CONTENTS

PREFACE

DAVID BASIN, SEAN MATTHEWS

LOGICAL FRAMEWORKS

1 INTRODUCTION

One way to define a logic is to specify a language and a deductive system. For example, the language of first-order logic consists of the syntactic categories of terms and formulae, and its deductive system establishes which formulae are theorems. Typically we have a specific language in mind for a logic, but some flexibility about the kind of deductive system we use; we are able to select from, e.g., a Hilbert calculus, a sequent calculus, or a natural deduction calculus. A *logical framework* is an abstract characterization of one of these kinds of deductive system that we can use to formalize particular examples. Thus a logical framework for natural deduction should allow us to formalize natural deduction for a wide range of logics from, e.g., propositional logic to intuitionistic type-theories or classical higher-order logic.

Exactly how a logical framework abstractly characterizes a kind of deductive system is difficult to pin-down formally. From a high enough level of abstraction, we can see a deductive system as defining sets; i.e. we have a recursive set corresponding to well-formed syntax, a recursive set corresponding to proofs, and a recursively enumerable set of provable formulae.[1] But this view is really too coarse: we expect a logical framework to be able to capture more than just the sets of well-formed and provable formulae associated with a logic. If this were all that we wanted, then any Turing complete programming language would constitute a logical framework, in so far as it can implement a proof-checker for any logic.

In this chapter we present and examine two different kinds of frameworks, each representing a different view of what a deductive system is: $\rightarrow$-frameworks (deduction interpreted as reasoning in a weak logic of implication) and ID-frameworks (deduction interpreted as showing that a formula is a member of an inductively defined set). Either of these can be used to formalize any recursively enumerable relation. However, before calling a system a logical framework we will demand that it preserves additional structure. Thus we first consider what are the important and distinguishing characteristics of the different kinds of deductive systems, then we examine frameworks based on different sorts of possible *metalogic* (or *metatheory*) and we show that these are well-suited to representing the deductive systems

[1] We make the assumption in this chapter that the property of being a well-formed syntactic entity or a proof is recursive; i.e. we know one when we see it.

for certain classes of *object logics* (or *object theories*). By providing procedures by which the deductive system of any object logic in a class can be naturally encoded in some metalogic, we show how effective frameworks are for formalizing particular logics.

When we say that an encoding is *natural* we shall mean not only that it is high-level and declarative but also that there is an appropriate bijection between derivations in the object logic and derivations manipulating the encoding in the metalogic. For example, a Hilbert system can be naturally encoded using an inductive definition where each rule in the object logic corresponds to a rule in the metalogic. Similarly, there are natural encodings of many natural deduction systems in a fragment of the language of higher-order logic, which make use of a simple, uniform, method for writing down rules as formulae of the metalogic, so that it is possible to translate between natural deduction proofs of the object logic and derivations in the metalogic.

The term 'logical framework' came into use in the 1980s; however the study of metalogics and methods of representing one logic in another has a longer history in the work of logicians interested in languages for the metatheoretic analysis of logics, and computer scientists seeking conceptual and practical foundations for implementing logic-based systems. Although these motivations differ and are, at least in part, application oriented, behind them we find something common and more general: logical frameworks clarify our ideas of what we mean by a 'deductive system' by reducing it to its abstract principles. Thus work on logical frameworks contributes to the work of Dummett, Gentzen, Hacking, Prawitz, and others on the larger question of what is a logic.

1.1 Some historical background

Historically, the different kinds of deductive systems have resulted from different views of logics and their applications. Thus the idea of a deductive system as an inductive definition developed out of the work of Frege, Russell, Hilbert, Post, Gödel and Gentzen attempting to place mathematics on firm foundational ground. In particular, Hilbert's program (which Gödel famously showed to be impossible) was to use the theory of proofs to establish the consistency of all of classical mathematics, using only finitary methods. From this perspective, of *metatheory as proof theory*, a deductive system defines a set of objects in terms of an initial set and a set of rules that generate new objects from old, and a proof is the tree of rule applications used to show that an object is in the set. For Frege and Hilbert these objects were simply theorems, but later Gentzen took them to be sequents, i.e. pairs of collections of formulae. But, either way, a deductive system defines a recursively enumerable set, which is suitable for analysis using inductive arguments.

How should the metatheory used to define these inductive definitions be characterized? Hilbert required it to be finitary, so it has traditionally been taken to be the primitive recursive fragment of arithmetic (which is essentially, e.g., what Gödel [1931] used for his incompleteness theorems). However, despite the endorsement by Hilbert and Gödel, arithmetic is remarkably unsuitable, in the primitives it offers, as a general theory of inductive definitions. Thus, more recent investigations, such as those of Smullyan [1961] and Feferman [1990], have proposed theories of inductive definitions based on sets of strings or S-expressions, structures more tractable in actual use.

A different view of deductive systems is found in work in computer science and artificial intelligence, where logics have been formalized for concrete applications like machine checked proof. The work of, e.g., de Bruijn [Nederpelt *et al.*, 1994] does not attempt to analyze the meta-theoretic properties of deductive systems, but concentrates rather on common operations, such as binding and substitution. The goal is to provide an abstract characterization of such operations so that deductive systems can be easily implemented and used to prove theorems; i.e. metatheory is seen as providing a *unifying language* for implementing, and ultimately using, deductive systems. A result of this concern with ease in use (i.e. building proofs) rather than ease of metatheoretic analysis (i.e. reasoning about proofs) is that work has emphasized technically more complex, but more usable, notations such as natural deduction, and resulted in frameworks based on higher-order logics and intuitionistic type-theories instead of the inductive definitions of the older proof theory tradition.

1.2 Focus and organization

This chapter presents the concepts underlying the various logical frameworks that have been proposed, examining the relationship between different kinds of deductive systems and metatheory. Many issues thus fall outside our scope. For example, we do not investigate semantically based approaches to formalizing and reasoning about logics, such as the *institutions* of Goguen and Burstall [1992] or the *general logics* of Meseguer [1989]. Even within our focus, we do not attempt a comprehensive survey of all the formalisms that have been proposed. Neither do we directly consider implementation questions, even though computer implementations now play an important role in the field. Further references are given in the bibliography; the reader is referred in particular to the work of Avron *et al.*, Paulson, Pfenning, Pollack, and ourselves, where descriptions of first-hand experience with logical frameworks can be found.

The remainder of this chapter is organized as follows. In §2 we briefly survey three kinds of deductive systems, highlighting details relevant for formalizing abstractions of them. In §3 we consider a logic based on minimal implication as an abstraction of natural deduction. It turns out that this abstraction is closely related to a generalization of natural deduction due to Schroeder-Heister. In §4 we consider in detail a particular metatheory that formalizes this abstraction. We also consider quantification and so-called 'higher-order syntax'. In §5 we present a case study: the problem of formalizing modal logics. In §6 we examine sequent calculi and their more abstract formalization as consequence relations. In §7 we investigate the relationship between sequent systems and inductive definitions, and present a particular logic for inductive definitions. Finally, we draw conclusions and point to some current and possible future research directions.

2 KINDS OF DEDUCTIVE SYSTEMS

Many different kinds of formalization of deduction have been proposed, for a range of purposes. However in this chapter we are interested in deductive systems designed for the traditional purposes of logicians and philosophers, of investigating the foundations of language and reasoning; in fact we restrict our interest even further, to three kinds of system, which are commonly called *Hilbert calculi*, *sequent calculi* and *natural deduction*. But this restriction is more apparent than real since, between them, these three cover the vast majority of deductive systems that have been proposed. We describe only the details that are important here, i.e. the basic mechanics; for deeper and more general discussion, the reader is referred to Sundholm's articles on systems of deduction elsewhere in this handbook [1983; 1986].

For the purposes of comparison, we shall present, as an example, the same simple logics in each style: propositional logics of minimal and classical implication. The language we will work with then is as follows.

DEFINITION 2.1 *Given a set of atomic propositions P, the language of propositions, L, is defined as the smallest set containing P, where if A and B are in L, then $(A \supset B)$ is in L.*

For the sake of readability we assume that $\supset$ associates to the right and omit unnecessary parentheses; for example, we abbreviate $(A \supset (B \supset C))$ as $A \supset B \supset C$. We now proceed to the different presentations.

2.1 Hilbert calculi

Historically, Hilbert calculi are the oldest kind of presentation we consider. They are also, in some technical sense, the simplest. A Hilbert calculus defines a set of *theorems* in terms of a set of *axioms* Ax and a set of *rules of proof* R. A *rule of proof* is a relation between formulae $A_1, \ldots, A_n$ and a formula A. A rule is usually written in a two-dimensional format and sometimes decorated with its name. For example

$$\frac{A_1 \ldots A_n}{A}\, name$$

says that by rule $name$, given $A_1, \ldots, A_n$, it follows that A. The set of formulae defined by a Hilbert calculus is the smallest set containing Ax and closed under R; we call this set a *theory*, and the formulae in it *theorems*.

The set of theorems in a logic of minimal implication can be defined as a Hilbert calculus where the set of axioms contains all instances of the axiom schemata K

$$A \supset B \supset A$$

and S

$$(A \supset B) \supset (A \supset B \supset C) \supset A \supset C\,.$$

We call these schemata because A, B and C stand for arbitrary formulae in L. In addition, we have one rule of proof, *detachment*, which takes the form

$$\frac{A \supset B \quad A}{B}\,Det.$$

The Hilbert calculus defined by K, S and Det is the set of theorems of the minimal (or intuitionistic) logic of implication. We call this theory $\mathrm{HJ}^{\supset}$.[2]

The set of theorems in a logic of classical implication, $\mathrm{HK}^{\supset}$, is slightly larger than $\mathrm{HJ}^{\supset}$. We can formalize $\mathrm{HK}^{\supset}$ by adding to $\mathrm{HJ}^{\supset}$ a third axiom schema, for *Peirce's law*

$$((A \supset B) \supset A) \supset A\,. \tag{1}$$

A proof in a Hilbert calculus consists of a demonstration that a formula is in the (inductively defined) set of theorems. We can think of a proof as either a tree, where the leaves are axioms, and the branches represent rule applications, or as a list of formulae, ending in the theorem, where each entry is either an axiom, or follows from previous entries by a rule. Following common practice, we use the list notation here.

The following is an example of a proof of $A \supset A$ in $\mathrm{HJ}^{\supset}$ and thus also in $\mathrm{HK}^{\supset}$:

1.	$A \supset A \supset A$	K
2.	$(A \supset A \supset A) \supset (A \supset (A \supset A) \supset A) \supset A \supset A$	S
3.	$A \supset (A \supset A) \supset A$	K
4.	$(A \supset (A \supset A) \supset A) \supset A \supset A$	Det 2,1
5.	$A \supset A$	Det 4,3

[2] For the proof systems presented in this section, we follow the tradition where the first letter indicates the kind of deduction system (H for Hilbert, N for natural deduction, and L for sequent calculus), the second letter indicates the logic (J for minimal, or intuitionistic logic, and K for classical logic), and superscripts name the connectives.

It turns out that proving theorems in a Hilbert calculus by hand, or even on a machine, is not practical: proofs can quickly grow to be enormous in size, and it is often necessary (e.g. in the proof we have just presented) to invent instances of axiom schemata that have no intuitive relationship to the formula being proven. However Hilbert calculi were never really intended to be used to build proofs, but rather as a tool for the metatheoretic analysis of logical concepts such as deduction. And from this point of view, the most important fact about Hilbert calculi is that they are essentially inductive definitions;[3] i.e. well-suited for arguments (about provability) by induction.

2.2 Sequent calculi

Useful though Hilbert calculi are, Gentzen [1934] found them unsatisfactory for his particular purposes, and so developed a very different style that has since become the standard notation for much of proof theory, and which is known as *sequent calculus*.

With a sequent calculus we do not define directly the set of theorems; instead we define a binary 'sequent' relation between collections of formulae, Γ and Δ, and identify a subset of instances of this relation with theorems. We shall write this relation as $\Gamma \vdash \Delta$, where Γ is called the *antecedent* and Δ the *succedent*. This is often read as 'if all the formulae in Γ are true, then at least one of the formulae in Δ is true'.

The rules of a sequent calculus are traditionally divided into two subsets consisting of the *logical* rules, which define logical connectives, and the *structural* rules, which define the abstract properties of the sequent relation itself. The basic properties of any sequent system are given by the following rules which state that $\vdash$ is reflexive for singleton collections ($Basic$) and satisfies a form of transitivity (Cut):

$$\frac{}{A \vdash A}\,Basic \qquad \frac{\Gamma \vdash A, \Delta \quad \Gamma', A \vdash \Delta'}{\Gamma, \Gamma' \vdash \Delta, \Delta'}\,Cut \tag{2}$$

A typical set of structural rules is then

$$\begin{array}{cc} \dfrac{\Gamma \vdash \Delta}{\Gamma, A \vdash \Delta}\,WL & \dfrac{\Gamma \vdash \Delta}{\Gamma \vdash A, \Delta}\,WR \\ \dfrac{\Gamma, A, A \vdash \Delta}{\Gamma, A \vdash \Delta}\,CL & \dfrac{\Gamma \vdash A, A, \Delta}{\Gamma \vdash A, \Delta}\,CR \\ \dfrac{\Gamma, A, B, \Gamma' \vdash \Delta}{\Gamma, B, A, \Gamma' \vdash \Delta}\,PL & \dfrac{\Gamma \vdash \Delta, A, B, \Delta'}{\Gamma \vdash \Delta, B, A, \Delta'}\,PR \end{array} \tag{3}$$

[3] The two are so closely related that, e.g., Aczel [1977] identifies them.

which define $\vdash$ to be a relation on finite sets that is also monotonic in both arguments (what we will later call *ordinary*). The names WL, CL and PL stand for *Weakening*, *Contraction* and *Permutation Left* of the sequent, while WR, CR and PR name the same operations on the right.

We can give a deductive system for classical implication, which we call $\mathrm{LK}^{\supset}$, in terms of this sequent calculus by adding logical rules for $\supset$:

$$\frac{\Gamma \vdash A, \Delta \quad \Gamma', B \vdash \Delta'}{\Gamma, \Gamma', A \supset B \vdash \Delta, \Delta'} \supset\text{-}L \qquad \frac{\Gamma, A \vdash B, \Delta}{\Gamma \vdash A \supset B, \Delta} \supset\text{-}R$$

The names $\supset$-L and $\supset$-R indicate that these are rules for introducing the implication connective on the left and the right of $\vdash$. We get a sequent calculus for minimal implication, $\mathrm{LJ}^{\supset}$, by adding the restriction on $\vdash$ that its succedent is a sequence containing exactly one formula.[4]

Like Hilbert proofs, sequent calculus proofs can be equivalently documented as lists or trees. Since proofs are less unwieldy than for Hilbert calculi, trees are a practical notation. A sequent calculus proof of $A \supset A$ is trivial, so instead we take as an example a proof of Peirce's law (1):

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{A \vdash A}\,Basic}{A \vdash B, A}\,WR}{\vdash (A \supset B), A}\,\supset\text{-}R \qquad \dfrac{}{A \vdash A}\,Basic}{((A \supset B) \supset A) \vdash A, A}\,\supset\text{-}L}{((A \supset B) \supset A) \vdash A}\,CR}{\vdash ((A \supset B) \supset A) \supset A}\,\supset\text{-}R$$

Notice that it is critical in this proof that the succedent can consist of more than one formula, and that CR can be applied. Neither this proof (since CR is not available), nor any other, of Peirce's law is possible in $\mathrm{LJ}^{\supset}$.

Technically we can regard a sequent calculus as a Hilbert calculus for a binary connective $\vdash$; however the theorems of this system are in the language of sequents over L, not formulae in the language of L itself. The set of theorems a sequent calculus defines is taken to be the set of formulae A in L such that $\vdash A$ is provable, i.e., there is a proof of the sequent $\Gamma \vdash \Delta$, where Γ is empty, and Δ is the singleton A.

2.3 Natural deduction

A second important kind of deductive system that Gentzen [1934] (and subsequently Prawitz [1965]) developed was *natural deduction*. In contrast to the se-

[4] As a result, in $\mathrm{LJ}^{\supset}$ the structural rules (WR, CR and PR) become inapplicable, and $\Delta = \emptyset$ in the logical rule $\supset$-L.

quent calculus, which is intended as a formalism that supports metatheoretic analysis, natural deduction, as its name implies, is intended to reflect the way people 'naturally' work out logical arguments. Thus Gentzen suggested that, e.g., in order to convince ourselves that the S axiom schema

$$(A \supset B) \supset (A \supset B \supset C) \supset A \supset C$$

is true, we would, reading $\supset$ as 'implies', informally reason as follows: 'Assuming $A \supset B$ and then $A \supset B \supset C$ we have to show that $A \supset C$, and to do this, it is enough to show that C is true assuming A. But if A is true then, from the first assumption, B is true, and, given that A and B are true, by the second assumption C is true. Thus the S schema is true (under the intended interpretation)'.

To represent this style of argument *under assumption* we need a new kind of rule, called a *rule of inference*, that allows temporary hypotheses to be made and discharged. This is best explained by example. Consider implication; we can informally describe some of its properties as: (i) if, assuming A, it follows that B, then it follows that $A \supset B$, and (ii) if $A \supset B$ and A, then it follows that B. We might represent this reasoning diagrammatically as follows:

$$\begin{array}{c}[A]\\ \vdots \\ B \\ \hline A \supset B\end{array}\ \supset\text{-}I \qquad \frac{A \supset B \quad A}{B}\ \supset\text{-}E \tag{4}$$

where $\supset$-I and $\supset$-E are to be pronounced as 'Implication Introduction' and 'Implication Elimination', since they explain how to introduce and to eliminate (i.e. to create and to use) a formula with $\supset$ as the main connective.

We call $A \supset B$ in $\supset$-E the *major premise*, and A the *minor premise*. In general, the major premise of an elimination rule is the premise in which the eliminated connective is exhibited and all other premises are minor premises. Square brackets around assumptions indicate that they are considered to be temporary, and made only for the course of the derivation; when applying the rule we can *discharge* these occurrences. Thus, when applying $\supset$-I, we can discharge (zero or more) occurrences of the assumption A which has been made for the purposes of building a derivation of B, which shows that $A \supset B$. Of course, when applying the $\supset$-I rule, there may be other assumptions (so called open assumptions) that are not discharged by the rule. Similarly, the two premises of $\supset$-E may each have open assumptions, and the conclusion follows under the union of these.

To finish our account of $\supset$, we must explain how these rules can be used to build formal proofs. Gentzen formalized natural deduction derivations just like in sequent and Hilbert calculi, as trees, explaining how formulae are derived from formulae. With natural deduction though, there is an added complication: we also have to track the temporary assumptions that are made and discharged. Thus along with the tree of formulae, a natural deduction derivation has a *discharge function*, which associates with each node N in the tree leaf nodes above N that the rule application at node N discharges. Moreover, there is the proviso that each leaf node can be discharged by at most one node. A proof, then, is a derivation where all the assumptions are discharged (i.e. all assumptions are temporary); the formula proven is a theorem of the logic.

A natural deduction proof, with its discharge function, is a complex object in comparison with a Hilbert or sequent proof. However, it has a simple two-dimensional representation: we just decorate each node with the name of the rule to which it corresponds, and a unique number, and decorate with the same number each leaf node of the tree that it discharges. In this form we can document the previously given informal argument of the truth of the S axiom schema as a formal proof (we only number the nodes that discharge some formula):[5]

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[A \supset B \supset C]_2 \quad [A]_3}{B \supset C}\supset\text{-}E \qquad \dfrac{[A \supset B]_1 \quad [A]_3}{B}\supset\text{-}E}{C}\supset\text{-}E}{A \supset C}\supset\text{-}I_3}{(A \supset B \supset C) \supset A \supset C}\supset\text{-}I_2}{(A \supset B) \supset (A \supset B \supset C) \supset A \supset C}\supset\text{-}I_1$$

Once we have committed ourselves to this formalization of natural deduction, we find that the two rules (4) together define exactly minimal implication, in a deductive system which we call $\mathrm{NJ}^{\supset}$.

While the claims of natural deduction to be the most intuitive way to reason are plausible, the style does have some problems. First, there is the question of how to encode classical implication. We can equally easily give a direct presentation of either classical or minimal implication, using a Hilbert or sequent calculus, but not using natural deduction. While $\mathrm{NJ}^{\supset}$ is standard for minimal implication, there is, unfortunately, no simple equivalent for classical implication (what we might imagine calling $\mathrm{NK}^{\supset}$): the standard presentation of classical implication in natural deduction is as part of a larger, functionally complete, set of connectives, including, e.g., negation and disjunction, through which we can appeal to the law of excluded middle. Alternatively we can simply accept Peirce's law as an axiom. We cannot, however, using the language of natural deduction, define classical implication simply in terms of introduction and elimination rules for the $\supset$ connective.

A second problem concerns proofs themselves, which are complex in comparison to the formal representations of proofs in Hilbert or sequent calculi. It is

[5]It may help the reader trying to compare this formal proof diagram with the previous informal argument, to read it 'bottom up'; i.e. upwards from the conclusion at the root.

worth noting that a Hilbert calculus can be seen as a special simpler case of natural deduction, since axioms of a Hilbert calculus can be treated as rules with no premises, and the rules of a Hilbert calculus correspond to natural deduction rules where no assumptions are discharged.

3 NATURAL DEDUCTION AND THE LOGIC OF IMPLICATION

Given the previous remarks about the complexity of formalizations of natural deduction, it might seem unlikely that a satisfactory logical framework for it is possible. In fact, this is not the case. In this section we show that there is an abstraction of natural deduction that is the basis of an effective logical framework. Preliminary to this, however, we consider another way of formalizing natural deduction using sequents.

3.1 Natural deduction and sequents

We can describe natural deduction informally as 'proof under assumption', so the basic facility needed to formalize such a calculus is a mechanism for managing assumptions. Sequents provide this: assumptions can be stored in the antecedent of the sequent and deleted when they are discharged. Thus, if we postulate a relation $\vdash$ satisfying $Basic$ and the structural rules (3), where the succedent is a singleton set, then we can encode $\mathrm{NJ}^{\supset}$ using the rules:[6]

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \supset B}\supset\text{-}I \qquad \frac{\Gamma \vdash A \supset B \quad \Gamma \vdash A}{\Gamma \vdash B}\supset\text{-}E \tag{5}$$

This view of natural deduction is often technically convenient (we shall use it ourselves later) but it is unsatisfactory in some respects. Gentzen, and later Prawitz, had a particular idea in mind of how natural deduction proofs should look, and they made a distinction between it and the sequent calculus, using proof trees with discharge functions for natural deduction, and sequent notation for sequent calculus. We also later consider 'natural' generalizations of natural deduction that cannot easily be described using a sequent notation.

3.2 Encoding rules using implication

The standard notation for natural deduction, based on assumptions and their discharge, while intuitive, is formally complex. Reducing it to sequents allows us to formalize presentations in a simpler 'Hilbert' style; however we have also said that this is not altogether satisfactory. We now consider another notation that can

[6] Importantly, we can prove that the relation defined by this encoding satisfies the Cut rule above; it thus defines a consequence relation (see §6).

encode not only natural deduction but also generalizations that have been independently proposed.

A natural 'horizontal' notation for rules can be based on lists $\langle A_1, \ldots, A_n \rangle$ and an arrow $\rightarrow$. Consider the rules (4): we can write the elimination rule in horizontal form simply as

$$\frac{A \supset B \quad A}{B} \quad \equiv \quad \langle A \supset B, A \rangle \rightarrow B$$

and the introduction rule as

$$\frac{\begin{array}{c}[A] \\ \vdots \\ B\end{array}}{A \supset B} \quad \equiv \quad \langle A^{\dagger} \rightarrow B \rangle \rightarrow A \supset B\,.$$

In the introduction rule we have marked the assumption A with the symbol † to indicate that it is discharged. But this is actually unnecessary; using this linear notation, precisely the formulae (here there is only one) occurring on the left-hand side of some $\rightarrow$-connective in the premise list are discharged.

This new notation is suggestive: Gentzen explicitly motivated natural deduction as a formal notation for intuitive reasoning, and we have now mapped natural language connectives such as 'follows from' onto the symbol $\rightarrow$. Why not make this explicit and read $\rightarrow$ as formal implication, and '$\langle \ldots \rangle$' as a conjunction of its components? The result, if we translate into the conventional language of (quantifier free) predicate logic, is just:

$$\begin{array}{c}(T(A \supset B)\ \&\ T(A)) \rightarrow T(B) \\ (T(A) \rightarrow T(B)) \rightarrow T(A \supset B)\end{array} \tag{6}$$

which, reading the unary predicate symbol T as 'True', is practically identical with Gentzen's natural language formulation.

What is the significance of this logical reading of rules? The casual relationship observed above is not sufficient justification for an identification, and we will see that we must be careful. However, after working out the details, this interpretation provides exactly what we need for an effective logical framework, allowing us to trade the complex machinery of trees and discharge functions for a pure 'logical' abstraction.

We separate the investigation of such interpretations into several parts. First, we present a metalogic based on (minimal) implication and conjunction and give a uniform way of translating a set of natural deduction rules $\mathcal{R}$ into a set of formulae $\mathcal{R}^*$ in the metalogic. For an object logic $\mathcal{L}$ presented by a set of natural deduction rules $\mathcal{R}$, this yields a metatheory $\mathcal{L}^*$ given by the metalogic extended with $\mathcal{R}^*$. Second, we demonstrate that for any such $\mathcal{L}$, the translation is *adequate*. This means that $\mathcal{L}^*$ is 'strong enough' to derive (the representative of) any formula derivable in $\mathcal{L}$. Third, we demonstrate that the translation is *faithful*. That is that $\mathcal{L}^*$ is not too strong; we can only derive in $\mathcal{L}^*$ representatives of formulae that are derivable in $\mathcal{L}$, and further, given such a derivation, we can recover a proof in $\mathcal{L}$.[7]

3.3 The metatheory and translation

We have previously defined natural deduction in terms of formula-trees and discharge functions. We now describe the logic with which we propose to replace it.

Thus we assume that $\to$ and & have at least the properties they have in minimal logic. Above we have provided three separate formalizations of (the implicational fragment of) minimal logic: as a Hilbert calculus, as a sequent calculus, and as natural deduction. Since these all formalize the same set of theorems, we could base our analysis on any of them. However, it will be convenient for our development to use natural deduction.[8] Even though this may seem circular, it isn't: we simply need some calculus to fix the meaning of these connectives and later to formalize the correctness of our embeddings.

For the rest of this section, we will formalize deductive systems associated with *propositional* logics. We leave the treatment of quantifiers in the object logic, for which we need quantification in the metalogic (here we need only, as we will see, quantifier-free schemata), and a general theory of syntax, to §4. For now, we simply assume a suitable term algebra formalizing the language of the object logic. We will build formulae in the metalogic from the unary predicate T and the connectives $\to$ and &. To aid readability we assume that & binds tighter than $\to$, which (as usual) associates to the right. Now, let $\mathrm{NJ}^{\to,\&}$ be the natural deduction calculus based on the following rules:

$$\frac{A \quad B}{A \,\&\, B}\,\&\text{-}I \qquad \frac{A \,\&\, B \qquad \begin{array}{c}[A, B]\\ \vdots \\ C\end{array}}{C}\,\&\text{-}E$$

$$\frac{\begin{array}{c}[A]\\ \vdots \\ B\end{array}}{A \to B}\,\to\text{-}I \qquad \frac{A \to B \quad A}{B}\,\to\text{-}E$$

[7] Notice that this is a stronger requirement than the model-theoretic requirement of soundness, which requires that we should not be able to prove anything false in the model, but not that we can recover a proof of anything that we have shown to be true.

[8] One reason is that we can then directly relate derivability in the metalogic with derivability in the object logic. As is standard [Prawitz, 1965], derivability refers to natural deduction derivations with possibly open assumptions. Provability is a special case where there are no open assumptions. This generalization is also relevant to showing that we can correctly represent particular consequence relations (see §6).

We also formally define the translation, given by the mapping '$*$', from rules of natural deduction to the above language. Here A_i varies over formulae in the logic, and Φ_i over the premises (along with the discharged hypotheses) of a rule Θ.

$$\left(\frac{\Phi_1 \ldots \Phi_n}{A}\right)^* \quad \rightsquigarrow \quad \Phi_1^* \ \& \cdots \& \ \Phi_n^* \to T(A)$$
$$\begin{pmatrix} [A_1, \ldots, A_n] \\ \vdots \\ A \end{pmatrix}^* \quad \rightsquigarrow \quad T(A_1) \ \& \cdots \& \ T(A_n) \to T(A) \tag{7}$$

Note that axioms and rules that discharge no hypotheses constitute degenerate cases, i.e. $A^* \rightsquigarrow T(A)$. We extend this mapping to sets of rules and formulae, i.e. $\mathcal{R}^* \equiv \{\Theta^* \mid \Theta \in \mathcal{R}\}$.

3.4 Adequacy

If A is a theorem of an object logic $\mathcal{L}$ defined by rules $\mathcal{R}$, then we would like $T(A)$ to be provable in the metatheory $\mathcal{L}^* = \mathrm{NJ}^{\to,\&} + \mathcal{R}^*$ (i.e. $\mathrm{NJ}^{\to,\&}$ extended by the rules $\mathcal{R}^*$). More generally, if A is derivable in $\mathcal{L}$ under the assumptions $\Delta = \{A_1, \ldots, A_n\}$, then we would like for $T(A)$ to be derivable in $\mathcal{L}^*$ under the assumptions Δ^*.

Consider, for example, the object logic $\mathcal{L}$ over the language $\{\oplus, \otimes, \times, +\}$ defined by the following rules $\mathcal{R}$:

$$\frac{+}{\otimes}\,\alpha \qquad \frac{+}{\times}\,\beta \qquad \frac{\otimes \quad \times}{\oplus}\,\gamma \qquad \cfrac{\begin{matrix}[+] \\ \vdots \\ \oplus\end{matrix}}{\oplus}\,\delta \tag{8}$$

We can prove, for example, $\oplus$ by:

$$\cfrac{\cfrac{\cfrac{[+]_1}{\otimes}\,\alpha \quad \cfrac{[+]_1}{\times}\,\beta}{\oplus}\,\gamma}{\oplus}\,\delta_1 \tag{9}$$

Under our proposed encoding, the rules are translated to the following set $\mathcal{R}^*$:

$$T(+) \to T(\otimes) \tag{α^*}$$
$$T(+) \to T(\times) \tag{β^*}$$
$$T(\otimes) \ \& \ T(\times) \to T(\oplus) \tag{γ^*}$$
$$(T(+) \to T(\oplus)) \to T(\oplus) \tag{δ^*}$$

and we can prove $T(\oplus)$ in the metatheory $\mathcal{L}^* = \mathrm{NJ}^{\to,\&} + \mathcal{R}^*$ by:

$$\dfrac{\dfrac{}{(T(+) \to T(\oplus)) \to T(\oplus)}\,\delta^* \qquad \dfrac{\begin{matrix}[T(+)]_1\\ \vdots\ \Sigma\\ T(\oplus)\end{matrix}}{T(+)\to T(\oplus)}\,{\to}\text{-}I_1}{T(\oplus)}\,{\to}\text{-}E$$

where Σ is:

$$\dfrac{\dfrac{}{T(\otimes)\ \&\ T(\times)\to T(\oplus)}\,\gamma^* \qquad \dfrac{\dfrac{\dfrac{}{T(+)\to T(\otimes)}\,\alpha^* \quad T(+)}{T(\otimes)}\,{\to}\text{-}E \qquad \dfrac{\dfrac{}{T(+)\to T(\times)}\,\beta^* \quad T(+)}{T(\times)}\,{\to}\text{-}E}{T(\otimes)\ \&\ T(\times)}\,\&\text{-}I}{T(\oplus)}\,{\to}\text{-}E$$

Notice how assumptions in $\mathcal{L}$ are modeled directly by assumptions in $\mathcal{L}^*$, and how, in general, a rule application in $\mathcal{L}$ corresponds to a fragment of the derivation in $\mathcal{L}^*$. We have, for instance, the equivalence

$$\dfrac{\begin{matrix}[+]\\ \vdots\\ \oplus\end{matrix}}{\oplus}\,\delta \qquad \equiv \qquad \dfrac{\dfrac{}{(T(+) \to T(\oplus)) \to T(\oplus)}\,\delta^* \qquad \dfrac{\begin{matrix}[T(+)]_1\\ \vdots\\ T(\oplus)\end{matrix}}{T(+)\to T(\oplus)}\,{\to}\text{-}I_1}{T(\oplus)}\,{\to}\text{-}E$$

Intuitively, then, we use ${\to}\text{-}E$ to unpack the rule, and ${\to}\text{-}I$ to gather together the subproofs and discharged hypotheses.

We call the right-hand side of this the corresponding *characteristic fragment* for the rule δ. In the same way there are characteristic fragments for each of the other rules, and out of these we can build a meta-derivation corresponding to any derivation in our original logic. Moreover, these characteristic fragments can be restricted to have a special form. Given a natural deduction rule Θ then Θ^* (in the general case) has the form

$$\begin{aligned}(T(A_{1_1})\ \&\ \cdots\ \&\ T(A_{1_{m^1}}) \to T(A_1))\ \&\ \cdots&\\ \&\ (T(A_{n_1})\ \&\ \cdots\ \&\ T(A_{n_{m^n}}) \to T(A_n)) \to T(A)\,,&\end{aligned}$$

and we can show:

LEMMA 3.1 *Given a rule Θ, there is a characteristic fragment Σ where for $1 \leq i \leq n$, atomic assumptions $T(A_{i_1}), \ldots T(A_{i_{m^i}})$ are discharged in subderivations* *of $T(A_i)$, and then these subderivations are combined together with Θ^* to prove $T(A)$. Further, if we insist also that the major premise of an elimination rule is never the result of the application of an introduction rule (i.e. $\rightarrow$-I or &-I), while the minor premise is always either atomic or the result of the application of an introduction rule, then Σ is unique.*

Proof. By an analysis of the possible structure of Θ. ■

For a rule Θ that does not discharge assumptions, i.e., of the form

$$\frac{A_1 \ldots A_n}{A},$$

the characteristic fragment is just a demonstration that, given Θ^*, the rule

$$\frac{T(A_1) \ldots T(A_n)}{T(A)}$$

is derivable in $\mathcal{L}^*$. That is, there is a derivation of $T(A)$ where the only open assumptions are $T(A_1), \ldots, T(A_n)$. Note that this standard notion of derivability (see, e.g., Troelstra [1982] or Hindley and Seldin [1986]), can be extended (see Schroeder-Heister [1984b] and §3.6 below) to account for rules that discharge assumptions; in this extended sense, each characteristic fragment justifies a derived rule that allows us to simulate $\mathcal{L}$-derivations in $\mathcal{L}^*$.

Given the existence of characteristic fragments, it is a simple matter to model $\mathcal{L}$-derivations in $\mathcal{L}^*$. To begin with, since our metalogic is a natural deduction calculus, we can model assumptions in the encoded logic as assumptions in the metalogic. Each such assumption B_i is modeled by an assumption $T(B_i)$. Now, given a derivation in the object logic, we can inductively replace rules by corresponding characteristic fragments to produce a derivation in $\mathcal{L}^*$. For example, in (9) we proved $\oplus$ using four rule applications; after we gave a proof in $\mathcal{L}^*$ of $T(\oplus)$ built from the four characteristic fragments that correspond to these applications.

In this form, however, this observation assumes not just a metalogic ($\mathrm{NJ}^{\rightarrow,\&}$) but also a particular proof calculus for the metalogic (natural deduction), which, since we want a purely logical characterization, we want to avoid. It is easy to remove this assumption by observing that A follows from the assumptions A_1 to A_n in $\mathrm{NJ}^{\rightarrow,\&}$ iff $A_1 \mathbin{\&} \cdots \mathbin{\&} A_n \rightarrow A$ follows without the assumptions. Thus we have a theorem that states the adequacy of encodings of natural deduction calculi in $\mathrm{NJ}^{\rightarrow,\&}$.[9]

THEOREM 3.2 (Adequacy) *For any natural deduction calculus $\mathcal{L}$ defined by a set of rules $\mathcal{R}$, if the formula A is derivable under assumptions $A_1, \ldots, A_n$, then $T(A_1) \mathbin{\&} \cdots \mathbin{\&} T(A_n) \rightarrow T(A)$ is provable in $\mathcal{L}^* = \mathrm{NJ}^{\rightarrow,\&} + \mathcal{R}^*$.*

[9] However notice that the translation is only defined on rules without side conditions, i.e. for 'pure' natural deduction; for more idiosyncratic logics, see the discussion in §6.

Notice that the proof is based on translation, and hence is constructive: given a derivation in the object logic we can construct one in the metalogic.

3.5 Faithfulness

In proving adequacy, we only used the metatheory $\mathcal{L}^*$ in a limited way where derivations had a simple structure built by pasting characteristic fragments together. Of course, there are other ways to build proofs in $\mathcal{L}^*$ and it could be that this freedom allows us to derive representations of formulae that are not derivable in $\mathcal{L}$. We now show that our translation is faithful, i.e. this is not the case.

To see why we have to be careful about faithfulness, consider the following: in the deductive system $\mathrm{NJ}^{\to,\&}$, the $\to$ connective is minimal implication (the logic is a conservative extension of $\mathrm{NJ}^{\to}$). But what happens if we strengthen the metalogic to be classical, e.g. by adding classical negation or assuming Peirce's law? We can still show adequacy of our encodings, since derivations in $\mathrm{NJ}^{\to,\&}$ remain valid when additional rules are present, but we can also use the encoding to derive formulae that are not derivable in the original system. Consider what happens if we try to prove something that is classically but not minimally valid, like, for instance, Peirce's law itself:

$$T(((A \supset B) \supset A) \supset A)\,. \tag{10}$$

Using the axioms in (6), we can reduce this to the problem of proving

$$((T(A) \to T(B)) \to T(A)) \to T(A)\,. \tag{11}$$

But this is itself an instance of Peirce's law, and is provable if $\to$ is classical implication. Thus $T(\cdot)$ does not here define the set of minimal logic theorems.

If $\to$ is read as minimal implication, then the same trick does not work. We can still reduce (10) to (11), but we are not able to take the final step of appealing to Peirce's law in the metalogic.

We now show that assuming $\to$ to be minimal really does lead to a faithful presentation of object logics, by demonstrating a direct relationship between derivations of formulae $T(A)$ in the $\mathcal{L}^*$ and derivations of A in $\mathcal{L}$.

The desired relationship is not a simple isomorphism: we pointed out in discussing adequacy above, that for each natural deduction rule translated into our logical language, it is possible to find a characteristic fragment, and using these fragments we can translate derivations in $\mathcal{L}$ into derivations in $\mathcal{L}^*$. However an arbitrary derivation in $\mathcal{L}^*$ may not have a corresponding derivation in $\mathcal{L}$ (it is a simple exercise to construct a proof of $T(\oplus)$ that does not use characteristic fragments). But if we look more carefully at the derivation of $T(\oplus)$ we have given, and the fragments from which it is constructed, we can see that it has a particularly simple structure, and this structure has a technical characterization, similar to that we used for the characteristic fragments themselves. The derivations we build to show adequacy are in what is called *expanded normal form* (ENF): the major premise of an elimination rule (i.e. $\to$-E or &-E) is never the result of the application of an introduction rule (i.e. $\to$-I or &-I), and all minor premises are either atomic or the result of introduction rules. Not all derivations are in ENF, but any derivation can be transformed into one that is. We have the following:

FACT 3.3 (Prawitz [1971]) *There is an algorithm transforming any derivation in* $\mathrm{NJ}^{\to,\&}$ *into an equivalent derivation in ENF, and this equivalent derivation is unique.*

From this we get the theorem we need; again, like for the statement of adequacy, we abstract away from the deductive system to get a pure logical characterization:

THEOREM 3.4 (Faithfulness) *For any natural deduction system $\mathcal{L}$ defined by a set of rules $\mathcal{R}$, if we can prove $T(A_1)\ \&\ \cdots\ \&\ T(A_n) \to T(A)$ in $\mathcal{L}^*$, then A is derivable in $\mathcal{L}$ from the assumptions $A_1, \ldots, A_n$.*

Proof. The theorem follows immediately from the existence of ENF derivations and Lemma 3.5 below. ■

LEMMA 3.5 *There is an effective transformation from ENF derivations in $\mathcal{L}^*$ of $T(A)$ from atomic assumptions Δ, to natural deduction derivations in $\mathcal{L}$ of A from assumptions $\{B \mid T(B) \in \Delta\}$.*

Proof. We prove this by induction on the structure of ENF derivations. Intuitively, we show that an ENF derivation is built out of characteristic fragments, and thus can easily be translated back into the original deductive system. Consider a derivation Σ of $T(A)$.

Base case: $T(A)$ corresponds to either an assumption in Δ or a (premisless) rule of the encoded theory. The translation then consists either of the assumption A or of the corresponding premisless rule with conclusion A.

Step case: Since the derivation is in ENF and the conclusion is atomic, the last rule applied is an elimination rule; more specifically, the derivation must have the form

$$\dfrac{\dfrac{}{\Phi_1^*\ \&\ \cdots\ \&\ \Phi_n^* \to T(A)}\,\Theta^* \qquad \dfrac{\begin{array}{ccc}\vdots\,\Sigma_1 & & \vdots\,\Sigma_n\\ \Phi_1^* & \cdots & \Phi_n^*\end{array}}{\Phi_1^*\ \&\ \cdots\ \&\ \Phi_n^*}\,\sharp}{T(A)}\,\to\text{-}E \tag{12}$$

where $\sharp$ consists only of applications of &-I and Θ^* is

$$\Phi_1^*\ \&\ \cdots\ \&\ \Phi_n^* \to T(A)$$

for some rule Θ of the encoded system.

By the definition of the encoding, each Φ_i^*, derived by a proof Σ_i, is then of the form

$$T(A_{i_1}) \mathbin{\&} \cdots \mathbin{\&} T(A_{i_{m^i}}) \rightarrow T(A_i).$$

Since the conclusion of Σ_i proves an implication, and is in ENF, the last rule applied must be $\rightarrow$-I; thus Σ_i must have the form

$$\dfrac{\dfrac{[T(A_{i_1}) \mathbin{\&} \cdots \mathbin{\&} T(A_{i_{m^i}})]_1 \qquad \begin{array}{c}[T(A_{i_1})] \ldots [T(A_{i_{m^i}})] \\ \vdots\, \Sigma_i' \\ T(A_i)\end{array}}{T(A_i)}\,\sharp}{T(A_{i_1}) \mathbin{\&} \cdots \mathbin{\&} T(A_{i_{m^i}}) \rightarrow T(A_i)}\,\rightarrow\text{-}I_1$$

with open assumptions Δ, where $\sharp$ consists only of applications of &-E, which 'unpack' the assumption $T(A_{i_1}) \mathbin{\&} \cdots \mathbin{\&} T(A_{i_{m^i}})$ into its component propositions. In other words, (12) corresponds to the characteristic fragment for Θ.

By induction we can apply the same analysis to each Σ_i', taking account not only of the undischarged assumptions Δ, but also the new atomic assumptions $\Delta + \{T(A_{i_1}), \ldots, T(A_{i_{m^i}})\}$ which have been introduced, and we are finished. ■

3.6 Generalizations of natural deduction

The distinction we mentioned above (in §3.1) that Gentzen and Prawitz make between natural deduction and sequent calculi has recently been emphasized by Schroeder-Heister [1984a; 1984b], who has proposed a 'generalized natural deduction'. This extension, which follows directly from Gentzen's appeal to intuition to justify natural deduction, is not formalizable using sequents in the way suggested in §3.[10]

Consider again Gentzen's proposed 'natural language' analysis of logical connectives. In these terms we can characterize Implication Elimination as

> If $A \supset B$ and A, then it follows that B

which we formalize as:

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

Now consider the slightly more convoluted

> If $A \supset B$ and assuming that if, given that from A we could derive B, we could derive C, then we can derive C.

[10] Though see Avron [1990] for relevant discussion.

which is also true. There is a natural generalization of rules that allows us to express this in the spirit of natural deduction; namely,

$$\dfrac{A \supset B \qquad \begin{array}{c}\left[\dfrac{A}{B}\right]\\ \vdots \\ C\end{array}}{C} \tag{13}$$

where we can assume rules as hypotheses (and by generalization, have rules as hypotheses of already hypothetical rules, and so on). Schroeder-Heister shows that it is quite a simple matter to extend the formula-tree/discharge-function formalization to cope with this extension: we simply allow discharge functions to point to subtrees rather than just leaves.

Why is such a generalization interesting? Schroeder-Heister uses it to analyze systematically the intuitionistic logical connectives. However it has more general use, and at least one immediate application: we can use it to encode Peirce's law without introducing new connectives, as

$$\dfrac{\begin{array}{c}\left[\dfrac{A}{B}\right]\\ \vdots \\ A\end{array}}{A}$$

which, in English, is equivalent to

> if, by assuming that some B follows from A, it follows that A, then it follows that A.

This provides, finally, a self-contained generalized natural deduction formulation of classical implication. (At least in the sense that the meaning of implication can be defined by rules involving no auxiliary connectives.)

The process of generalization can obviously be continued beyond allowing rules as hypotheses, though. There is no reason why we cannot also have them as conclusions. Thus, for instance, it is clearly true that

$$\dfrac{A \supset B}{\left(\dfrac{A}{B}\right)} \to\text{-}E_C$$

(where round brackets do not, in the manner of square brackets, represent the discharge of an assumption, but simply grouping), which might reasonably be read as:

If $A \supset B$ then from A it follows that B.

By now, however, while our intuition still seems to be functioning soundly, the formula-tree/discharge-function formalization is beginning to break down. It can cope with rules as assumptions, but the more general treatment of both rules and formulae in derivations that we are now proposing is more difficult. With our proposed alternative of the metalogic $\mathrm{NJ}^{\to,\&}$, on the other hand, the same problems do not arise. In fact one way of looking at the faithfulness argument developed above is as essentially a demonstration that this sort of 'confusion' can be unwound for the purpose of recovering a proof in a system that does not in the end allow it. The question then is, how closely does our logic match the traditional formulation?

In the most general case, allowing rules as both hypotheses and conclusions, such a question is not quite meaningful, since we do not have a traditional formulation against which we can compare. However if we limit ourselves to the restricted case where we have rules as assumptions, it is still possible to be properly formal, since we can still compare our encoding to Schroeder-Heister's more traditional formalization in terms of formula-trees and discharge-functions.

Such a formulation is enough to allow us, by a 'natural' generalization of the encoding in (7), to formalize, for instance, (13) as

$$(T(A \supset B)\ \&\ ((T(A) \to T(B)) \to T(C))) \to T(C)$$

and $\to$-E_C as

$$T(A \supset B) \to T(A) \to T(B)\,.$$

It is now possible, by a corresponding generalization of the notion of deduction (see Schroeder-Heister [1984b]) to show adequacy and faithfulness for this larger class of deductive systems.

Generalized natural deduction and curryed rules

Until now we have been using $\mathrm{NJ}^{\to,\&}$ to encode and reason with deductive systems. If we want to show a direct relationship between the traditional notation of natural deduction and our encoding, then we seem to need both the $\to$ and $\&$ connectives in the metalogic. However, compare $\to$-E and $\to$-E_C; while there seems to be large differences between the two as rules, there is very little either in natural language or in logical framework terms: one is simply the 'curryed' form of the other; in this case the conjunction seems almost redundant.

On the other hand, one generalization that we have not made so far, and one that is suggested by $\mathrm{NJ}^{\to,\&}$, is to allow a natural deduction rule with multiple conclusions; e.g. assuming an ordinary conjunction $\wedge$, have a (tableau-like) rule

$$\frac{A \wedge B}{A, B}\ \&\text{-}E_{MC}$$

corresponding to the natural language

If $A \wedge B$ then it follows that A and B.

Both of which correspond to the framework formalization

$$T(A \wedge B) \to T(A) \,\&\, T(B)\,.$$

However, there seems to be less of a need for rules of this form: a single encoded rule $C \to A \,\&\, B$ can be replaced by a pair $C \to A$ and $C \to B$.

If we are willing to accept this restriction to the language of $\to$ alone, then we can simplify the metalogic we are using: if we have conjunctions only in the antecedents of implications $\to$, they can always be eliminated by 'currying' the formulae conjoined in the antecedent, allowing us to dispose of conjunction completely. For example, the curryed form of the axiom for $\to$-E in (6) is then

$$T(A \supset B) \to T(A) \to T(B)\,.$$

Notice that this form of $\to$-E is indistinguishable from the encoding we give above for $\to$-E_C: we are no longer able to distinguish some different rules in generalized natural deduction, and thus we lose the faithfulness/adequacy bijection that we have previously demonstrated. However this problem is not serious: we are only identifying proofs that differ in simple ways, e.g., by the application of curryed versus uncurryed rules. Furthermore, this possible confusion arises in the case of *generalized* natural deduction, but not in the traditional form.

In the next section we describe a full logical framework based on the ideas we have developed here. In fact it turns out that we can adopt the same notation to formalize not only deductive systems, but also languages, in a uniform manner.

4 FRAMEWORKS BASED ON TYPE-THEORIES

In the last section we established a relationship between natural deduction and the logic of implication. However we considered only reasoning about fragments of propositional logic, and when we turn to predicate logics, we find that the mechanisms of binding and substitution introduce some entirely new problems for us to solve.

The first problem is simply how to encode languages where operators bind variables. Such variable binding operators include standard logical quantifiers, the λ of the λ-calculus, fixedpoint operators like μ in fixedpoint logics, etc. Until now we have been using a simple term algebra to represent syntax, where, e.g. a binary connective like implication is represented by a binary function. However, with the introduction of binding and substitution this approach is less satisfactory. For instance $\forall x.\, \phi(x)$ and $\forall y.\, \phi(y)$ are distinct syntactically, but not in terms of the deductive system (any proof of one proves the other). Binding operators also complicate operations performed on syntax; e.g. substitution. The second problem is that proof-rules become more complex: the rules for quantifiers place conditions on the contexts (e.g. insisting that certain variables do not appear free) in which they can be applied.

Now we extend our investigation to deal with these problems, and complete the development of a practical $\rightarrow$-framework. We tackle the two problems of language encoding and rule encoding together, by introducing the λ-calculus as a representation language into our system. This provides us with a way to encode quantifiers using *higher-order* syntax and then to encode rules for these quantifiers.

There are different ways that we can combine the λ-calculus with a metalogic. One possibility is simply to add it to the term language, extending, e.g., the theory $\mathrm{NJ}^{\rightarrow}$ to a fragment of higher-order logic.[11] Another, similar, possibility, and one which offers some theoretical advantages, is to use a *type-theory*. We investigate the type-theoretic approach in this section. Type-theories based on the λ-calculus are well-known to be closely related to intuitionistic logics like $\mathrm{NJ}^{\rightarrow}$ via 'Curry-Howard' (see Howard [1980]), or *propositions-as-types*, isomorphisms, a fact which allows us to carry across much of what we already know about encoding deductive systems from $\mathrm{NJ}^{\rightarrow}$. Moreover, an expressive enough type-theory provides a unified language for representing not just syntax, but also proof-rules and proofs. Thus a type-theoretic logical framework can provide a single solution to the apparently distinct problems of encoding languages and deductive systems: The encoding problems are reduced to declaration of appropriate (higher-order) signatures and the checking problems (e.g. well-formedness of syntax and the correctness of derivations) to the problem of type checking against these signatures.

4.1 Some initial observations

We begin by considering the problem of formalizing a binding operator, taking $\forall$ as our example. Consider what happens if we follow through the analysis given above for natural deduction in propositional logic. First we state, in Gentzen style natural language, some of the properties of $\forall$; e.g.

> if, for arbitrary t, it follows that $\phi(t)$, then it follows that $\forall x.\, \phi(x)$

and

> if it follows that $\forall x.\, \phi(x)$, then for any t it follows that $\phi(t)$.

These are traditionally represented, in the notation of natural deduction, as

$$\frac{\phi}{\forall x.\, \phi}\,\forall\text{-}I \qquad \text{and} \qquad \frac{\forall x.\, \phi}{\phi[x \leftarrow t]}\,\forall\text{-}E$$

[11] See, e.g., Felty [1989], Paulson [1994] or Simpson [1992], for examples of this approach.

where, in $\forall$-I, the variable x does not appear free in any undischarged assumption, and the notation $\phi[x \leftarrow t]$ denotes the formula ϕ where the term t has been substituted through for x (care being taken to avoid capturing variables). The relationship between these rules and their informal characterizations is less direct here than in the propositional case; for example, we model the statement 'If, for arbitrary t, it follows that $\phi(t)$' indirectly, by assuming that an arbitrary variable x can stand for an arbitrary term, then ensuring that x really is arbitrary by requiring that it does not occur free in the current assumptions.

Consider how we might use a 'logical' language instead of rules by extending our language based on minimal implication. To start with, we need a way of saying that a term is arbitrary, which we can accomplish with universal quantification. Furthermore, unlike in the propositional case, we have two syntactic categories. As well as formulae (fm), we now have terms (tm), and we will use types to formally distinguish between them. If we combine quantification with typing, by writing $(x{:}y)$ to mean 'for all x of type y', then a first attempt at translating natural language into (semi)formal language results in the following:

$$\begin{array}{l} T(\forall x.\, \phi(x)) \to (t{:}tm)T(\phi(t)) \\ (t{:}tm)T(\phi(t)) \to T(\forall x.\, \phi(x)) \end{array} \tag{14}$$

However while this appears to capture our intuitions, it is not clear that it is formally meaningful. If nothing else, one might question the cavalier way we have treated the distinction between object-level and metalevel languages. In the following sections we show that this translation does in fact properly correspond to the intuitive reading we have suggested.

4.2 Syntax as typed terms

We begin by considering how languages that include variable-binding operators can be represented. We want a method of representing syntax where terms in the object logic are represented by terms in the metalogic. This representation should be computable and we want too that it is compositional; i.e. that the representative of a formula is built directly from the representatives of its subformulae.

Types provide a starting point for solving these problems. A type system can be used to classify terms into types where well-typed terms correspond to well-formed syntax. Consider the example of first-order logic; this has two syntactic categories, terms and formulae, and, independent of whether we view syntax as strings, trees, or something more abstract, we must keep track of the categories to which subexpressions belong. One way to do this is to tag expressions with their categories and have rules for propagating these tags to ensure that entire expressions are 'well-tagged'. Even if there is only one syntactic category we still need some notion of well-formed syntax; in minimal logic, for instance, some strings built from implication and variables, such as $\phi \supset \psi$, are well-formed formulae, while others, like $\phi{\supset}{\supset}\psi$, are not.

In a typed setting, we can reduce the problem of syntactic well-formedness to the problem of well-typedness by viewing syntax as a typed term algebra: we associate a type of data with each syntactic category and regard operators over syntax as typed functions. For instance, $\supset$ corresponds to a function that builds a formula given two formulae, i.e., a function of type $fm \times fm \to fm$. Under this reading, and using infix notation, $\phi \supset \psi$ is a well-typed formula provided that ϕ and ψ are both well-typed formulae of type *fm*, whereas $\phi{\supset}{\supset}\psi$ is ill-typed.

This view of syntax as typed terms provides a formalization of the treatment of propositional languages as term algebras that we have informally adopted up to now. We will see that it also provides a basis for formalizing languages with quantification. In fact the mechanism by which this is done is so flexible that it is possible to claim:

THESIS 4.1 *The typed λ-calculus provides a logical basis for representing many common kinds of logical syntax.*

Note that we do not claim that the typed λ-calculus, as we shall use it, is a universal solution applicable to formalizing any language. We cannot, for instance, formalize a syntax based on a non-context-free grammar. Neither can we give a finite signature for languages that require infinitely many (or parameterized) sets of productions. The notation is remarkably flexible nevertheless, and, especially if we refine the type system a little further, can deal with a great many subtle problems — a good example of this can be found in the analysis of the syntax of higher-order logic developed by Harper *et al.* [1993].

The simply typed λ-calculus

We assume that the reader is familiar with the basics of the λ-calculus, e.g. reduction (we denote one-step reduction by $\to_\beta$ and its reflexive-transitive closure by $\xrightarrow{*}_\beta$) and conversion ($=_\beta$), and review only syntax and typing here; further details can be found in Barendregt [1984; 1991] or Hindley and Seldin [1986]. We now define a type theory based on this, called the *simply typed* λ-calculus, or more succinctly $\lambda^\to$. Our development is based loosely on that of Barendregt and anticipates extensions we develop later.

We start by defining the syntax we use:

DEFINITION 4.2 *Fix a denumerable set of variables $\mathcal{V}$. The terms and types of $\lambda^\to$ are defined relative to a signature consisting of a non-empty set of* base types *$\mathcal{B}$, a set of* constants *$\mathcal{K}$, and a* constant signature *Σ, which is a set $\{\, c_i{:}A_i \mid 1 \leq i \leq n, c_i \in \mathcal{K}, A_i \in \mathrm{Ty} \,\}$, where the c_i are distinct, and:*

- *The set of* types Ty *is given by*

$$\mathrm{Ty} ::= \mathcal{B} \mid \mathrm{Ty} \to \mathrm{Ty}\,.$$

- *The set of* terms $\mathcal{T}$ *is given by*

$$\mathcal{T} ::= \mathcal{V} \mid \mathcal{K} \mid \mathcal{T}\mathcal{T} \mid \lambda\mathcal{V}^{\mathrm{Ty}}.\, \mathcal{T}\,.$$

  A variable x occurs bound *in a term if it is in the scope of a $\lambda\, x$ and x occurs* free *otherwise. We implicitly identify terms that are identical under a renaming of bound variables.*

- *A* typing context *is a sequence $x_1{:}A_1, \ldots, x_n{:}A_n$ of* bindings *where the x_i are distinct variables and $A_i \in \mathrm{Ty}$ for $(1 \leq i \leq n)$.*

  For convenience, we shall overload the $\in$ relation, extending it from sets to sequences in the obvious manner (i.e. so that $x_i{:}A_i \in x_1{:}A_1, \ldots, x_n{:}A_n$ iff $1 \leq i \leq n$).

- *A* type assignment relation $\models_{\Sigma}$ *is a binary relation, indexed by Σ, defined between typing contexts Γ and typing assertions $M{:}A$ where $M \in \mathcal{T}$ and $A \in \mathrm{Ty}$, by the inference rules:*

$$\frac{c{:}A \in \Sigma}{\Gamma \models_{\Sigma} c{:}A}\,assum \qquad \frac{\Gamma, x{:}A \models_{\Sigma} M{:}B}{\Gamma \models_{\Sigma} (\lambda x^A.\, M){:}(A \to B)}\,abst$$

$$\frac{x{:}A \in \Gamma}{\Gamma \models_{\Sigma} x{:}A}\,hyp \qquad \frac{\Gamma \models_{\Sigma} M{:}A \to B \quad \Gamma \models_{\Sigma} N{:}A}{\Gamma \models_{\Sigma} (MN){:}B}\,appl$$

This definition states that types consist of the closure of a set of base types under the connective $\to$ and that terms are either variables, constants, applications or (typed) λ-abstractions. The rules constitute a deductive system that defines when a term is well-typed relative to a context and a signature, which in turn assign types to free variables and constants. Notice that we do not have to make explicit the standard side condition for $abst$ that the variable x does not appear free in Γ, since this is already implicitly enforced by the requirement that variables in a type context must be distinct.

As an example, if $\{A, B\} \subseteq \mathrm{Ty}$ then

$$\frac{\dfrac{\dfrac{x{:}A \in x{:}A, y{:}B}{x{:}A, y{:}B \models_{\Sigma} x{:}A}\,assum}{x{:}A \models_{\Sigma} \lambda y^B.\, x{:}B \to A}\,abst}{\models_{\Sigma} \lambda x^A.\, \lambda y^B.\, x{:}A \to (B \to A)}\,abst \tag{15}$$

is a proof (for any Σ). Further, it is easy to see that:

FACT 4.3 *Provability for this system (and thus well-typing) is decidable.*

A Curry-Howard isomorphism for $\lambda^{\rightarrow}$

The rules for $\lambda^{\rightarrow}$ suggest the natural deduction presentation for $\mathrm{NJ}^{\rightarrow}$ (see §3.1): if we rewrite all instances of $\Gamma \vdash_{\Sigma} M{:}A$ in a proof in $\lambda^{\rightarrow}$ as $\Gamma, \Sigma \vdash M{:}A$ and uniformly replace each typing assertion $c{:}A$, $x{:}A$, and $M{:}A$ by the type A, then $abst$ and $appl$ correspond to $\rightarrow$-I and $\rightarrow$-E, while hyp and $assum$ together correspond to $Basic$. The following makes this correspondence more precise.

FACT 4.4

1. *There is a bijection between types in $\lambda^{\rightarrow}$ and propositions in $\mathrm{NJ}^{\rightarrow}$ where a proposition in $\mathrm{NJ}^{\rightarrow}$ is provable precisely when the corresponding $\lambda^{\rightarrow}$ type is inhabited (has a member).*

2. *There is a bijection between members of types in $\lambda^{\rightarrow}$ and proofs of the corresponding propositions in $\mathrm{NJ}^{\rightarrow}$.*

Part 1 of this characterizes the direct syntactic bijection between types and propositions where base types correspond to atomic propositions[12] and the function space constructor corresponds to the implication connective. The correspondence between provability and inhabitation then follows from the correspondence between the proof-rules of the two systems. Then part 2 refines the bijection to the level of inhabiting terms on the one hand, and proofs on the other.

Fact 4.4 states an isomorphism between types and propositions that we can characterize as 'truth is inhabitation': if $M{:}A$ then the proposition corresponding to A is provable and, moreover, there is a corresponding notion of reduction in the two settings where β-reduction of M in the λ-calculus corresponds to reduction of the proof A (in the sense of Prawitz).

We exploit this isomorphism in reasoning about encodings in this chapter. In this section we show the correctness of encodings by reasoning about normal forms of terms in type-theory and in §5 we will reason about normal forms of proofs.

REMARK 4.5 *For reasoning about the correctness of encodings it is sometimes necessary to use a more complex isomorphism based on both β and η-conversion. It is possible to establish a correspondence between so called long $\beta\eta$-normal forms[13] and corresponding classes of derivations in $\mathrm{NJ}^{\rightarrow}$. More details can be found in [Harper* et al.*, 1993].*

---

[12] This fact holds for the pure version of $\lambda^{\rightarrow}$ without constants. We also implicitly assume a bijection between base types and propositional variables.

[13] A term $M = \lambda x_1 \ldots x_n . x M_1 \ldots M_m$ is in long $\beta\eta$-normal form when 1) x is an x_i, a constant, or a free variable of M, 2) $x M_1 \ldots M_n$ is of base type (i.e. x is applied fully to all possible arguments) and 3) each M_i is in long $\beta\eta$-normal form.

Representation

In order to represent syntax in $\lambda^{\rightarrow}$ we define a suitable signature Σ and establish a correspondence between terms in the metalogic and syntax in the object logic. Our signature will consist of a base type for each syntactic category of the object logic and a constant for each constructor in the object logic. We will see that we do not need a type *variable* to formalize variables in the object logic because we can use instead variables of the metalogic. Syntax belonging to a given category in the object logic then corresponds to terms in the metalogic belonging to the type of that category.

This is best illustrated with a simple example: we represent the syntax of minimal logic itself. We follow Church's [1940] convention, where o is the type of propositions, so the signature is the (singleton) set of base types $\{o\}$, and the constant signature is

$$\Sigma \equiv \{imp\colon o \to o \to o\}\,. \tag{16}$$

The constructor imp builds a proposition from two propositions. With respect to this signature, $imp\ x\ \ (imp\ y\ z)$ is a term that belongs to o when x, y, and z belong to o; i.e. $x{:}o, y{:}o, z{:}o \vdash_{\Sigma} imp\ x\ (imp\ y\ z){:}o$. This corresponds to the fact that $x \supset (y \supset z)$ is a proposition of minimal logic provided that x, y, and z are propositions.

Thus formulae of minimal logic correspond to well-typed terms of type o. Formulating this correspondence requires some care though: we have to check adequacy and faithfulness for the represented syntax; this amounts to showing that (i) every formula in minimal logic is represented by a term of type o, and (ii) that every term of type o represents a formula in minimal logic.

As a first attempt to establish such a correspondence, consider the following mapping $\ulcorner\cdot\urcorner$, from terms in minimal logic to λ-calculus terms:

$$\begin{aligned} \ulcorner x \urcorner &= x \\ \ulcorner t_1 \supset t_2 \urcorner &= imp\, \ulcorner t_1 \urcorner\, \ulcorner t_2 \urcorner \end{aligned}$$

As an example, under this representation the formula $x \supset (y \supset z)$ corresponds to the term $imp\ x\ (imp\ y\ z)$. The representation function is an injection from propositions to terms of type o, provided that variables are declared to be of type o in the context. However, it is not surjective: there are too many terms of type o. For example in a context where x is of type $o \to o$ and y is of type o, then $x\ y$ is of type o, and so is $imp\ ((\lambda z^{o}.\, z)\ y)\ y$; but neither of these is in the image of $\ulcorner\cdot\urcorner$. The problem with the first example is that the variable x is of higher type (i.e. the function type $o \to o$) and not a base type in $\mathcal{B}$. The problem with the second example is that it is not in normal form. Any such term, however, has a unique equivalent β-normal form, which we can compute. In the above example the term has the normal form $imp\ y\ y$, which is $\ulcorner y \supset y \urcorner$. Our correspondence is thus a bijection when we exclude the cases above: we consider only β-normal form terms where free variables are of type o.

THEOREM 4.6 *$\ulcorner\cdot\urcorner$ is a bijection between propositions in minimal logic with propositional variables $x_1, \ldots, x_n$, and β-normal form terms of type o containing only free variables $x_1, \ldots, x_n$, all of type o.*

First-order syntax

The syntax of minimal logic is very simple, and we do not need all of $\lambda^{\rightarrow}$ to encode it. We can be precise about what we mean by 'not all' if we associate types with orders: observe that any type γ has a unique representation as $\alpha_1 \rightarrow \ldots \rightarrow \alpha_n \rightarrow \beta$, where $\rightarrow$ associates to the right and β is a base type. Now define the *order* of γ to be 0 if $n = 0$, and $1 + max(Ord(\alpha_1), \ldots, Ord(\alpha_n))$ otherwise.[14] In our encoding of minimal propositional logic we have used only the first-order fragment of $\lambda^{\rightarrow}$; i.e. variables are restricted to the base type o and the only function constant imp is first-order (since its two arguments are of base type); this is another way of saying that an encoding using a simple term algebra is enough.

This raises an obvious question: why adopt a full higher-order notation if a simple first-order notation is enough? Indeed, in a first-order setting, results like Theorem 4.6 are much easier to prove because there are no complications introduced by reduction and normal forms. The answer is that the situation changes when we introduce quantifiers and other variable binding operators. A 'naive' encoding of syntax is still possible but is much more complicated.

Consider, as an example, the syntax of first-order arithmetic. This is defined in terms of two syntactic categories, terms and formulae, which are usually specified as:

$$\begin{aligned} \text{terms } T &::= x \mid 0 \mid sT \mid T + T \mid T \times T \\ \text{formulae } F &::= T = T \mid \neg F \mid F \wedge F \mid F \vee F \mid F \supset F \mid \forall x.\, F \mid \exists x.\, F \end{aligned}$$

How should we represent this? A possible first-order notation is as follows: we define a base type v of variables, in addition to types i for terms (i.e. 'individuals') and o for formulae. Since variables are also terms, our signature requires a 'coercion' function mapping elements of type v to elements of type i. The rest of the signature is formalized as we would expect; e.g. $plus$ is a constant of type $i \rightarrow i \rightarrow i$, atomic formulae are built from the equality relation *eq* of type $i \rightarrow i \rightarrow o$, connectives are defined as propositional functions over o, and a quantifier like $\forall$ is formalized by declaring a function constant *all* of type $v \rightarrow o \rightarrow o$ taking a variable and a formula to a formula.

[14] Note that there is not complete agreement in the literature about the order of base types, which is sometimes defined to be 1, not 0.

This part of the encoding presents no difficulties. However the problems with first-order representations of a language with binding are not directly in the representation, but rather in the way we will use that representation. It is in the formalization of proof-rules where we encounter the problems, in particular with substitution. Consider, for instance, $\forall$-E in (4.1), where we use the notation $\phi[x \leftarrow t]$; we need to formalize an analogue for our encoding, which we can do by introducing a ternary function, sub of type $o \to i \to v \to o$, where $sub(\phi, t, x) = \psi$ is provable precisely when $\phi[x \leftarrow t] = \psi$. With this addition $\forall$-E is axiomatizable as

$$\forall \phi, \psi{:}o. \forall t{:}i. \forall x{:}v. \, (sub(\phi, t, x) = \psi) \to T(all\ x\ \phi) \to T(\psi)\,. \qquad (17)$$

There are several ways we might axiomatize the details of sub. The most direct approach is simply to formalize the standard textbook account of basic concepts such as free and bound variables, capture, and equivalence under bound variable renaming, which we can easily do by structural recursion on terms and formulae. The definitions are well-known, although complex enough that some care is required, e.g. bound variables must sometimes be renamed to avoid capture. Note too that in (17) we have used the equality predicate over formulae (not to be confused with equality over terms in the object logic, i.e. *eq*), which must either be provided by the metalogic or additionally formalized. Examples of such equational encodings of logics are given by Martí-Oliet and Meseguer [2002].

Other encodings have also been explored: for instance we can use a representation of terms that finesses problems involving bound variable names by eliminating them entirely. Such an approach was originally suggested by de Bruijn [1972] who represents terms with bound variables by replacing occurrences of such variables with 'pointers' to the operators that bind them. A related approach has been proposed by Talcott [1993], who has axiomatized a general theory of binding structure and substitution, which can be used as the basis for encoding logics that require such facilities. Another recent development, which is gaining some popularity, is to provide extensions of the λ-calculus that formalize operators for 'explicit substitutions' [Abadi *et al.*, 1991].

In all of these approaches, direct or indirect, there is considerable overhead, and not only at the formalization stage itself: when we are building proofs, we have to construct subproofs using the axiomatization of sub at every application of $\forall$-E. Being forced to take such micro-steps in the metatheory simply to apply a rule in the object logic is both awkward and tedious. And there is another serious problem that appears when we consider a rule like $\forall$-I in (4.1), where we have a side condition that *the variable x does not appear free in any undischarged assumption*. There is no direct way, in the language we have been developing, to complete the formalization of this: we cannot reify the semiformal

$$\forall \phi{:}o. \forall x{:}v. \, x \text{ not free in context} \to T(\phi) \to T(all\ x\ \phi) \qquad (18)$$

into a formal statement (we cannot, in general, refer to contexts in this way).

Again there are ways to get around the problem by complicating the formalization. For instance while we cannot tell which variables are free in the context, we can, under certain circumstances, keep track of those that might be. We can define a new type sv, of sets of variables, with associated algebra and predicates, where, e.g., $notin(x, c)$ is a predicate on types v and sv that is true iff x does not occur in the set c, and $union$ is a function of type $sv \to sv \to sv$ that returns the union of its arguments. In this setting we can then expand $T(\cdot)$ so that rather than being a predicate on o, it is a predicate on o and sv; this yields the formalization

$$\forall \phi{:}o.\, \forall x{:}v.\, \forall c{:}sv.\; notin(x, c) \to T(\phi, c) \to T(all\ x\ \phi, c)\,.$$

In addition, we have, of course, to modify all the other rules so they keep track of the variables that might be added to the context; for instance $\to$-I now has the formalization

$$\forall \phi, \psi{:}o. \forall c{:}sv.\, (T(\phi, c) \to T(\psi, union(c, fv(\phi)))) \to T(imp\, \phi\, \psi, c)\,,$$

where *fv* returns the set of variables free in a formula (i.e. for ψ we have added all the free variables of ϕ to the free variables in the context).

Clearly, first-order syntax in combination with $\to$ is becoming unacceptably complicated at this point, and is far removed from the 'sketched' characterization in (14). If part of the motivation of a logical framework is to provide a high-level abstraction of a deductive system, which we can then use to implement particular systems, then, since substitution and binding are standard requirements, we might expect them to be built into the framework, rather than encoded from scratch each time we need them. We thus now examine a very different sort of encoding in terms of $\lambda^{\to}$ that does precisely this.

Higher-order syntax

When using a first-order encoding of syntax, each time we apply a proof-rule like $\forall$-E, we must construct a subproof about substitution. But in $\lambda^{\to}$ we already have a substitution mechanism available that we can exploit if we formalize variable binding operators a bit differently. The formalization is based on what is now commonly called *higher-order* syntax.

We start by observing that in $\lambda^{\to}$ the λ operator binds variables, β-reduction provides the sort of substitution we want, and we have a built-in equivalence that accounts for renaming of bound variables. Higher-order syntax is a way of exploiting this, using higher-order functions to formalize the variable binding operators of an encoded logic directly, avoiding the complications associated with a first-order encoding.

The idea is best illustrated with an example. If we return to the problem of how to encode the language of arithmetic, then, using higher-order syntax our signature need contain only the two sorts, for terms and formulae; i.e. $\mathcal{B} = \{i, o\}$.

We do not need a sort corresponding to a syntactic category of variables, because now we will represent them directly using variables of the metalogic itself, which are either declared in the context or bound by λ in the metalogic. The signature Σ is then

$$\{0{:}i, s{:}i \to i, plus{:}i \to i \to i, times{:}i \to i \to i, eq{:}i \to i \to o,\\ falsum{:}o, neg{:}o \to o, or{:}o \to o \to o, and{:}o \to o \to o,\\ imp{:}o \to o \to o, all{:}(i \to o) \to o, exists{:}(i \to o) \to o\}\,. \tag{19}$$

In this signature all and $exists$ no longer have the (first-order) type $v \to o \to o$; instead they are second order, taking as their arguments predicate valued functions (which have first-order types).

Using higher-order syntax, an operator that binds a variable can be conceptually decomposed into two parts. First, if ϕ is an encoded formula, i.e. of type o, possibly including a free metavariable x of type i, then $\lambda x^i.\,\phi$ is the abstraction of ϕ over x, where x is now bound. The result is, however, of type $i \to o$, not of type o. Second, we convert $\lambda x^i.\,\phi$ back into an object of type o, and indicate the variable binding operator we want, by applying that operator to it. For example, applying all to $\lambda x^i.\,\phi$ yields the term $all\,(\lambda x^i.\,\phi)$ of type o. Similarly, for a substitution we reverse the procedure. Given $all\,(\lambda x^i.\,\phi)$, for which we want to generate the substitution instance $\phi[x \leftarrow t]$, we first strip off the operator all and apply (in $\lambda^{\to}$) the result to t, to get $(\lambda x^i.\,\phi)t$. But in $\lambda^{\to}$ this reduces to $\phi[x \leftarrow t]$. Hence, we needn't formalize explicitly any substitution mechanism for the object logic since we can exploit the substitution that is (already) formalized for the metalogic.

Of course we must check that all of this works. But it is easy to extend adequacy (Theorem 4.6) for this signature to show that the terms and formulae of first-order arithmetic are correctly represented by normal form members of types i and o respectively.

Now we can formalize $\forall$-E and $\forall$-I in a way that directly reflects the sketch in (14):

$$\forall\phi^{i\to o}.\,(T(all\,\phi) \to \forall x^i.\,T(\phi\ x))\\ \forall\phi^{i\to o}.\,((\forall x^i.\,T(\phi\ x)) \to T(all\,\phi)) \tag{20}$$

If we compare this with (18) above, we can see how, by using metavariables as object variables, we are able to formalize the side condition 'x not free in context' in (20) by having x bound directly by a universal quantifier at the metalevel.

In conclusion, higher-order syntax provides strong supporting evidence for Thesis 4.1 by providing a mechanism for using the λ of the metatheory to provide directly the machinery needed for variable binding, substitution, variable renaming, and the like, which are typically needed for representing and using object logics that contain variable binding operators.

4.3 Rules of proof and dependent types

We have shown how we can use a type-theory to represent syntax, reducing the problem of syntactic well-formedness to the problem of decidable well-typing. We now extend the language we are developing so that we can do the same with proof-rules.

We can use $\lambda^{\to}$ as a representation language for proofs too, but it is too weak to reduce proof checking to type checking alone. To see why, consider the two function symbols $times$ and $plus$ in the signature defined in (19). Both are of type $i \to i \to i$, which means that as far as the typing enforced by $\lambda^{\to}$ is concerned, they are interchangeable; i.e., if t is a well-typed term in $\lambda^{\to}$, and we replace every occurrence of the constant $times$ with the constant $plus$, we get a well-typed term t'. If t is supposed to represent a piece of syntax, this is what we want; for instance if we have used the type-checking of $\lambda^{\to}$ to show that $t \equiv eq\,(times\;0\;(s0))\;0$ is a well-formed formula, i.e. that $t{:}o$, then we immediately know that t' is a well-formed formula too. Unfortunately, what is useful for encoding syntax makes it impossible to define a type of proofs: in arithmetic we want t, but not t', to be provable, but we cannot make this distinction in $\lambda^{\to}$: we cannot define a type pr such that $a{:}pr$ iff a is the encoding of a proof, since we would not be able to tell whether a ‘proof’ is of t or of t'.

This observation may seem at odds with the relationship between $\mathrm{NJ}^{\to}$ and $\lambda^{\to}$ established in §3, since we have already used $\mathrm{NJ}^{\to}$ to encode (propositional) proofs. But in our discussion of the Curry-Howard isomorphism, we were careful to talk about the *propositional fragment* of $\mathrm{NJ}^{\to}$, but in order to encode even propositional proofs in $\mathrm{NJ}^{\to}$ we have used the language of *quantifier-free predicate* logic, not propositional logic, and in order to encode the rules for quantifiers, we needed explicit quantification.

To represent proofs we proceed by extending $\lambda^{\to}$ with *dependent* types, that is, with types that can depend on terms. Specifically, we introduce a new operator, Π, where, if A is a type, and for every $t \in A$, $B[x \leftarrow t]$ is a type, then so is $\Pi x^{A}.\,B$. In other words, we use Π to build families of types, $B[x \leftarrow t]$, indexed by A. Π is sometimes called a dependent function space constructor because its members are functions f where, for every $t \in A$, $f(t)$ belongs to the type $B[x \leftarrow t]$. The addition of dependent types generalizes $\lambda^{\to}$ since when x does not occur free in B, the type $\Pi x^{A}.\,B$ is simply $A \to B$ because its members are just the functions from A to B that we have in $\lambda^{\to}$.

Given dependent function types, we can define the provability relation for a logic as a type-valued function: instead of having pr be a single type, we index it over the formulae it might prove, i.e. we define it to be a function from objects ϕ of type o (i.e. formulae) to the type of proofs of ϕ. Using this, we define typed function constants that correspond to rules of proof. For example, we can now formalize implication elimination as

$$impe{:}\Pi x^o.\, \Pi y^o.\, pr(imp\ x\ y) \to pr(x) \to pr(y)$$

i.e., $impe$ is a function which, given formulae x and y (objects of type o), and terms proving $x \supset y$ and x, returns a term proving y.

We now provide the formal details of an extension of $\lambda^{\to}$ in which we can build dependent types, and show that the approach to representing deductive systems using type systems actually works the way we want.

The metalogic λ^{P}

The particular theory we present, which we call λ^{P}, is closely related to the Edinburgh LF type-theory [Harper *et al.*, 1993] and the Π fragment of the AUTOMATH language AUT-PI [de Bruijn, 1980]. Our presentation is based on a similar presentation by Barendregt [1991; 1992], which we have chosen for its relative simplicity.

We define the expressions and types of λ^{P} together as follows:

DEFINITION 4.7 (Pseudo-Terms) *Let $\mathcal{V}$ be an infinite set of variables and $\mathcal{K}$ be a set of constants that contains at least two elements, $*$ and $\Box$, which are called* sorts. *A set of* pseudo-terms *$\mathcal{T}$ is described by the following grammar*

$$\mathcal{T} ::= \mathcal{V} \mid \mathcal{K} \mid \mathcal{T}\mathcal{T} \mid \lambda\mathcal{V}^{\mathcal{T}}.\,\mathcal{T} \mid \Pi\mathcal{V}^{\mathcal{T}}.\,\mathcal{T}$$

Π binds variables exactly like λ. Substitution (respecting bound variables) and bound variable renaming are defined in the standard manner.

DEFINITION 4.8 (A deductive system for λ^{P}) *We define, together, judgments for a* valid signature*, a* valid context *and a* valid typing*. In the following, s ranges over $\{*, \Box\}$:*

- *A* signature *is a sequence given by the grammar*

  $$\Sigma ::= \langle\,\rangle \mid \Sigma, c{:}A$$

  where c ranges over $\mathcal{K}$. A signature Σ is valid *when it satisfies the relation $\vdash_s$ defined by:*

  $$\frac{}{\vdash_s \langle\,\rangle} \qquad \frac{\vdash_s \Sigma \quad \vdash_\Sigma A{:}s \quad c \notin dom(\Sigma)}{\vdash_s \Sigma, c{:}A}$$

- *A* context *is a sequence given by the grammar*

  $$\Gamma ::= \langle\,\rangle \mid \Gamma, x{:}A$$

  where x ranges over $\mathcal{V}$. A context is valid *with respect to a valid signature Σ when it satisfies the relation $\vdash_c$ defined by:*

  $$\frac{}{\vdash_c \langle\,\rangle} \qquad \frac{\vdash_c \Gamma \quad \Gamma \vdash_\Sigma A{:}s \quad x \notin dom(\Gamma)}{\vdash_c \Gamma, x{:}A}$$

- *A* type assignment relation $\vdash_\Sigma$, *indexed by a valid signature* Σ, *is defined between valid typing contexts* Γ *and typing assertions* $a{:}B$ *where* $a, B \in \mathcal{T}$ *is given by the rules:*

$$\frac{}{\Gamma \vdash_\Sigma *{:}\Box}\ axiom \qquad \frac{\Gamma \vdash_\Sigma A{:}* \quad \Gamma, x{:}A \vdash_\Sigma B{:}s}{\Gamma \vdash_\Sigma \Pi x^A.\, B{:}s}\ form$$

$$\frac{c{:}A \in \Sigma}{\Gamma \vdash_\Sigma c{:}A}\ assum \qquad \frac{\Gamma, x{:}A \vdash_\Sigma b{:}B \quad \Gamma \vdash_\Sigma \Pi x^A.\, B{:}s}{\Gamma \vdash_\Sigma \lambda x^A.\, b{:}\Pi x^A.\, B}\ abst$$

$$\frac{x{:}A \in \Gamma}{\Gamma \vdash_\Sigma x{:}A}\ hyp \qquad \frac{\Gamma \vdash_\Sigma f{:}\Pi x^A.\, B \quad \Gamma \vdash_\Sigma a{:}A}{\Gamma \vdash_\Sigma f(a){:}B[x \leftarrow a]}\ appl$$

$$\frac{\Gamma \vdash_\Sigma a{:}B \quad \Gamma \vdash_\Sigma B'{:}s \quad B =_\beta B'}{\Gamma \vdash_\Sigma a{:}B'}\ conv$$

We use the two sorts $*$ and $\Box$ to classify entities in $\mathcal{T}$ into levels. We say that $*$ is the set of types and $\Box$ is the set of kinds. As in $\lambda^{\rightarrow}$, terms, which are here a subset of the pseudo-terms, belong to types; unlike in $\lambda^{\rightarrow}$, types, which are here also pseudo-terms, belong to $*$. For example, if o is a type (i.e. $o{:}*$), then $\lambda x^o.\, x$ is a term of type $\Pi x^o.\, o$, which we can abbreviate as $o \rightarrow o$, since x does not occur free in o. It is possible to build kinds, in limited ways, using the constant $*$; in particular, the rules we give allow the formation of kinds with range $*$, e.g. $o \rightarrow *$ but exclude kinds with domain $*$, e.g. $* \rightarrow o$. Hence we can form kinds like $o \rightarrow *$ that have type-valued functions as members, but we cannot form kinds by quantifying over the set of types.

We state without proof a number of facts about this system. They have been proven in the more general setting of the so-called λ-cube (a family of eight related type systems) and generalized type systems examined by Barendregt [1991; 1992], but see also Harper *et al.* [1993] who show that the closely related LF type-theory has similar properties.

FACT 4.9 *Term reduction in* λ^{P} *is Church-Rosser: given* $A, B, B' \in \mathcal{T}$, *then if* $A \xrightarrow{*}_\beta B$ *and* $A \xrightarrow{*}_\beta B'$ *there exists* $C \in \mathcal{T}$ *where both* $B \xrightarrow{*}_\beta C$ *and* $B' \xrightarrow{*}_\beta C$.

FACT 4.10 *Terms in* λ^{P} *are strongly normalizing: if* $\Gamma \vdash_\Sigma A{:}B$, *then* A *and* B *are strongly normalizing (all* β*-reductions starting with* A *or* B *terminate).*

FACT 4.11 λ^{P} *satisfies unicity of types: if* $\Gamma \vdash_\Sigma A{:}B$ *and* $\Gamma \vdash_\Sigma A{:}B'$, *then* $B =_\beta B'$.

From the decidability of these operations it follows that:

FACT 4.12 *All judgments of* λ^{P} *are decidable.*

Relationship to other metalogics

The proof-rules for λ^{P} extend the rules given for $\lambda^{\to}$ in Definition 4.2. The rules for $\lambda^{\to}$ essentially correspond to the identically named rules for λ^{P} restricted so that in every typing assertion $a{:}B$, a is a term of $\lambda^{\to}$ and B is a simple type. The correspondence is not quite exact since in λ^{P} we have to prove that a signature, context, and type are well-formed (i.e., the first three parts of Definition 4.2), whereas this is assumed to hold in $\lambda^{\to}$. The need for this explicit demonstration of well-formedness is also reflected in the second premise of the λ^{P} rule for abstraction.

An example should clarify the connection between the two systems. In §4.2 we gave a signature Σ for minimal logic

$$\{\, imp{:}\ o \to o \to o \,\}\,.$$

In λ^{P} we have

$$\Sigma = o{:}{*}, imp{:}o \to o \to o\,.$$

According to this, if x is of type o, then we can show in λ^{P} that $imp\ x\ x$ is a well-formed proposition, i.e., $\Gamma \vdash_{\Sigma} imp\ x\ x{:}o$ where $\Gamma = x{:}o$, as follows:

$$\cfrac{\cfrac{\cfrac{imp{:}o \to o \to o \in \Sigma}{\Gamma \vdash_{\Sigma} imp{:}o \to o \to o}\ assum \qquad \cfrac{x{:}o \in \Gamma}{\Gamma \vdash_{\Sigma} x{:}o}\ hyp}{\Gamma \vdash_{\Sigma} imp\ x{:}o \to o}\ appl \qquad \cfrac{x{:}o \in \Gamma}{\Gamma \vdash_{\Sigma} x{:}o}\ hyp}{\Gamma \vdash_{\Sigma} imp\ x\ x{:}o}\ appl$$

However, the rules of λ^{P} formalize a strictly more expressive type-theory than $\lambda^{\to}$, and correspond, via a Curry-Howard isomorphism, to a more expressive logic. Terms are built, as we have already seen, by declaring function constants that form typed objects from other typed objects, e.g., $imp\ x\ x$ above corresponds to a term of type o. An n-ary predicate symbol P, which takes arguments of types $s_1, \ldots, s_n$, has the kind $s_1 \to \ldots \to s_n \to *$. The Π-type constructor corresponds either to universal quantification or (in its non-dependent form) implication. For example, given the signature

$$\Sigma = s_1{:}{*}, s_2{:}{*}, p{:}s_1 \to s_2 \to *$$

we can show that there is a t such that

$$\vdash_{\Sigma} t{:}(\Pi x^{s_1}.\, \Pi y^{s_2}.\, p(x,y)) \to \Pi y^{s_2}.\, \Pi x^{s_1}.\, p(x,y)\,,$$

which corresponds to demonstrating the provability of the formula

$$\vdash_{\Sigma} (\forall x.\, \forall y.\, p(x,y)) \to \forall y.\, \forall x.\, p(x,y)$$

in a traditional sorted first-order setting.

In λ^{P} we generalize $\lambda^{\to}$ so that types can depend on terms. We have not carried through this generalization to allow, e.g., types depending on types, which would allow impredicative higher-order quantification. As a result, and given the above discussion, logics like λ^{P} and the LF are often described as first-order. Alternatively, since we can also quantify over functions (as opposed to predicates) at all types, some authors prefer to talk about minimal implicational predicate logic with quantification over all higher types [Simpson, 1992], or ω-order logic (hh^{ω}) [Felty, 1991], to emphasize that these logics are more than first-order, but are not fully higher-order.

4.4 *Representation in* λ^{P}

We have reached the point where the intuitions we have formulated about the relationship between natural deduction calculi and the logic of implication are reduced to a single formal system, the type-theory λ^{P}. In this system, the problems of encoding the syntax and proof rules of a deductive system are reduced to the single problem of providing a signature Σ and the problems of checking well-formedness of syntax and proof checking are reduced to (decidable) type-checking. We will expand on these points with two examples.

A simple example: minimal logic

A deductive system is encoded in λ^{P} by a signature that encodes

1. The language of the object logic and
2. The deductive system.

In §4.3 we gave a signature suitable for encoding the language of minimal logic. As we have seen, this consists first of an extension of the signature with types corresponding to syntactic categories and then with function constants over these types. The encoding of the deductive system also proceeds in two stages. First, we represent the basic judgments of the object logic.[15] To do this, for each judgment we augment the signature with a function from the relevant syntactic categories to a type. For minimal logic we have one judgment, that a formula is provable, so we add to the signature a function pr, of kind $o \to *$, where for any proposition $p \in o$, $pr(p)$ should be read as saying that the formula represented by p is provable. Second, we add constants to the signature that build (representatives of) proofs. Each

[15] Recall that *judgments* are assertions such as, e.g., that a proposition is provable. Typically, a logic only has a single judgment, but not always; for instance λ^{P} itself, in our presentation, has three judgments: a signature is well-formed, a context is well-formed, and a typing assertion is provable relative to a well-formed signature and context. The reader should be aware of the following possible source of confusion. By using a metalogic we have judgments at two levels: we use the judgment in λ^{P} that a typing assertion is provable relative to a signature and a context to demonstrate the truth of judgments in some object logic.

constant is associated with a type that encodes (under the propositions-as-types correspondence) a proof rule of the object logic. For minimal logic we add constants with types that encode the formulae given in (6) from §3.2, which axiomatize the rules for minimal logic.

Putting the above pieces together, minimal logic is formalized by the following signature.

$$\begin{aligned}\Sigma \equiv o{:}*, pr{:}o \to *, imp{:}o \to o \to o,\\ impi{:}\Pi A^o B^o.\,(pr(A) \to pr(B)) \to pr(imp\ A\ B),\\ impe{:}\Pi A^o B^o.\,pr(imp\ A\ B) \to pr(A) \to pr(B)\end{aligned}$$

It is an easy exercise to prove in λ^P that this is a well-formed signature.

Now consider how we use this signature to prove the proposition $A \supset A$. We encode this as $imp\ A\ A$ and prove it by showing that the judgment $pr(imp\ A\ A)$ has a member. We require, of course, that A is a proposition and we formalize this by the context $A{:}o$, which is well-formed relative to Σ. (In the following proof we have omitted rule names, but these can be easily reconstructed.)

Part I:

$$\dfrac{\dfrac{\dfrac{impi{:}\Pi A^o B^o.\,(pr(A) \to pr(B)) \to pr(imp\,A\,B) \in \Sigma}{A{:}o \vdash_\Sigma impi{:}\Pi A^o B^o.\,(pr(A) \to pr(B)) \to pr(imp\,A\,B)} \quad \dfrac{A{:}o \in A{:}o}{A{:}o \vdash_\Sigma A{:}o}}{A{:}o \vdash_\Sigma impi\ A{:}\Pi B^o.\,(pr(A) \to pr(B)) \to pr(imp\,A\,B)} \quad \dfrac{A{:}o \in A{:}o}{A{:}o \vdash_\Sigma A{:}o}}{A{:}o \vdash_\Sigma impi\ A\,A{:}(pr(A) \to pr(A)) \to pr(imp\,A\,A)}$$

Part II:

$$\dfrac{\dfrac{y{:}pr(A) \in A{:}o, y{:}pr(A)}{A{:}o, y{:}pr(A) \vdash_\Sigma y{:}pr(A)} \quad \dfrac{\cdots}{A{:}o \vdash_\Sigma pr(A) \to pr(A){:}*}}{A{:}o \vdash_\Sigma \lambda y^{pr(A)}.\,y{:}pr(A) \to pr(A)}$$

(We have elided the subproof showing that $pr(A) \to pr(A)$ is well-formed, which is straightforward using the formation rule $form$.) Putting the two parts together gives:

$$\dfrac{\text{Part I} \qquad \text{Part II}}{A{:}o \vdash_\Sigma impi\ A\,A\,(\lambda y^{pr(A)}.\,y){:}pr(imp\,A\,A)}$$

Note that the reader interested in actually using a metalogic for machine supported proof construction should not be frightened away by the substantial 'metalevel overhead' that is associated with carrying out a proof of even very simple propositions like $A \supset A$. Real implementations of logical frameworks can hide much of this detail by partially automating the work of proof construction. Because all the judgments of λ^P are decidable, the well-formedness of signatures and contexts can be checked automatically, as can the typing of the terms that encode proofs.[16]

This example shows how both well-formedness and proof checking are uniformly reduced to type checking. With respect to well-formedness of syntax, a proof that the type $pr(imp\,A\,A)$ is inhabited is only possible if $imp\,A\,A$ is of type o, i.e. it represents a well-formed proposition. That members of type o really represent well-formed propositions follows from adequacy and faithfulness of the representation of syntax, which for this example was argued (for $\lambda^{\rightarrow}$) in §4.2. With respect to proof checking, we have proven that the term $impi\,A\,A\,(\lambda y^{pr(A)}.\,y)$ inhabits the type $pr(imp\,A\,A)$. In the same way that a term of type o represents a proposition, a term of type $pr(p)$ represents a proof of p. In this example, the term represents the following natural deduction proof of $A \supset A$.

$$\dfrac{[A]_y}{A \supset A}\supset\text{-}I_y$$

The exact correspondence (adequacy and faithfulness) between terms and the proofs that they encode can be formalized (see Harper *et al.* [1993] for details), though we do not do this here since it requires first formalizing natural deduction proof trees and the representation of discharge functions. The idea is simple enough though: the proof rule $\supset$-I is encoded using a constant $impi$, and a well-typed application of this constructs a proof-term formalizing the operation of discharging an assumption. Specifically, $impi$ builds an object (proof representative) of the type (proposition) $pr(imp\ A\ B)$ given an object of type $pr(A) \rightarrow pr(B)$, i.e. a proof that can take any object of proof $pr(A)$ (the hypothesis), and from it produce an object of type $pr(B)$.

In the example above the function must construct a proof of $pr(A)$ from $pr(A)$, and $\lambda y^{pr(A)}.\,y$ does this. In general, the question of which occurrences of $pr(A)$ are discharged and how the proof of B is built is considerably more complex. Consider for example

$$impi\,A\,(imp\,B\,A)\,(\lambda x^{pr(A)}.\,impi\,B\,A\,(\lambda y^{pr(B)}.\,x))\,, \tag{21}$$

which is a member of the type $pr(imp\,A\,(imp\,B\,A))$ in a context where $A{:}o$ and $B{:}o$. This term represents a proof where Implication Introduction has been applied twice and the first (reading left to right) application discharges an assumption x and the second discharge (of y) is vacuous. This proof-term corresponds to the following natural deduction proof.

$$\dfrac{\dfrac{[A]_x}{B \supset A}\supset\text{-}I_y}{A \supset (B \supset A)}\supset\text{-}I_x \tag{22}$$

[16]This second point is not so important: Although the decidability of syntactic well-formedness is important, in practice, a framework is not used to decide if a given proof is valid, but as an interactive tool for building proofs.

$$\begin{aligned}
oril&:\Pi A^o B^o.\, pr(A) \to pr(or\ A\, B)\\
orir&:\Pi A^o B^o.\, pr(B) \to pr(or\ A\, B)\\
ore&:\Pi A^o B^o C^o.\, pr(or\ A\, B) \to (pr(A) \to pr(C))\\
&\qquad\qquad \to (pr(B) \to pr(C)) \to pr(C)\\
raa&:\Pi A^o.\, (pr(imp\ A\ falsum) \to pr(falsum)) \to pr(A)\\
alli&:\Pi A^{i\to o}.\, (\Pi x^i.\, pr(A(x))) \to pr(all(A))\\
alle&:\Pi A^{i\to o} x^i.\, pr(all(A)) \to pr(A(x))\\
existsi&:\Pi A^{i\to o}.\, \Pi x^i.\, pr(A(x)) \to pr(exists(A))\\
existse&:\Pi A^{i\to o}.\, \Pi C^o.\, pr(exists(A)) \to (\Pi x^i.\, pr(A(x)) \to pr(C)) \to pr(C)\\
ind&:\Pi A^{i\to o}.\, pr(A(0)) \to (\Pi x^i.\, pr(A(x)) \to pr(A(sx))) \to pr(all(A))
\end{aligned}$$

Figure 1. Some proof-rules for arithmetic

A larger example: first-order arithmetic

A more complex example of a theory that we can easily formalize in λ^{P} is first-order arithmetic, and in fact we can define this as a direct extension of the system we have already formalized. We extend the signature with the formalization of the syntax of arithmetic that we developed in §4.2 then we formalize the new rules, axioms and axiom-schemas that we need.

We have formalized some of the proof-rules in Figure 1 and most are self-explanatory. The first five extend minimal logic to propositional logic by adding rules for disjunction and falsum. We use the constant $falsum$ to encode $\perp$ (from which we can define negation as $not\ A \equiv imp\ A\ falsum$).

In our rules we assume a 'classical' falsum, i.e. the rule:

$$\dfrac{\begin{matrix}[A \supset \perp]\\ \vdots\\ \perp\end{matrix}}{A}\ \perp_c$$

encoded as raa. If we wanted an 'intuitionistic' falsum, we would replace this with the simpler rule encoded by

$$\Pi A^o.\, pr(falsum) \to pr(A).$$

For the quantifier rules we have not only given the rules for universal quantification ($alli$ and $alle$) but also the rules for existential quantification, given by

$existsi$ and $existse$.

$$
\dfrac{A[x \leftarrow t]}{\exists x.\, A}\ \exists\text{-}I
\qquad
\dfrac{\exists x.\, A \quad \begin{array}{c}[A]\\ \vdots \\ C\end{array}}{C}\ \exists\text{-}E
$$

These come with the usual side conditions: in $\forall$-I, x cannot be free in any undischarged assumptions on which A depends and, for $\exists$-E, x cannot be free in C or any assumptions other than A upon which (in the subderivation) C depends.

If we stop with the quantifier rules, the result is an encoding of first-order logic over the language of arithmetic. We have to add more rules to formalize the theory of equality and arithmetic. Thus, for example, ind formalizes the induction rule

$$
\dfrac{A[x \leftarrow 0] \quad \begin{array}{c}[A]\\ \vdots \\ A[x \leftarrow sx]\end{array}}{\forall x.\, A}
$$

and enforces the side condition that x does not occur free in any assumptions other than those discharged by the application of the rule. The other rules of arithmetic are formalized in a similar fashion.

4.5 Summary

Figure 2 contains a summary. It is worth emphasizing that there is a relationship between the metalogic and the way that it is used, and an $\rightarrow$-framework like λ^P is well-suited to particular kinds of encodings. The idea behind higher-order syntax and the formalization of judgments using types is to internalize within the metalogic as much of the structure of terms and proofs as possible. By this we mean that syntactic notions and operations are subsumed by operations provided by the framework logic. In the case of syntax, we have seen how variable binding in the object logic is implemented by λ-abstraction in the framework logic and how substitution is implemented by β-reduction. Similarly, when representing proof-rules and proof-terms, rather than our having to formalize, and then reason explicitly about, assumptions and their discharging, this is also captured directly in the metalogic. Support for this sort of internalization is one of the principles behind the design of these framework logics. The alternative (also possible in λ^P) is to *externalize*, i.e., explicitly represent, such entities. The external approach is taken when using frameworks based on inductive definitions, which we will consider in §7.

Object Logic		Metalogic
Syntactic Categories terms, individuals	$\leadsto$	Base Types $\{i{:}*, o{:}*\}$
Connectives & Constructors $\vee$	$\leadsto$	First-Order Constants $or{:}o \to o \to o$
Variable Binding Operators $\forall$	$\leadsto$	Higher-Order Constants $all{:}(i \to o) \to o$
Judgment $\vdash p$	$\leadsto$	Type Valued Functions $pr{:}o \to *$
Inference Rule $\dfrac{A}{A \vee B}\ \vee\text{-}IL$	$\leadsto$	Constant Declaration $oril{:}\Pi A^o B^o.\, pr(A) \to pr(or\ A\ B)$
Deductive System		Signature Declaration
Deduction		Typing Proof

Figure 2. Correspondence between object logics and their encodings

5 ENCODING LESS WELL BEHAVED LOGICS

So far, we have restricted our attention to fairly standard, e.g. intuitionistic or classical, logics. We now consider how an $\to$-framework can treat the more 'unconventional' logics that we encounter in, for example, philosophy or artificial intelligence, for which such simple calculi are not available. As previously observed, most metalogics (and all the examples examined in this chapter) are 'universal' in the sense that they can represent any recursively enumerable relation, and thus any logic expressible in terms of such relations. However, there is still the question of how effective and natural the resulting encodings are.

We take as our example one of the more common kinds of philosophical logics: modal logic, i.e. propositional logic extended with the unary $\Box$ connective and the *necessitation* rule (see Bull and Segerberg [1984]). Modal logics, as a group, have common features; for example, 'canonical' presentations use Hilbert calculi and, when natural deduction presentations are known, the proof-rules typically are not encodable in terms of the straightforward translation presented in §3.2. In §7 we will see how Hilbert presentations of these logics can be directly encoded as inductive definitions. Here we consider the problem of developing natural deduction presentations in an $\to$-framework. We explore two different possibilities, *labelled deductive systems* and *multiple judgment systems*, consider how practical they are, and how they compare.

5.1 Modal logic

We consider two modal logics in this section: K and an important extension, $\mathrm{S4}$. A standard presentation of K is as a Hilbert system given by the axiom schemata

$$(A \supset B) \supset (A \supset B \supset C) \supset A \supset C$$
$$A \supset B \supset A$$
$$((A \supset \bot) \supset \bot) \supset A$$
$$\Box(A \supset B) \supset \Box A \supset \Box B$$

and the rules

$$\frac{A \supset B \quad A}{B}\, Det \qquad \text{and} \qquad \frac{A}{\Box A}\, Nec$$

The first of these rules is just the rule of detachment from §2.1, and the second is called necessitation. We get S4 from K by adding the additional axiom schemata:

$$\Box A \supset \Box\Box A$$
$$\Box A \supset A$$

As noted above, we can see a Hilbert calculus as a special case of a natural deduction calculus, where the rules discharge no assumptions and axioms are premisless rules.

There is an important difference between Hilbert and natural deduction calculi however, which is in the nature of what they reason about: Hilbert calculi manipulate formulae that are true in all contexts, i.e. valid (theorems), in contrast to natural deduction calculi, which typically manipulate formulae that are true under assumption. This difference causes problems when we try to give natural deduction-like presentations of modal logics, i.e. presentations that allow reasoning under temporary assumptions. The problem can be easily summarized:

PROPOSITION 5.1 *The deduction theorem (see §7.3) fails for* K *and* S4.

Proof. First, observe (e.g., semantically using Kripke structures; see §5.2) that $A \supset \Box A$ is not provable in K or $\mathrm{S4}$. However, if the deduction theorem held, we could derive this formula as follows: assume A, then, by necessitation, we have $\Box A$, and by the deduction theorem we would have that $A \supset \Box A$ is a theorem. This is a contradiction. ∎

The deduction theorem is a justification for the natural deduction rule $\supset$-I, but this in turn is precisely the rule that distinguishes natural deduction-like from Hilbert calculi: without it, one collapses into the other.

The problem of natural deduction encodings of modal logics is well known, and various fixes have been proposed. In some of these, the rules $\supset$-I and $\supset$-E

are kept intact by extending the language of natural deduction itself. For instance if we allow global side conditions on rules then (following Prawitz [1965]) for S4 we have the rules

$$\dfrac{\begin{array}{c}\vdots\, * \\ A\end{array}}{\Box A}\;\Box\text{-}I \qquad \text{and} \qquad \dfrac{\Box A}{A}\;\Box\text{-}E$$

where $*$ means that *all undischarged assumptions are boxed; i.e. of the form* $\Box B$. Notice that given this side condition, the argument we have used to illustrate the failure of the deduction theorem no longer works. But the language of $\rightarrow$ does not provide the vocabulary to express this side condition on $\Box$-I, so we cannot encode such a proof rule in the same fashion as proof-rules were encoded in §3.2.

5.2 A Kripke semantics for modal logics

A common way of understanding the meaning of formulae in a modal logic is in terms of the *Kripke*, or *possible worlds* semantics (see Kripke [1963] or van Benthem [1984] for details). We shall use this style of interpretation in developing our encodings.

A Kripke model (W, R, V) for a modal logic consists of a nonempty set of *worlds* W, a binary *accessibility* relation R defined over W, and a *valuation* predicate V over W and the propositional variables. We then define a *forcing* relation $\Vdash$ between worlds and formulae as follows: $a \Vdash A$ iff $V(a, A)$ for A atomic; $a \Vdash A \supset B$ iff $a \Vdash A$ implies $a \Vdash B$; and $a \Vdash \Box A$ iff for all $b \in W$ if $a\ R\ b$ then $b \Vdash A$.

Using the Kripke semantics, we can classify modal logics by the behavior of R alone. For instance we have

FACT 5.2 *Let R be the accessibility relation of a Kripke model.*

- *A formula A is a theorem of* K *iff A is forced at all worlds of all Kripke models.*
- *A formula A is a theorem of* S4 *iff A is forced at all worlds of all Kripke models where R is reflexive ($x\ R\ x$) and transitive (if $x\ R\ y$ and $y\ R\ z$, then $x\ R\ z$).*

It is now possible to see why the deduction theorem fails. Consider a Kripke model (W, R, V) and a formula $A \supset B$. In the deduction theorem we assume, for the sake of argument, A as a new axiom, and show that B is then a theorem; i.e. assuming $\forall a \in W.\, a \Vdash A$, we show that $\forall a \in W.\, a \Vdash B$. But it does not follow from this that $A \supset B$ is a theorem; i.e. that $\forall a \in W.\, a \Vdash A \supset B$.

It is however easy to find a correct 'semantic' analogue of the deduction theorem:

FACT 5.3 *For any Kripke model* (W, R, V)

$$\forall a \in W. \, (a \Vdash A \to a \Vdash B) \to a \Vdash A \supset B \, .$$

The problem of providing a natural deduction encoding of a modal logic can be reduced to the problem of capturing this semantic property of $\supset$ in rules that can be directly encoded in the language of implication. We will consider two ways of doing this, which differ in the extent to which they make the Kripke semantics explicit.

5.3 *Labelled deductive systems*[17]

The above analysis suggests one possible solution to our problem: we can internalize the semantics into the deductive calculus. Hence, instead of reasoning with formulae, we reason about formulae in worlds; i.e. we work with pairs $a{:}A$ where a is a world and A is a formula.

Taking this approach, the rules

$$\dfrac{\begin{array}{c}[x{:}A]\\ \vdots \\ x{:}B\end{array}}{x{:}A \supset B}\ \supset\text{-}I \qquad \dfrac{\begin{array}{c}[x \; R \; y]\\ \vdots \\ y{:}A\end{array}}{x{:}\Box A}\ \Box\text{-}I$$

$$\dfrac{\begin{array}{c}[x{:}A \supset \bot]\\ \vdots \\ y{:}\bot\end{array}}{x{:}A}\ \bot\text{-}E \qquad \dfrac{x{:}A \supset B \quad x{:}A}{x{:}B}\ \supset\text{-}E \qquad \dfrac{x{:}\Box A \quad x \; R \; y}{y{:}A}\ \Box\text{-}E$$

define a natural deduction calculus, which we call K_L. (We also require the side conditions that in $\Box$-I y is different from x and does not occur in the assumptions on which $y{:}A$ depends, except those of the form $x \; R \; y$ that are discharged by the inference.) These rules formalize the meaning of both $\supset$ and $\Box$ in terms of the Kripke semantics; i.e., we locate applications of $\supset$-I in some particular world, and take account of the other worlds in defining the behavior of $\Box$ and $\bot$ (where it suffices to derive a contradiction in any world). We can show:

FACT 5.4 (Basin *et al.* [1997a]) $a{:}A$ *is provable in* K_L *iff* A *is true in all Kripke models, and therefore, by the completeness of* K *with respect to the set of all Kripke models, iff* A *is a theorem of* K.

As an example of a proof in K_L of a K theorem, we show that $\Box$ distributes

[17] The work described in this section was done in collaboration with Luca Viganò.

over $\supset$.

$$\dfrac{\dfrac{\dfrac{\dfrac{[a{:}\Box(A \supset B)]_1 \quad [a\ R\ b]_3}{b{:}A \supset B}\ \Box\text{-}E \qquad \dfrac{[a{:}\Box A]_2 \quad [a\ R\ b]_3}{b{:}A}\ \Box\text{-}E}{\dfrac{b{:}B}{a{:}\Box B}\ \Box\text{-}I_3}\ \supset\text{-}E}{a{:}\Box A \supset \Box B}\ \supset\text{-}I_2}{a{:}\Box(A \supset B) \supset \Box A \supset \Box B}\ \supset\text{-}I_1$$

Further, and essential for our purpose, there are no new kinds of side conditions on the rules of K_L, so we have no difficulty in formalizing these in an $\rightarrow$-framework.[18] The following is a signature for K_L in λP (note that for the sake of readability we write ':' and R in infix form):

$$\begin{aligned}\Sigma_{\mathrm{K}_L} \equiv\ & w{:}*,\ o{:}*,\ \text{`:'}{:}w \rightarrow o \rightarrow *,\ R{:}w \rightarrow w \rightarrow *,\\ & \mathit{falsum}{:}o,\ \mathit{imp}{:}o \rightarrow o \rightarrow o,\ \mathit{box}{:}o \rightarrow o,\\ & \mathit{FalseE}{:}\Pi A^o x^w y^w.\,(x{:}\mathit{imp}\ A\ \mathit{falsum} \rightarrow y{:}\mathit{falsum}) \rightarrow x{:}A,\\ & \mathit{impI}{:}\Pi A^o B^o x^w.\,(x{:}A \rightarrow x{:}B) \rightarrow x{:}\mathit{imp}\ A\ B,\\ & \mathit{impE}{:}\Pi A^o B^o x^w.\,x{:}A \rightarrow x{:}\mathit{imp}\ A\ B \rightarrow x{:}B,\\ & \mathit{boxI}{:}\Pi A^o x^w.\,(\Pi y^w.\,x\ R\ y \rightarrow y{:}A) \rightarrow x{:}\mathit{box}\ A,\\ & \mathit{boxE}{:}\Pi A^o x^w y^w.\,x{:}\mathit{box}\ A \rightarrow x\ R\ y \rightarrow y{:}A\end{aligned}$$

The signature reflects that there are two types, a type of worlds w and type of formulae o, and two judgments; one about the relationship between worlds and formulae, asserting that a formula is true at that world, and a second, between two worlds, asserting that the first accesses the second. Adequacy and faithfulness follow by the style of analysis given in §3.

K_L *as a base for other modal logics*

We can now take K_L as a base upon which to formalize other modal logics. Since modal logics are characterized, in terms of Kripke models, purely in terms of their accessibility relations, to get other modal logics we must simply modify the behavior of R in our encoding. Thus, since S4 corresponds to the class of Kripke models with transitive and reflexive accessibility relations, we can enrich our signature with:

$$\begin{aligned}& \mathit{Ref}{:}\Pi x^w.\,x\ R\ x\\ & \mathit{Trans}{:}\Pi x^w y^w z^w.\,x\ R\ y \rightarrow y\ R\ z \rightarrow x\ R\ z\end{aligned}$$

[18] There is of course the side condition on $\Box$-I. But this can be formalized in the same way that eigenvariable conditions are formalized in logics with quantifiers, by using universal quantification in the metalogic.

Again, we can show [Basin *et al.*, 1997a] that this really formalizes S4.

The limits of K_L

It might appear from the discussion above that we can implement any modal logic we want, simply by adding the axioms for the appropriate accessibility relation to K_L. That is, we represent a logic by embedding its semantics in the metalogic, a formalization technique that is sometimes called *semantic embedding* (see van Benthem [1984] or Ohlbach [1993] for details on this approach). We must be careful though; not every embedding based on labelling accurately captures the semantics and different kinds of embeddings capture more structure than others.

Consider K_L again: the rules for $\supset$ and $\Box$ reflect the meaning that the Kripke semantics gives the connectives. On the other hand, the semantics does not explicitly state how the rules for $\perp$ should function using labels. Following from the rules for $\supset$, a plausible formalization is

$$\frac{\begin{array}{c}[x{:}A \supset \perp]\\ \vdots \\ x{:}\perp\end{array}}{x{:}A}\ \perp\text{-}E^{-}$$

which, like the rule for implication, stays in one world and ignores the existence of different possible worlds. But if we investigate the logic that results from using this rule (instead of $\perp$-E), we find that it is not complete with respect to the Kripke semantics.

An examination of the Gentzen-style natural language characterization of $\perp$ shows where the problem lies:

> If the assumption that $A \supset \perp$ in world x is inconsistent with the interpretation, then A is true in world x.

This says nothing about where the inconsistency might be and specifically does not say that it should be in the world x itself. The role of negation in encoding the semantics of logics is subtle and we lack space to develop this topic here. We therefore restrict ourselves to a few comments; much more detail can be found in [Basin *et al.*, 1997a]. In K_L we assumed that it is enough to be able to show that the inconsistency is in some world. It turns out that this is sufficient for a large class of logics; but again this does not reflect the complete behavior of $\perp$. Some accessibility relations require a richer metalogic than one based on minimal implication and this may in turn require formalizing all of first or even higher-order logic. In such formalizations, we must take account of the possibility that the inconsistency of an assumption about a world might manifest itself as a contradiction in the theory of the accessibility relation, or *vice versa*. It is possible then to use classical first (or higher-order) logic as a metatheory to formalize the Kripke semantics in a complete way, however the result also has drawbacks. In particular, we lose structure in the proofs available in K_L. In K_L we reason in two separate systems. We can reason in just the theory of the accessibility relation and then use the results of this in the theory of the labelled propositions; however, we cannot go in the other direction, i.e. we cannot use reasoning in the theory of labelled propositions as part of an argument about the relations. This enforced separation provides extra structure that we can exploit, e.g., to bound proof search (see, e.g., [Basin *et al.*, 1997b]). And in spite of enforcing this separation, K_L is a sufficient foundation for a very large class of standard logics.[19]

5.4 Alternative multiple judgment systems

Using a labelled deductive system, we built a deductive calculus for a modal logic based on two judgments: a formula is true in a world and one world accesses another. But this is only one of many possible presentations. We now consider another possibility, using multiple judgments that distinguish between truth (in a world) and validity that is due originally to Avron *et al.* [1992] (and further developed in [Avron *et al.*, 1998]).

Validity

Starting with K, we can proceed in a Gentzen-like manner, by writing down the behavior of the logical connectives as given by the Kripke semantics. If we abbreviate 'A is true in all worlds' to $V(A)$ (A is valid; i.e. A is a theorem), then we have

$$\frac{V(A \supset B) \quad V(A)}{V(B)} \supset\text{-}E_V \qquad \text{and} \qquad \frac{V(A)}{V(\Box A)} \Box\text{-}I_V\,, \tag{23}$$

which can be easily verified against the Kripke semantics (they directly reflect the two rules Det and Nec). Since the deduction theorem fails, we do not have an introduction rule for $\supset$ in terms of V, neither do we have a rule for $\perp$.

Thus the first part of the signature for K simply records the rules (23):

$$\begin{aligned}\Sigma_1 \equiv\ & o{:}*,\ V{:}o \to *,\\ & False{:}o,\ imp{:}o \to o \to o,\ box{:}o \to o,\\ & impE_V{:}\Pi A^o B^o.\, V(A) \to V(imp\ A\ B) \to V(B),\\ & boxI_V{:}\Pi A^o.\, V(A) \to V(box\ A)\end{aligned}$$

And, as observed above, we have

[19] In [Basin *et al.*, 1997a] we show that it is sufficient to define almost all the modal logics of the so-called Geach hierarchy, which includes most of those usually of interest, i.e. K, T, $\mathrm{S4}$, $\mathrm{S5}$, etc., though not, e.g., the modal logic of provability G [Boolos, 1993].

LEMMA 5.5 *The rules encoded in* Σ_1 *are sound with respect to the standard Kripke semantics of* K*, if we interpret* $V(A)$ *as* $\forall a.\, a \Vdash A$*.*

The rest of V. The rest of the details about $V(\cdot)$ could be summarized simply by declaring all the axioms of K to be valid. In this case $V(\cdot)$ would simply encode a Hilbert presentation of K. However there is a more interesting possibility, which supports proof under assumption.

Truth in a world

As previously observed we do have a kind of semantic version of the deduction theorem relativized to any given world. We can use this to formalize more about the meaning of implication and $\perp$. If we abbreviate 'A is true in some arbitrary but fixed world c' to $T(A)$, then we have:

$$\frac{T(A \supset B) \quad T(A)}{T(B)} \supset\text{-}E_T \qquad \frac{\begin{array}{c}[T(A)]\\ \vdots \\ T(B)\end{array}}{T(A \supset B)} \supset\text{-}I_T \qquad \frac{\begin{array}{c}[T(A \supset \perp)]\\ \vdots \\ T(\perp)\end{array}}{T(A)} \perp\text{-}E_T$$

When we talked above about validity we did not have a rule for introducing implication, or reflecting the behavior of $\perp$. Similarly, when we are talking about truth in some world, we do not have a rule reflecting the behavior of $\Box$, since that is not dependent on just the one world. Thus for instance, there is no introduction (or elimination) rule for this operator. All we can say is that

$$\frac{T(\Box(A \supset B)) \quad T(\Box A)}{T(\Box B)}\, Norm_T$$

And again we can verify these rules against the semantics.

Thus the second part of the signature is

$$\begin{aligned}
\Sigma_2 \equiv\ & T{:}o \to *, \\
& impE{:}\Pi A^o B^o.\, T(imp\ A\ B) \to T(A) \to T(B)\,, \\
& impI{:}\Pi A^o B^o.\, (T(A) \to T(B)) \to T(imp\ A\ B)\,, \\
& FalseE{:}\Pi A^o.\, (T(imp\ A\ falsum) \to T(falsum)) \to T(A)\,, \\
& norm{:}\Pi A^o B^o.\, T(box\ (imp\ A\ B)) \to T(box\ A) \to T(box\ B)
\end{aligned}$$

and we have

LEMMA 5.6 *The rules encoded in* Σ_1, Σ_2 *are sound with respect to the standard Kripke semantics of* K*, if we interpret* $V(A)$ *as in Lemma 5.5 and* $T(A)$ *as* $c \Vdash A$ *where* c *is a fixed constant.*

Connecting V and T

We have now formalized two separate judgments, defined by the predicates V and T, which we have to connect together. To this end, we introduce two rules, C and R, which intuitively allow us to introduce a validity judgment, given a truth judgment, and eliminate (or use) a validity judgment.

C states that if A is true in an arbitrary world, then it is valid.

$$\frac{T(A)}{V(A)}\,C$$

For this rule to be sound, we require that the world where A is true really is arbitrary, and this will hold so long as there are no other assumptions $T(A')$ current when we apply it. It is easy to see that this condition is ensured for any proof of the atomic proposition $V(A)$, so long as C is the only rule connecting T and V together, since the form of the rules then ensures that there can be no hypotheses $T(A')$ at a place where C is applied.

The addition of C yields a complete inference system for reasoning about valid formulae. However, the resulting deductive system is awkward: once we end up in the V fragment of the system, which by itself is essentially a Hilbert calculus, we are forced to stay there.

We thus extend our system with an elimination rule for V, to allow us to return to the natural deduction-like T fragment. Important to our justification of the rule C was that the premise followed in an *arbitrary* world. Any further rule that we add must not invalidate this assumption. However we observe that given $V(A)$, then, since A is valid, it *is* true in an arbitrary world, so adding $T(A)$ as an open assumption to an application of C does not harm the semantic justification of that rule application. We can encode this as the following rule R.

$$\frac{V(A) \quad \begin{array}{c}[T(A)] \\ \vdots \\ V(B)\end{array}}{V(B)}\,R$$

Taken together, these rules complete our proposed encoding of K.

$$\begin{aligned}\Sigma_{\mathrm{K}_{MJ}} \equiv\ & \Sigma_1,\ \Sigma_2, \\ & C{:}\Pi A^o.\, T(A) \to V(A)\,, \\ & R{:}\Pi A^o B^o.\, V(A) \to (T(A) \to V(B)) \to V(B)\end{aligned}$$

To establish correctness formally, we begin by proving that:

PROPOSITION 5.7 *If A is a theorem of* K*, then $V(A)$ is a theorem of the proof calculus encoded as* $\Sigma_{\mathrm{K}_{MJ}}$.

Proof. We observe that if A is one of the listed axioms of K, then we can show that $T(A)$, and thus, by C, that $V(A)$. Therefore we need not declare these to be 'valid'. These, and the rules encoded in Σ_1 allow us to reconstruct any proof in Hilbert K (see also the remarks after Lemma 5.5 above). ■

Next that:

PROPOSITION 5.8 *If $V(A)$ is a theorem of the proof calculus encoded as* $\Sigma_{\mathrm{K}_{MJ}}$*, then A is a theorem of* K.

We prove a slight generalization, for which we need

LEMMA 5.9 *For an arbitrary set Γ of theorems of* K*, if $T(A)$ is a theorem of* $\Sigma_{\mathrm{K}_{MJ}}$ *extended with the axiom set* $\{ T(x) \mid x \in \Gamma \}$*, then A is a theorem of* K.

Proof. First notice that only the rules encoded as $impI$, $impE$, $FalseE$ and $norm$ can occur in a proof of $T(A)$. By Lemma 5.6, reading $T(A)$ as $c \Vdash A$, only theorems of K follow from this fragment of the proof calculus extended with Γ, where Γ consists of theorems of K. ■

The generalization of Proposition 5.8 is then

LEMMA 5.10 *For an arbitrary set Γ of theorems of* K*, if $V(A)$ is a theorem of the proof calculus encoded as* $\Sigma_{\mathrm{K}_{MJ}}$ *extended with* $\{ T(x) \mid x \in \Gamma \}$*, then A is a theorem of* K.

Proof. The proof is by induction on the size of a proof Π of $V(A)$. We need to consider three cases: (i) The last rule in the proof is an application of one of the rules encoded in Σ_1 from theorems $V(A_i)$, in which case, by appeal to the induction hypothesis, A_i are theorems of K and thus A is a theorem of K. (ii) The last rule is an application of C, in which case the sub-proof is a proof of the theorem $T(A)$ (there are no undischarged assumptions for the theorem $V(A)$ and hence $T(A)$) and by Lemma 5.9, A is a theorem of K. (iii) The last rule is an application of R to proofs Π_1 of $V(B)$ from $T(A)$ and Π_2 of the theorem $V(A)$. Since Π_2 is smaller than Π, by the induction hypothesis A is a theorem of K. Then we can transform Π_1 into a proof of the theorem $V(B)$ in the proof calculus formalized as $\Sigma_{\mathrm{K}_{MJ}}$ extended with $\{ T(x) \mid x \in \Gamma \cup \{A\} \}$ by replacing any appeal to a hypothesis $T(A)$ in Π_1 by an appeal to the axiom $T(A)$. Since the result is a proof no bigger than Π_1, which in turn is smaller than Π, by appeal to the induction hypothesis, B is a theorem of K. ■

We can combine Propositions 5.7 and 5.8 as

THEOREM 5.11 *A is a theorem of* K *iff $V(A)$ follows from* $\Sigma_{\mathrm{K}_{MJ}}$.

Encoding other modal logics

We can extend the encoding of K easily to deal with S4; there are in fact several ways we can do this: one possibility (see Avron *et al.* [1992] for more discussion) is to add the rules

$$\frac{\begin{array}{c}[V(\Box A)]\\ \vdots \\ V(B)\end{array}}{V(\Box A \supset B)}\supset\text{-}I_V \qquad \text{and} \qquad \frac{V(\Box A)}{V(A)}\Box\text{-}E_V$$

(given these rules we can show that the $Norm_T$ rule is redundant). This produces an encoding that is closely related to the version of S4 suggested by Prawitz.

An alternative view of K_{MJ}

We have motivated and developed the K_{MJ} presentation of K using a Kripke semantics, as a parallel with K_L. Unlike K_L, however, the interpretation is implicit, not explicit (there is no mention of particular worlds in the final proof calculus) so we are not committed to it. In fact, and importantly, this presentation can be understood from an entirely different perspective that uses just the Hilbert axiomatization itself, as follows.

We observe in §7 that it is possible to prove a deduction theorem for classical propositional logic, justifying the $\supset$-I rule by proof-theoretic means in terms of the Hilbert presentation. If we examine that proof, then we can see that it is easily modified for the fragment of (our Hilbert presentation of) K *without* the rule Nec. But this is precisely the fragment of K that is defined by the T fragment of K_{MJ}. Equally, the system defined by V, can be seen as the full Hilbert calculus.

Thus we can alternatively view the two judgments of K_{MJ} not as indicating whether we are speaking of truth in some world, or truth in all worlds, but rather whether or not we are allowed to apply the deduction theorem. This perspective provides the possibility of an entirely different proof of the correctness of our encoding, based on the standard Hilbert encoding, and without any intervening semantic argument.

5.5 Some conclusions

We have presented two different encodings of two well-known modal logics in this section as examples of approaches to representing nonstandard logics in an $\to$-framework. Which approach is preferable depends, in the end, on how the resulting encoding will be used. K_L and $\mathrm{S4}_L$ make the semantic foundation of the presentation more explicit. This is advantageous if we take for granted the view that modal logics are the logics of Kripke models since the user is able to exploit the associated intuitions in building proofs. On the other hand this may be a problem if we want to use our encoding in circumstances where our intuitions are different. The opposite holds for K_{MJ} and $S4_{MJ}$: it is more difficult to make direct use of any intuitions we might have from the Kripke semantics, but, since the proof systems involve no explicit, or even necessary, commitment to that interpretation, we have fewer problems in assuming another.[20]

The solutions we have examined, while tailored for modal logics, do have some generality. However, each different logic must be considered in turn and the approaches presented here may not always be applicable, or the amount of effort in modifying them may be considerable. For instance it is possible to interpret relevance logics in terms of a Kripke semantics that can be adopted as the basis of a labelled deductive system, similar in style to (though considerably more complex than) K_L (see Basin *et al.* [1998b]). But such an implementation is tricky to use and we reach, eventually, the limits of encodings that are understandable and usable by humans.

6 CONSEQUENCE RELATIONS

In the previous sections we considered an abstraction of natural deduction calculi and in the following section we will consider an abstraction of Hilbert calculi. Here, we consider the third style of proof calculus we mention in the introduction: the sequent calculus. It turns out that, unlike the other two, little work has been done on systems that directly abstract away from sequent calculi in a way that we can use as a logical framework. This certainly does not mean that there has been no work on the principles of the sequent calculus, just that work has concentrated not on the concrete implementational aspects so much as on the abstract properties of the sequent relation, $\vdash$, which when investigated in isolation is called a *consequence relation*.

Consequence relations provide a powerful tool for systematically analyzing properties of a wide range of logics, from the traditional logics of mathematics to modal or substructural logics, in terms that we can then use as the starting point of an implementation. In fact it is often possible to encode the results of a sequent calculus analysis directly in an $\rightarrow$-framework.

What, then, is a consequence relation? There are several definitions in the literature (e.g. Avron [1991; 1992], Scott [1974] and Hacking [1979]); we adopt that of Avron, along with his vocabulary, where possible.

[20] In fact it would be fairly easy to adapt a multiple judgment presentation of S4 to the different circumstances of say relevance, or linear, propositional logic, which share many properties with traditional S4. Such a project would be considerably more difficult, not to mention questionable, starting from $S4_L$. This issue of commitment to a particular interpretation is discussed at length with regard to labelled deductive systems in general by Gabbay [1996].

DEFINITION 6.1 *A* consequence relation *is a binary relation between finite multisets of formulae* Γ, Δ*, usually written* $\Gamma \vdash \Delta$*, and satisfying at least* $Basic$ *and* Cut *in* (2) *and PL and PR in* (3).[21]

This is, however, a very general definition. In fact most logics that we might be interested in encoding have natural presentations in terms of more constrained *ordinary* consequence relations:[22]

DEFINITION 6.2 *A consequence relation is said to be* ordinary *if it satisfies the rules WL, CL, WR and CR of* (3).

Examples of ordinary consequence relations are $\mathrm{LJ}^{\supset}$ and $\mathrm{LK}^{\supset}$ defined in §2.2, as is the encoding in terms of $\vdash$ of $\mathrm{NJ}^{\supset}$ in §3.1 (even though Cut is not a basic property of this presentation, we can show that it is admissible; i.e. we do not change the set of provable sequents if we assume it). Most of the traditional logics of mathematics can be presented as ordinary (in fact, *pure* ordinary, see below) consequence relations.

6.1 The meaning of a consequence relation

While it is possible to treat a consequence relation purely in syntactic terms, often one can be understood, and may have been motivated, by some idea of the 'meaning' of the formulae relative to one another. For instance we can read a sequent $\Gamma \vdash \Delta$ of $\mathrm{LK}^{\supset}$ as 'if all the formulae in Γ are true, then at least one of the formulae in Δ is true.' Because they have this reading in terms of truth, we call the systems defined by $\mathrm{LJ}^{\supset}$, $\mathrm{LK}^{\supset}$ and $\mathrm{NJ}^{\supset}$ 'truth' consequence relations. Notice that the meaning of 'truth' here is not fixed: when we say that something is true we might mean that it is classically true, intuitionistically true, Kripke-semantically true, relevance true, or even something else more exotic.

We can derive a different sort of consequence relation from a Hilbert calculus: if we assume (1) $Basic$, i.e. that $A \vdash A$, (2) that if A is an axiom then $\Gamma \vdash A$, and (3) that for each rule of proof

$$\frac{A_1 \ldots A_n}{A}$$

we have that

$$\frac{\Gamma_1 \vdash A_1 \ldots \Gamma_n \vdash A_n}{\Gamma_1, \ldots, \Gamma_n \vdash A}$$

[21] In the rules given in §2.2, the antecedent and succedent are *sequences* of formulae whereas here they are *multisets*. In practice, the permutation rules PL and PR are often omitted and multi-sets are taken as primitive, as here. This is not always possible though, e.g., in λ^{P}, where the ordering in the sequence matters. Note also that this definition does not take account of variables; for an extension to that case, see Avron [1992].

[22] We do not have space to consider the best known exception, the *substructural* logics, e.g. relevance and linear logic. However what we say in this section generalizes, given a suitably modified $\rightarrow$-framework, to these cases. Readers interested in the (non-trivial) technical details of this modification are referred to [Cervesato and Pfenning, 1996] and, especially, [Ishtiaq and Pym, 1998].

then it is easy to show that the resulting system satisfies Cut, and that $\vdash A$ iff A is a theorem. This is not a truth consequence relation: the natural reading of $\Gamma \vdash A$ is 'if Γ are *theorems*, then A is a *theorem*'. We thus call $\vdash$ a *validity* consequence relation.

Of course truth and validity are not the only possibilities. We can define consequence relations any way we want,[23] the only restriction we might impose is that in order to be effectively mechanizable on a computer, the relation $\vdash$ should be recursively enumerable.

6.2 *Ordinary pure consequence relations and $\rightarrow$-frameworks*

Part of the problem with mechanizing consequence relations, if we formalize them directly, is their very generality. Many systems are based on ordinary consequence relations, and if an encoding forces us to deal explicitly with all the rules that formalize this, then proof construction will often require considerable and tedious structural reasoning. This may explain, in part, why there has been little practical work on logical frameworks based directly on consequence relations.[24] However another reason is that an $\rightarrow$-framework can very effectively encode many ordinary consequence relations directly. In the remainder of this section we explore this reduction, which clarifies the relationship between consequence relations and $\rightarrow$-frameworks as well as illuminating some of the strengths and weaknesses of $\rightarrow$-frameworks.

In order to use an $\rightarrow$-framework for representing consequence relations, it helps if we impose a restriction in addition to ordinaryness.

DEFINITION 6.3 *We say that a consequence relation is* pure *if, given that*

$$\frac{\Gamma_1 \vdash \Delta_1 \quad \ldots \quad \Gamma_n \vdash \Delta_n}{\Gamma_0 \vdash \Delta_0}$$

holds, then there are Γ_i', Δ_i', which are sub-multisets of Γ_i, Δ_i, such that for arbitrary Γ_i'', Δ_i''

$$\frac{\Gamma_1', \Gamma_1'' \vdash \Delta_1', \Delta_1'' \quad \ldots \quad \Gamma_n', \Gamma_n'' \vdash \Delta_n', \Delta_n''}{\Gamma_0', \Gamma_0'' \vdash \Delta_0', \Delta_0''}.$$

Notice that the consequence relations discussed at the beginning of this section all satisfy the definition of purity; this is also the case for most of the logics we encounter in mathematics. In order to find counterexamples we must look to some

[23] For some examples of other consequence relations which can arise in the analysis of a modal logic, see Fagin *et al.* [1992].

[24] There are, however, many computer implementations of particular sequent calculi, e.g. the PVS system for higher-order logic, by Owre *et al.* [1995], not to mention many tableau calculi, which are, essentially, sequent calculi.

of the systems arising in philosophical logic. For instance in modal logic (see §5, and the discussion by Avron [1991]) we get a natural truth consequence relation satisfying the rule

$$\frac{\vdash A}{\vdash \Box A}$$

but not, for arbitrary Γ,

$$\frac{\Gamma \vdash A}{\Gamma \vdash \Box A}$$

We shall not, here, consider frameworks that can handle general impure consequence relations satisfactorily.

Single conclusioned consequence

As we said earlier, most of the standard logics of mathematics have intuitive presentations as ordinary pure consequence relations. Avron [1991] adds one more restriction before claiming that

> Every ordinary, pure, single-conclusioned [consequence relation] system can, e.g., quite easily be implemented in the Edinburgh LF.

We begin by considering the single conclusioned case, and then follow it with multiple conclusioned consequence relations and systems based on multiple consequence relations.

The encoding is uniform and consists of two parts. We first explain how to encode sequents and then rules. We can encode

$$A_1, \ldots, A_n \vdash A$$

as

$$T(A_1) \to \cdots \to T(A_n) \to T(A)$$

and it is easy to show that this satisfies Definition 6.2 of an ordinary consequence relation (in this case, single conclusioned). Notice how the structural properties of the consequence relation are directly reflected by the logical properties of $\to$.

We can then represent basic and derived rules expressed in terms of such a consequence relation, *assuming it is pure* as follows. Consider a rule of such a consequence relation $\vdash$, where, for *arbitrary* Γ_i:

$$\frac{\Gamma_1, A_{(1,1)}, \ldots, A_{(1,n_1)} \vdash A_1 \quad \cdots \quad \Gamma_m, A_{(m,1)}, \ldots, A_{(m,n_m)} \vdash A_m}{\Gamma_1, \ldots, \Gamma_m, A_{(0,1)}, \ldots, A_{(0,n_0)} \vdash A_0}$$

We can encode this as

$$\begin{aligned}(T(A_{(1,1)}) \to \cdots \to T(A_{(1,n_1)}) \to T(A_1)) \to \cdots \\ \to (T(A_{(m,1)}) \to \cdots \to T(A_{(m,n_m)}) \to T(A_m)) \\ \to T(A_{(0,1)}) \to \cdots \to T(A_{(0,n_1)}) \to T(A_0)\end{aligned}$$

leaving the Γ_i implicit (the condition of purity is important because it allows us to do this).

As an example, consider $\mathrm{LJ}^{\supset}$ from §2.2, for which we have the rules

$$\frac{\Gamma \vdash A \quad \Gamma', B \vdash C}{\Gamma, \Gamma', A \supset B \vdash C} \supset\text{-}L \qquad \frac{\Gamma, A \vdash B}{\Gamma \vdash A \supset B} \supset\text{-}R$$

We can encode these rules (assuming a suitable formalization of the language) as

$$T(A) \to (T(B) \to T(C)) \to T(A \supset B) \to T(C) \tag{24}$$

and

$$(T(A) \to T(B)) \to T(A \supset B)\,. \tag{25}$$

Then we can simulate the effect of applying $\supset$-L, i.e.

$$\frac{F_1, \dots, F_m \vdash A \quad G_1, \dots, G_n, B \vdash C}{F_1, \dots, F_m, G_1, \dots, G_n, A \supset B \vdash C},$$

encoded as (24), to the encoded assumptions

$$T(F_1) \to \cdots \to T(F_m) \to T(A) \tag{26}$$

and

$$T(G_1) \to \cdots \to T(G_n) \to T(B) \to T(C)\,. \tag{27}$$

By (26) and (24) we have

$$T(F_1) \to \cdots \to T(F_m) \to (T(B) \to T(C)) \to T(A \supset B) \to T(C) \tag{28}$$

which combines with (27) to yield

$$T(F_1) \to \cdots \to T(F_m) \to T(G_1) \to \cdots \to T(G_n) \to T(A \supset B) \to T(C)\,,$$

which encodes the conclusion of $\supset$-L.

From this, it is easy to see that our encoding of $\mathrm{LJ}^{\supset}$ is adequate. We show faithfulness by a modification of the techniques discussed above.

An observation about implementations

Note that the last step, 'combining' (28) with (27), actually requires a number of steps in the metalogic; e.g. shuffling around formulae by using the fact that, in the metalogic of the framework itself, $T(A) \to T(B) \to T(C)$ is equivalent to $T(B) \to T(A) \to T(C)$. But such reasoning must come out somewhere in any formalism of a system based on consequence relations. There is no getting around some equivalent of structural reasoning; the question is simply of how, and where, it is done, and how visible it is to the user.

In fact in actual implementations of $\rightarrow$-frameworks, e.g., Isabelle [Paulson, 1994], the cost of this structural reasoning is no greater than in a custom implementation since the framework theory itself will be implemented in terms of a pure, ordinary single conclusioned consequence relation; i.e. as a sequent or natural deduction calculus. If we write the consequence relation of the implementation as $\Longrightarrow$, then it is easy to see that an encoded sequent such as (26) can be quickly transformed into the logically equivalent

$$T(F_1), \ldots, T(F_m) \Longrightarrow T(A)$$

at which point it is possible to 'piggy-back' the rest of the proof off of $\Longrightarrow$, and thus make use of the direct (and thus presumably efficient) implementation of its structural properties.

Multiple conclusioned consequence

Having examined how we might encode ordinary pure single conclusioned consequence relations in an $\rightarrow$-framework, we now consider the general case of multiple conclusioned relations, which occur in standard presentations of many logics (e.g. $\mathrm{LK}^{\supset}$ described in §2.2).

There is no obvious correspondence between such relations and formulae in the language of $\rightarrow$. However, by refining our encoding a little, we can easily extend Avron's observation to the general case and thereby give a direct and effective encoding of multiple conclusioned consequence relations. We take as our example the system $\mathrm{LK}^{\supset}$. For larger developments and applications of this style, the reader is referred to [Pfenning, 2000].

A multiple conclusioned consequence relation is a pair of multisets of formulae, which we can refer to as 'left' and 'right'; i.e.

$$\overbrace{A_1, \ldots, A_n}^{left} \vdash \overbrace{B_1, \ldots, B_m}^{right} . \tag{29}$$

To encode this we need not one judgment T, but *two* judgments which we call T_L and T_R; we also define a new propositional constant E. We can encode (29) in an $\rightarrow$-framework as

$$T_L(A_1) \rightarrow \cdots \rightarrow T_L(A_n) \rightarrow T_R(B_1) \rightarrow \cdots \rightarrow T_R(B_m) \rightarrow E .$$

This is not quite enough to give us a consequence relation though; unlike in the single conclusioned case, we do not automatically get that $Basic$ is true. However we can remedy this by declaring that

$$T_L(A) \rightarrow T_R(A) \rightarrow E$$

and then we can show that the encoding defines an ordinary consequence relation.

We can then extend the style of encoding described to define, e.g., the right and left rule for implication in $LK^{\supset}$, which are

$$(T_L(A) \to T_R(B) \to E) \to T_R(A \supset B) \to E$$

and

$$(T_R(A) \to E) \to (T_L(B) \to E) \to T_L(A \supset B) \to E\,.$$

As in the single conclusioned case, a necessary condition for this encoding is that the consequence relation is pure. It is also easy to show that the encoding is adequate and faithful with respect to the derivability of sequents in $LK^{\supset}$.

Multiple consequence relation systems

Finally, we come to the problem of how to formalize systems based on more than one consequence relation. Here we will briefly consider just the single conclusioned case; the same remarks, suitably modified, also apply to the multiple conclusioned case.

One approach to analyzing a formal system (and quite useful if we want the analysis to be in terms of pure consequence relations) is, rather than using a single relation, to decompose it into a set of relations $\vdash_1$ to $\vdash_n$. We can encode each of these relations, and their rules, just like in the single relation case, using predicates T_1 to T_n; i.e.

$$A_1, \ldots, A_n \vdash_i A$$

as

$$T_i(A_1) \to \cdots \to T_i(A_n) \to T_i(A).$$

We encounter a new encoding problem with multiple consequence systems. These typically (always, if they are interesting) contain rules that relate consequence relations to each other, i.e., rules where the premises are built from different consequence relations than the conclusion. Consider the simplest example of this, which simply declares that one consequence relation is a subrelation of another (what we might call a $bridge$ rule):

$$\frac{\Gamma \vdash_1 A}{\Gamma \vdash_2 A}\, bridge$$

We can encode this as the pair of schemata

$$T_1(A) \to T_2(A) \tag{30}$$

and

$$(T_1(A) \to T_2(B)) \to T_2(A) \to T_2(B) \tag{31}$$

and show that the resulting encoding is properly closed. That is, given

$$T_1(A_1) \to \cdots \to T_1(A_n) \to T_1(B)$$

by (30) we get

$$T_1(A_1) \to \cdots \to T_1(A_n) \to T_2(B)$$

then by n applications of (31) we eventually get

$$T_2(A_1) \to \cdots \to T_2(A_n) \to T_2(B)\,.$$

As this example shows, working with a multiple consequence relation system in an $\to$-framework may require many metalevel steps to simulate a proof step in the object logic (here the number depends on the size of the sequent). More practical experience using such encodings is required to judge whether they are really usable in practice as opposed to just being a theoretically interesting way of encoding multiple consequence relation systems in the logic of $\to$.

In fact we have already encountered an example of this kind of an encoding: the multiple judgment encoding of modal logic developed in [Avron *et al.*, 1992; Avron *et al.*, 1998], described in §5.4, can be seen as the encoding of a two consequence relation system for truth and validity where T_1 is T, T_2 is V and $bridge$ is implemented by R and C.

7 DEDUCTIVE SYSTEMS AS INDUCTIVE DEFINITIONS

In the introduction we discussed two separate traditions of metatheory: metatheory as a unifying language and metatheory as proof theory. We have shown too how $\to$-frameworks fit into the unifying language tradition, and the way different logics can be encoded in them and used to carry out proofs. However, $\to$-frameworks are inadequate for proof theory: in exchange for ease of reasoning *within* a logic, reasoning *about* the logic becomes difficult or impossible.

In order better to understand this point, and some of the subtleties it involves, consider the following statements about the (minimal) logic of $\supset$.

1. $A \supset A$ is a theorem.

2. $A \supset B \supset C$ is true on the assumption that $B \supset A \supset C$ is true.

3. The deduction theorem holds for $\mathrm{HJ}^{\supset}$.

4. The deduction theorem holds for all extensions of $\mathrm{HJ}^{\supset}$ with additional axioms.

Statement 1 can be formalized in a metalogic as a statement about provability in any complete presentation of the logic of $\supset$; e.g. $\mathrm{NJ}^{\supset}$, $\mathrm{LJ}^{\supset}$ or $\mathrm{HJ}^{\supset}$. As a statement about provability we might regard it as, in some sense, a proof-theoretic statement. But as such, it is very weak since, by completeness, it must be provable irrespective of the deductive system used.

Statement 2 expresses a derived rule of the logic of $\supset$. Its formalization requires that we can reason about the truth of one formula relative to others. As explained in §6, representing this kind of a (truth) consequence relation can easily be reduced to a provability problem in an $\rightarrow$-framework. For example, in the λ^{P} encoding of $\mathrm{NJ}^{\supset}$ we prove that the type $\Pi A^o B^o C^o.\, pr(B \supset A \supset C) \rightarrow pr(A \supset B \supset C)$ is inhabited.

As with statement 1, the metalogic must be able to build proofs in the object logic (in this case though under assumptions, encoded using $\rightarrow$ in the metalogic), but proofs themselves need not be further analyzed.

Statement 3 (which we prove in this section) is an example of a metatheoretic statement that is more in the proof theory tradition. In order to prove it we must analyze the structure of arbitrary proofs in the deductive system of $\mathrm{HJ}^{\supset}$ using an inductive argument. The difference between this and the previous example is important: for a proof of statement 2 we need to know which axioms and rules are available in the formalized deductive system, while for a proof of statement 3 we also need to know that no other rules are present, since this is what justifies an induction principle over proofs (the rules of $\mathrm{HJ}^{\supset}$ can be taken as constituting an inductive definition). An $\rightarrow$-framework like λ^{P} contains no provisions for carrying out induction over the structure of an encoded deductive system.

Statement 4 is an extension of statement 3, which introduces the problem of *theory structuring*. Structuring object theories to allow metatheoretic results to be 'imported' and used in related theories is not a substantial problem in the kinds of metamathematical investigations undertaken by proof theorists, who are typically interested in proving particular results about particular systems. However, for computer scientists working on formal theorem proving, it is enormously important: it is good practice for a user reasoning about complex theories to formalize a collection of simpler theories (e.g. for numbers and arithmetic, sequences, relations, orders, etc.) that later are combined together as needed. So some kind of theory structuring facility is practically important and many systems provide support for it.[25]

Unfortunately, hierarchical structure in $\rightarrow$-frameworks is the result of an assumption (necessary anyway for other reasons) that the languages and deductive systems of encoded logics are 'open to extensions' (see §7.4 below), something that automatically rules out any arguments requiring induction on the structure of

[25] For example HOL [Gordon and Melham, 1993], Isabelle [Paulson, 1994] and their predecessor LCF [Gordon *et al.*, 1979] support simple theory hierarchies where theorems proven in a theory may be used in extensions.

the language[26] or proofs of a theory, e.g. the deduction theorem. If we 'close' the language or deductive system by explicitly adding induction over the language or proofs, in order to prove metatheorems, it is unsound later to assume those theorems in extensions of the deductive system or language. In §7.4, we suggest that there is a way to avoid this problem if we formulate metatheorems in a metalogic based on inductive definitions.

These examples illustrate that there are different sorts of metatheoretic statements, which are distinguished by how conscious we have to be of the metatheoretic context in order to prove them. The central role of induction in carrying out many kinds of the more general metatheoretic arguments is the reason we now consider logical frameworks based on inductive definitions.

7.1 Historical background: From Hilbert to Feferman

What kind of a metalogic is suited for carrying out metatheoretic arguments that require induction on the language or deductive system of the object logic? To answer this question we draw on experience gained by proof theorists dating back to Hilbert, in particular Post and Gödel, and later Feferman, who have asked very similar questions. We can also draw on practical experience over the last 30 years in the use of computers in work with formal systems.

The work of Post and Gödel

In the early part of this century, Post [1943] (see Davis [1989] for a short survey of Post's work) investigated the decidability of logics like that of *Principia Mathematica*. He showed that such systems could be formalized as (what we now recognize as) recursively enumerable classes of strings and that this provided a basis for metatheoretic analysis. Although Post's work is a large step towards answering our question, one important aspect, from our point of view, is missing from his formalization. There is no consideration of *formal* metatheory. Post was interested in a formal characterization of deductive systems, not of their metatheories; he simply assumed, reasonably for his purposes, that arbitrary mathematical principles could be adopted as necessary for the metatheory.

We cannot make the same assumption with a logical framework: we must decide first which mathematical principles we need, and then formalize them. The work of Post is thus complemented, for our purposes, by Gödel's [1931] work on the incompleteness of systems like Principia Mathematica, which shows that a fragment of arithmetic is sufficient for complex and general metatheory. Logicians have since been able to narrow that fragment down to the theory of primitive

[26] An example of a metatheorem for which we need induction over the language, not the derivations, is that *In classical logic, a propositional formula that contains only the $\Leftrightarrow$ connective is valid if and only if each propositional variable occurs an even number of times*.

recursive arithmetic, which seems sufficient for general syntactic metatheory.[27]

The natural numbers, although adequate for Gödel's purposes, are too unwieldy to use as a logical framework. Indeed, a large part of Gödel's paper is taken up with a complicated technical development showing that arithmetic really can represent syntax. The result is unusable in a practical framework: the relationship between syntax and its encoding is only indirectly given by (relatively) complicated arithmetic functions and the numbers generated in the encoding can be enormous. This is in contrast to Post's strings (further investigated in the early sixties by mathematicians such as Smullyan [1961]), which have a simple and direct correspondence with the structures we want to encode, and a compact representation.

S-expressions and FS_0

It is possible to build a formal metatheory based on strings, in the manner of Post. However, experience in computer science in formalizing and working with large symbolic systems has shown that there is an even more natural language for modeling formal (and other kinds of symbolic) systems. The consensus of this experience is found in Lisp [McCarthy, 1981; Steele Jr. and Gabriel, 1996], which for more than 30 years has remained the most popular language for building systems for symbolic computation.[28] Further, Lisp is not only effective, but its basic data-structure, which has, in large part, contributed to its effectiveness, is remarkably simple: the *S-expression*. This is the data-type freely generated from a base type by a binary function 'Cons'. Experience with Lisp has shown that just about any syntactic structure can be mapped directly onto S-expressions in such a way that it is very easy to manipulate. In fact, we can even restrict the base type of S-expressions to be a single constant (often called '*nil*') and still get this expressiveness. Further evidence for the effectiveness of Lisp and S-expressions is that one of the most successful theorem proving systems ever built, NQTHM [Boyer and Moore, 1981], verifies recursive functions written in a version of Lisp that uses precisely this class of S-expressions.

An example of a theory that integrates all the above observations is FS_0, due to Feferman. This provides us with a language in which we can define exactly all the recursively enumerable classes of S-expressions. Moreover, it permits inductive arguments over these inductively defined classes, and thus naturally subsumes both the theory of recursively enumerable classes and primitive recursive arithmetic. It

[27] This is a thesis, of course, not a theorem; and exceptions have to be made for, e.g., proof normalization theorems, for which (as a corollary of Gödel's result itself) we know there can be no single general metatheory.

[28] This does not mean that there have not been successful programming languages that use strings as their basic data-structure; the SNOBOL family [Griswold, 1981] of languages, for example, is based on the theory of Markoff string transformation algorithms. However it is significant that SNOBOL, in spite of its mathematical elegance in many ways, has never been seen as a general purpose symbolic programming language like Lisp, or been adopted so enthusiastically by such a large and influential programming community.

has proved usable too in case-studies of computer supported metatheory, in the proof-theoretic tradition (see Matthews [1992; 1993; 1994; 1997b]). In the rest of this chapter we will use a slightly abstracted version of FS_0 for our discussion.

7.2 *A theory of inductive definitions:* FS_0

FS_0 is a simple minimal theory of inductive definitions, which we present here in an abstract form (we elide details in order to emphasize the general, rather than the particular, features). A full description of the theory is provided by Feferman [1990], while an implementation is described by Matthews [1996].

FS_0 is a theory of inductively defined sets, embedded in first-order logic and based on Lisp-style S-expressions. The class of S-expressions is the least class containing nil and closed under the pairing $(cons)$ operator, which we write as an infix comma $(\cdot,\cdot)$, such that $nil \neq (a, b)$ for all S-expressions a and b; we also assume, for convenience, that comma associates to the right, so that $(a, (b, c))$ can be abbreviated to (a, b, c). We then have functions car and cdr, which return the first and second elements of a pair.

Comprehension over first-order predicates is available. We write

$$x \in S \Leftrightarrow P(x) \qquad \text{or} \qquad S \equiv \{\, x \mid P(x) \,\}$$

to indicate a set S so defined. Such definitions can be parameterized and the parameters are treated in a simple way, thus, for example, we can write

$$x \in S(a, b) \Leftrightarrow (x, a) \in b.$$

We can also define sets explicitly as inductive definitions using the $I(\cdot,\cdot)$ construction: if A and B are sets, then $I(A, B)$ is the least set containing A and closed under the rule

$$\frac{t_1 \quad t_2}{t}$$

where $(t, t_1, t_2) \in B$. Note that we only have inductive definitions with exactly two 'predecessors', but this is sufficient for our needs here and, with a little more effort, in general.

Finally, we can reason about inductively defined sets using the induction principle

$$\begin{aligned} &Base \subseteq S \to \forall a,\ b,\ c.\,(b \in S \to c \in S \to \\ &\qquad\qquad (a, b, c) \in Step \to a \in S) \to I(Base, Step) \subseteq S\,. \end{aligned} \tag{32}$$

This says that a set S contains all the members of a set $I(Base, Step)$ if it contains the members of $Base$ and whenever it contains two elements b and c, and (a, b, c) is an instance of the rule $Step$, then it also contains a. This induction principle applies to sets, not predicates, but it is easy, by comprehension, to generate one from the other, so this is not a restriction.[29]

7.3 A Hilbert theory of minimal implication

Having sketched a theory of inductive definitions, we now consider how it might actually be used, both to encode an object logic, and to prove metatheorems. As an example, we encode the theory $\mathrm{HJ}^{\supset}$ (of §2.1) and prove a deduction theorem.

The definition of HJ

The language L_{HJ} We define an encoding $\ulcorner\cdot\urcorner$ of the standard language of implicational logic L_{HJ} (as usual we distinguish metalevel ($\to$) from object level ($\supset$) implication) as follows. We define two distinct S-expression constants (e.g. *nil* and (nil, nil)) which we call `atom` and `imp`, then we have

$$\begin{array}{rcll} \ulcorner a \urcorner & = & (\mathtt{atom}, \ulcorner\!\ulcorner a \urcorner\!\urcorner) & \qquad (a \text{ atomic}) \\ \ulcorner a \supset b \urcorner & = & (\mathtt{imp}, \ulcorner a \urcorner, \ulcorner b \urcorner) & \end{array}$$

(assuming $\ulcorner\!\ulcorner a \urcorner\!\urcorner$ for atomic a to be already, separately, defined). It is easy to see that $\ulcorner\cdot\urcorner$ is an injection from L_{HJ} into the S-expressions, on the assumption that $\ulcorner\!\ulcorner\cdot\urcorner\!\urcorner$ is. For the sake of readability, in the future we will abuse notation, and write simply $a \supset b$ when we mean the schema $(\mathtt{imp}, a, b)$; i.e. a and b here are variables in the theory of inductive definitions, not propositional variables in the encoded language L_{HJ}.

The theory HJ We now define a minimal theory of implication, HJ, as follows. We have two classes of axioms

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

and

$$x \in S \Leftrightarrow \exists a,\, b,\, c.\, x = ((a \supset b) \supset (a \supset b \supset c) \supset a \supset c)$$

and a rule of detachment

$$x \in Det \Leftrightarrow \exists a,\, b.\, x = (b, (a \supset b), a)$$

from which we define the theory HJ to be

$$\mathrm{HJ} \equiv I(K \cup S, Det)\,.$$

[29] In FS_0 comprehension is restricted to essentially Σ^0_1 predicates with the result that the theory is recursion-theoretically equivalent to primitive recursive arithmetic.

Using HJ

We can now use HJ to prove theorems of the Hilbert calculus $\mathrm{HJ}^{\supset}$ in the same way that we would use an encoding to carry out natural deduction proofs in an $\rightarrow$-framework. One difference is that we do not get a direct correspondence between proof steps in the encoded theory and steps in the derivation. However, in this case it is enough to prove first the following lemmas:

$$a \supset b \supset a \in \mathrm{HJ} \tag{33}$$

$$(a \supset b) \supset (a \supset b \supset c) \supset a \supset c \in \mathrm{HJ} \tag{34}$$

$$a \in \mathrm{HJ} \rightarrow (a \supset b) \in \mathrm{HJ} \rightarrow b \in \mathrm{HJ} \tag{35}$$

Here, and in future, we assume theorems are universally closed. From these, it follows that if A is a theorem of minimal implication, then $\ulcorner A \urcorner \in \mathrm{HJ}$.

For example, we show

$$a \supset a \in \mathrm{HJ} \tag{36}$$

with the derivation

1. $a \supset a \supset a \in \mathrm{HJ}$	by (33)
2. $a \supset (a \supset a) \supset a \in \mathrm{HJ}$	by (33)
3. $(a \supset a \supset a) \supset (a \supset (a \supset a) \supset a) \supset a \supset a \in \mathrm{HJ}$	by (34)
4. $(a \supset (a \supset a) \supset a) \supset a \supset a \in \mathrm{HJ}$	by (35),1,3
5. $a \supset a \in \mathrm{HJ}$	by (35),2,4

The steps of this proof correspond, one to one, with the steps of the proof that we gave for $\mathrm{HJ}^{\supset}$ in §2.1. As this example suggests, at least for propositional Hilbert calculi, inductive definitions support object theory where proofs in the metatheory closely mirror proofs in the object theory. (For logics with quantifiers the relationship is less direct, since we have to encode and explicitly reason about variable binding and substitution.)

Proving a deduction theorem for HJ

Let us now consider an example that requires induction over proofs themselves: the deduction theorem for HJ. We only sketch the proof; the reader is referred to Basin and Matthews [2000] for details. Informally, the deduction theorem says:

> If B is provable in HJ with the additional axiom A then $A \supset B$ is provable in HJ.

Note that this theorem relates different deductive systems: HJ and its extension with the axiom A. Moreover, as A and B are schematic, ranging over all formulae of HJ, the theorem actually relates provability in HJ with provability in infinitely many extensions, one for each proposition A.

To formalize this theorem we define

$$\mathrm{HJ}[\Gamma] \equiv I(K \cup S \cup \Gamma, Det)\,; \tag{37}$$

i.e. $\mathrm{HJ}[\Gamma]$ is the deductive system HJ where the axioms are extended by all formulae in the class Γ. Now we can formalize the deduction theorem as

$$b \in \mathrm{HJ}[\{a\}] \to (a \supset b) \in \mathrm{HJ}\,, \tag{38}$$

which in FS_0 can be transformed into

$$I(K \cup S \cup \{a\}, Det) \subseteq \{\ x \mid (a \supset x) \in \mathrm{HJ}\ \}\,.$$

This in turn can be proved by induction on the inductively defined set $I(K \cup S \cup \{a\}, Det)$ using (32). The proof proceeds as follows:

Base case We have to show $K \cup S \cup \{a\} \subseteq \{\ x \mid (a \supset x) \in \mathrm{HJ}\ \}$. This reduces, via $x \in K \cup S \cup \{a\} \to (a \supset x) \in \mathrm{HJ}$, to showing that $(a \supset x) \in \mathrm{HJ}$ given either (i) $x \in K \cup S$ or (ii) $x \in \{a\}$. For (i) we have $x \in \mathrm{HJ}$ and $(x \supset a \supset x) \in \mathrm{HJ}$, and thus, by Det that $(a \supset x) \in \mathrm{HJ}$. For (ii) we have $x = a$ and thus have to show that $(a \supset a) \in \mathrm{HJ}$, which we do following the proof of (36) above.

Step case There is only one rule (Det) and thus one case: given $b \in \{\ x \mid (a \supset x) \in \mathrm{HJ}\ \}$ and $(b \supset c) \in \{\ x \mid (a \supset x) \in \mathrm{HJ}\ \}$, prove that $c \in \{\ x \mid (a \supset x) \in \mathrm{HJ}\ \}$. This reduces to proving, given $(a \supset b) \in \mathrm{HJ}$ and $(a \supset b \supset c) \in \mathrm{HJ}$ that $(a \supset c) \in \mathrm{HJ}$. This in turn follows by observing that $((a \supset b) \supset (a \supset b \supset c) \supset a \supset c) \in \mathrm{HJ}$ by (34), from which, by (35) twice with the hypotheses, $(a \supset c) \in \mathrm{HJ}$.

Once we have proved the deduction theorem we can use it to build proofs where we reason under assumption in the style of natural deduction. This is useful, indeed in practice essential, if we really wish to use Hilbert calculi to prove anything. However it is also limited since this metatheorem can be applied *only* to HJ. Thus, we next consider how this limitation can be partially remedied.

7.4 Structured theory and metatheory

At the beginning of this section, we discussed two examples of the deduction theorem (statements 3 and 4) where the second stated that the deduction theorem holds not just in $\mathrm{HJ}^{\supset}$ but also in extensions. We return to this example, which illustrates an important difference between ID-frameworks and $\to$-frameworks.

Structuring in $\rightarrow$-frameworks

Let us first examine how theories can be structured in $\rightarrow$-frameworks. Consider the following: we can easily encode $\mathrm{HJ}^{\supset}$ as (assuming the encoding of the syntax is given separately) the axiom set $\Gamma_{\mathrm{HJ}^{\supset}}$.

$$T(A \supset B \supset A)$$
$$T((A \supset B) \supset (A \supset B \supset C) \supset A \supset C)$$
$$T(A) \rightarrow T(A \supset B) \rightarrow T(B)$$

Then A is a theorem of $\mathrm{HJ}^{\supset}$ iff $\Gamma_{\mathrm{HJ}^{\supset}} \vdash T(A)$, i.e., $T(A)$ is provable in the metalogic under the assumptions $\Gamma_{\mathrm{HJ}^{\supset}}$. Now consider the deduction theorem in this setting; we would have to show that

$$\Gamma_{\mathrm{HJ}^{\supset}} \vdash (T(A) \rightarrow T(B)) \rightarrow T(A \supset B)\,. \tag{39}$$

This is not possible, however.

In an $\rightarrow$-framework, a basic property of $\vdash$ is weakening; i.e. if $\Gamma \vdash \phi$ then $\Gamma, \Delta \vdash \phi$. This is very convenient for structuring theories: a theorem proven under the assumptions Γ holds under extension with additional assumptions Δ. For example, Δ might extend our formalization of $\mathrm{HJ}^{\supset}$ to a classical theory of $\supset$ or perhaps to full first-order logic.[30] By weakening, given $\Gamma_{\mathrm{HJ}^{\supset}} \vdash \phi$ we immediately have $\Gamma_{\mathrm{HJ}^{\supset}}, \Delta \vdash \phi$. Thus we get a natural hierarchy on the object theories we define: theory T' is a subtheory of theory T when its axioms are a subset of those of T. This allows us to reuse proven metatheorems since anything proven for a subtheory automatically follows in any supertheory.

Consider, on the other hand, the extension Δ_{K} consisting of the axioms

$$T(A) \rightarrow T(\Box A)$$
$$T(\Box A \supset \Box(A \supset B) \supset \Box B)$$

with which we can extend $\Gamma_{\mathrm{HJ}^{\supset}}$ to a fragment of the modal logic K. The deduction theorem *does not* follow in K; therefore, since by faithfulness we have

$$\Gamma_{\mathrm{HJ}^{\supset}}, \Delta_{\mathrm{K}} \nvdash (T(A) \rightarrow T(B)) \rightarrow T(A \supset B)\,,$$

we must also have, by weakening and contraposition,

$$\Gamma_{\mathrm{HJ}^{\supset}} \nvdash (T(A) \rightarrow T(B)) \rightarrow T(A \supset B)\,.$$

This suggests that there is an either/or situation: we can have either hierarchically structured theories, as in an $\rightarrow$-framework, or general inductive metatheorems (like the deduction theorem), as in an ID-framework, but not both. In fact,

[30] An extension of the deduction theorem to first-order logic, however, is not trivial—we have to treat a new rule, which introduces complex side conditions to the statement of the theorem (Kleene [1952] discusses one way to do this).

as we will see, in an ID-framework things are not quite so clear-cut: there is the possibility both to prove metatheorems by induction and to use them in certain classes of extensions.

Structuring in an ID-framework

Part of the explanation of why we can prove (38), but not (39), is that it is not possible to extend the deduction theorem for HJ to arbitrary supertheories: (38) is a statement about HJ and it tells us nothing about anything else. However a theorem about HJ alone is of limited use: in practice we are likely to be interested in $\mathrm{HJ}^{\supset}$ as a fragment of some larger theory. We know, for instance, that the deduction theorem follows for many extensions of HJ (e.g. extensions to larger fragments of intuitionistic or classical logic). The problem is that the induction principle we use to prove the theorem is equivalent to a closure assumption, and such an assumption means that we are not able to use the theorem with extensions.

We seem to have confirmed, from the other side, the trade-off we have documented above for $\rightarrow$-frameworks: either we can have induction and no theory structuring (as theories are 'closed'), or vice versa. However, if we look harder, there is sometimes a middle way that is possible in ID-frameworks. The crucial point is that an inductive argument does not always rely on *all* the closure assumptions of the most general case. Consider the assumptions that are made in the proof of (38):

- The proof of the base case relies on the fact that the axioms $a \supset b \supset a$ and $a \supset a$ are available in HJ.

- The proof of the step case relies on the fact that the *only* rule that can be applied is Det, and that the axiom

  $$(a \supset b) \supset (a \supset b \supset c) \supset a \supset c$$

  is available.

What bears emphasizing is that we do not need to assume that no axioms other than those explicitly mentioned are available, only that no rules other than Det are available.

We can take account of this observation to produce a more general version of (38)

$$b \in \mathrm{HJ}[\Gamma \cup \{a\}] \rightarrow (a \supset b) \in \mathrm{HJ}[\Gamma], \tag{40}$$

which we can still prove in the same way as (38). We call this version *open-ended* since it can be used with any axiomatic extension Γ of HJ. In particular (38) is just (40) where we take Γ to be $\emptyset$.

Structuring theories with the deduction theorem Unlike (38), we can make effective use of (40) in a hierarchy of theories in a way similar to what is possible in an $\rightarrow$-framework. The metatheorem can be applied to any extension $\mathrm{HJ}[\Gamma]$ where Γ is a collection of axioms. The fact that in the $\rightarrow$-framework we can add new rules, not just new axioms, is not as significant as it at first appears, so long as we have that

$$T(A) \rightarrow T(B) \quad \text{iff} \quad T(A \supset B) \tag{41}$$

since we can use this to find an axiomatic equivalent of any rule schema built from $\rightarrow$ and T in terms of $\supset$.

The above observation, of course, only holds for theories that can be defined in terms of a single predicate T and which include a connective $\supset$ for which (41) is true.[31]

A further generalization of the deduction theorem

We arrived at the open-ended (40) by observing that other axioms could be present. And as previously observed, no such generalization is possible with arbitrary rules, e.g., the deduction theorem does not hold in K (which requires extensions by rules, as opposed to axioms). However, a more refined analysis of the step case of the proof is possible, and this leads to a further generalization of our metatheorem.

In the step case we need precisely that the theory is closed under

$$\frac{A \supset B \quad A \supset C}{A \supset D}$$

for each instance of a basic rule

$$\frac{B \quad C}{D}.$$

In the case of Det (the only rule in $\mathrm{HJ}^{\supset}$) we can show this by a combination of Det and the S axiom.

Using our ID-framework we can explicitly formalize these statements as part of the deduction theorem itself, proving a further generalization. If we extend the notation of (37) with a parameter Δ for rules, i.e.,

$$\mathrm{HJ}[\Gamma, \Delta] \equiv I(K \cup S \cup \Gamma, Det \cup \mathit{\Delta}) \tag{42}$$

then for the base case we have

$$b \in \mathrm{HJ}[\Gamma \cup \{a\}, \Delta] \rightarrow (a \supset b) \in \mathrm{HJ}[\Gamma, \Delta] \tag{43}$$

[31] And, of course, some encodings use more than one metalevel predicate; e.g. in §5.4 we introduce a second predicate V for which there is no equivalent of (41). For these systems we have rules for which no axiomatic equivalent is available. This does not, however, mean that ID-frameworks are necessarily less effective for structuring collections of theories; it just means that we have to be more sophisticated in the way we exploit (41). See, e.g., Matthews [1997b] for discussion of how we can do this by introducing an 'extra' layer between the ID-framework and the theory to be encoded.

and

$$(a \supset a) \in \mathrm{HJ}[\Gamma, \Delta] \tag{44}$$

while for the step case

$$\begin{aligned}(d, b, c) \in \Delta \to (a \supset b) \in \mathrm{HJ}[\Gamma, \Delta] \\ \to (a \supset c) \in \mathrm{HJ}[\Gamma, \Delta] \to (a \supset d) \in \mathrm{HJ}[\Gamma, \Delta]\,.\end{aligned} \tag{45}$$

The formulae (43) and (44) follow immediately for any $\mathrm{HJ}[\Gamma, \Delta]$, but (45) isn't always true. Thus, our third deduction theorem has the form

$$(45) \to b \in \mathrm{HJ}[\Gamma \cup \{a\}, \Delta] \to (a \supset b) \in \mathrm{HJ}[\Gamma, \Delta]\,, \tag{46}$$

which can be proved in the same way as (40). Note too that this metatheorem generalizes (40), since (40) is just (38) where Δ is $\emptyset$ and the antecedent, which is therefore true, has been removed.

The deduction theorem can even be further generalized, but doing so would take us too far afield. In [Basin and Matthews, 2000] we show how a further generalization of (46) can be specialized to modal logics that extend S4. This generalization allows us to prove

$$b \in \mathrm{S4}[\Gamma \cup \{a\}] \to (\Box a \supset b) \in \mathrm{S4}[\Gamma].$$

That is, in S4 we can prove a deduction theorem that allows us to reason under 'boxed' assumptions $\Box a$.

7.5 Admissible and derived rules

Our examples suggest that inductive definitions offer considerable power and simplicity in organizing metatheories. Each metatheorem states the conditions an extension has to satisfy for it to apply; so once proved, we need only check these conditions before making use of it. Most metatheorems require only that certain axioms and rules are available and therefore hold in all extensions with additional axioms and rules. Others depend on certain things being absent (e.g. rules that do not satisfy certain properties, in the case of the deduction theorem); in such cases, we can prove more restricted theorems that are still usable in appropriate extensions.

How does this kind of metatheory compare with what is possible in theorem provers supporting hierarchical theories? We begin by reviewing the two standard notions of proof-rules. Our definitions are those of Troelstra [1982, § 1.11.1] translated into our notation, where $\mathcal{T}[\Gamma, \Delta]$ is a deductive system $\mathcal{T}$ extended with sets of axioms Γ and rules Δ, e.g. (42).

Fix a language of formulae. A *rule* is an $n + 1$-ary relation over formulae $\langle F_1, \ldots, F_n, F_{n+1} \rangle$ where the $F_1, \ldots, F_n$ are the *premises* and F_{n+1} the *conclusion*. A rule is *admissible* for $\mathcal{T}$ iff

$$\vdash_{\mathcal{T}[\emptyset,\emptyset]} F_1 \to \cdots \to \vdash_{\mathcal{T}[\emptyset,\emptyset]} F_n \to \vdash_{\mathcal{T}[\emptyset,\emptyset]} F_{n+1}\ , \qquad (adm)$$

and *derivable* for $\mathcal{T}$ iff

$$\forall \Gamma.\vdash_{\mathcal{T}[\Gamma,\emptyset]} F_1 \to \cdots \to \vdash_{\mathcal{T}[\Gamma,\emptyset]} F_n \to \vdash_{\mathcal{T}[\Gamma,\emptyset]} F_{n+1}\ . \qquad (der)$$

It follows immediately from the definitions that derivability implies admissibility; however, the converse does not always hold. It is easy to show that Troelstra's definition of derivability is equivalent to that of Hindley and Seldin [1986]; i.e. $\vdash_{\mathcal{T}[\{F_1,\ldots,F_n\},\emptyset]} F_{n+1}$, and that if a rule is derivable it holds in all extensions of $\mathcal{T}$ with new axioms and rules.

Whereas in $\to$-frameworks we can only prove derived rules, logical frameworks based on inductive definitions allow us to prove that rules are admissible, as well as reason about other kinds of rules not fitting the above categories. For example, the languages or deductive systems for the F_i can be different, like in the various versions of the deduction theorem that we have formalized; our deduction theorems are neither derived nor admissible since their statements involve different deductive systems.

7.6 Problems with ID-frameworks

Our examples provide evidence that a framework based on inductive definitions can serve as an adequate foundation for carrying out metatheory in the proof theory tradition and can be used to structure metatheoretic development. However, some aspects of formal metatheory are more difficult than with an $\to$-framework. The most fundamental difficulty, and one that is probably already clear from our discussion in this section, is the way languages are encoded. This is quite primitive in comparison to what is possible in an $\to$-framework: for the propositional examples that we have treated here, the view of a language as a recursively enumerable class is direct and effective. But this breaks down for logics with quantifiers and other variable binding operators where the natural equivalence relation for syntax is no longer identity but equivalence under the renaming of bound variables (α-congruence). We have shown (in §4.2) that language involving binding has a natural and direct treatment in an $\to$-framework as higher-order syntax. Nothing directly equivalent is available in an ID-framework; we are forced to build the necessary facilities ourselves.

Since the user must formalize many basic syntactic operations in an ID-framework, any treatment of languages involving variable binding operators will be more 'primitive' than what we get in an $\to$-framework, but how much more primitive is not clear. So far, most experience has been with *ad hoc* implementations of binding (e.g. [Matthews, 1992]) but approaches that are both more sophisticated and more modular are possible, such as the binding structures proposed by Talcott [1993], a generalization of de Bruijn indices as an algebra. As yet, we do not know how effective such notations are.

The other property of ID-frameworks that might be criticized is that they are biased towards Hilbert calculi, which are recognized to be difficult to use. But metatheorems, in particular the deduction theorem, can play an important role in making Hilbert calculi usable in practice. And, if a Hilbert style presentation is not suitable, it may be possible to exploit a combination of the deduction theorem and the intuitions of $\rightarrow$-frameworks to provide direct encodings of natural deduction [Matthews, 1997b]. The same provisos about lack of experience with effective notations for handling bindings apply here though, since this work only discusses the propositional case.

8 CONCLUSIONS

This chapter does not try to give a final answer to the question of what a logical framework might be. Rather it argues that the question is only meaningful in terms of some particular set of requirements, and in terms of 'goodness of fit'; i.e. the relationship between the properties of a proposed metalogic and logics we want to encode.

Our central theme has been the relationship between different kinds of deductive systems and their abstractions as metatheories or metalogics, which we can use to encode and work with instances of them. We have showed that a logic of minimal implication and universal quantification can be used to encode both the language and the proof-rules of a natural deduction or sequent calculus, and then described in detail a particular logic of this type, λ^{p}. As a contrast, we have also considered how (especially Hilbert) calculi can be abstracted as inductive definitions and we sketched a framework based on this view, FS_0. We then used the metatheoretic facilities that we get with an ID-framework like FS_0 to explore the relationship between metatheory and object theory, especially in the context of structured collections of theories.

The simple binary distinction between ID and $\rightarrow$-frameworks, which we make for the sake of space and explication, of course does not describe the whole range of frameworks that have been proposed and investigated. It does, however, help to define the space of possibilities and current research into frameworks can mostly be categorized in its terms.

For instance, research into the problem of the 'goodness of fit' relation between the metalogic and the object logic, especially for $\rightarrow$-frameworks, can be separated into two parts. We have shown that natural deduction calculi for standard mathematical (i.e., classical or intuitionistic, first or higher-order) logics fit well into an $\rightarrow$-framework. But the further we diverge from standard, mathematical, logics into philosophical (i.e. modal, relevance, etc.) logic the more complex and artificial the encodings become. In order to encode modal logics, for instance, we might introduce either multiple-judgment encodings or take explicit account of a semantics via a labelling. The particular encodings that we have described here are only some among a range of possibilities that could be imagined. A more radical possibility, not discussed here, can be found in, e.g., [Matthews, 1997a], where the possibility of extending a framework directly with a modal 'validity' connective is explored. However we do not yet know what the practical limits of these approaches are.[32] A similar problem of goodness of fit is also encountered in ID-frameworks, where the 'natural' deductive systems are Hilbert calculi and the 'obvious' encodings of consequence style systems are impractically unwieldy. Matthews [1997b] suggests how we might encode pure ordinary consequence relation based systems in such a framework in a way that is more effective than, and at least as intuitive as, the 'naive' approach of using inductively defined classes.

The particular problems of substructural logics (e.g. linear or relevance logics) have been the subject of substantial research. The consequence relations associated with these logics are not ordinary and hence cannot be encoded using techniques such as those suggested in §6. While labelled or multiple judgment presentations of these logics in an $\rightarrow$-framework are possible, they seem to be unwieldy; i.e. they 'fit' particularly badly. An alternative approach has been explored where the framework itself is modified: minimal implication is replaced or augmented in the framework logic with a substructural or linear implication, which does permit a good fit. In Ishtiaq and Pym [1998] and Cervesato and Pfenning [1996], systems are presented that are similar to λ^{P} except that they are based on linear implication, which is used to encode variants of linear and other relevance logics. There is also work that, rather than employing either $\rightarrow$ or ID-frameworks, attempts to combine features of both (e.g. McDowell and Miller [1997] and Despeyroux *et al.* [1996]).

In short, then, there are many possibilities and, in the end, no absolute solutions: the suitability of a particular logical framework to a particular circumstance depends on empirical as well as theoretical issues; i.e. before we can choose we have to decide on the range of object logics we envision formalizing, the nature of the metatheoretic facilities that we want, and the kinds of compromises that we are willing to accept.

REFERENCES

[Abadi *et al.*, 1991] Martín Abadi, Luca Cardelli, Pierre-Louis Curien, and Jean-Jacques Lévy. Explicit substitutions. *J. Functional Programming*, 1:375–416, 1991.

[Aczel, 1977] Peter Aczel. An introduction to inductive definitions. In Jon Barwise, editor, *Handbook of Mathematical Logic*. North-Holland, Amsterdam, 1977.

[32] This is essentially a practical, not a theoretical question, since, as we pointed out earlier, an $\rightarrow$-framework can be used as a Turing complete programming language, so with sufficient ingenuity any deductive system can be encoded.

[Avron *et al.*, 1992] Arnon Avron, Furio Honsell, Ian Mason, and Robert Pollack. Using typed lambda calculus to implement formal systems on a machine. *J. Auto. Reas.*, 9:309–352, 1992.

[Avron *et al.*, 1998] Arnon Avron, Furio Honsell, Marino Miculan, and Cristian Paravano. Encoding modal logics in logical frameworks. *Studia Logica*, 60(1):161–208, 1998.

[Avron, 1990] Arnon Avron. Gentzenizing Shroeder-Heister's natural extension of natural deduction. *Notre Dame Journal of Formal Logic*, 31:127–135, 1990.

[Avron, 1991] Arnon Avron. Simple consequence relations. *Inform. and Comput.*, 92:105–139, 1991.

[Avron, 1992] Arnon Avron. Axiomatic systems, deduction and implication. *J. Logic Computat.*, 2:51–98, 1992.

[Barendregt, 1984] Henk Barendregt. *The Lambda Calculus: Its Syntax and Semantics*. North-Holland, Amsterdam, 2nd (revised) edition, 1984.

[Barendregt, 1991] Henk Barendregt. Introduction to generalized type sytems. *J. Functional Programming*, 2:125–154, 1991.

[Barendregt, 1992] Henk Barendregt. Lambda calculi with types. In Samson Abramsky, Dov Gabbay, and Tom S. E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 2. Oxford University Press, 1992.

[Basin and Matthews, 2000] David Basin and Sean Matthews. Structuring metatheory on inductive definitions. *Information and Computation*, 162(1–2), October/November 2000.

[Basin *et al.*, 1997a] David Basin, Seán Matthews, and Luca Viganò. Labelled propositional modal logics: theory and practice. *J. Logic Computat.*, 7:685–717, 1997.

[Basin *et al.*, 1997b] David Basin, Seán Matthews, and Luca Viganò. A new method for bounding the complexity of modal logics. In Georg Gottlob, Alexander Leitsch, and Daniele Mundici, editors, *Proc. Kurt Gödel Colloquium, 1997*, pages 89–102. Springer, Berlin, 1997.

[Basin *et al.*, 1998a] David Basin, Seán Matthews, and Luca Viganò. Labelled modal logics: Quantifiers. *J. Logic, Language and Information*, 7:237–263, 1998.

[Basin *et al.*, 1998b] David Basin, Seán Matthews, and Luca Viganò. Natural deduction for non-classical logics. *Studia Logica*, 60(1):119–160, 1998.

[Boolos, 1993] George Boolos. *The Logic of Provability*. Cambridge University Press, 1993.

[Boyer and Moore, 1981] Robert Boyer and J. Strother Moore. *A Computational Logic*. Academic Press, New York, 1981.

[Bull and Segerberg, 1984] Robert Bull and Krister Segerberg. Basic modal logic. In Gabbay and Guenthner [1983–89], chapter II.1.

[Cervesato and Pfenning, 1996] Iliano Cervesato and Frank Pfenning. A linear logical framework. In *11th Ann. Symp. Logic in Comp. Sci.* IEEE Computer Society Press, 1996.

[Church, 1940] Alonzo Church. A formulation of the simple theory of types. *J. Symbolic Logic*, 5:56–68, 1940.

[Davis, 1989] Martin Davis. Emil Post's contributions to computer science. In *Proc. 4th IEEE Ann. Symp. Logic in Comp. Sci.* IEEE Computer Society Press, 1989.

[de Bruijn, 1972] Nicolas G. de Bruijn. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem. *Indagationes Mathematicae*, 34:381–392, 1972.

[de Bruijn, 1980] Nicolas G. de Bruijn. A survey of the project Automath. In J. R. Hindley and J. P. Seldin, editors, *To H. B. Curry: Essays in Combinatory Logic, Lambda Calculus and Formalism*, pages 579–606. Academic Press, New York, 1980.

[Despeyroux *et al.*, 1996] Joëlle Despeyroux, Frank Pfenning, and Carsten Schürmann. Primitive recursion for higher-order abstract syntax. Technical Report CMU-CS-96-172, Dept. of Computer Science, Carnegie Mellon University, September 1996.

[Dummett, 1978] Michael Dummett. The philosophical basis of intuitionistic logic. In *Truth and other enigmas*, pages 215–247. Duckworth, London, 1978.

[Fagin *et al.*, 1992] Ronald Fagin, Joseph Y. Halpern, and Moshe Y. Vardi. What is an inference rule? *J. Symbolic Logic*, 57:1018–1045, 1992.

[Feferman, 1990] Solomon Feferman. Finitary inductive systems. In *Logic Colloquium '88*, pages 191–220. North-Holland, Amsterdam, 1990.

[Felty, 1989] Amy Felty. *Specifying and Implementing Theorem Provers in a Higher Order Programming Language*. PhD thesis, University of Pennsylvania, 1989.

[Felty, 1991] Amy Felty. Encoding dependent types in an intuitionistic logic. In Huet and Plotkin [1991].

[Gabbay and Guenthner, 1983–89] Dov Gabbay and Franz Guenthner, editors. *Handbook of Philosophical Logic, vol. I–IV*. Reidel, Dordrecht, 1983–89.

[Gabbay, 1996] Dov Gabbay. *Labelled Deductive Systems, vol. 1*. Clarendon Press, Oxford, 1996.

[Gardner, 1995] Philippa Gardner. Equivalences between logics and their representing type theories. *Math. Struct. in Comp. Science*, 5:323–349, 1995.

[Gentzen, 1934] Gerhard Gentzen. Untersuchen über das logische Schließen. *Math. Z.*, 39:179–210, 405–431, 1934.

[Gödel, 1931] Kurt Gödel. Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I. *Monatsh. Math.*, 38:173–198, 1931.

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

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

[Gordon *et al.*, 1979] Michael J. Gordon, Robin Milner, and Christopher P. Wadsworth. *Edinburgh LCF: A Mechanized Logic of Computation*. Springer, Berlin, 1979.

[Griswold, 1981] Ralph E. Griswold. A history of the Snobol programming languages. In Wexelblat [1981], pages 601–660.

[Hacking, 1979] Ian Hacking. What is logic? *J. Philosophy*, 76(6), 1979.

[Harper *et al.*, 1993] Robert Harper, Furio Honsell, and Gordon Plotkin. A framework for defining logics. *J. Assoc. Comput. Mach.*, 40:143–184, 1993.

[Hindley and Seldin, 1986] J. Roger Hindley and Jonathan P. Seldin. *Introduction to Combinators and λ-Calculus*. Cambridge University Press, Cambridge, 1986.

[Howard, 1980] William Howard. The formulas-as-types notion of construction. In J. P. Seldin and J. R. Hindley, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda-Calculus, and Formalism*. Academic Press, New York, 1980.

[Huet and Plotkin, 1991] Gérard Huet and Gordon Plotkin, editors. *Logical Frameworks*. Cambridge University Press, Cambridge, 1991.

[Ishtiaq and Pym, 1998] Samin Ishtiaq and David Pym. A relevant analysis of natural deduction. *J. Logic Computat.*, 8:809–838, 1998.

[Kleene, 1952] Stephen C. Kleene. *Introduction to Metamathematics*. North-Holland, Amsterdam, 1952.

[Kripke, 1963] Saul A. Kripke. Semantical analysis of modal logic I: normal propositional modal logic. *Z. Math. Logik Grundlag. Math*, 8:67–96, 1963.

[Martí-Oliet and Meseguer, 2002] Narciso Martí-Oliet and José Meseguer. Rewriting logic as a logical and semantic framework. In *Handbook of Philosophical Logic*. second edition, 2002.

[Matthews *et al.*, 1993] Seán Matthews, Alan Smaill, and David Basin. Experience with FS_0 as a framework theory. In Gérard Huet and Gordon Plotkin, editors, *Logical Environments*. Cambridge University Press, Cambridge, 1993.

[Matthews, 1992] Seán Matthews. *Metatheoretic and Reflexive Reasoning in Mechanical Theorem Proving*. PhD thesis, University of Edinburgh, 1992.

[Matthews, 1994] Seán Matthews. A theory and its metatheory in FS_0. In Dov Gabbay, editor, *What is a Logical System?* Clarendon Press, Oxford, 1994.

[Matthews, 1996] Seán Matthews. Implementing FS_0 in Isabelle: adding structure at the metalevel. In Jacques Calmet and Carla Limongelli, editors, *Proc. Disco'96*. Springer, Berlin, 1996.

[Matthews, 1997a] Seán Matthews. Extending a logical framework with a modal connective for validity. In Martín Abadi and Takayasu Ito, editors, *Proc. TACS'97*. Springer, Berlin, 1997.

[Matthews, 1997b] Seán Matthews. A practical implementation of simple consequence relations using inductive definitions. In William McCune, editor, *Proc. CADE–14*. Springer, Berlin, 1997.

[McCarthy, 1981] John McCarthy. History of Lisp. In Wexelblat [1981], pages 173–197.

[McDowell and Miller, 1997] Raymond McDowell and Dale 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.

[Meseguer, 1989] José Meseguer. General logics. In Heinz-Dieter Ebbinghaus, J. Fernandez-Prida, M. Garrido, D. Lascar, and M. Rodriguez Artalejo, editors, *Logic Colloquium, '87*, pages 275–329. North-Holland, 1989.

[Nederpelt *et al.*, 1994] Rob P. Nederpelt, Herman J. Geuvers, and Roel C. de Vrijer, editors. *Selected papers on Automath*. Elsevier, Amsterdam, 1994.

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

[Owre *et al.*, 1995] Sam Owre, John Rushby, Natarajan Shankar, and Friedrich von Henke. Formal verification for fault-tolerant architectures: Prolegomena to the design of PVS. *IEEE Trans. Software Eng.*, 21:107–125, 1995.

[Paulson, 1994] Lawrence C. Paulson. *Isabelle: A Generic Theorem Prover*. Springer, Berlin, 1994.

[Pfenning, 1996] Frank Pfenning. The practice of logical frameworks. In Helene Kirchner, editor, *Proc. CAAP'96*. Springer, Berlin, 1996.

[Pfenning, 2000] Frank Pfenning. Structural cut elimination I. intuitionistic and classical logic. *Information and Computation*, 157(1–2), March 2000.

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

[Post, 1943] Emil Post. Formal reductions of the general combinatorial decision problem. *Amer. J. Math.*, 65:197–214, 1943.

[Prawitz, 1965] Dag Prawitz. *Natural Deduction*. Almqvist and Wiksell, Stockholm, 1965.

[Prawitz, 1971] Dag Prawitz. Ideas and results in proof theory. In J. E. Fensted, editor, *Proc. Second Scandinavian Logic Symp.*, pages 235–307. North-Holland, Amsterdam, 1971.

[Pym and Wallen, 1991] David J. Pym and Lincoln Wallen. Proof-search in the $\lambda\Pi$-calculus. In Huet and Plotkin [1991], pages 309–340.

[Schroeder-Heister, 1984a] Peter Schroeder-Heister. Generalised rules for quantifiers and the completeness of the intuitionistic operators $\&$, $\vee$, $\supset$, $\perp$, $\forall$, $\exists$. In M. M. Richter et al., editors, *Computation and proof theory*. Springer, Berlin, 1984.

[Schroeder-Heister, 1984b] Peter Schroeder-Heister. A natural extension of natural deduction. *J. Symbolic Logic*, 49:1284–1300, 1984.

[Scott, 1974] Dana Scott. Rules and derived rules. In S. Stenlund, editor, *Logical Theory and Semantical Analysis*, pages 147–161. Reidel, Dordrecht, 1974.

[Simpson, 1992] Alex K. Simpson. Kripke semantics for a logical framework. In *Proc. Workshop on Types for Proofs and Programs*, Båstad, 1992.

[Smullyan, 1961] Raymond Smullyan. *Theory of Formal Systems*. Princeton University Press, 1961.

[Steele Jr. and Gabriel, 1996] Guy L. Steele Jr. and Richard P. Gabriel. The evolution of Lisp. In Thomas J. Bergin and Richard G. Gibson, editors, *History of Programming Languages*, pages 233–330. ACM Press, New York, 1996.

[Sundholm, 1983] Göran Sundholm. Systems of deduction. In Gabbay and Guenthner [1983–89], chapter I.2.

[Sundholm, 1986] Göran Sundholm. Proof theory and meaning. In Gabbay and Guenthner [1983–89], chapter III.8.

[Talcott, 1993] Carolyn Talcott. A theory of binding structures, and applications to rewriting. *Theoret. Comp. Sci.*, 112:99–143, 1993.

[Troelstra, 1982] A. S. Troelstra. *Metamathematical Investigation of Intuitionistic Arithmetic and Analysis*. Springer, Berlin, 1982.

[van Benthem, 1984] Johan van Benthem. Correspondence theory. In Gabbay and Guenthner [1983–89], chapter II.4.

[Wexelblat, 1981] Richard L. Wexelblat, editor. *History of Programming Languages*. Academic Press, New York, 1981.