# Reflective Metalogical Frameworks

David Basin
*Institut für Informatik, Universität Freiburg, Germany*

Manuel Clavel
*Department of Philosophy, University of Navarre, Spain*

José Meseguer
*Computer Science Laboratory, SRI International, USA*

**Abstract**

A metalogical framework is a logic with an associated methodology that is used to represent other logics and to reason about their metalogical properties. We propose that logical frameworks can be good metalogical frameworks when their logics support reflective reasoning and their theories always have initial models. We present a concrete realization of this idea in rewriting logic and report on experiments with the Maude system.

## 1 Introduction

A logical framework is a formal logic with an associated methodology that is employed for representing and using other logics, theories, and, more generally, formal systems. A minimum requirement for a logical framework is that object logics and their entailment relations can be conservatively represented in the framework logic. Typically we also demand more. For example, that the representation preserves appropriate kinds of structure and that there is a small conceptual distance between the object logic and its representation and use in the framework logic.

To compare logical frameworks and analyze their effectiveness, it is helpful to make further distinctions concerning their intended application. In particular, we can distinguish between *logical frameworks*, where the emphasis is on reasoning *in* a logic, in the sense of simulating its derivations in the framework logic, and *metalogical frameworks*, where the emphasis is on reasoning *about* logics. Metalogical frameworks are more powerful, as they include the ability to reason about a logic's entailment relation as opposed to merely being adequate to demonstrate entailment. Moreover, if a metalogical framework should provide a basis for formal metatheory, it should also support reasoning about relationships *between* logics. This is standard in metamathematics and is common practice when reasoning about formal systems in computer science.

The different kinds of applications make different demands on the framework logic. In a logical framework it is sufficient to use representations of proof rules to construct demonstrations of (object logic) entailments. This is the approach taken in logical frameworks like Isabelle [28] and the Edinburgh LF [15]. Note that under this approach one may formalize logics and theories where induction is present *within* the theory (e.g., induction over the natural numbers in Peano Arithmetic), but induction is not present *over* the encoded theories. That is, the framework logic does not support induction over the terms and proofs of a theory, and in general there is no reason to assume that sound induction principles exist.

In a metalogical framework, it is essential to have induction over theories. When reasoning about logics, standard proof-theoretic arguments usually require induction over the formulae or derivations of the object logic. Induction is also one of the key concepts in reasoning about formal systems in computer science, e.g., programming language semantics.

## Standard approaches to metalogical frameworks

Several approaches have been considered in the past to strengthen logical frameworks so that they can function as metalogical frameworks. One approach, which we might summarize as "modules with explicit induction," is to formalize theories in a framework logic supporting some notion of a module, where each module comes with its own, explicitly given, induction principle. For example, in [1], theories were specified by collections of parameterized modules ($\Sigma$-types) within the Nuprl type theory (a constructive, higher-order logic), and each module included its own induction principle for reasoning about terms or proofs. This approach can be very powerful and can be used to show, e.g., that rules are admissible, or to relate different theories.

An alternative approach is to formalize theories directly using inductive definitions in a framework logic or framework theory that is strong enough to formalize the corresponding induction principles. A simple example of this is the first-order theory $FS_0$ of [13], which has been used by [20] to carry out experiments in formal metatheory. In $FS_0$, inductive definitions are terms in the framework logic, and the framework logic has an induction rule for reasoning about such terms. Another common choice is formalization of inductive definitions in strong "foundational" framework logics like higher-order logic or set-theory [27, 14], or calculi like the calculus of constructions with inductive definitions [26]. In higher-order logic and set theory one can internally develop a theory of inductive definitions, where inductive definitions correspond to terms in the metatheory (e.g., formalized as the least-fixedpoint of a monotonic function) and, from the definition, induction principles are formally derived within the framework logic. Alternatively, in the calculus of constructions, given an inductive definition, induction principles are simply added, soundly, to the metalogic. Current research in this area focuses on appropriate induction principles for logics with higher-order quantification, which support higher-order abstract syntax [11, 21, 29].

## A new paradigm

In this paper we propose a new alternative: in some cases we can take the step from a logical framework to a metalogical framework by augmenting the logical framework with reflection and induction. This is the case when formalization of theories in the framework logic support induction principles that can be reflected back into the framework logic. This proposal can be summarized with the slogan *"logical frameworks with reflection and initiality are metalogical frameworks,"* which can in turn be expressed by the formula

$$\text{Logical Framework} + \text{Reflection} + \text{Initiality} \Longrightarrow \text{Metalogical Framework}. \tag{1}$$

After making this idea precise, we present a concrete realization of it using rewriting logic, which we use to support the thesis that combining a logical framework with reflection can result in an effective metalogical framework. Rewriting logic is not the only candidate for a reflective logical framework, but we believe it is a good one. Rewriting logic has been demonstrated to be a good logical framework [17, 24]. Moreover the logic is balanced on a point where it is strong enough to naturally formalize different entailment systems, but it is weak enough that its theories or modules always have initial models. This means that induction on these initial models is a sound reasoning principle. The key then is to *reflect* these reasoning principles into the logic.

To sum up, we see our contributions as both theoretical and practical. Theoretically, our work contributes to answering the question "*what is a metalogical framework?*" by proposing reflective logical frameworks, whose theories have initial models, as a possible answer. Moreover, it illuminates the interrelationship between logical and metalogical frameworks, and the role of reflection as a key ingredient for turning a logical framework with initial models into a metalogical one. Practically, our case study shows that rewriting logic, combined with reflection, is an effective metalogical framework that is well suited for nontrivial kinds of metatheoretic reasoning.

# 2 Reflective metalogical frameworks

## 2.1 Reflective logics

Intuitively, a reflective logic is a logic in which important aspects of its metatheory can be represented at the object level — that is, *in* the logic. Two standard metatheoretic notions that can be so reflected are theories[1] and entailment relations.

A general axiomatic notion of reflective logic was recently proposed in [9, 4] . The key concept is the notion of a *universal theory.* Let $\vdash$ be the entailment relation defined in a logic, let $T$ be a theory over a signature $\Sigma$, and let $sen(T)$ be the set of $\Sigma$-sentences. Then, given a set of theories $\mathcal{C}$, a theory $U$ is $\mathcal{C}$-*universal* if there is a function, called a *representation function,*

$$\overline{(\_ \vdash \_)} : \bigcup_{T \in \mathcal{C}} \{T\} \times sen(T) \longrightarrow sen(U)\,,$$

such that for each $T \in \mathcal{C}, \varphi \in sen(T)$,

$$T \vdash \varphi \iff U \vdash \overline{T \vdash \varphi}\,. \tag{2}$$

A logic is *reflective* when it contains a theory $U$ that is $\mathcal{C}$-*universal* and, in addition, $U \in \mathcal{C}$. Note that in a reflective logic, since $U$ itself is representable, representation can be iterated; hence we immediately have a "reflective tower"

$$T \vdash \varphi \iff U \vdash \overline{T \vdash \varphi} \iff U \vdash \overline{\overline{U} \vdash \overline{T \vdash \varphi}} \ldots .$$

Note that if a framework logic is to support flexible metatheoretic reasoning, e.g., where we can compare theories and reason about families of theories, then, in practice we require more than a representation function. Namely, we require a *theory representation calculus.* We will not formalize requirements for this here (see [12]), but such a calculus will typically treat theories as *first-class objects* and provide constructors, destructors, and discriminators, for building and reasoning about theories.

## 2.2 Reflecting induction

In the introduction, we proposed that logical frameworks with reflection and initiality can be used as effective metalogical frameworks. Above we have given a logic-independent account of reflection. We now consider what we require from a logical framework so that we can use reflection to augment theories with induction principles. In particular, if the combination is to be a useful metalogical framework, we also require that:

1. the logical framework must be *weak* enough so that there are valid induction principles for reasoning about all formalized theories, and

2. *strong* enough so that it really is a viable logical framework.

We argue this as follows. If 2 is satisfied, then object logics and their entailment relations can be represented as theories in the logical framework, and if 1 is also satisfied, then theories in the logical framework, including those representing object logics, admit induction. Now, if the logical

[1] In this paper we consider a *theory* as a pair $T = (\Sigma, \Gamma)$ consisting of a language syntax $\Sigma$, called a *signature*, and a collection of axioms $\Gamma$. A logician will typically treat the theory's language and its theorems extensionally as sets. However, it is more practical (computationally) to specify the sets using constructors: $\Sigma$ for building formulae, and $\Gamma$ for building proofs of theorems.

framework is reflective, then it contains a universal theory where theories can be represented. It is then possible to extend the universal theory so that sound reasoning principles — in particular, induction — for each theory in the logic can be *reflected*; that is, sound reasoning principles for each theory can be added to the universal theory.

### 2.3 Induction and initiality

How can we capture the notion of induction in an abstract and logic-independent way? If the framework logic is such that its theories have *initial models*, then inductive reasoning principles can be soundly added to a theory to derive sentences valid in its initial model. This method is very general; for example, for equational logic, induction and initiality are equivalent concepts [25], and for different propositional calculi, cut elimination results can be seen as inductive properties of their initial categorical models [16].

We are therefore interested in a reflective metalogical framework having a universal theory $U$ such that each theory in the framework logic has initial models. Under such circumstances it can be possible (as indeed it is the case for rewriting logic) to extend the theory $U$ to a theory $I$ that *adds sound inductive principles* to each theory in the logic, including $U$ itself. This means that we can use reflection in $I$ to reason soundly about the inductive properties of any theory $T$.

This approach can be surprisingly powerful. Since $U$ represents *all* theories in the logic, by reasoning by induction on $U$ or its extension $I$ we may be able to inductively reason about properties satisfied not only by a single theory, but by entire *families* of theories. We will give an example of this later, namely, a metatheorem not just for a single logic, but for a family of logics.

## 3 Rewriting logic as a reflective metalogical framework

We now show how the above, abstractly presented, ideas can be concretely realized. Our realization in rewriting logic supports reflection and initiality as described above, and theories are first-class objects in the universal theory. Moreover, its implementation in the Maude system supports object level reasoning via metalevel computation in (an extension of) the universal theory: any reasoning *in* the object logics (e.g., to show that formulae are syntactically well-formed or provable) can be performed by reflection down, that is, by computation in the theory that represents the object logic.

### 3.1 Rewriting logic

Rewriting logic [22] is a simple logic whose sentences are sequents of the form

$$t \longrightarrow t'$$

with $t$ and $t'$ $\Sigma$-terms on a given signature $\Sigma$. From the logical point of view, we can think of rewriting logic as a framework logic in which any inference system can be naturally formalized by expressing each inference rule as a (possibly conditional) rewrite rule.

Theories in rewriting logic are triples $(\Sigma, E, R)$, with $\Sigma$ a signature of operators, $E$ a set of $\Sigma$-equations, and $R$ a collection of (possibly conditional) rewrite rules. The inference rules of rewriting logic [22] allow the derivation of all rewrites possible in a given theory. Rewriting is understood *modulo* the equations $E$. This makes inference flexible and abstract since the equations $E$ can take care of structural bookkeeping. For example, structural rules for sequents can be "internalized" by rewriting modulo appropriate equational axioms.

Since a rewrite theory $(\Sigma, E, R)$ has an underlying equational theory $(\Sigma, E)$, rewriting logic is parameterized by the choice of the underlying equational logic. An attractive choice in terms of expressiveness is *membership equational logic* [23], a logic that has sorts, subsorts, and overloading of function symbols, and is capable of expressing partiality using equational conditions. Atomic sentences are equations $t = t'$ and membership assertions $t : s$, with $s$ a sort. General axioms are Horn clauses on such atoms. Since we can view an equational theory $(\Sigma, E)$ as a rewrite theory $(\Sigma, E, \emptyset)$, there is an obvious sublogic inclusion $MEqtl \subseteq RWLogic$, from membership equational logic into rewriting logic. Both membership equational logic and rewriting logic have initial models [23, 22].

### 3.2 Rewriting logic is a good logical framework

Rewriting logic is noncommittal about the structure and properties of the formulae expressed by $\Sigma$-terms. They are user-definable as an algebraic data type satisfying equational axioms, so that rewriting deduction takes place modulo such axioms. Because of this ecumenical neutrality, rewriting logic has good properties as a logical framework. In [17, 18, 19], many examples of logic representations are given, including first-order linear logic, sequent presentations of modal and propositional logics, Horn logic with equality, and so on. In all such examples, the *representational distance* between the object logic and its representation is practically zero, that is, the representations are direct and faithfully mimic the original logics.

Note that there are several ways of conservatively representing a logic (with a finitary syntax and inference system) within rewriting logic. A simple and direct way is to turn the inference rules into rewrite rules, which may be conditional if the inference rules have side conditions. Alternatively, we can use the underlying membership equational logic to represent theoremhood in a logic as a sort in a membership equational theory. Conditional membership equations then directly support the representation of rules as schemas, which is typically used in presenting logics and formal systems. This is the approach we have adopted in the experimental work that we report in Section 4.

### 3.3 Rewriting logic is reflective

Rewriting logic is reflective [10, 4]. There is a universal theory `UNIVERSAL`, and a representation function $\overline{(\_ \vdash \_)}$ encoding pairs consisting of a rewrite theory $T$ and a sentence in it as sentences in `UNIVERSAL`. For any finitely presented rewrite theory $T$ (including `UNIVERSAL` itself) and any terms $t$, $t'$ in $T$, the representation function is defined by

$$\overline{T \vdash t \longrightarrow t'} = \langle \overline{T}, \overline{t} \rangle \longrightarrow \langle \overline{T}, \overline{t'} \rangle\,,$$

where $\overline{T}$, $\overline{t}$, $\overline{t'}$ are terms in `UNIVERSAL`. Then, the equivalence (2) for rewriting logic that is proved in [10, 4] takes the form

$$T \vdash t \longrightarrow t' \Longleftrightarrow \texttt{UNIVERSAL} \vdash \langle \overline{T}, \overline{t} \rangle \longrightarrow \langle \overline{T}, \overline{t'} \rangle\,.$$

## 4 Maude and experimental work

In this section we report on a case study in metatheoretic reasoning that is based on the above ideas. For our study we used Maude [8, 6], which is a reflective logic based on rewriting logic. Maude's implementation has been designed with the explicit aims of supporting executable specification and reflective computation.

### 4.1 Maude's metalevel

Maude's language design and implementation make systematic use of the fact that rewriting logic is reflective to give the user a well-defined gateway to the metatheory of rewriting logic [5]. This entry point is the predefined module `META-LEVEL`, which provides the user with the functionality necessary to exploit the universal theory for rewriting logic. In the module `META-LEVEL`, terms in modules are reified as elements of a data type `Term`, and Maude modules (that is, theories with initial semantics) are reified as elements of a data type `Module`.

We illustrate the general syntax for representing modules, with a simple example: a module `NAT` for natural numbers with zero and successor and with a commutative addition operator.

```
fmod NAT is
  sorts Zero Nat .
  subsort Zero < Nat .
  op 0 : -> Zero .
  op s : Nat -> Nat .
  op _+_ : Nat Nat -> Nat [comm] .
  vars N M : Nat .
  eq 0 + N = N .
  eq s(N) + M = s(N + M) .
endfm
```

The representation $\overline{\texttt{NAT}}$ of `NAT` in `META-LEVEL` is the term

```
fmod 'NAT is
  nil
  sorts 'Zero ; 'Nat .
  subsort 'Zero < 'Nat .
  op '0 : nil -> 'Zero [none] .
  op 's : 'Nat -> 'Nat [none] .
  op '_+_ : 'Nat 'Nat -> 'Nat [comm] .
  var 'N : 'Nat . var 'M : 'Nat .
  none
  eq '_+_[{'0}'Nat, 'N] = 'N .
  eq '_+_['s['N], 'M] = 's['_+_['N, 'M]] .
endfm
```

of sort `Module`.

The processes of reducing a term to normal form in a functional module (that is, a Church-Rosser and terminating equational theory) and of rewriting a term in a system module (that is, a rewrite theory) using Maude's default interpreter are reified respectively by functions `meta-reduce` and `meta-apply`. In particular, `meta-reduce` takes as arguments the representations of a module $T$, and of a term $t$ or a membership predicate $t : s$ in that module. When the second argument is the representation $\overline{t}$ of a term $t$ in $T$, `meta-reduce` returns the representation of the fully reduced form of the term $t$ using the equations in $T$. Similarly, when the second argument of `meta-reduce` is the representation of a membership predicate $t : s$, the term $t$ is fully reduced using the equations in $T$ and then the representation of the Boolean value of the corresponding predicate is returned. Hence `meta-reduce` returns `{'true}'Bool` if $T \vdash t : s$; otherwise, it returns `{'false}'Bool`.

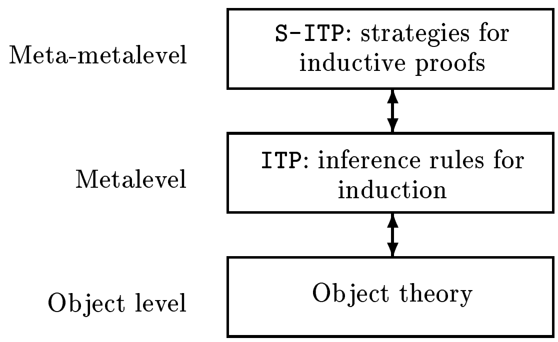
## 4.2 Internal strategies

Since the Maude system is a particular implementation of the metatheory of rewriting logic, the module `META-LEVEL` also provides gateway to the Maude system itself. By extending `META-LEVEL`, the user can effectively customize Maude (in Maude) to fit his particular computational needs. Using rewriting rules at the metalevel, user-definable *internal strategy languages* can be defined to change the (default) operational semantics of Maude for system modules (that is, for rewrite theories that need not be Church-Rosser or terminating) [10, 4]. The idea is to use the functions `meta-reduce` and `meta-apply` as basic strategies, and then to extend the module `META-LEVEL` by additional (arbitrarily complex) strategy functions, defined by rewrite rules.

### 4.3 An inductive theorem prover in Maude

To reflect and use induction principles, we formalize an appropriate deductive system in Maude. Furthermore, we specify strategies for applying rules in this system by specifying rewriting strategies.

In general, based on the concepts of reflection and internal strategy languages, theorem-proving tools have a simple "reflective" design in Maude [7]. An inductive theorem prover, which we implemented for metatheoretic reasoning, illustrates this. The idea is that the theory $T$, for which we want to prove inductive theorems, is at the object level; an inference system $\mathcal{I}$ for inductive proofs uses $T$ as data and therefore should be specified as a rewrite theory at the metalevel; then, different *proof tactics* to guide the application of the rewrite rules specifying the inference rules in $\mathcal{I}$ are strategies that can be represented at the meta-metalevel. This is illustrated by the following picture:




The module `ITP` is an extension of the module `META-LEVEL` and realizes, for the case of rewriting logic, the extension $I$ of the universal theory $U$ with inductive principles discussed in Section 2.3.

Formulas are represented in `ITP` as terms of sort `Formula` built with the constructors `equality`, `implication`, `conjunction`, and `VQuantification`. For example, the formula

$$\forall\{\mathtt{N,M}\}\mathtt{+(N,\ M)} = \mathtt{+(M,\ N)}$$

is represented in `ITP` by the term

```
VQuantification(('N ; 'M), equality('+['N, 'M], '+['M, 'N])).
```

The (sub)goals for the inductive theorem prover are represented with the constructors `proveinInitial` and `proveinVariety`, for proofs in the initial model and proofs in the variety, respectively. Sets of (sub)goals are built with the constructor `goalSet`, with `emptyGoalSet` the empty set of goals. For example, the goal

$$\mathtt{NAT} \vdash_{\text{ind}} (\forall\{\mathtt{N,\ M}\})\mathtt{+(N,\ M)} = \mathtt{+(M,\ N)}$$

is represented in `ITP` by the term

$\mathtt{proveinInitial(}I\mathtt{,\ }\overline{\mathtt{NAT}}\mathtt{,}$
```
   VQuantification(('N ; 'M), equality('+['N, 'M], '+['M, 'N]))),
```

where $I$ should be a string of positive numbers. The strings of positive numbers are used to number the (sub)goals in a proof.

With this machinery in hand, it is possible to formalize in `ITP` induction principles for Maude modules. In our work, we formalize rewrite rules that specify the rules of inference for proving that a universally quantified formula is an inductive consequence of a given membership algebra specification. For example, the rule `induction` below rewrites a (sub)goal representing the task of proving inductively in a module $M$ a given formula $\forall\{x, X\}\phi$ to a set of subgoals representing the tasks of proving inductively the base case(s) and the induction step(s) that result from induction on the variable $x$. The function `getVars` extracts the variable declaration from the metarepresentation of the module $M$. The function `findSortV` finds the metarepresentation of the sort $s$ of the variable $x$ in the module $M$. The function `extractRuleSystem` extracts from the metarepresentation of the module $M$ all the clauses that define the set $s$ in $M$. (Notice that specifications in membership equational logic coincide with a special case of many-sorted Horn logic with equality.) Finally, the function `makeNewGoalSetF` generates from the defining clauses of $s$ the corresponding base case(s) and induction step(s).

```
var Idx : IntString . var Mod : Module .
var X : Qid . var Xs : QidSet .
vars Alpha Beta : Formula .

rl [induction]:
    proveInitial(Idx, Mod,
                 VQuantification((X ; Xs), Alpha))
   =>
   makeNewGoalSetF(intString(Idx, 1), Mod, Xs, X,  Alpha,
                  extractRuleSystem(Mod, findSortV(X, getVars(Mod)))) .
```

Proving a theorem consists then in applying (with a strategy and, therefore, in the module `S-ITP`, at the meta-metalevel) the rewrite rules in the module `ITP` to the term representing the initial (sub)goals until it is rewritten to the term `empty`.

## 4.4 An example: the deduction theorem

As an example, consider the deduction theorem for *minimal logic (of implication)*. This theorem is interesting for several reasons. To begin with, it is a central metatheorem that holds for many Hilbert systems and justifies proof under temporary assumption in the manner of a natural deduction proof system. Moreover, although relatively simple, it illustrates some subtle aspects of formal metareasoning. For example, it is actually a metatheorem not about *a particular* deductive system, but rather one that relates *different* deductive systems: one in which $A \to B$ is proven and a second (which is the first, augmented by the axiom $A$) in which $B$ is proven. Indeed, as $A$ is an arbitrary formula, the standard statement of the deduction theorem is actually a statement about the relationship between a family of pairs of deductive systems. And as we will see, it can be formalized even more generally than this.

The deduction theorem is proven by induction over the structure of derivations. We start by specifying minimal logic as a module in Maude. The formulae of minimal logic correspond to members of the set $\mathsf{L}_{\mathcal{M}}$, built from the binary connective $\to$ (written infix, associating to the right) and sentential constants. Theorems correspond to members of a second set $\mathsf{T}_{\mathcal{M}}$, and are either instances of the standard Hilbert axiom schemata

$$A \to B \to A$$

and

$$(A \to B) \to (A \to B \to C) \to (A \to C)\,,$$

or are generated by applying the rule *modus ponens*:

$$\frac{A \quad A \to B}{B}$$

The module `MINIMAL` below represents minimal logic within membership equational logic (and rewriting logic), in the sense that a formula $\phi$ is a theorem in minimal logic if and only if its representation $\overline{\phi}$ is a term of sort `Theorem`, that is, the membership assertion $\overline{\phi}$ : `Theorem` is true in `MINIMAL`.

```
mod MINIMAL is
  sorts SentConstant Formula Theorem .
  subsort SentConstant < Formula .
  subsort Theorem < Formula .
  op -> : Formula Formula -> Formula .
  vars A B C : Formula .
  mb A -> (B -> A) :  Theorem .
  mb (A -> B) -> ((A ->(B -> C)) -> (A -> C)) : Theorem .
  cmb B : Theorem if (A -> B) : Theorem and  A : Theorem .
endm
```

We write $\vdash_{\mathcal{M}} A$ to denote that $A \in \mathsf{T}_{\mathcal{M}}$, and $A \vdash_{\mathcal{M}} B$ to denote that if minimal logic is extended with the additional axiom $A$, then $B$ belongs to the resulting set of theorems. The deduction theorem then states that for any $A$ and $B$ in $\mathsf{L}_{\mathcal{M}}$,

$$A \vdash_{\mathcal{M}} B \quad \Longrightarrow \quad \vdash_{\mathcal{M}} A \to B\,.$$

This metatheorem is proven by induction on the structure of derivations in minimal logic extended with the axiom $A$.

According to our representation of minimal logic in rewriting logic, we can rephrase the deduction theorem in the following terms: for any formulae $A$ and $B$, if $\overline{B}$:`Theorem` is true in the module `MINIMAL` extended with the membership axiom `mb` $\overline{A}$`:Theorem`, then $\overline{A \longrightarrow B}$`:Theorem` is true in `MINIMAL`.

Notice that this theorem states an implication between the truth of two membership assertions over *two different* membership equational theories. Since the truth of membership assertions over theories is defined in the metatheory of rewriting logic, the "object" theory about which we have to prove the deduction theorem is in fact, in our setting, the universal theory for rewriting logic. This corresponds to the following goal for the inductive theorem prover, where $\boxed{\texttt{A}}$ and $\boxed{\texttt{B}}$ are variables of sort `Term` in the module `META-LEVEL`:

```
META-LEVEL ⊢ind ∀([A],[B])
meta-reduce(
    (mod'ARROW is
        including 'BOOL .
        sorts('SentConstant ; 'Formula ; 'Theorem) .
        subsort 'SentConstant < 'Formula .
        subsort 'Theorem < 'Formula .
        op 'impl :  'Formula 'Formula -> 'Formula [none] .
        var 'A : 'Formula .  var 'B : 'Formula .  var 'C : 'Formula .
        mb [A] :  'Theorem .
        mb 'impl['A, 'impl['B, 'A]] : 'Theorem .
        mb 'impl['impl['A, 'B],
                 'impl['impl['A, 'impl['B, 'C]], 'impl['A, 'C]]] : 'Theorem .
        cmb'B : 'Theorem if
           '_and_[('impl['A, 'B] : 'Theorem), ('A : 'Theorem)] = {'true}'Bool .
        none
        none
    endm),
    [B] :  'Theorem) = {'true}'Bool
⟹
meta-reduce(
    (mod'ARROW is
        including 'BOOL .
        sorts('SentConstant ; 'Formula ; 'Theorem) .
```

```
      subsort 'SentConstant < 'Formula .
      subsort 'Theorem < 'Formula .
      op 'impl :  'Formula 'Formula -> 'Formula [none] .
      var 'A : 'Formula .  var 'B : 'Formula .  var 'C : 'Formula .
      mb 'impl['A, 'impl['B, 'A]] : 'Theorem .
      mb 'impl['impl['A, 'B],
               'impl['impl['A, 'impl['B, 'C]], 'impl['A, 'C]]] : 'Theorem .
      cmb'B : 'Theorem if
         '_and_[('impl['A, 'B] : 'Theorem), ('A : 'Theorem)] = {'true}'Bool .
      none
      none
   endm),
   'impl[$\boxed{A}$, $\boxed{B}$] :  'Theorem) = {'true}'Bool .
```

Observe that applying induction on the variable $\boxed{\texttt{B}}$ using the `induction` rule introduced above will be of little use here: $\boxed{\texttt{B}}$ is a variable of sort `Term` and, therefore, the base case(s) and the induction step(s) that the function `makeNewGoalSetF` will generate correspond to the clauses that define the set `Term` in the module `META-LEVEL`. Instead what we need are the base case(s) and the induction step(s) that correspond to the clauses defining the subset of the set `Term` that includes only those terms of sort `Term` representing at the metalevel terms of sort `Theorem` in the module `MINIMAL` extended with the membership axiom `mb` $\boxed{\texttt{A}}$`:Theorem`.

To generate the appropriate induction, we extend the module `ITP` with a new rule `induction*`. This rule generates the appropriate base case(s) and induction step(s) when proving in the module `META-LEVEL` a universally quantified implicative formula $\forall\{x, X\}(\beta_1 \wedge \cdots \wedge \beta_n) \longrightarrow \alpha$ by induction on a variable $x$ of sort `Term`, if the implicative formula includes in its antecedent a clause $\beta_i$ that restricts the scope of the variable $x$ to metarepresentations of terms of a sort $s$ in a module $T$. The function `makeNewGoalSetF*` uses the set of clauses that define the set $s$ in the module $T$ (obtained with the function `extractRuleSystem`) to generate the appropriate base case(s) and induction step(s).

```
rl [induction*]:
   proveInitial(Idx, $\overline{\text{META-LEVEL}}$,
        VQuantification((X ; Xs),
        implication(
            conjunction(Beta,
                  equality('meta-reduce[$\overline{\overline{\overline{T}}}$, X : $\overline{\overline{s}}$] = $\overline{\overline{\text{\{'true\}'Bool}}}$)),
            Alpha)))
   =>
   makeNewGoalSetF*(intString(Idx, 1), $\overline{\text{META-LEVEL}}$, Xs, X, Beta, Alpha,
          extractRuleSystem($\overline{T}$, $\overline{s}$)) .
```

Using the rule `induction*`, along with the rest of inference rules specified in `ITP`, we have proven this metatheorem with a strategy defined in `S-ITP` that mirrors the standard presentation of the proof of the deduction theorem.

## 4.5 Proving a parameterized deduction theorem in ITP

In [2, 3], Basin and Matthews showed how metatheorems that are parameterized by their scope of application can be proved using a theory of parameterized inductive definitions as a metatheory. To illustrate the notion of a *scoped metatheorem* they present a generalized version of the deduction theorem that can be applied to all extensions of the language and axioms of minimal logic as well as extensions of rules that satisfy certain conditions. From their theorem it follows, for example, that the deduction theorem holds not just for minimal logic of implication, but also for any propositional or first-order extension, but not necessarily for extensions to modal logics (which would require adding new rules, as opposed to axioms).

Since the requirements demanded of the metatheory in [2, 3] — namely, that we can build families of sets using parameterized inductive definitions, and that we can reason about their elements by induction — are indeed satisfied by rewriting logic and our theory representation calculus, we should be able to formalize scoped metatheorems as goals in the extended module `ITP` and prove them (probably using strategies) in the module `S-ITP`.

To illustrate this idea, we consider a generalized version of the deduction theorem that applies to all extensions of minimal logic with a new rule of the form

$$\frac{C \quad D}{E}$$

that satisfies a certain condition; namely, in the step case we can use the assumptions $A \longrightarrow C$ and $A \longrightarrow D$ to prove $A \longrightarrow E$. This metatheorem corresponds to the following goal for the inductive theorem prover, where $\boxed{\texttt{A}}$, $\boxed{\texttt{B}}$, $\boxed{\texttt{C}}$, $\boxed{\texttt{D}}$, and $\boxed{\texttt{E}}$ are variables of sort `Term` in the module `META-LEVEL`:

$$
\begin{array}{l}
\texttt{META-LEVEL} \vdash_{\text{ind}} \forall(\boxed{\texttt{A}},\boxed{\texttt{B}},\boxed{\texttt{C}},\boxed{\texttt{D}},\boxed{\texttt{E}})\\
\texttt{((meta-reduce(ARROW+, 'impl[}\boxed{\texttt{A}}\texttt{, }\boxed{\texttt{C}}\texttt{] : 'Theorem) = \{'true\}'Bool}\\
\quad \wedge\\
\quad \texttt{meta-reduce(ARROW+, 'impl[}\boxed{\texttt{A}}\texttt{, }\boxed{\texttt{D}}\texttt{] : 'Theorem) = \{'true\}'Bool}\\
\quad \Longrightarrow\\
\quad \texttt{meta-reduce(ARROW+, 'impl[}\boxed{\texttt{A}}\texttt{, }\boxed{\texttt{E}}\texttt{] : 'Theorem) = \{'true\}'Bool)}\\
\;\wedge\\
\;\texttt{meta-reduce(ARROW+A, }\boxed{\texttt{B}}\texttt{ : 'Theorem) = \{'true\}'Bool)}\\
\Longrightarrow\\
\texttt{meta-reduce(ARROW+, 'impl[}\boxed{\texttt{A}}\texttt{, }\boxed{\texttt{B}}\texttt{] : 'Theorem) = \{'true\}'Bool),}
\end{array}
$$

where `ARROW+` is shorthand for the term

```
(mod'ARROW is
    including 'BOOL .
    sorts('SentConstant ; 'Formula ; 'Theorem) .
    subsort 'SentConstant < 'Formula .
    subsort 'Theorem < 'Formula .
    op 'impl :  'Formula 'Formula -> 'Formula [none] .
    var 'A : 'Formula .  var 'B : 'Formula .  var 'C : 'Formula .
    mb 'impl['A, 'impl['B, 'A]] : 'Theorem .
    mb 'impl['impl['A, 'B],
             'impl['impl['A, 'impl['B, 'C]], 'impl['A, 'C]]] : 'Theorem .
    cmb'B : 'Theorem if
       '_and_['impl['A, 'B] : 'Theorem, 'A : 'Theorem] = {'true}'Bool .
    cmb E :  'Theorem if
       '_and_[C :  'Theorem, D :  'Theorem] = {'true}'Bool .
    none
    none
endm)
```

and `ARROW+A` is shorthand for the term

```
(mod'ARROW is
    including 'BOOL .
    sorts('SentConstant ; 'Formula ; 'Theorem) .
    subsort 'SentConstant < 'Formula .
```

```
    subsort 'Theorem < 'Formula .
    op 'impl :  'Formula 'Formula -> 'Formula [none] .
    var 'A : 'Formula .  var 'B : 'Formula .  var 'C : 'Formula .
    mb A : 'Theorem .
    mb 'impl['A, 'impl['B, 'A]] : 'Theorem .
    mb 'impl['impl['A, 'B],
             'impl['impl['A, 'impl['B, 'C]], 'impl['A, 'C]]] : 'Theorem .
    cmb'B : 'Theorem if
       '_and_['impl['A, 'B] : 'Theorem, 'A : 'Theorem] = {'true}'Bool .
    cmb [E] :  'Theorem if
       '_and_[[C] :  'Theorem, [D] :  'Theorem] = {'true}'Bool .
    none
    none
endm).
```

Using the rule `induction*`, along with the rest of the inference rules specified in `ITP`, we have proven this metatheorem with a strategy defined in `S-ITP` that follows the expected proof strategy, beginning with induction on the variable $\boxed{\texttt{B}}$.

# 5 Conclusion

We have presented, both abstractly and concretely, a new approach to metatheoretic reasoning based on using reflective logical frameworks whose theories have initial models. Initial experiments with these ideas are encouraging. We can formalize theories as modules in Maude and use the Maude system as a logical framework to prove theorems *in* the theories. Moreover, using reflective reasoning we can exploit the initiality of these modules by reflectively formalizing induction principles over them. This yields a formalization well-suited for reasoning *about* theories and their interrelationships.

# References


[1] D. Basin and R. Constable. Metalogical frameworks. In G. Huet and G. Plotkin, editors, *Logical Environments*, pages 1–29. Cambridge University Press, 1993.

[2] D. Basin and S. Matthews. Scoped metatheorems. In *Second International Workshop on Rewriting Logic and its Applications*, volume 15, pages 1–12. Electronic Notes in Theoretical Computer Science (ENTCS), September 1998.

[3] D. Basin and S. Matthews. Structuring metatheory on inductive definitions. *Information and Computation*, 1999. To appear.

[4] M. Clavel. *Reflection in General Logics and in Rewriting Logic with Applications to the Maude Language.* PhD thesis, University of Navarre, 1998.

[5] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, and J. Meseguer. Metalevel computation in Maude. In C. Kirchner and H. Kirchner, editors, *Second International Workshop on Rewriting Logic and its Applications*, volume 15 of *Electronic Notes in Theoretical Computer Science*, pages 3–23, Pont-à-Mousson, France, September 1998. Elsevier.

[6] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and J. Quesada. Maude: Specification and programming in rewriting logic. SRI International, January 1999, `http://maude.csl.sri.com`.

[7] M. Clavel, F. Durán, S. Eker, and J. Meseguer. Building equational proving tools by reflection in rewriting logic. In *Proceedings of the CafeOBJ Symposium '98, Numazu, Japan.* CafeOBJ Project, April 1998.

[8] M. Clavel, S. Eker, P. Lincoln, and J. Meseguer. Principles of Maude. In J. Meseguer, editor, *First International Workshop on Rewriting Logic and its Applications*, volume 4 of *Electronic Notes in Theoretical Computer Science*, pages 65–89, Asilomar (California), September 1996. Elsevier.

[9] M. Clavel and J. Meseguer. Axiomatizing reflective logics and languages. In G. Kiczales, editor, *Proceedings of Reflection'96*, pages 263–288, San Francisco (California), April 1996. Xerox PARC.

[10] M. Clavel and J. Meseguer. Reflection and strategies in rewriting logic. In J. Meseguer, editor, *First International Workshop on Rewriting Logic and its Applications*, volume 4 of *Electronic Notes in Theoretical Computer Science*, pages 125–147, Asilomar (California), September 1996. Elsevier.

[11] J. Despeyroux, F. Pfenning, and C. Schürmann. Primitive recursion for higher-order abstract syntax. In *Proceedings of the 3rd International Conference on Typed Lambda Calculi and Applications (TLCA'97)*, volume 1210 of *Lecture Notes in Computer Science*, Nancy, France, April 1997. Springer-Verlag.

[12] F. Durán. *A Reflective Module Algebra with Applications to the Maude Language*. PhD thesis, University of Málaga, 1999. `http://maude.csl.sri.com`.

[13] S. Feferman. Finitary inductively presented logics. In *Logic Colloquium '88*. North-Holland, 1988.

[14] M. Gordon and T. Melham. *Introduction to HOL: A Theorem Proving Environment for Higher Order Logic*. Cambridge University Press, 1993.

[15] R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. *J. ACM*, 40(1):143–184, January 1993.

[16] S. M. Lane. Why commutative diagrams coincide with equivalent proofs. *Contemporary Mathematics*, 13:387–401, 1982.

[17] N. Martí-Oliet and J. Meseguer. Rewriting logic as a logical and semantic framework. Technical Report SRI-CSL-93-05, SRI International, Computer Science Laboratory, August 1993. To appear in D. Gabbay, ed., *Handbook of Philosophical Logic*, Kluwer Academic Publishers.

[18] N. Martí-Oliet and J. Meseguer. General logics and logical frameworks. In D. Gabbay, editor, *What is a Logical System?*, pages 355–392. Oxford University Press, 1994.

[19] N. Martí-Oliet and J. Meseguer. Rewriting logic as a logical and semantic framework. In J. Meseguer, editor, *First International Workshop on Rewriting Logic and its Applications*, volume 4 of *Electronic Notes in Theoretical Computer Science*. Elsevier, September 1996.

[20] S. Matthews, A. Smaill, and D. Basin. Experience with $FS_0$ as a framework theory. In G. Huet and G. Plotkin, editors, *Logical Environments*, pages 61–82. Cambridge University Press, 1993.

[21] R. McDowell and D. Miller. A logic for reasoning with higher-order abstract syntax. In *Twelfth Annual IEEE Symposium on Logic in Computer Science*, June 1997.

[22] J. Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theoretical Computer Science*, 96(1):73–155, 1992.

[23] J. Meseguer. Membership algebra as a semantic framework for equational specification. In F. Parisi-Presicce, editor, *Proceedings of WADT'97*, volume 1376 of *Lecture Notes in Computer Science*, pages 18–61. Springer-Verlag, 1998.

[24] J. Meseguer. Research directions in rewriting logic. In U. Berger and H. Schwichtenberg, editors, *Computational Logic, NATO Advanced Study Institute, Marktoberdorf, Germany, July 29 - August 6, 1997*. Springer-Verlag, 1998.

[25] J. Meseguer and J. A. Goguen. Initiality, induction and computability. In M. Nivat and J. C. Reynolds, editors, *Algebraic Methods in Semantics*, pages 459–541. Cambridge University Press, 1985.

[26] C. Paulin-Mohring. Inductive Definitions in the System Coq - Rules and Properties. In M. Bezem and J.-F. Groote, editors, *Proceedings of the conference Typed Lambda Calculi and Applications*, volume 664 of *Lecture Notes in Computer Science*, 1993. LIP research report 92-49.

[27] L. C. Paulson. A fixedpoint approach to implementing (co)inductive definitions. In *Proceedings of the 12th International Conference on Automated Deduction (CADE-12)*, volume 814 of *Lecture Notes in Artificial Intelligence*, Nancy, France, June 1994. Springer-Verlag.

[28] L. C. Paulson. *Isabelle : a generic theorem prover; with contributions by Tobias Nipkow*, volume 828 of *Lecture Notes in Computer Science,*. Springer, Berlin, 1994.

[29] C. Schürmann and F. Pfenning. Automated theorem proving in a simple meta-logic for LF. In C. Kirchner and H. Kirchner, editors, *Proceedings of the 15th International Conference on Automated Deduction (CADE-15)*, volume 1421 of *Lecture Notes in Computer Science*, pages 286–300, Lindau, Germany, July 1998. Springer-Verlag.