Electronic Notes in Theoretical Computer Science 15 (1998)
URL: `http://www.elsevier.nl/locate/entcs/volume15.html` 14 pages

Scoped Metatheorems

David Basin

Institut für Informatik, Universität Freiburg
Am Flughafen 17, D-79110 Freiburg, Germany

Seán Matthews

Max-Planck-Institut für Informatik
Im Stadtwald, D-66123 Saarbrücken, Germany

Abstract

Proof development systems traditionally structure theories hierarchically: Theorems established in a subtheory hold in all supertheories. While often effective, this is sometimes too restrictive as there are certain facts that are true of some but not all extensions. We present a solution where instead of first formalizing a theory and then establishing facts, we parameterize each statement with its scope of application. We present this idea abstractly and consider concrete implementations based on parameterized inductive definitions.

1 Introduction

It is a truism of software engineering that large programming projects should be factored into modular subdevelopments. The same holds for large formal mathematical projects carried out using computer support in the form of a *proof development system*, as we see if look at what has been achieved by the user communities of, e.g., Nuprl, HOL, Coq, Isabelle, or Mizar.

However the nature of facilities for structuring developments varies from system to system, and we should examine the alternatives. The problem is one of how to manage, intellectually and practically, the complexity that arises 'in the large' (e.g. in the course of proving complex mathematical theorems or establishing the correctness of complex programs) and of how to support the development of generic reusable theorems. Refining the analogy with software engineering, we want subdevelopments, e.g. collections of theorems, definitions, and the like, with well-defined interfaces, which allow composition and as much reuse as possible.

In software engineering, structuring is often organized around modules of code. We can define a theorem-proving analogue of a module in terms of contexts (also called *theories* in the theorem proving community). Call a *context* C a pair (L, T) consisting of a language L and a collection of theorems T. Although a logician can, without problems, treat L and T extensionally, simply as sets, this view is not so useful for designing theorem provers, since the sets are usually infinite, and thus not directly implementable. However these sets can be specified using constructors: thus L is better taken as a collection of formula constructors for building formulae and T as a collection of proof constructors (axioms and rules) for building proofs of theorems; the corresponding sets are then the closures of the constructors. Note that we will go back and forth between these two views without much further comment; in §3 we will show the reader several ways to distinguish unambiguously between a set of constructors and the set it inductively defines.

When theorem proving is organized around contexts, a natural structuring facility to provide, and one found in many provers, is one that allows users to *enrich* a context. Given a set of formula constructors L_δ, we write $\mathsf{L} + \mathsf{L}_\delta$ (and similarly for $\mathsf{T} + \mathsf{T}_\delta$) to denote both the combined collection of constructors and the enlarged set of formulae built from these constructors. Then, given a context $C = (\mathsf{L}, \mathsf{T})$ and extension $\delta = (\mathsf{L}_\delta, \mathsf{T}_\delta)$, we write $C + \delta$ to denote the pointwise enrichment of L and T. Enrichment gives rise to a natural partial order on contexts ordered by inclusion, and theorems established in C hold in all extensions $C + \delta$. Further, since two contexts can be combined (provided the constructor sets are disjoint), by enriching one with another, we are able to compose contexts into larger development.

This model, however, is sometimes too simple. We may know things (call them *facts*) about a context (L, T) that do not, and even cannot, correspond to anything in T. Consider the following examples:

(i) In the context C_{PA} for arithmetic, any ground term has a normal form built only from 'zero' and the 'successor' function.

(ii) $A \to B$ is theorem of the context $C_\to$ of minimal logic (of implication) whenever B is a theorem of $C'_\to$, defined as the enrichment of $\mathsf{T}_\to$ with A.

(iii) If A is a theorem of $C_\Box$, a context defining the modal logic K, then there is a translation function $(\cdot)^* : \mathsf{L}_\Box \to \mathsf{L}_{K_\Box}$ such that A^* is a theorem of $C_{K_\Box}$, a context formalizing the Kripke semantics for K whenever A is a theorem of $C_\Box$.[1]

These facts are not just theoretically interesting; their formalizations can be used in machine supported theorem proving (for examples, see [6,8] and §2 of this paper). The problem is that these statements depend on the precise details of the context for which they are proven, and are not compatible with

[1] Such a translation function is called a *semantic embedding*, and is used to reduce the problem of reasoning about non-classical logics to that of reasoning in first-order theories (assuming of course that it is possible to characterize the semantics of the logics under consideration in first-order logic) [10].

the simple view of structuring based on context enrichment that we discuss above. Fact (i) fails for simple extensions of the context, e.g. if the language of terms in C_{PA} is extended with notation for general recursive functions. Fact (ii) relates structurally similar but distinct contexts based on $C_{\rightarrow}$; it fails if $C_{\rightarrow}$ is extended to, e.g., modal logic (since the deduction theorem fails for modal logics), but holds if it is extended to larger fragments of intuitionistic or classical propositional logic. Fact (iii) relates the pair of contexts $(C_{\Box}, C_{K_{\Box}})$ which are structurally distinct (the first formalizes a propositional modal logic, the second its first-order semantics). This fact is antimonotonic in the first argument, and monotonic in the second; i.e. it holds for any pair of contexts (C_1, C_2) where there exist δ_1, δ_2 such that $C_1 + \delta_1 = C_{\Box}$ and $C_2 = C_{K_{\Box}} + \delta_2$.

Such facts cannot be established in provers where structuring is based on context enrichment. The provers mentioned above avoid this incompatibility by the simple expedient of not providing the necessary vocabulary of contexts, which is needed if such statements are to be formalized in the first place. Instead they assume open-ended languages and deductive systems, i.e. any fact holding for a context C must hold in every extension $C + \delta$. Facts (i–iii), on the other hand, assume *fixed* (closed) languages or deductive systems, so that arguments about terms or proofs can appeal to induction, something that is not possible in a setting that assumes the possibility of enrichment.

Our approach

In this paper we explore a new approach to structuring developments that generalizes the hierarchical approach based on enrichment. The idea is simple but surprisingly powerful: Each fact formalizes its scope, i.e. the class of contexts in which it holds. Rather than associating a fact with a context C, and having it automatically hold for all enrichments δ, we formalize explicitly the conditions that a context must satisfy; i.e. rather than having a context determine the scope of a statement, we invert the relationship by having each statement determine the scope of its contexts. Said another way, facts are scoped with contexts rather than context scopes containing facts.

Different realizations of this idea are possible. However a necessary requirement is that the proof development system supports a logic that can formalize contexts *within* the logic. Such a logic is typically called a metalogic, since the contexts themselves represent (object) logics. Theorems of the metalogic are often called metatheorems; reasoning about provability in a context C requires a metatheorem stating, for example, that 'ϕ is provable in C', where ϕ is a formula in the language L. In §3 we shall consider different possible realizations. The more promising ones are based on parameterized inductive definitions; the inductive definitions formalize the base context and the parameters the allowed extensions.

Overall, we see our contributions as being both conceptual and practical. Conceptually, we explore a new idea in an abstract setting: that metatheorems scope the applicability context of theorems, i.e. specify the class of theories for which a fact is true. Practically, we sketch several different realizations of these ideas and speculate on how future proof development systems that incorporate these ideas might look.

Related work

This paper can be seen as the abstract complement to previous work [2] where we described a particular experiment using FS_0, a theory of inductive definitions, to explore the distinction between derived and admissible rules of a logic. The primary focus there was on formalizing such distinctions and the resulting consequences for machine supported metatheory.

We have motivated our approach here by making comparisons with provers that use enrichment to structure developments hierarchically. There is, of course, more than one possible way to place scoping constraints on a metatheorem. While we include the constraints explicitly as part of the statement of the metatheorem itself, Rohwedder [14] locates them at a separate 'meta-meta' level. His motivation for this decision is a wish to retain facilities of a type-theoretic logical framework that are otherwise incompatible with scoping facilities or general metatheory. An alternative attempt to find a compromise solution to the same problem is the work of McDowell and Miller [9], who propose an extension of an Edinburgh LF-like language with controlled induction facilities (by adding a type of natural numbers).

There has also been systematic work on structuring logics that is semantically based. In particular, Goguen and Burstall have proposed the notion of an *institution*, which formalizes abstract model theory and provides a semantic framework for formalizing different logics and relationships between them [4]. Meseguer has shown how institutions can be combined with *entailment systems*, which formalize syntactic and deductive notions, to obtain *general logics*. Moreover, within this framework (which can be mechanized, e.g. in rewriting logic), it is possible to formalize mappings between general logics based on maps between their entailment systems [7]. There are similarities here too with the type-theoretic work of Basin and Constable on *metalogical frameworks* in which Σ-types were used to formalize entailment systems, and mappings between entailment systems were formalized by functions within the type theory [1]. These approaches are very powerful, and more general than what we are proposing here. However, the user is obliged, with each new logic, to provide mappings and to prove metatheorems that relate it to previously given logics.

2 Some examples

Let us motivate scoped metatheorems by further developing one of the examples above: fact (ii), a result which is of fundamental importance for Hilbert systems. Hilbert systems are interesting because they correspond to simple inductive definitions and are therefore relatively simple to prove theorems *about*. However they are also notoriously hard to prove theorems *in*. The purpose of (ii), also known as the deduction theorem, is to make proof in a Hilbert system easier, by allowing proof under temporary assumption in the manner of a natural deduction proof system.

Consider the development of a hierarchy of related Hilbert systems. A possible root is minimal (implicational propositional) logic. The language, $\mathsf{L}_{\rightarrow}$, is defined by the constructor $\rightarrow : \mathsf{L}_{\rightarrow} \times \mathsf{L}_{\rightarrow} \rightarrow \mathsf{L}_{\rightarrow}$ (written infix, associating to the right) and sentential constants. $\mathsf{T}_{\rightarrow}$ is defined by all instances of the standard Hilbert axiom schemata

$$A \rightarrow B \rightarrow A \tag{K}$$

and

$$(A \rightarrow B) \rightarrow (A \rightarrow B \rightarrow C) \rightarrow A \rightarrow C, \tag{S}$$

and the rule *modus ponens*:

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

(i.e. given theorems A and $A \rightarrow B$, we get the new theorem B). We write $\vdash_{\rightarrow} A$ to indicate that $A \in \mathsf{T}_{\rightarrow}$.

Given a metatheory containing at least the 'implies' connective $\Rightarrow$, two *meta*theorems we should be able to prove are:

$$\vdash_{\rightarrow} A \rightarrow A \tag{I}$$

$$(\vdash_{\rightarrow} A \rightarrow C) \Rightarrow (\vdash_{\rightarrow} A \rightarrow B \rightarrow C) \tag{$thin$}$$

The first asserts the provability of a theorem (that implication is an identity) in the object logic and the second represents a derived proof rule. Given universal quantification over $\mathsf{L}_{\rightarrow}$ in the metatheory we can make both of the above formulae schematic by taking their universal closures; in future we will assume this reading, leaving the quantifiers implicit.

Now consider the deduction theorem. For this we need to extend our metalanguage: We define $A \vdash_{\rightarrow} B$ as metanotation meaning that if $\mathsf{T}_{\delta} = \{A\}$ (i.e. the enrichment with the new axiom A) then $B \in \mathsf{T}_{\rightarrow} + \mathsf{T}_{\delta}$. The deduction theorem is then

$$A \vdash_{\rightarrow} B \Rightarrow \vdash_{\rightarrow} A \rightarrow B\,,$$

which is proven by induction on the structure of $\mathsf{T}_{\rightarrow} + \mathsf{T}_{\delta}$ derivations. Notice that just to state this metatheorem, which is schematic in A (and B), we refer to infinitely many axiomatic extensions of the original Hilbert system.

An obvious candidate for a supertheory of minimal logic is full propositional logic (we consider here only conjunction, other operators are similar):

We extend $\mathsf{L}_{\rightarrow}$ to L_P by adding the new binary connective $\wedge$, and $\mathsf{T}_{\rightarrow}$ to T_P by adding axiom schemata, which for $\wedge$ are:

$$A \rightarrow B \rightarrow A \wedge B$$
$$A \wedge B \rightarrow A$$
$$A \wedge B \rightarrow B$$

What is the status of the theorems and metatheorems of minimal logic in the context of propositional logic? We expect that all theorems, e.g. (I), are still theorems of propositional logic provided that the original axioms (S) and (K) and inference rule (*modus ponens*) are interpreted over the new language. Of course the theorems *as proven* cannot be used for this purpose, but we could (and practicing foresight, would) prove instead, e.g,

$$\forall \mathsf{T}_\delta.\ \vdash_{\mathsf{T}_{\rightarrow}+\mathsf{T}_\delta} A \rightarrow A;$$

i.e. not just that (I) is a theorem in context $C_{\rightarrow}$ but also in an arbitrary enrichment of the deductive system.

The metatheorem (*thin*) generalizes in exactly the same way. But what about the deduction theorem? In fact it does hold for propositional logic, but since the proof suggested above is by induction on the structure of proofs in minimal logic, it is not valid in this new context; we must check the new cases corresponding to the new axioms. Before considering what a generalization of the deduction theorem would look like though, first consider an extension where it fails: a fragment of the normal propositional modal logic K. Extend the language $\mathsf{L}_{\rightarrow}$ to $\mathsf{L}_{\Box}$ by adding the unary modality $\Box$ and extend $\mathsf{T}_{\rightarrow}$ to $\mathsf{T}_{\Box}$ by adding an axiom stating the distributivity of $\Box$ over $\rightarrow$,

$$\Box(A \rightarrow B) \rightarrow (\Box A \rightarrow \Box B)\,,$$

and a rule of *necessitation*

$$\frac{A}{\Box A}.$$

Again we can ask, what is the status of the theorems and metatheorems of minimal logic in $\mathsf{T}_{\Box}$? (I) holds in any extension of $\mathsf{T}_{\rightarrow}$. Furthermore, it holds when $\mathsf{T}_{\rightarrow}$ is extended with new rules and even when the language of $\mathsf{L}_{\rightarrow}$ is extended to $\mathsf{L}_{\Box}$, provided the old axioms and inference rule *modus ponens* are interpreted over the new language; the same follows for (*thin*). The deduction theorem also holds under extensions of $\mathsf{T}_{\rightarrow}$ with new axioms. But it does not hold under extensions with arbitrary new rules; in particular, it fails for $\mathsf{T}_{\Box}$, which includes the additional rule for necessitation.

Now we consider how to generalize the deduction theorem. First we define a predicate $P(\mathsf{T}_\delta)$ to mean 'T_δ is a purely axiomatic extension', then we write

$$\forall \mathsf{T}_\delta.\, P(\mathsf{T}_\delta) \Rightarrow\ \vdash_{(\mathsf{T}_{\rightarrow}+\mathsf{T}_\delta)+\{A\}} B \Rightarrow\ \vdash_{\mathsf{T}_{\rightarrow}+\mathsf{T}_\delta} A \rightarrow B\,, \tag{1}$$

which relates the provability of $A \to B$ in the parameterized theory $\mathsf{T}_{\to} + \mathsf{T}_{\delta}$ to the provability of B in the extension of $\mathsf{T}_{\to} + \mathsf{T}_{\delta}$ with the axiom A. In general, the way provisos on, and scope of, metatheorems can be formalized depends on the way that constraints on parameter sets can be imposed.

In the same way, it is possible to state fact (iii). We can capture the monotonicity conditions simply as follows. Given a suitable function $(\cdot)^* : \mathsf{L}_{\Box} \to \mathsf{L}_{K_{\Box}}$, then, quantifying over the theory components of the two contexts, we write

$$\forall \mathsf{T}_1, \mathsf{T}_2. (\exists \mathsf{T}_{\delta_1}, \mathsf{T}_{\delta_2}. \mathsf{T}_1 + \mathsf{T}_{\delta_1} = \mathsf{T}_{\Box} \wedge \mathsf{T}_2 = \mathsf{T}_{K_{\Box}} + \mathsf{T}_{\delta_2}) \Rightarrow \models_{\mathsf{T}_1} A \Rightarrow \models_{\mathsf{T}_2} A^* . \qquad (2)$$

3 Implementation

Our discussion so far has left open the precise technical definition of a context C, a context extension δ, and how the two are combined with $+$. This has allowed us to avoid commitment to any particular formal metatheory. However, implicit in our discussion of 'languages' and 'collections of theorems' is the idea that these are, essentially, generalized inductive definitions. In this section we concretize this and consider how different formalisms can provide a suitable foundation for implementation in a way that allows explicit statements about both contexts and extensions.

3.1 Arithmetic

The traditional vehicle for general metatheory, i.e. proof theory, has been finitary, or primitive recursive, arithmetic. The reasons for this choice are, however, historical and philosophical, not practical. As a practical matter, arithmetic is hopelessly unsuitable; however it is a good starting point for our discussion since it makes explicit the facilities that are needed in practice.

The basic procedure for encoding an object logic using predicates over the natural numbers is familiar to anyone who knows Gödel's incompleteness theorems. Gödel's object theory was arithmetic, but other deductive systems can be treated similarly. First we define an encoding of the syntax of the formulae and proofs of the object logic. We might represent the former by the notation $(\cdot)^* : \mathsf{L} \to \mathbb{N}$ and then define a relation 'proves$_C$' between proofs and formulae, that formalizes provability in the relevant context C. Then with a collection of suitable lemmata of the form

$$\exists x. (x \ \text{proves}_C \ A^*) \Rightarrow \exists x. (x \ \text{proves}_C \ (A \vee B)^*)$$

it becomes a straightforward task to prove theorems in the encoded theory, or metatheorems about the encoded theory. This straightforwardness is, though, as we have said, purely theoretical. The enormous numbers involved, and the computation necessary to recover even simple structural information about encoded formulae, make it impractical.

A further problem, for the purposes of the current discussion, is that the provability predicates that result from this kind of encoding do not naturally support theory structuring, either the simple hierarchical kind based on enrichment, or the more sophisticated scoping mechanisms that we have considered. Fortunately, since Gödel, logicians and computer scientists have developed other notations which, although technically similar to the traditional theories of arithmetic, are more tractable in practice. Below we consider several examples of this.

3.1.1 Theories related to arithmetic

One of the most successful proof development systems in the history of theorem proving is the Boyer-Moore Theorem Prover. The language of the Boyer-Moore system is rather different from the language of arithmetic which Gödel used: It is a purely functional notation inspired by Lisp. Whereas in arithmetic the fundamental (indeed, only) data-structure is that of the numbers, in Lisp it is the *S-expression.* S-expressions are defined as the set of terms generated from the constant *nil* and the pairing operation *cons*, usually written simply as a pair $(\cdot, \cdot)$, where the comma associates to the right; i.e. (a, b, c) is short for $(a, (b, c))$. Also provided are inverse operations for *cons*, known, for reasons of historical accident, as *car* and *cdr*, which return the left and right subexpressions of a *cons* pair; i.e. $car(x, y) = x$ and $cdr(x, y) = y$.

A representation formalism based on S-expressions has significant practical advantages over one based on numbers: terms and proofs are naturally represented as S-expressions, and it is easier to manipulate large S-expressions (i.e. access their subterms) than large numbers (i.e. compute their prime factors). The result is a theory that, with some support for automating theorem proving, is effective enough to be used to prove complex metatheorems of the sort we are interested in here, including, among other things, Gödel's incompleteness theorem and the Church-Rosser theorem [15,16]. Unfortunately however, the Boyer-Moore logic still shares the problem with arithmetic of lacking facilities that allow a user to scope theorems with contexts in the manner discussed in §2, and illustrated by the examples (1) and (2).

A recent development that provides a solution to this problem is the theory FS_0, proposed originally by Feferman [3] as a vehicle for formal proof theory. FS_0 also develops the idea of a Lisp-like language but takes it in a slightly different direction. FS_0 presents a theory of S-expressions in first-order logic, rather than in a purely functional notation, and augments it with an explicit notation for inductively defined classes and induction over these classes. This extension is precisely what we need not only to represent the language and proofs of encoded object logics, but also to naturally represent extensions: Languages and proofs correspond to classes, and extensions corresponds to the addition of new constructors for generating these classes.

An inductively defined class is formalized in FS_0 by the term $I(A, B)$. A and B are both classes of S-expressions, possibly inductively defined themselves, and the result is a class of S-expressions. (In order to get off the ground, classes in FS_0 may also be formed by limited kinds of comprehension, and can be combined by intersection and union.) A can be thought of as the base-case of the inductive definition. In representing a language, A might be a set representing propositional constants or atoms. In representing a deductive system, A would represent the set of axioms or instances of the axiom schemas. B can be thought of as the step-case of the inductive definition; it constitutes a class of S-expressions where tuples of the form (a, b, c) are interpreted as 'if b and c then a'. In other words, B formalizes the rule that whenever b and c belong to the class $I(A, B)$ then so does a.

An advantage of FS_0 is that inductively defined classes are defined purely out of other classes and that $I(A, B)$, A and B are all *terms* in the theory, and thus can all be quantified over in statements of theorems. We can quantify over the formulae or proofs of a language by quantifying over the classes that represent them. For example, if we use classes to formalize contexts, enrichment is trivial: given a class $I(A, B)$ we can add new rules R simply by writing, instead, $I(A, B \cup R)$. This gives us a formal interpretation of expressions like $\mathsf{T} + \mathsf{T}_\delta$ given in §1. Moreover, since R is a class, we can replace it with a variable, quantify over it, and even constrain it to satisfy predicates. This gives a formal meaning to the quantification over contexts seen in §2. To see in more detail how this works, we consider a simple example below.

3.1.2 An example encoding

To see how FS_0 supports scoped metatheorems, we sketch an encoding of minimal implicational logic, which we define as the context $(\mathsf{L}_\rightarrow, \mathsf{T}_\rightarrow)$.

The first thing we need is an encoding of the language. Assuming a predefined class of atomic propositions $\mathbf{A}$ (itself encoded in terms of S-expressions), then we propose an encoding $(\cdot)^*$ such that

$$\begin{aligned} a^* &\rightsquigarrow (atom, a), \qquad \text{if } a \in \mathbf{A} \\ (a \rightarrow b)^* &\rightsquigarrow (imp, a^*, b^*) \end{aligned}$$

where *atom* and *imp* are distinct constant S-expressions. We formalize this in FS_0 by first defining our constants

$$\begin{aligned} atom &\triangleq nil \\ imp &\triangleq (nil, nil) \end{aligned}$$

then using comprehension we can solve for At and Ru, as specified by the logical equations:

$$\begin{aligned} x \in At &\Leftrightarrow \operatorname{car} x = atom \wedge \operatorname{cdr} x \in \mathbf{A} \\ x \in Ru &\Leftrightarrow \exists a, b.\, x = ((imp, a, b), a, b) \end{aligned}$$

And, given this, we can define the language of implication as

$$\mathsf{L}_{\to} \triangleq I(At, Ru).$$

An extension of $\mathsf{L}_{\to}$ with more formula constructors L_{δ} is written simply as

$$\mathsf{L}_{\to} + \mathsf{L}_{\delta} \triangleq I(At, Ru \cup \mathsf{L}_{\delta})\,.$$

The set of theorems $\mathsf{T}_{\to}$ is defined in the same way. For example the class of instances of the K axiom is the solution for K_ax of

$$x \in K\_ax \Leftrightarrow \exists a, b.\, x = (imp, a, (imp, b, a))$$

(S_ax follows similarly), while modus ponens is

$$x \in MP \Leftrightarrow \exists a, b.\, x = (b, (imp, a, b), a)\,.$$

Finally $\mathsf{T}_{\to}$ is the closure of the axiom instances under modus ponens, i.e.

$$\mathsf{T}_{\to} \triangleq I((K\_ax \cup S\_ax) \cap \mathsf{L}_{\to}, MP)\,.$$

At this point fact (ii) from the introduction can be stated as

$$\begin{aligned}\forall a, b.\, b \in I((K\_ax \cup S\_ax \cup \{a\}) \cap \mathsf{L}_{\to}, MP)& \\ \Rightarrow (imp, a, b) \in I((K\_ax \cup S\_ax) \cap \mathsf{L}_{\to}, MP)&\,,\end{aligned}$$

which can be easily proved by induction.

3.2 Higher-order logic and set theory

A logic or theory that is strong enough to formalize and prove some variant of the Knaster-Tarski fixedpoint theorem, e.g. higher-logic or set-theory, is an alternative possible starting point for a theory of inductive definitions. In particular, theorem provers like the HOL system [5] and Isabelle (which implements both HOL [11] and ZF [12]) provide such logics. These logics form the basis of packages that provide users with syntactic support for formalizing inductive definitions from which the package automatically constructs and derives the corresponding induction principles.

In terms of the Knaster-Tarski theorem, an inductive definition is the least fixedpoint of a function from sets to sets that satisfies certain conditions like monotonicity. The least fixedpoint is an operator (a term formalized in the logic) that takes the intersection of all pre-fixedpoints, i.e.

$$lfp(f) \triangleq \bigcap\{u \mid f(u) \subseteq u\}\,.$$

Inductive definitions can be parameterized by simply abstracting the set $lfp(f)$ over (finitely many) other sets.

Although higher-order logic, as a metalogic, is quite different from FS_0, we can formalize contexts in it using inductive definitions in a similar way. Consider the previous example where again $\mathbf{A}$ is a set of suitably encoded atomic propositions. Now, however, let imp be a relation such that $(a, b, c) \in imp$ iff c encodes the proposition $a \to b$. Then the set of propositions may be defined as

$$\mathsf{L}_{\to} \triangleq \mathit{lfp}(\lambda X.\, \mathbf{A} \cup (ap_2\ imp\ X\ X))\,,$$

where

$$ap_2\ R\ S_1\ S_2 \triangleq \{z \mid \exists x : S_1.\, \exists y : S_2.\, (x, y, z) \in R\}\,.$$

And, again like before, we can also parameterize over extensions with new (here, binary) term constructors; we get

$$\mathsf{L}^{\mathrm{ex}}_{\to} \triangleq \lambda \mathsf{L}_{\delta}.\, \mathit{lfp}(\lambda X.\, \mathbf{A} \cup (ap_2\ (imp \cup \mathsf{L}_{\delta})\ X\ X))$$

so that a rule extension $\mathsf{L}_{\to} + \mathsf{L}_{\delta}$ is $\mathsf{L}^{\mathrm{ex}}_{\to}\, \mathsf{L}_{\delta}$. The definition of $\mathsf{T}_{\to}$ is similar.

4 Conclusion and Prospects

We have argued that for certain kinds of theory structuring we need to be able to say in which classes of theories a theorem holds. In order to provide this control it is necessary to cross the line between formal theories and formal metatheories. Although much of our presentation is abstract, we have sketched two particular kinds of logical foundations that provide good support for scoped metatheorems. Both involve a metalogic: FS_0 or higher-order logic/set theory.

We have performed machine checked experiments with both of these foundations using the Isabelle system. Isabelle is an excellent tool for carrying out such experiments since it is a logical framework, i.e. it provides a metalogic for encoding and using other object logics. Thus each of our experiments was carried out in a system with two separate metalogics: the metalogic of Isabelle, and the metalogic of the encoded theory of inductive definitions. Of course, this layering is not necessary, since it is possible to implement a theory of inductive definitions directly. Still, the reader may wonder if the use of a formal metatheory requires too much effort. With the enrichment approach it is possible to declare a context and then work in it, without having to repeat the entire context definition with each theorem proved. Of course, the enrichment approach is limited in its support for formal metatheory, but it might seem preferable for the purpose of 'merely' proving theorems in a theory.

Here we suggest that scoped metatheorems can be as comfortable to use as conventionally structured theories, provided proof development systems for them provide appropriate architectural support. First, the use of definitions can make a large difference. Just as in §2, where we use definitions like $\vdash_{\rightarrow}$ to refer to defined theories, a system can support the definition of parameterized inductive definitions where each definition is made once and the definiendum is used in subsequent theorem statements. This is not so different from theory development in logical frameworks like Isabelle, where each theorem statement is given to the system with the name of a theory;[2] the main difference is that, in the scoped metatheorem approach, the names of the theories and languages have a formal logical status. Second, if the user intends to use a particular language and deductive system for proving a collection of theorems, it is possible to support this by declaring the language and deductive system, and automatically scoping each (meta)theorem proved for them. This could be similar to the kind of support provided by theorem provers like LEGO [13] where a user begins (in the metalogic of the Edinburgh LF) by declaring a language and deductive system, and then proves a sequence of theorems in the scope of this context; afterwards, the system closes the scope by discharging the context, universally quantifying over all the term and proof constructors.

The above, of course, is speculative. However it reflects our belief that with appropriate architectural support, scoped metatheorems are not just a curiosity, but a realistic proposition for structured development, even (and especially) in the large.

References

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

[2] D. Basin and S. Matthews. Structuring metatheory on inductive definitions. *Inform. and Comput.*, (to appear).

[3] S. Feferman. Finitary inductive systems. In *Logic Colloquium '88*. North-Holland, Amsterdam, 1990.

[4] J. A. Goguen and R. M. Burstall. Institutions: Abstract model theory for specifications and programming. *J. Assoc. Comput. Mach.*, pages 95–146, 1992.

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

[6] D. J. Howe. Computational metatheory in Nuprl. In E. Lusk and R. Overbeek, editors, *Proc. CADE–9*, pages 238–257. Springer, Berlin, 1988.

[7] N. Marti-Oliet and J. Meseguer. General logics and logical frameworks. In D. Gabbay, editor, *What is a Logical System?*, pages 355–391. Clarendon Press, Oxford, 1994.

[2] This information is used to determine, for example, how to parse the theorem statement and to control which lemmata may be used in its proof.

[8] S. Matthews. *Metatheoretic and Reflexive Reasoning in Mechanical Theorem Proving*. PhD thesis, Edinburgh, 1992.

[9] R. McDowell and D. Miller. A logic for reasoning with higher-order abstract syntax. In *Proc. 12th IEEE Ann. Symp. Logic in Comp. Sci.*, pages 434–446. IEEE Computer Society Press, 1997.

[10] H.-J. Ohlbach. Translation methods for non-classical logics: an overview. *Bulletin of the IGPL*, 1:69–89, 1993.

[11] L. C. Paulson. A fixedpoint approach to implementing (co)inductive definitions. In A. Bundy, editor, *Proc. CADE–12*, pages 148–161. Springer, Berlin, 1994.

[12] L. C. Paulson. Set theory for verification: II. Induction and recursion. *J. Auto. Reas.*, 15:167–215, 1995.

[13] R. Pollack. *The Theory of LEGO: A Proof Checker for the Extended Calculus of Constructions*. PhD thesis, University of Edinburgh, 1994.

[14] E. Rohwedder. *Verifying the Meta-Theory of Deductive Systems*. PhD thesis, Carnegie Mellon University, 1996.

[15] N. Shankar. A mechanical proof of the Church-Rosser theorem. *J. Assoc. Comput. Mach.*, 35:475–522, 1988.

[16] N. Shankar. *Metamathematics, machines, and Gödel's proof*, volume 38 of *Cambridge tracts in theoretical computer science*. Cambridge University Press, New York, NY, 1994.