"("x") ":-" "Q"("x")).
demo(true).
demo("Q(a)").
demo(y) :- clause("y" ":-" "z"), demo(z).
```

Here, we cannot satisfactorily compute `demo("P(a)")` as the quoted terms are effectively meta-level constants; the meta-level system only sees the clauses

```
clause(A(c) ":-" B(c)).
demo(true).
demo(d).
demo(y) :- clause(e ":-" f), demo(z).
```

where `c`, `d`, `e` and `f` are new constants and `A` and `B` are new predicate symbols.

Thus, in order to implement an interpreter for the object-language in the meta-language, the mechanisms for dealing with non-ground terms must be re-coded in the meta-language. For example, unification for object-level terms has to be re-implemented in the meta-language.

This is obviously inefficient and also makes the interpreter description much more cumbersome.

In an attempt to avoid this inefficiency, a *non-ground* representation for terms can be used.

4.2 Non-Ground Representation

Rather than use a ground representation, where object-level variables are represented as meta-level constants, we now consider the representation of object-level variables as meta-level *variables*. This allows the object-level terms to be dealt with using the inference mechanisms already available in the meta-level.

There are two varieties of non-ground representation, namely *quoted* and *typed*.

4.2.1 Quoted Representation

A *quoted representation* of object-level terms uses the quoting mechanism described earlier. This produces a meta-level constant from an object level term and would seem, at first sight, to only be capable of producing a ground representation. However, Perlis [16] shows how terms can be *partially* quoted, thus preserving certain variables within the term. For example,

$$p(\text{“}q(x)\text{”}, \text{“}r\text{”}(y))$$

is a partially quoted term. Here “$q(x)$” is a new meta-level term, $s(a)$, where s is a new meta-level function symbol and a is a new meta-level constant. However, though “r” is a new meta-level function symbol, say t, y is still a variable symbol. Thus the meta-level term becomes

$$p(s(a), t(y))$$

This allows meta-level unification to be carried out on object-level variables as, in the case of y above, such variables can be treated as meta-level variables.

By appropriate nesting of quotes and by the insertion of unquoted variables within quoted expressions, we develop a meta-language suitable for describing most practical aspects of meta-reasoning. (Again, see [10] for more examples and theoretical limitations of such an approach. Note that the quoting of expressions is used in many functional programming languages, for example Scheme [19].)

4.2.2 Typed Representation

In a *typed representation*, object-level variables are represented directly as variables in the meta-level. However, all variables are typed. Consequently, variables of type $\mathcal{O}$ (object-level) can range over object-level terms, whereas variables of type $\mathcal{M}$ (meta-level) can range over meta-level terms.

Thus, the meta-level unification procedures can safely be used on object-level variables as the unification will be restricted by the type of the variable being unified.

This typing of levels of variables can (though not always — see [12]) lead to the use of higher-order logics.

When any non-ground representation is used, simple meta-interpreters can be described, such as the following example in Prolog.

```
is_true(true).
is_true(Y) :- clause(implies(X,Y)), is_true(X)
```

Since such interpreters utilize the meta-level mechanisms for dealing with variables, procedures such as unification need not be duplicated.

However, the price for this extra flexibility comes with the possibility of defining *comprehension axioms*.

4.3 Technical Problems with Comprehension Axioms

Frege introduced the first quantificational logic at the beginning of the 20th Century. In this he used certain *comprehension axioms*, such as

$$\text{is\_true}(c, \text{“}P\text{”}) \;\Leftrightarrow\; P(c).$$

Russell showed that such axioms cause Frege's system to be inconsistent. For example, if we define P by

$$P(x) \;\Leftrightarrow\; \neg\text{is\_true}(x, x)$$

and instantiate x with "P", then the system is inconsistent.

If we don't allow such axioms, there are no problems with the 'quoting' approach. However, it is often necessary to 'reflect' from meta-level statements to object-level statements and, in this case, the comprehension axioms are very useful. Russell found a way round the inconsistencies of Frege's system by introducing a 'typed' system (effectively a higher-order logic). Since that time, many researchers have used higher-order logics to enable object-level and meta-level statements to be used together (this is the method used when a typed representation is used).

Perlis [16] showed that Frege's comprehension axioms could be used and Russell's paradox avoided without recourse to higher-order logic. He described a meta-level reasoning system where the object and meta-level languages were the same (i.e., predicate calculus). Though he is forced to use an intuitionistic interpretation, his system shows how meta-languages and object languages can be described using the same language. Note that, as an alternative to Perlis' approach, Priest [17] showed how such results could be achieved by the use of paraconsistent logics.

5 Categorization of Meta-level Architectures

Van Harmelen [23] categorizes various forms of meta-level architecture. One of his axes of categorization is the level at which the majority of the activity of the system takes place. This ranges from object-level (i.e., systems where the activity takes place mainly at the object-level) to meta-level (i.e., systems where the activity takes place mainly at the meta-level).

Orthogonal to this categorization is the type of linguistic relation between levels in the meta-level system. This category consists of monolingual, bilingual and amalgamated systems.

Orthogonal to this categorization is the type of linguistic relation between levels in the meta-level system. This category consists of monolingual, bilingual and amalgamated systems.

In *monolingual* systems, the object language L and the meta-language M are the same language, and no syntactic distinction is made between object-level and meta-level expressions.

In *bilingual* systems, the meta-language is different from the object language (in fact it is often a different *type* of language from the object language). The languages are related via a naming convention which translates objects of L into terms (possibly variable-free — see [12]) of M.

In *amalgamated* systems, L and M are the same language, but a naming convention is used to separate the subsets of the language. Associated with each object-level term is a meta-level term (usually variable-free), representing the name of the object-level term.

6 Meta-level Reasoning in METATEM

We have seen from the discussion that in general terms, a meta-language is a language that is used to reason about another language, the object language. The (names of the) terms, sorts and constructors of the object language (or a subset of it) constitute terms in the meta-language. If the meta-language is a programming language, then the terms, sorts and constructors of the object language are the data for the programs.

Within the logic programming field, meta-level reasoning is of importance in interpreters, debuggers, loop checkers, etc. Effectively, meta-level reasoning provides a way of manipulating a program during the course of its execution. In particular, it provides an opportunity to extend the object language with the portion of the meta-language that deals with object-level provability relation. For logic programming, this was first explored by Bowen and Kowalski [4].

From a practical view-point, the utilisation of a meta-level in a programing language, including METATEM, allows the separation of the programming specification – represented at the object-level – from the control specification – represented at the meta-level. This access to the control specification via Meta Level makes it possible to write various useful programming tools such as compilers, interpreters and debuggers within a logical framework. It also enables loop checking and execution selection strategies to be explicitly expressed in the logic rather than having to use a separate 'control language'.

Using a logic for a meta-level for tools such as interpreters and debuggers allows us to prove properties of these tools. Such tools potentially complement other model building, model checking and theorem proving techniques. This, together with formal partial evaluation techniques also opens up the possibility of rigorously addressing the problems of comptuational efficiency in meta-level reasoning by formal partial evaluation techniques.

Finally, the language used in METATEM needs to be of a type appropriate to its usage in the application of METATEM systems. One possibility developed is a logic called FML* based on quotes [9]. This approach corresponds to the ground representation described in [12]. An alternative approach based on a typed representation [12] for the meta language with the variables partitioned into two sorts, representing object level and meta level terms respectively. Here the system can be seen either as monolingual or amalgamated. This approach follows Perlis' work, but differs slightly by using implicit, rather than explicit, quoting. For details and examples of its use see [1]. The system can be seen either as monolingual or amalgamated. This approach follows Perlis' work, but differs slightly by using implicit, rather than explicit, quoting.

Research in this direction is continuing with the goal of harnassing the power of Meta level approach for the Temporal reasoning using METATEM.

Acknowledgements

This work was supported by ESPRIT under Basic Research Action 3096 (SPEC) and Project No. 2469 (TEMPORA), and by SERC under Project GR/F/30123 (MetateM).

References

[1] H. Barringer, M. Fisher, D. Gabbay, G. Gough, and R. Owens. METATEM: A Framework for Programming in Temporal Logic. In *Proceedings of REX Workshop on Stepwise Refinement of Distributed Systems: Models, Formalisms, Correctness*, Mook, Netherlands, June 1989. (Published in *Lecture Notes in Computer Science*, volume 430, Springer Verlag).

[2] H. Barringer, M. Fisher, D. Gabbay and A. Hunter. Meta-reasoning in executable temporal logic. In *Principles of Knowledge Representation and Reasoning (KR'91)*, J. Allen, R. Fikes and E. Sandewall, pp. 40–49, Morgan Kaufmann, 1991.

[3] Nuel D. Belnap and Dorothy L. Grover. Quantifying In and Out of Quotes. In Leblanc [13], pages 17–47.

[4] K. A. Bowen and R. A. Kowalski. Amalgamating Language and Metalanguage in Logic Programming. In K. L. Clark and S.-A. Tärnlund, editors, *Logic Programming*, volume 16 of *A.P.I.C. Studies in Data Processing*, pages 153–172. Academic Press, Inc., 1982.

[5] K. L. Clark and S.-A. Tärnlund, editors. *Logic Programming*, volume 16 of *A.P.I.C. Studies in Data Processing*. Academic Press, Inc., 1982.

[6] Haskell B. Curry. *Foundations of Mathematical Logic*. Dover, New York, 1977.

[7] Donald Davidson. In Defense of Convention T. In Leblanc [13], pages 76–86.

[8] S. J. Feferman. Transfinite Recursive Progressions of Axiomatic Theories. *Journal of Symbolic Logic*, 27(3):259–316, 1962.

[9] Michael Fisher. Meta-Programming in METATEM (Draft). Department of Computer Science, University of Manchester, 1990.

[10] Michael R. Genesereth and Nils J. Nilsson. *Logical Foundations of Artificial Intelligence*. Morgan Kaufmann Publishers, Inc., Palo Alto, California, 1987.

[11] Susan Haack. *Philosophy of Logics*. Cambridge University Press, 1978.

[12] P. M. Hill and J. W. Lloyd. Analysis of Meta-Programs. In *Proceedings of the Workshop on Meta-Programming in Logic Programming*, pages 27–42, University of Bristol, U.K., June 1988.

[13] Hughes Leblanc, editor. Rees *Truth, Syntax and Modality*, volume 68 of *Studies in Logic and The Foundations of Mathematics*. North-Holland Publishing Company, Amsterdam, 1973.

[14] John Malpas. *PROLOG – A Relational Language and its Applications*. Prentice-Hall International, Englewood Cliffs, New Jersey, 1987.

[15] Ulf Nilsson and Małuszyński. *Logic, Programming and Prolog*. John Wiley and Sons, Chichester, U.K.,, 1990.

[16] Donald Perlis. Languages with Self Reference I: Foundations. *Artificial Intelligence*, 25:301–322, 1985.

[17] G. Priest. Reasoning About Truth. *Artificial Intelligence*, 39:231–244, 1989.

[18] W. V. O. Quine. *Mathematical Logic*. , Cambridge, Mass., 2nd edition edition, 1951.

[19] Jonathan A. and William Clinger. Revised^3 Report on the Algorithmic Language Scheme. *ACM Sigplan Notices*, 21(12), December 1986.

[20] J. Tanaka. An Experimental Reflective Programming System Written in GHC. Technical Report TR-506, ICOT, Minato-ku, Tokyo, Japan, September 1989.

[21] Alfred Tarski. The Semantic Conception of Truth. *Philosophy and Phenomenological Research*, 4:341–375, 1943.

[22] Alfred Tarski. *Logic, Semantics, Metamathematics: papers from 1923 to 1938*. Oxford University Press, Oxford, England, 1956.

[23] Frank van Harmelen. A Classification of Meta-Level Architectures. In *Proceedings of the Workshop on Meta-Programming in Logic Programming*, pages 81–94, University of Bristol, U.K., June 1988.

[24] R. Weyhrauch. Prologemena to a Theory of Mechanized Formal Reasoning. *Artificial Intelligence*, 13:133–170, 1980.