

Definable Naming Relations in Meta-level Systems

Frank van Harmelen

S.W.I., University of Amsterdam
e-mail: frankh@swi.psy.uva.nl

“Don’t stand chattering to yourself like that,” Humpty Dumpty said, looking at Alice for the first time, “but tell me your name and your business.”

“My *name* is Alice, but —”

“It’s a stupid name enough!” Humpty Dumpty interrupted impatiently. “What does it mean?”

“*Must* a name mean something?” Alice asked doubtfully.

“Of course it must,” Humpty Dumpty said with a short laugh: “*my* name means the shape I am — and a good handsome shape it is, too. With a name like yours, you might be any shape, almost.”

“Through the Looking Glass and What Alice Found There”,
Lewis Carroll, 1871.

Abstract. Meta-level architectures are always, implicitly or explicitly, equipped with a component that establishes a relation between their object- and meta-level layers. This so-called *naming relation* has been a neglected part of the architecture of meta-level systems. This paper argues that the naming relation can be employed to increase the expressiveness and efficiency of meta-level architectures, while preserving known logical properties. We argue that the naming relation should not be a fixed part of a meta-level architecture, but that it should be *definable* to allow suitable encoding of syntactic information. Once the naming relation is definable, we can also make it *meaningful*. That is, it can also be used to encode pragmatic and semantic information, allowing for more compact and efficient meta-theories. We explore the *formal constraints* that such a definable naming relation must satisfy, and we describe a *definition mechanism* for naming relations which is based on term rewriting systems.

1 Introduction

An important property of any meta-level system is the connection between its object- and meta-level layers. A crucial aspect of this connection is what is known as the *naming relation*. A naming relation associates elements of the object-layer with their names in the meta-layer, and thereby allows the meta-layer to refer to, and express properties of, the elements of the object-layer.

Below, we will first discuss the naming relations that have been used (either implicitly or explicitly) in the literature (Sect. 2). We will argue that these naming relations have been used in a very limited way, and that they can be used

more effectively if made *definable* (Sect. 3). Section 4 discusses the advantages of a definable naming relation, and Sect. 5 gives some examples of definable naming relations from different areas of AI. Naming relations cannot be defined arbitrarily, but must satisfy certain formal constraints which will be explored in Sect. 6. We describe a particular formalism based on term rewriting for constructing definable naming relations (Sect. 7). Comparison with related work (Sect. 8) concludes this paper.

Before we proceed a remark is in order on the scope of this paper. The discussion in this paper will be in the context of systems that use logic as their representation language: they consist of theories built out of logical expressions and are equipped with inference rules. However, many of the notions, arguments and conclusions of this paper can be easily generalised to meta-systems based on other types of languages, such as object-oriented languages [9], functional languages [12] or mixed representation languages [4].

2 Existing Naming Relations

As many important concepts in knowledge representation, the notion of a naming relation originates in work by logicians [13]. A naming relation is a mapping from syntactic constructs in the object-language \mathcal{L}_O to variable free terms in the meta-language \mathcal{L}_M . Through this mapping, object-level constructs become available for discourse in the meta-level theory. This makes it possible in the meta-level theory to quantify over object-level constructs while staying within a first order framework, without having to resort to second order logic to express properties of object-level predicates.

So far, we have been deliberately vague about the domain of the naming relation, and have described it with the term “object-level constructs”. There is a considerable amount of freedom in choosing the domain of a naming relation. This choice will depend on what constructions we wish to predicate on in the meta-theory, but in general the domain of the naming relations will always consist of constructions over the elements of \mathcal{L}_O . In the simplest case, the domain of the naming relation is \mathcal{L}_O itself, thus providing names for object-level sentences and terms. However, the domain can also be restricted to subsets of \mathcal{L}_O (say only terms), or extended to other constructions over \mathcal{L}_O such as object-level proofs (trees or sequences of object-sentences), object-theories (sets of object-sentences), etc.

Even though the domain of a naming relation can include any of these possibilities, for the purpose of examples in the remainder of this paper, we will often assume that the naming relation ranges over the elements of \mathcal{L}_O , although most of our arguments and conclusions will not crucially depend on this.

The co-domain of the naming relation must always be a set of variable free terms from \mathcal{L}_M . The fact that names are terms enables the meta-theory to express properties of object-level expressions while staying within a first order language. Names must be variable free in order to allow for the correct semantic interpretation for them in the model of a meta-theory (see Sect. 6 below).

Note that nothing in either the definition of the domain or co-domain of the naming relations excludes the possibility that we choose $\mathcal{L}_{\mathcal{O}} \subset \mathcal{L}_{\mathcal{M}}$, or even $\mathcal{L}_{\mathcal{O}} = \mathcal{L}_{\mathcal{M}}$. This possibility (discussed in [1]) allows for the famous self-referential constructions known from [5], but nothing in this paper depends on the equality or otherwise of $\mathcal{L}_{\mathcal{O}}$ and $\mathcal{L}_{\mathcal{M}}$.

Traditionally, two types of naming relations have been employed by logicians, starting again from [13], called *quotation-mark names* and *structural description names*. The simplest possible names are the quotation-mark names, which associate object-level expressions from $\mathcal{L}_{\mathcal{O}}$ with atomic constants from $\mathcal{L}_{\mathcal{M}}$. For mnemonic reasons, the name of an object-expressions $\phi \in \mathcal{L}_{\mathcal{O}}$ is often written as the constant $[\phi] \in \mathcal{L}_{\mathcal{M}}$, but the name can of course be any arbitrary constant in $\mathcal{L}_{\mathcal{M}}$.

Structural descriptive names turn object-expressions into complex terms in the meta-level theory, where these terms reflect the syntactic structure of the object-level expressions. An example would be the name

$$\text{all}(\text{var1}, \text{all}(\text{var2}, \text{imply}(\text{2pred}(\text{p}, \text{var1}, \text{var2}), \text{2pred}(\text{p}, \text{var2}, \text{var1}))))$$

in \mathcal{M} for the following sentence in $\mathcal{L}_{\mathcal{O}}$, which expresses the commutativity of the predicate p :

$$\forall x \forall y [p(x, y) \rightarrow p(y, x)]$$

(Notice that the object-variables x and y are named by meta-constants var1 and var2 to obtain a ground term in $\mathcal{L}_{\mathcal{M}}$, and that all predicates, logical constants and quantifiers of $\mathcal{L}_{\mathcal{O}}$ are named by function symbols or constants of $\mathcal{L}_{\mathcal{M}}$).

A large number of AI systems with a meta-level architecture have implemented a naming relation in the sense described above. For example, the Socrates knowledge representation system [8], directly implements a quotation-mark naming relation, whereas the logic programming systems Gödel [6] and Reflective Prolog [3] implement a structural descriptive naming relation

3 Naming Relations Should Be Definable

In all systems described in the literature (with the possible exception of FOL, but see Sect. 8 for more discussion on this system) the naming relation has been attributed an “external” status: In presentations by logicians, the naming relation is assumed to be fixed, and systems are studied with respect to different sets of axioms and rules of inference in \mathcal{O} and \mathcal{M} , but no variation in the naming relation is ever considered. Similarly, in presentations of AI systems, the user or programmer is allowed to define the contents of \mathcal{O} and \mathcal{M} (and sometimes even the languages $\mathcal{L}_{\mathcal{O}}$ and $\mathcal{L}_{\mathcal{M}}$ and their rules of inference), but the naming relation is assumed to fixed, and not available for redefinition.

This state of affairs seems to be based on a lack of appreciation of the importance of the choice of the naming relation. It appears to be a neglected but nevertheless crucial fact that the nature of the naming relation determines for a large part the expressiveness of the meta-theory. As argued in [11], a meta-theory is *model relative*, which means that the expressiveness of a meta-theory

depends on the model it has of the object-theory. A meta-theory can only refer to properties of object-expressions that are made visible on the meta-level through the naming relation, because it is the naming relation that determines for a large part the model that the meta-level has of the object-level theory. If, for example, we look at the following two naming relations:

$$\textit{object-sentence} : p(f(x)) \wedge q(g(x, y)) \quad (1)$$

$$\textit{quotation name} : [p(f(x)) \wedge q(g(x, y))] \quad (2)$$

$$\textit{structural name} : p'(f'(var1)) \wedge' q'(g'(var1, var2)) \quad (3)$$

(where \wedge' is a binary function symbol of $\mathcal{L}_{\mathcal{M}}$, written in infix notation), then it is clear that (3) is more expressive than (2), thereby allowing meta-theories to express properties which would otherwise not be expressible. An example is the statement which expresses that the commutativity of a given object-predicate p is provable in the meta-theory. Using the structural naming relation from (3), this statement can be expressed as:

$$\forall x \forall y [prove(p'(x, y)) \rightarrow prove(p'(y, x))] \quad (4)$$

but a similar statement would not be expressible using the quotation naming relation (2). The closest approximation that would be possible with this poorer naming relation is:

$$prove([\forall x \forall y [p(x, y) \rightarrow p(y, x)])]$$

which is different from (4) since it expresses that the commutativity of p is provable in the object-theory, whereas (4) asserts that this property of p is provable in the meta-theory. However, even though the structural names from (3) are more powerful than the atomic names from (2), they still restrict the expressibility of the meta-theory. For instance, they do not allow for quantification over object-level function symbols, since these are represented by meta-level function symbols, and this would require second order quantification in the meta-language. An alternative name of (1) is illustrated in the following example

$$\textit{alternative name} : p'(func(f, [var1])) \wedge' q'(func(g, [var1, var2])) \quad (5)$$

which does allow quantification over object-level function symbols. For instance the sentence

$$\forall f \forall arg [\dots func(f, [arg]) \dots]$$

is quantified over all unary function-symbols. However, this naming relation does not allow for quantification over object-level predicate symbols. Ever more complicated naming relations can be invented, for instance allowing quantification over predicate symbols, which would make the name of (1) look like:

$$pred(p, [func(f, [var1])]) \wedge' pred(q, [func(g, [var1, var2])])$$

or even allowing quantification over logical constants, yielding:

$$\textit{logical-const}(and, pred(p, [func(f, [var1])]), pred(q, [func(g, [var1, var2])]))$$

as the name of (1).

However, there is of course no end to the amount of properties that we may want to have available in the meta-theory: we may want to quantify over the arity of object-level function- or predicate-symbols, in a sorted logic we may want to include the sorts of object-level terms in their names, etc.

On the other hand, if none of these properties of object-level expressions is needed for a given meta-theory, it is advantageous to employ as simple a naming relation as possible, in order to reduce the complexity of the expressions in the meta-theory.

The conclusion of this progression of ever more general naming relations and the trade-off with complexity is of course that no single naming relation can be found that is optimal for all meta-theories, but that the richness (and therefore the complexity) of the naming relation *should be adapted to the particular requirements of a given meta-theory*. Since in many systems the definition of the contents of the meta-theory is up to the user or programmer, so should the nature of the naming relation between object- and meta-theories be left to the user or programmer. In other words, the naming relation between object- and meta-theories should be made *internal* to the system, *definable* by the user, instead of being external to the system and fixed once and for all by the system's designers, as is current practice.

4 Advantages of Definable Naming Relations

The previous section introduced the notion of a definable naming relation. In this section, we shall argue why this notion is indeed a useful one.

The first advantage of definable names has already been described above: they allow the complexity of names in $\mathcal{L}_{\mathcal{M}}$ to be tailored to the requirements of the specific meta-theory \mathcal{M} in which they are used. This means that names will only be as complex as they are required to be, and will not encode any syntactic details of expressions from $\mathcal{L}_{\mathcal{O}}$ which are not used by the axioms in \mathcal{M} . This is beneficial both from a human standpoint (since meta-level expressions will be easier to comprehend) and from a computational standpoint (since meta-level names will be easier to compute).

Readers will of course have recognised this argument as a special case of the general lesson in Computer Science that it is good to allow flexible use of datastructures, allowing users to formulate their own definitions separately from the rest of the program. We have simply applied this argument to the special case of naming relations in meta-level systems.

A second, and more fundamental advantage of a definable naming relation is that they can be used to reduce the computational complexity of \mathcal{M} . This will be the subject for the remainder of this section.

A property shared by all naming relations exploited in existing systems and in the literature, is that the names are defined uniformly across the syntax of $\mathcal{L}_{\mathcal{O}}$: syntactically similar expressions will have similar names. However, if the naming relation is made into a definable component of the system, then there is

no longer a need to only define the naming relation uniformly across the syntax of $\mathcal{L}_{\mathcal{O}}$, but we may employ the naming relation to introduce other distinctions between expressions of $\mathcal{L}_{\mathcal{O}}$. In particular, it becomes possible to encode semantic and pragmatic aspects of expressions from $\mathcal{L}_{\mathcal{O}}$ into their names in $\mathcal{L}_{\mathcal{M}}$. We have coined the phrase “*meaningful naming*” for this idea (as opposed to the more traditional “syntactic naming”), because it becomes possible to encode more than only syntactic structure into names, but we use the names also to encode both semantics (meaning) and pragmatics (use).

As we will show, this mechanism of meaningful names can be used to increase the efficiency and the intelligibility of the meta-theory, by reducing both the number of meta-level axioms and the size of the remaining axioms.

In first order logic, properties of individuals are represented by unary predicates applied to constants. Thus, properties of expressions from $\mathcal{L}_{\mathcal{O}}$ are expressed by unary predicates of from $\mathcal{L}_{\mathcal{M}}$. However, the notion of definable names allows us to encode such properties in a different way, namely no longer as unary predicates, which will occur in the axioms of \mathcal{M} , but instead in the names themselves. After all, it is no longer necessary to define names uniformly across the syntax of elements in $\mathcal{L}_{\mathcal{O}}$ (which would require that similar expressions have similar names); we can now take into account non-syntactic properties of expressions from $\mathcal{L}_{\mathcal{O}}$, and encode these properties in their names. Thus, if two expressions in $\mathcal{L}_{\mathcal{O}}$ are syntactically similar, but one has a certain property that we want to represent in \mathcal{M} , and the other does not, then we can assign different names to these expressions, which will encode these differences, even though the expressions are syntactically similar. In this way, we can exploit the naming relation to eliminate as many of the unary predicates from \mathcal{M} as we would like. We know from previous results that a transformation which removes unary predicates from a theory brings substantial gains. The transformation from an unsorted to a sorted first order theory, as described for instance in [14] also results in the removal of unary predicates from a theory (in that case, they are encoded in the sort structure), and problems which are known to be combinatorially explosive in unsorted theories can be efficiently dealt with in sorted theories (e.g. Schubert’s Steamroller). This close parallel between the introduction of sorts and the introduction of meaningful names shows how meaningful names will lead to more compact and efficient meta-theories in meta-level systems.

5 Examples of Naming Relations

In this section, we will illustrate the idea of exploiting the naming relation to encode properties of object-expressions in their names in the meta-language through a number of small examples.

5.1 Control Information in PRESS

A well-known meta-level system, and in fact one of the early systems to propagate the use of a meta-level architecture, is the equation solving system PRESS

[2]. As with many meta-level systems, the purpose of the meta-theory \mathcal{M} in PRESS is to express constraints on the search space of the object-theory \mathcal{O} . In PRESS, \mathcal{O} consists of algebraic rewrite rules, such as

$$\log_U V = W \implies V = U^W \quad (6)$$

$$T \times W + V \times W = U \implies (T + V) \times W = U \quad (7)$$

PRESS's meta-theory \mathcal{M} classifies all these rewrite rules into a number of categories, depending on the way the rule should be used in the process of solving an algebraic equation. For instance, PRESS distinguished between isolation rules and collection rules. Isolation rules isolated the single occurrence of the unknown on one side, thereby solving the equation, whereas collection rules combined multiple occurrences of the unknown into one, thereby enabling the application of isolation rules. Some axioms of PRESS's meta-theory are given in Fig. 1 (all variables are universally quantified if not stated otherwise). PRESS used structural names rather than the atomic names in Fig. 1, which have been introduced for simplified presentation. The crucial point here of course is that PRESS used *syntactic* names, be they atomic or structural.

-
- `isolation-rule(V, rule-6)`
 - `collection-rule(W, rule-7)`
 - `one-occurrence(var, lhs=rhs) ∧`
`isolation-rule(var, rule) ∧`
`apply(rule, lhs=rhs, var=rhs') → solve(var, lhs=rhs, rhs')`
 - `multiple-occurrence(var, lhs=rhs) ∧`
`collection-rule(var, rule) ∧`
`apply(rule, lhs=rhs, lhs'=rhs') ∧`
`solve(var, lhs'=rhs', rhs'') → solve(var, lhs=rhs, rhs'')`

Fig. 1. A meta-theory of PRESS

If we would employ a definable naming relation, we could exploit the names of rules (6) and (7) to encode that they are of different categories. Instead of the un-descriptive names *rule-6* and *rule-7* used in Fig. 1, we could use the more descriptive names *isolation-rule(V, 6)* and *collection-rule(W, 7)*. This would enable a new formulation of the meta-theory as in Fig. 2.

Clearly, the axiom set from Fig. 2 is smaller than the one from Fig. 1: there are fewer axioms (2 instead of 4), and the axioms that remain are simpler (both implications have lost one conjunct from their antecedent).

This transformation of the theory from Fig. 1 to the theory from 2 may look insignificant, but this is mainly due to the simplified nature of our example. As argued above, we know from parallels with sorted logic that such transformation do in fact have substantial benefits.

-
- `one-occurrence(var, lhs=rhs) ∧`
`apply(isolation-rule(var, rule), lhs=rhs, var=rhs') →`
`solve(var, lhs=rhs, rhs')`
 - `multiple-occurrence(var, lhs=rhs) ∧`
`apply(collection-rule(var, rule), lhs=rhs, lhs'=rhs') ∧`
`solve(var, lhs'=rhs', rhs'') → solve(var, lhs=rhs, rhs'')`

Fig. 2. A PRESS meta-theory using meaningful names

5.2 Control Information in Logic-programming

We can use definable names to construct a naming relation that encodes information about the degree of instantiation of formulae from $\mathcal{L}_{\mathcal{O}}$. If, for example, we have an object-theory encoding graphs as $edge(node_1, node_2)$ predicates, plus an axiom defining the transitive closure as

$$connected(N_1, N_2) \leftarrow edge(N_1, N_3) \wedge connected(N_3, N_2)$$

then we can construct a meta-theory containing the control assertion that the conjunction defining $connected(N_1, N_2)$ should be executed from left to right if N_1 is given, but from right to left if N_2 is given. We can exploit the naming relation for this purpose by assigning different names to formulae from $\mathcal{L}_{\mathcal{O}}$ depending on their degree of instantiation:

formula from $\mathcal{L}_{\mathcal{O}}$	name in $\mathcal{L}_{\mathcal{M}}$
$edge(node15, X)$	$groundvar(\lceil edge \rceil, \lceil node15 \rceil, \lceil X \rceil)$
$connected(X, node15)$	$groundvar(\lceil connected \rceil, \lceil X \rceil, \lceil node15 \rceil)$
$edge(X, node15)$	$varground(\lceil edge \rceil, \lceil X \rceil, \lceil node15 \rceil)$
$connected(X, node15)$	$varground(\lceil connected \rceil, \lceil X \rceil, \lceil node15 \rceil)$

This enables us to write down the meta-axiom:

$$before(groundvar(_, _, _), varground(_, _, _)),$$

which can be taken into account by a meta-interpreter in \mathcal{M} for \mathcal{O} .

We deliberately give this well-known example from meta-programming in logic-programming to show how a definable naming relation can be used to achieve more compact meta-theories.

5.3 Knowledge Roles in KBS

Knowledge-roles are a concept from the development of knowledge-based systems used to indicate the role that particular expressions play in the inference process. An example of knowledge roles (a term introduced by [10]) is for instance the difference between a causation-relation and a specialisation-relation. These will be used in very different ways in the inference process, even though they may be logically represented in similar ways. We can capture such different roles by using a definable naming relation, as follows:

formula from $\mathcal{L}_{\mathcal{O}}$	name in $\mathcal{L}_{\mathcal{M}}$
acute-meningitis \rightarrow bact-meningitis	type-of($[\text{acute-meningitis}]$, $[\text{bact-meningitis}]$)
meningococcus \rightarrow bact-meningitis	causes($[\text{meningococcus}]$, $[\text{bact-meningitis}]$)

and a meta-theory specifying the inference steps can then use these names to distinguish between the use of causation-relations and specialisation-relations.

6 Properties of Definable Names

Once a naming relation is a definable instead of a predefined component of a meta-level architecture, it becomes important to know what the formal properties are that a naming relation must satisfy, since these properties can no longer be enforced once and for all by the system implementer, but must be checked on the definition supplied by the user.

For the purpose of this section, we will use the 2-place symbol N to denote the naming relation, and we will use the notation $\bar{\phi}$ to denote any name of the object-expression ϕ . Thus, we have $\mathcal{N}(\phi, \bar{\phi})$ for any name $\bar{\phi}$ of ϕ . Notice that the relation \mathcal{N} is not definable in either object- or meta-theory, but is outside the scope of both. We will continue to use $[\phi]$ for the quotation name of ϕ .

6.1 Definable Names Are a Conservative Extension

It is easy to see that the introduction of non-syntactically determined names are “only” a notational device, and do not extend the logical power of our formalism. For any expression $\phi \in \mathcal{L}_{\mathcal{O}}$ with a meaningful (= non-syntactically determined) name $\bar{\phi} \in \mathcal{L}_{\mathcal{M}}$, we can use simply the syntactic name (say the atomic name $[\phi]$), and assert the additional equality $\bar{\phi} = [\phi]$ in \mathcal{M} . This shows that definable names are only a conservative extensions (everything that is expressible and provable in both object- and meta-theory with definable names is also expressible and provable without them). However, as we have argued in the previous section, the *combinatorial* properties of \mathcal{M} change for the better with the introduction of definable names.

6.2 Relational Properties of Naming

In Sect. 2, we have already discussed the domain and range of naming relations. If we write, $\mathcal{N} : \mathcal{D} \times \mathcal{R}$, then the range \mathcal{R} of the naming relation must be a set of ground terms from $\mathcal{L}_{\mathcal{M}}$. The domain \mathcal{D} can vary from definition to definition, and can be such things as sets of object-level terms, formulae, proofs, theories, etc.

Four general properties that characterise any relation are the following:

- total:** $\forall x \in \mathcal{D} \exists y \in \mathcal{R} : \mathcal{N}(x, y)$;
- surjective:** $\forall y \in \mathcal{R} \exists x \in \mathcal{D} : \mathcal{N}(x, y)$;
- functional:** $\forall x \in \mathcal{D} \forall y_1, y_2 \in \mathcal{R} : \mathcal{N}(x, y_1) \wedge \mathcal{N}(x, y_2) \rightarrow y_1 = y_2$;
- injective:** $\forall x_1, x_2 \in \mathcal{D} \forall y \in \mathcal{R} : \mathcal{N}(x_1, y) \wedge \mathcal{N}(x_2, y) \rightarrow x_1 = x_2$;

Of these, none is required (and even desired) with the exception of the last (injectivity). This is in contrast with the naming relations as defined by logicians and as implemented in the systems mentioned above, which are all total, functional and injective.

- A *non-total* naming relation implies that not all elements of \mathcal{D} have names in $\mathcal{L}_{\mathcal{M}}$, and thus some elements of \mathcal{R} cannot be described in \mathcal{M} . This can be used in systems where only some parts of $\mathcal{L}_{\mathcal{O}}$ need to be visible outside the theory. This corresponds closely to the “export” operation that is well known from programming and specification languages, and will lead to a reduction in the search space of proofs in \mathcal{M} .
- Surjectivity would imply that $\mathcal{L}_{\mathcal{M}}$ contains only terms that are names of elements of \mathcal{D} , and no terms denoting anything else. It would exclude such obvious meta-theories as those that reason about the length of proofs, which require natural numbers as terms besides terms that refer to $\mathcal{L}_{\mathcal{O}}$. This would be an absurd requirement, which is not enforced in any naming relation in the literature, either from logic or from AI. There is one case where some form of surjectivity would seem reasonable: if the meta-language $\mathcal{L}_{\mathcal{M}}$ is a sorted language, and contains sorts whose terms are meant to be only terms from $\mathcal{L}_{\mathcal{M}}$ that are names for \mathcal{D} , then we would expect \mathcal{N} to be surjective onto such sorts
- A non-functional \mathcal{N} can assign more than one name to a single element of \mathcal{D} . This is unusual in syntactic naming relations, but is useful in meaningful definition of \mathcal{N} , since it is well possible that we may want to reason about different properties of an element of \mathcal{D} . In such a case, it is more convenient to assign different names to an element from \mathcal{D} , each name encoding a different property, rather than constructing a single complicated name to encode all properties at once.
- The only property that is formally required of \mathcal{N} is its injectivity. This becomes clear when we realise that the proper semantics of a naming relation, as pointed out by [13], is that the object-theory should be regarded as a partial model of the meta-theory, in the sense that the denotations of names in \mathcal{M} (the elements of \mathcal{R}) are the corresponding expressions in \mathcal{O} (the elements of \mathcal{D}), in other words:

$$\forall x \in \mathcal{D} \forall y \in \mathcal{R} : \llbracket y \rrbracket = x \text{ iff } \mathcal{N}(x, y),$$

where $\llbracket y \rrbracket$ is the semantic denotation of y . This makes N the inverse relation of semantic denotation. Thus, if $\bar{\phi}$ is the name of ϕ , then ϕ is the denotation of $\bar{\phi}$. Since semantic denotation is required to be functional (no term is allowed to have more than one denotation), N must therefore be injective.

6.3 The Complexity of the Naming Relation

In the literature, naming is often tacitly assumed to be an operation of trivial complexity (e.g. quoting, or some direct form of syntactic isomorphism). The

following example (due to Luciano Serafini at IRST) illustrates that we must indeed put an upper bound on the complexity of N . Let \mathbb{N} be the standard model of arithmetic, and define N as follows:

$$\mathcal{N}(\phi, \bar{\phi}) \leftrightarrow \begin{cases} \bar{\phi} = \text{true}(\lceil \phi \rceil) & \text{if } \mathbb{N} \vdash \phi \\ \bar{\phi} = \text{false}(\lceil \phi \rceil) & \text{otherwise} \end{cases}$$

Suppose we have the following natural deduction rules:

$$\frac{\phi}{\text{prove}(\text{true}(\lceil \phi \rceil))} \text{ if } \mathcal{N}(\phi, \text{true}(\lceil \phi \rceil)) \qquad \frac{\phi}{\text{prove}(\text{false}(\lceil \phi \rceil))} \text{ otherwise}$$

and

$$\frac{\text{prove}(\text{true}(\lceil \phi \rceil))}{\phi} \qquad \frac{\text{prove}(\text{false}(\lceil \phi \rceil))}{\neg \phi}$$

In this way, the object-theory would be a consistent and complete axiomatization of the standard model of arithmetic, something which we know to be impossible. The construction of this example relied of course on the definition of \mathcal{N} , and in particular on the fact that the above definition was not a recursive function. Thus, we will have to require that \mathcal{N} is at least computable, but in practical systems we would expect it not only to be computable, but tractable as well. After all, a major reason for organising a system as a meta-architecture is to control the combinatorial explosion in the inference process, and we would thus expect N not to contribute to this explosion (for instance by being of exponential cost in the length of the object-formulae).

6.4 Equality of Names

If we allow the naming relation N to be non-functional (that is: some elements of \mathcal{D} have more than one name in \mathcal{R}), what is the relation among the different names $\bar{\phi}_i$ of an element $\phi \in \mathcal{D}$? The choice here is whether we want all names $\bar{\phi}_i$ of an element $\phi \in \mathcal{D}$ to be provably equal in the meta-theory or not. In other words, the question is whether we want the following:

$$\text{if } \mathcal{N}(\phi, \bar{\phi}_1) \text{ and } \mathcal{N}(\phi, \bar{\phi}_2) \text{ then } \mathcal{M} \vdash \bar{\phi}_1 = \bar{\phi}_2. \quad (8)$$

From a logical point of view, this is perfectly natural, since it amounts to having equality in the model as equality in the language. Taking the object-theory as a partial model for the meta-theory, as suggested above, this simply amounts to keeping \mathcal{M} complete with respect to its model \mathcal{O} .

It is less clear that (8) is also a desirable property from a computational point of view. After all, we would like the meta-level search space to be smaller than the object-level space. However, if we keep \mathcal{M} complete, this implies that everything provable in \mathcal{O} will also be provable in \mathcal{M} (and possibly more), thus resulting in an even larger search space. Thus, from a computational point of view, it may be advantageous to make \mathcal{M} incomplete, and in particular to have certain equalities such as those resulting from (8) *not* provable in \mathcal{M} .

Summarising: equality of names in \mathcal{M} is possible (and possibly even desirable) on logical grounds, but problematic on computational grounds.

7 Naming as a Term Rewriting System

7.1 Definitions

If we accept the arguments of the previous sections concerning the need for definable naming relations, we obviously require some mechanism that will enable us to write down definitions of naming relations. We need a formal language that we can use to express naming relations, and we would like to be able to use such a language to show that a particular definition fulfills the formal requirements that have been investigated above.

One possibility is to define a naming relation as a term-rewriting system. The motivation for this choice is the problem that the domain \mathcal{D} of a naming relation can vary between applications, and could consist of formulae, formulae-trees (proofs), formulae-sets (theories), etc. In order to minimise the assumptions we make about \mathcal{D} , we will define the naming relation as ranging over arbitrary term-trees, and assume that every possible choice for \mathcal{D} can be presented in such terms.

The choice of using rewrite systems to specify the naming relation is only meant as an example to illustrate the general point that it is both possible and advantageous to have user defined naming relation. Many other formalisms are possible to define the transformation between object-expressions and their meta-level names.

Neither the two place relation \mathcal{N} nor the rewrite rules specifying it are themselves expressible in first-order meta-level terms, since they require second order quantifications. The definition of the naming relation by the user is done *outside* the meta-object theories.

When using a set of rewrite rules to define a naming relation, a naming relation $\mathcal{N} : \mathcal{D} \rightarrow \mathcal{R}$ is realised by a finite set \mathcal{E} of rewrite rules of the form $name(\lambda) \mapsto \rho$, with λ and ρ taken from \mathcal{D} and \mathcal{R} respectively, but allowing for rule-variables ν_i to occur in both λ and ρ , where these rule-variables are taken from a set \mathcal{V} assumed to be disjoint from form \mathcal{D} and \mathcal{R} . We also allow the occurrence of subterms $name(\nu_i)$ in ρ , and assume the symbol $name$ does not occur in \mathcal{R} .

As usual, a rule $name(\lambda) \mapsto \rho$ is applicable to $\phi \in \mathcal{D}$ iff there exists a substitution for rule-variables σ such that $\sigma(\lambda) = \phi$. The result of such a rule application is then $\sigma(\rho)$. We then define the naming relation as the normal form under \mathcal{E} :

$$\mathcal{N}(\phi, \bar{\phi}) \text{ iff } name(\phi) \Downarrow = \bar{\phi}, \text{ and } \bar{\phi} \in \mathcal{R} \quad (9)$$

The intuition behind this definition is as follows: we use the rewrite system \mathcal{E} to gradually translate elements of \mathcal{D} into elements of \mathcal{R} . Each rule translates part of an element of \mathcal{D} , leaving the translation of other parts unspecified (the $name(\nu_i)$ subterms) which have to be computed by further applications of rewrite rules. Whenever the rewriting process terminates (when we have computed a normal form), we will either have completely translated our original element of \mathcal{D} into its name in \mathcal{R} , or we will have failed to do so because we have not arrived at a correct element of \mathcal{R} . This latter gives rise to a non-total naming relation.

It is important to realise that the set of rule variables \mathcal{V} is disjoint from the variables in either \mathcal{D} or \mathcal{R} . During the application of rewrite rules, only the rule variables are bound by matching λ with ϕ . During the rewrite process, the elements from \mathcal{D} are regarded as ground terms, and variables from \mathcal{O} occurring in elements of \mathcal{D} play no distinguished role in the rewrite process.

Notice that in testing whether a rule $name(\lambda) \mapsto \rho$ is applicable to an expression ϕ , only λ can possibly contain rule variables, and not ϕ , and we therefore do not require unification, but only matching in order to test for rule applicability.

As we will see in the examples below, it will turn out to be useful to impose a sort structure on \mathcal{V} and \mathcal{D} , and to require that the substitution σ respects this sort-structure.

7.2 Examples

In this section we shall illustrate naming relations as rewrite rules by defining some of the naming relations used earlier in this paper in this way.

Example 1 Binding information.

For defining the naming relation from Sect. 5.2 which encodes instantiation information, we can usefully exploit the obvious sort-structure on syntactic elements of $\mathcal{L}_{\mathcal{O}}$:

$$\begin{array}{l} \dots \\ name(\nu_1^{ps}(\nu_2^c, \nu_3^v)) \mapsto groundvar([\nu_1^{ps}], [\nu_2^c], [\nu_3^v]) \\ name(\nu_1^{ps}(\nu_3^c, \nu_2^v)) \mapsto varground([\nu_1^{ps}], [\nu_3^c], [\nu_2^v]) \\ \dots \end{array}$$

(We write ν_i^s for a rule-variable ν_i of sort s , and write ps and fs for predicate and function symbols, c for constants, v for variables, t for terms and s for sentences, arranged in the obvious hierarchy).

Example 2 Knowledge roles.

The definition of the naming relation that captures the different roles that expressions from $\mathcal{L}_{\mathcal{O}}$ play in the inference process (Sect. 5.3) differs from the previous example because it explicitly refers to particular non-logical symbols from $\mathcal{L}_{\mathcal{O}}$:

$$\begin{array}{l} name(\text{acute-meningitis} \rightarrow \text{bact-meningitis}) \mapsto \\ \quad \text{type-of}([\text{acute-bact-meningitis}], [\text{bact-meningitis}]) \\ name(\text{meningococcus} \rightarrow \text{bact-meningitis}) \mapsto \\ \quad \text{causes}([\text{meningococcus}], [\text{bact-meningitis}]) \end{array}$$

The naming relation used in the example concerning PRESS (Sect. 5.1) has a similar form, showing that the effectiveness of PRESS's meta-theory can be explained as the assignment of different knowledge roles to the rewrite rules from its object-theory.

Example 3 Purely syntactic names.

A segment of a structural naming relation, illustrated above in 3 can be formulated as follows:

$$\begin{aligned}
 \text{name}(\nu_1^c) &\mapsto [\nu_1^c]' \\
 \text{name}(\nu_2^{\text{fs}}(\nu_3^t)) &\mapsto [\nu_2^{\text{fs}}](\text{name}(\nu_3^t)) \\
 \text{name}(\nu_4^{\text{ps}}(\nu_3^t)) &\mapsto [\nu_4^{\text{ps}}](\text{name}(\nu_3^t)) \\
 \text{name}(\nu_5^s \wedge \nu_6^s) &\mapsto \text{name}(\nu_5^s) \wedge' \text{name}(\nu_6^s) \\
 &\dots
 \end{aligned}$$

This naming relation is in fact (very close to) the one used in the Gödel logic programming language (see Sect. 8 below).

By contrast, an quotation naming relation can be captured by the following trivial rewrite system:

$$\text{name}(\nu_1^{\text{any}}) \mapsto [\nu_1^{\text{any}}]$$

7.3 Properties

Section 6 argued that naming relations should be allowed to be non-functional and non-total, but are required to be injective. How is all this reflected in our realisation of the naming relation as a finite set of rewrite rules?

It is clear that defining a naming relation as rewrite rules allows for non-total and non-functional naming relations. After all, we can have rule sets with no rule applicable to a particular element of \mathcal{D} (non-total).

Similarly, \mathcal{E} is allowed to be non-canonical, allowing some elements of \mathcal{D} to have more than one normal form, thereby making the naming relation non-functional.

The only requirement that we had to enforce on a naming relation was its injectivity. A well-known result from term-rewriting theory (e.g. [7]) tells us that this property is decidable, since the injectivity of \mathcal{E} corresponds to the set of reverse rules \mathcal{E}^{-1} (mapping from \mathcal{R} to \mathcal{D}) being canonical, and being canonical is decidable for terminating finite rewrite systems. Clearly, \mathcal{E}^{-1} is finite (since \mathcal{E} is finite), and it is also terminating (proof by induction on the size of terms to be rewritten).

Finally, naming as rewriting as defined in this section is clearly efficient: it is linear in the size of the element of \mathcal{D} to be rewritten.

8 Comparison With Existing Systems

Not very many of the meta-systems in the literature consider the naming relation as a definable part of the system. In this section, we discuss the only exception we know of, namely the theorem-proving system FOL [15].

The FOL system consists of a set of theories, called contexts, some of which may form object-meta pairs, in the sense that terms from one theory refer to formulae of the other. These object-meta pairs are connected by means of the standard reflection rules for upwards and downwards reflection, and by a naming relation as required by these rules.

The FOL system is one of the few in the literature where the notion of a definable name already occurs. To be more precise, in FOL the user does not actually define naming, but instead defines the inverse operation (denotation). It is possible, by means of the **attach** command, to associate terms from the meta-theory with (the data-structures underlying) the formulae from the object-theory. This denotation relation is required to be functional, and as a result the naming relation satisfies the injectivity requirement from Sect. 6. It is also possible for a single (datastructure underlying an) object-formula to serve as the denotation of multiple meta-level terms, and as a result, naming in FOL is not required to be functional, again in accordance with Sect. 6. Among such multiple names of a single object-formula, FOL distinguishes a *preferred name*. This user-defined naming relation is used in FOL during the application of the reflection rules for the purposes of substituting object-level computation with meta-level computation or vice versa.

The major difference between FOL and the approach taken in this paper is the mechanism by which the naming relation is defined: FOL is restricted to a “point-wise” definition of the naming relation (where names are assigned to individual formulae of \mathcal{L}_O , much as in example 2), whereas our rewrite -rule formulation can be used to define the names of classes of expressions. For example, the rule

$$\nu_1^{\text{pr}}(\nu_2^{\text{t}}) \mapsto \text{unary}([\nu_1^{\text{pr}}], [\nu_2^{\text{t}}])$$

assigns a name to all unary predicates from \mathcal{L}_O , whereas an FOL user would be forced to write down the name for each separate unary predicate from \mathcal{L}_O in turn.

Furthermore, none of the papers on FOL ever use the naming relation for anything else than a purely syntactic (typically structurally descriptive) naming relation. Although FOL’s mechanisms seem to allow meaningful naming relations, this idea has not been investigated in the FOL community.

Acknowledgements

Both the contents and the presentation of this paper have greatly benefitted from extensive discussions with Fausto Giunchiglia, Alex Simpson and Luciano Serafini (all at IRST in Trento, Italy), Alan Smaill (University of Edinburgh, Scotland), and Pat Hill (University of Leeds, England). The research reported here was carried out in the course of the REFLECT project. This project is partially funded by the Esprit Basic Research Programme of the Commission of the European Communities as project number 3178. The partners in this project are The University of Amsterdam (NL), the German National Research Institute for Computer-Science GMD (D), the Netherlands Energy Research Foundation ECN (NL), and BSR Consulting (D).

References

1. K.A. Bowen and R.A. Kowalski. Amalgamating language and metalanguage in logic programming. In K. Clark and S. Tarnlund, editors, *Logic Programming*, pages 152–172. Academic Press, 1982.
2. A. Bundy and B. Welham. Using meta-level inference for selective application of multiple rewrite rule sets in algebraic manipulation. *Artificial Intelligence*, 16:189–212, 1981.
3. S. Costantini and G.A. Lanzarone. A metalogic programming language. In G. Levi and M. Martelli, editors, *Proceedings of the Sixth International Conference on Logic Programming*, pages 218–233, Lisbon, 1989. The MIT Press.
4. L. Erman, A. Scott, and P. London. Separating and integrating control in a rule-based tool. In *Proceedings of the IEEE Workshop on Principles of Knowledge Based Systems*, pages 37–43, Denver, Colorado, December 1984.
5. K. Gödel. Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I. *Monatsh. Math. Phys.*, 38:173–98, 1931. English translation in *From Frege to Gödel: a source book in Mathematical Logic, 1879-1931*, J. van Heijenoort (ed.), Harvard University Press, 1967, Cambridge, Mass.
6. P.M. Hill and J.W. Lloyd. The Gödel report (preliminary version). Technical Report TR-91-02, Computer Science Department, University of Bristol, March (Revised September '91) 1991.
7. G. Huet and D.C. Oppen. Equations and rewrite rules: a survey. In R. Book, editor, *Formal languages: perspectives and open problems*. Academic Press, 1978. Presented at the conference on formal language theory, Santa Barbara, 1979. Also: technical report CSL-111, SRI International, Menlo Park, California.
8. P. Jackson, H. Reichgelt, and F. van Harmelen. *Logic-Based Knowledge Representation*. The MIT Press, Cambridge, MA, 1989.
9. P. Maes. Reflection in an object-oriented language. In P. Maes and D. Nardi, editors, *Meta-Level Architectures and Reflection*, Amsterdam, 1988. North-Holland. Also: AI-Laboratory Memon 86-8, Vrije Universiteit Brussel.
10. J. McDermott. Preliminary steps towards a taxonomy of problem-solving methods. In S. Marcus, editor, *Automating Knowledge Acquisition for Expert Systems*, pages 225–255. Kluwer Academic Publishers, The Netherlands, 1988.
11. B. Smith. Reflection and semantics in a procedural language. Technical Report TR-272, MIT, Computer Science Lab., Cambridge, Massachusetts, 1982.
12. B. Smith. Reflection and semantics in Lisp. In *Proc. 11th ACM Symposium on Principles of Programming Languages*, pages 23–35, Salt Lake City, Utah, 1984. also: Xerox PARC Intelligent Systems Laboratory Technical Report ISL-5.
13. A. Tarski. Der Wahrheitsbegriff in den formalisierten Sprachen. *Studia Philosophica*, 1:261–405, 1936. English translation in *Logic, Semantics, Metamathematics*, A. Tarski, Oxford University Press, 1956.
14. C. Walther. A mechanical solution of Schuberts steamroller by many-sorted resolution. *Artificial Intelligence*, 26(2):217–224, 1985.
15. R. Weyhrauch. Prolegomena to a theory of mechanized formal reasoning. *Artificial Intelligence*, 13, 1980. Also in: *Readings in Artificial Intelligence*, Webber, B.L. and Nilsson, N.J. (eds.), Tioga publishing, Palo Alto, CA, 1981, pp. 173-191. Also in: *Readings in Knowledge Representation*, Brachman, R.J. and Levesque, H.J. (eds.), Morgan Kaufman, California, 1985, pp. 309-328.