LFCS

Laboratory for Foundations of Computer Science
Department of Computer Science - University of Edinburgh

Using Typed Lambda Calculus to Implement Formal Systems on a Machine

Using Typed Lambda Calculus to Implement Formal Systems on a Machine

by

Arnon Avron, Furio A. Honsell and Ian A. Mason

LFCS Report Series

ECS-LFCS-87-31
(also published as CSR-237-87)

July 1987

LFCS
Department of Computer Science
University of Edinburgh
The King's Buildings
Edinburgh EH9 3JZ

Contents

Using Typed Lambda Calculus to Implement Formal Systems on a Machine

Arnon Avron, Furio Honsell and Ian A. Mason

June 12, 1987

1 Introduction

In recent years there has been a growing interest in using computers as an aid for correctly manipulating formal systems. Much research has been devoted in building computer systems for checking proofs (Automath project [5,13] and the Theory of Constructions [8]) or for developing interactively correct proofs (Edinburgh and Gothenburg LCF [11,18,20]) and Nuprl [7] in specific logical systems.

The Edinburgh Logical Framework (ELF) is a logic independent proof editor and proof checker currently under development at the Laboratory for Foundations of Computer Science, Edinburgh.

The diversity amongst logics is almost as great as the diversity amongst the people that use them. For example free, tense, linear, quantum and relevance logics are of interest to philosophers, physicists, linguists and computer scientists, not to mention the case of logicians and mathematicians and first order logic, or even the various types of formal systems (for example natural deduction, Hilbert style, sequent calculus and tableaux). Since the task of implementing any particular logic on a machine is quite demanding (e.g. one must deal with such issues as: syntax; parsing; pretty printing; proof rules; derived rules; higher-order rules; tactics; strategies; definitions; abbreviations; lemmas; theorems; theories and libraries) a somewhat more uniform and less time consuming alternative is desirable. It is therefore indispensable, if such an application of computers is going to progress, to understand which aspects of this task are common to all logics so that these could be carried out once and for all. This is essentially the idea behind the ELF: to provide a conceptual buffer between the people who use these logics and the machine they are to be implemented on. The scientist, philosopher or linguist need then only specify the logic in question within the ELF to obtain a proof editor (with facilities such as those mentioned above) for that logic. The ELF is thus an attempt to provide such a general theory of logics,

one that captures the uniformities and idiosyncrasies of as large a class of logics as possible without sacrificing generality for tractability. It should be emphasized that we wish to capture a logic's peculiarities (for example side conditions on rule application, variable occurrence and formula formation) at the level of the framework rather than that of the implementation.

The paper [12] describes the basic features of such general logical framework. In this paper we are going to give a broader illustration of the applicability of the ELF and discuss to what extent it is successful in expressing uniformities among logics or in capturing their peculiarities.

2 The Edinburgh Logical Framework

The ELF is a weak constructive type theory i.e. a Π-typed λ-calculus closely related to AUT-PI and AUT-QE [5], to Martin Löf's early type theory [14] and to Meyer and Reinhold's λ^* [17]. The expressive power of ELF is enough to specify both the language of a logic, its axioms, rules schemes and its proofs.

Ordinary descriptions of formal systems or of the inferential activity within them rely heavily on schematic, parametric and abstracted reasoning. For instance syntax and rules are always described through schemas. Ordinary proofs, and even rules, are implicitly treated as functional objects mapping proofs of premisses to conclusions (in the case of rules) and conjectures or proofs of lemmas to complete proofs (in the case of proofs). One could also go as far as to say that proof checking itself consists mainly of checking the correctness of instantiation of schemas and the application of rules (both basic and derived) to premisses or proofs thereof. This explains why a λ-calculus can naturally play a fundamental role in the specification and manipulation of logics.

One of the basic ideas of ELF is to reduce, whenever possible,

1. all forms of schematic abstraction and parameterization involved in defining and using a formal system to λ-abstractions;

2. all forms of schematic instantiation to λ-application;

3. all forms of substitution to β-reduction;

The correctness of any of the above activities can then be enforced through type matching and checking.

The ELF type theory is a language with three levels of entities:

1. objects,

2. types and families of types, and

3. kinds.

Objects are classified by types, types and families of types by kinds. The kind Type classifies the types; the other kinds classify functions f which yield a type $f(x_1)\ldots(x_n)$ when applied to objects $x_1,\ldots,x_n$ of certain types determined by the kind of f. Any function definable in the system has a type as domain, while its range can either be a type, if it is an object, or a kind, if it is a family of types. The ELF type theory is therefore predicative.

A number of different presentations of this system can be given. We shall describe a more compact version of the one given in [12]. The theory we shall deal with is a formal system for deriving assertions of one of the following shapes:

$\Gamma \vdash_\Sigma K$	K is a kind
$\Gamma \vdash_\Sigma A : K$	A has kind K
$\Gamma \vdash_\Sigma M : A$	M has type A

where the syntax is specified by the following grammar:

Signatures	Σ	::=	$\langle\rangle \mid \Sigma, c : K \mid \Sigma, c : A$
Contexts	Γ	::=	$\langle\rangle \mid \Gamma, x : A$
Kinds	K	::=	$\text{Type} \mid \prod_{x:A} . K$
Type Families	A	::=	$c \mid \prod_{x:A} . B \mid \lambda x : A.B \mid AM$
Objects	M	::=	$c \mid x \mid \lambda x : A.M \mid MN$

We let M and N range over expressions for objects, A and B for types and families of types, K for kinds, x and y over variables, and c over constants. We write $A \to B$ for $\prod_{x:A} . B$ when x does not occur free in B. The inference rules of the ELF type theory are listed below, according to which of the three forms of assertions they concern, α-conversion is assumed throughout.

1. **Valid Kinds**

$$\frac{}{\vdash \text{Type}} \tag{1}$$

$$\frac{\Gamma \vdash_\Sigma A : \text{Type} \quad \Gamma, x : A \vdash_\Sigma K}{\Gamma \vdash_\Sigma \prod_{x:A} . K} \tag{2}$$

2. **Valid Elements of a Kind**

$$\frac{\Gamma \vdash_\Sigma \text{Type} \quad c : K \in \Sigma}{\Gamma \vdash_\Sigma c : K} \tag{3}$$

$$\frac{\vdash_\Sigma K \quad c \notin \text{Dom}(\Sigma)}{\vdash_{\Sigma, c:K} \text{Type}} \tag{4}$$

$$\frac{\Gamma \vdash_\Sigma A : \text{Type} \quad \Gamma, x : A \vdash_\Sigma B : \text{Type}}{\Gamma \vdash_\Sigma \prod_{x:A} . B : \text{Type}} \tag{5}$$

$$\frac{\Gamma \vdash_\Sigma A : \text{Type} \quad \Gamma, x : A \vdash_\Sigma B : K}{\Gamma \vdash_\Sigma \lambda x : A.B : \prod_{x:A} . K} \tag{6}$$

$$\frac{\Gamma \vdash_\Sigma B : \prod_{x:A} . K \quad \Gamma \vdash_\Sigma N : A}{\Gamma \vdash_\Sigma BN : [N/x]K} \tag{7}$$

$$\frac{\Gamma \vdash_\Sigma A : K \quad \Gamma \vdash_\Sigma K' \quad K =_{\beta\eta} K'}{\Gamma \vdash_\Sigma A : K'} \tag{8}$$

3. Valid Elements of a Type

$$\frac{\vdash_\Sigma A : \text{Type} \quad c \notin \text{Dom}(\Sigma)}{\vdash_{\Sigma, c:A} \text{Type}} \tag{9}$$

$$\frac{\Gamma \vdash_\Sigma A : \text{Type} \quad x \notin \text{Dom}(\Gamma)}{\Gamma, x : A \vdash_\Sigma \text{Type}} \tag{10}$$

$$\frac{\Gamma \vdash_\Sigma \text{Type} \quad M : A \in \Sigma \cup \Gamma}{\Gamma \vdash_\Sigma M : A} \tag{11}$$

$$\frac{\Gamma \vdash_\Sigma A : \text{Type} \quad \Gamma, x : A \vdash_\Sigma M : B}{\Gamma \vdash_\Sigma \lambda x : A.M : \prod_{x:A} . B} \tag{12}$$

$$\frac{\Gamma \vdash_\Sigma M : \prod_{x:A} . B \quad \Gamma \vdash_\Sigma N : A}{\Gamma \vdash_\Sigma MN : [N/x]B} \tag{13}$$

$$\frac{\Gamma \vdash_\Sigma M : A \quad \Gamma \vdash_\Sigma A' : \text{Type} \quad A =_{\beta\eta} A'}{\Gamma \vdash_\Sigma M : A'} \tag{14}$$

A term is said to be *well-typed in a signature and context* if it can be shown to either be a kind, have a kind, or have a type in that signature and context. A term is *well-typed* if it is well-typed in some signature and context. The notion of $\beta\eta$–contraction, written $\rightarrow_{\beta\eta}$, can be defined both at the level of objects and at the level of types and families of types in the obvious way. Rules (8) and (14) make use of a relation $=_{\beta\eta}$ between terms which is defined as follows: $M =_{\beta\eta} N$ iff $M \rightarrow^*_{\beta\eta} P$ and $N \rightarrow^*_{\beta\eta} P$ for some term P. The following theorem from [12] summarizes the basic theoretical facts about ELF (here α ranges over the basic assertions of the type theory):

Theorem 1

1. **Thinning:** *thinning is an admissible rule: if* $\Gamma \vdash_\Sigma \alpha$ *and* $\Gamma, \Gamma' \vdash_{\Sigma,\Sigma'} \text{Type}$, *then* $\Gamma, \Gamma' \vdash_{\Sigma,\Sigma'} \alpha$.
2. **Transitivity:** *transitivity is an admissible rule: if* $\Gamma \vdash_\Sigma M : A$ *and* $\Gamma, x : A, \Delta \vdash_\Sigma \alpha$, *then* $\Gamma, [M/x]\Delta \vdash_\Sigma [M/x]\alpha$.

3. **Uniqueness of types and kinds:** *if* $\Gamma \vdash_\Sigma M : A$ *and* $\Gamma \vdash_\Sigma M : A'$, *then* $A =_{\beta\eta} A'$, *and similarly for kinds.*

4. **Subject reduction:** *if* $\Gamma \vdash_\Sigma M : A$ *and* $M \rightarrow^*_{\beta\eta} M'$, *then* $\Gamma \vdash_\Sigma M' : A$, *and similarly for types.*

5. **Confluence:** *all well-typed terms are Church–Rosser, while in general this is false.*

6. **Strong Normalization:** *all well-typed terms are strongly normalizing.*

7. **Decidability:** *each of the three relations defined by the inference system of the ELF is decidable, as is the property of being well-typed.*

8. **Predicativity:** *if* $\Gamma \vdash_\Sigma M : A$ *then the type free* λ*-term obtained by erasing all type information from* M *can be typed in the Curry type assignment system.*

3 The ELF Paradigm for Specifying a Logical System

In the ELF a logical system is specified by means of an ELF valid signature (i.e. a signature, Σ, in which $\vdash_\Sigma$ Type can be derived, Σ is of course finite). The ELF theory of expressions deals with the issue of determining that part of the signature which encodes the language of the logical system. On the other hand the ELF theory of judgementsdeals with the encoding in the signature of the rules of the logical system and the representation in ELF of the proofs of the given logical system (see [12] for a detailed description of these theories and their philosophical motivations).

The ELF paradigm for specifying the language of a logical system is based on a thorough exploitation of the typed λ-calculus structure of the framework. It is inspired by Church [6] and Martin-Löf [15]. The fundamental idea is to model syntactic categories with ELF types and expression constructors with ELF elements of the appropriate type. Object language variables and schematic variables are then modeled by ELF variables of the appropriate type. A schematic expression, of a given syntactic category, in certain schematic variables can therefore be expressed as the λ-abstraction of that expression with respect to the schematic variables. Finally binding operators are modeled as expression constructors ranging over schemas.

The paradigm outlined above is satisfactory when the language under consideration is given, roughly, by an unambiguous context-free grammar. Well formed expressions of ordinary logical systems, however, are often defined by formal systems of a much higher complexity which cannot be directly encoded in the weak

type theory of ELF. The ELF paradigm must therefore be refined by making additional use of the ELF theory of judgements. Rather than giving a theoretical account of the various possibilities available for handling complex languages, we will leave to the force of the examples in the sequel to illustrate the major techniques which can be used in this regard. We remark that the implicit identification the ELF utilizes between object language variable and schematic variables (over terms of the language) will often suggest similarities between the ELF translation of a system and a denotational model of that system (see for example the internalizations of the various lambda calculii).

An ELF translation of a language, however, will be satisfactory only if *adequate*. Informally an ELF signature will be adequate iff for each syntactic category of the language there is a *compositional* surjection from the ELF type, corresponding to that category, onto the category itself.

That part of an ELF signature, for a given logical system, that corresponds to the language is therefore a declaration of constants of appropriate kind or type. One for each basic syntactic category and each syntactic constructor of the language.

The ELF paradigm for specifying and handling rules and proofs is centred on the notion of *judgement*. This notion was introduced by Martin-Löf [15] and corresponds to the notion of *assertion* of a formal system. However the ELF does not commit itself to the intuitionistic viewpoint and actually, as explained below, extends the meaning of this notion.

A logical system can be construed as a system for establishing proofs of basic judgements (assertions) of certain forms by means of proofs of hypothetico-general judgements (rules). Again ELF thoroughly exploits the typed λ-calculus structure in modeling these notions. Judgements are modeled as types, which explicitly depend on the type of the expressions they are concerned with. Basic judgements correspond to basic dependent types while hypothetico-general judgements correspond to higher order judgement types. Elements of a judgement type are used to represent the evidence for that judgement (in Martin-Löf's terminology) i.e. proofs, rules and derived rules. The ELF signature corresponding to the rules of a logical system is a list of declarations of judgement types of the appropriate kind (corresponding to the assertions of the system) and of constants of the appropriate higher order judgement type (corresponding to the rules and axioms of the system). Also in this context schemas such as rule schemas are modeled by means of λ-abstractions. One of the major benefits of this approach is that proofs of theorems and of derived rules are treated on the same logical level.

An ELF type encodes an *open concept* (see theorem 1.1). This implies that the notion of proof ELF is actually encoding is not determined completely, it is not merely the notion of *proof of logical theorem* in a fixed system. Therefore a

judgement J, which is used for internalizing a formal system, truly corresponds to a *consequence relation* (which is definable in that system), not just to its theorems. Accordingly, a term of type $J(\phi) \to J(\psi)$ does not encode just a function which transfers a proof that ϕ is a logical theorem to a proof that ψ is also a logical theorem (the system may even lack logical theorems altogether —see example 1.1). Such a term encodes a proof that ψ *follows* from ϕ in the system. More accurately the assertion that J holds of ψ follows from the assumption that J holds of ϕ. A proof of a hypothetical judgement therefore corresponds to either a rule of derivation or a derivable rule of the system, not simply a rule of proof or an admissible rule. For example, a constant $\wedge I$ of type $T(\phi) \to T(\psi) \to T(\phi \wedge \psi)$ (which we have in the standard internalizations of classical and intuitionistic logics) would be inadequate for relevance or linear logic, despite the fact that $\phi \wedge \psi$ is a theorem of these logics whenever ϕ and ψ are. The reason is that it is not always possible to infer $\phi \wedge \psi$ from the two conjucts in these logics.

The consequence relations which are encoded by the ELF's judgements are ordinary, single-conclusioned consequence relations [1]. The basic idea is quite simple: a proof of a sequent $\phi_1, \ldots, \phi_n \vdash \psi$ is encoded by a term of type $J(\phi_1) \to J(\phi_2) \to \cdots J(\phi_n) \to J(\psi)$, where J is a judgement that is induced by $\vdash$. It is very important to note, however, that the type structure of the ELF makes it possible to formulate and prove also higher-order logical facts about an internalized consequence relation (see example 4.1) or even logical facts relating two or more such relations (see examples 4.3, 4.4 and 4.5).

As was already emphasized above, in the ELF we try to achieve as much uniformity as possible. The attempt to translate, whenever possible, the variables of a system with the ELF variables is one way in which this attempt manifests itself. Here we have another: whenever possible, rules of arbitrary order are encoded using the single ELF $\to$ primitive. For example, the full standard formulation of the elimination rule for $\vee$ (in classical logic) is

$$\frac{\Gamma_1, \phi \vdash \vartheta \qquad \Gamma_2, \psi \vdash \vartheta \qquad \Gamma_3 \vdash \phi \vee \psi}{\Gamma_1, \Gamma_2, \Gamma_3 \vdash \vartheta}$$

In this formulation $\vdash$ is the basic consequence relation between *sentences*, while the "fraction" symbol corresponds to a higher order rule concerning *sequents*. In the ELF both are translated using $\to$ by introducing a constant $\vee$E of type:

$$\prod_{\phi:o} \prod_{\psi:o} \prod_{\vartheta:o} (T(\phi) \to T(\vartheta)) \to (T(\psi) \to T(\vartheta)) \to T(\phi \vee \psi) \to T(\vartheta)$$

It is worth noting that $\vee$E corresponds, in fact, to something stronger than the rule it internalizes, since in the original rule the assumptions in the Γ_i's were intended to be just sentences, while in the ELF the application of $\vee E$ is possible under any set of assumptions of arbitrary order.

It is a major part of the ELF paradigm that such identifications and uniformities should be possible for a natural deduction formalism which deserves this name. It can be taken as a kind of an "ELF thesis" that well-behaved natural-deduction formalisms are those that can be directly encoded in the ELF, and also that given a formal representation of a consequence relation the notion of a derivable rule (of arbitrary order) is *defined* by the non-emptiness of the ELF type encoding its specification in the corresponding signature.

Definitions of adequacy and faithfullness of signatures can be given even with respect to consequence relations and proofs thereof.

4 Case Studies in the ELF

4.1 Kleene's Three-Valued Logic

4.1.1 The System

An interesting example is provided by Kleene's three-valued logic. Syntactically this is identical to classical propositional logic (with the connectives $\neg, \vee$ and $\wedge$). Semantically this logic is based on the truth-values 0, 1 and -1, of which only 1 is taken to be designated. The operations which correspond to the logical connectives are defined as follows:

- $\phi \vee \psi = \min(\phi, \psi)$
- $\phi \wedge \psi = \max(\phi, \psi)$
- $\neg\phi = -\phi$.

A fundamental property of the resulting logic is that it has no theorems. No sentence has the designated value 1 under all possible assignments. This property is not significant from the ELF point of view. What is significant is the consequence relation that is naturally associated with the above logic.

- $\phi_1, \ldots, \phi_n \vdash_{\mathrm{Kl}} \psi$ if and only if $\nu(\psi) = 1$ for all assignments ν such that $\nu(\phi_1) = \ldots = \nu(\phi_n) = 1$.

Accordingly, the status of Kleene's logic with respect to the ELF depends, as usual, on the existence of a reasonable natural deduction representation for this consequence relation. One in which $\phi_1, \ldots, \phi_n \vdash_{\mathrm{Kl}} \psi$ if and only if there is a proof of ψ from the assumptions $\phi_1, \ldots, \phi_n$. Such a formal system is described in [3]. Its internalization in the ELF is as easy a task as that of classical logic. The resulting signature is:

4.1.2 The Signature Σ_{KI}

- Syntactic Categories
  - o : Type
- Operations
  - $\neg : o \to o$
  - $\wedge : o \to o \to o$
  - $\vee : o \to o \to o$
- Judgement
  - $\mathrm{T} : o \to \mathrm{Type}$
- Axioms and Rules
  - $\wedge\mathrm{I} : \prod_{\phi,\psi:o} . \mathrm{T}(\phi) \to \mathrm{T}(\psi) \to \mathrm{T}(\phi \wedge \psi)$
  - $\vee\mathrm{I}_l : \prod_{\phi,\psi:o} . \mathrm{T}(\phi) \to \mathrm{T}(\phi \vee \psi)$
  - $\vee\mathrm{I}_r : \prod_{\phi,\psi:o} . \mathrm{T}(\psi) \to \mathrm{T}(\phi \vee \psi)$
  - $\neg\neg\mathrm{I} : \prod_{\phi:o} . \mathrm{T}(\phi) \to \mathrm{T}(\neg\neg\phi)$
  - $\neg\wedge\mathrm{I}_l : \prod_{\phi,\psi:o} . \mathrm{T}(\neg\phi) \to \mathrm{T}(\neg(\phi \wedge \psi))$
  - $\neg\wedge\mathrm{I}_r : \prod_{\phi,\psi:o} . \mathrm{T}(\neg\psi) \to \mathrm{T}(\neg(\phi \wedge \psi))$
  - $\neg\vee\mathrm{I} : \prod_{\phi,\psi:o} . \mathrm{T}(\neg\phi) \to \mathrm{T}(\neg\psi) \to \mathrm{T}(\neg(\phi \vee \psi))$
  - $\neg\mathrm{E} : \prod_{\phi,\psi:o} . \mathrm{T}(\phi) \to \mathrm{T}(\neg\phi) \to \mathrm{T}(\psi)$
  - $\wedge\mathrm{E}_l : \prod_{\phi,\psi:o} . \mathrm{T}(\phi \wedge \psi) \to \mathrm{T}(\phi)$
  - $\wedge\mathrm{E}_r : \prod_{\phi,\psi:o} . \mathrm{T}(\phi \wedge \psi) \to \mathrm{T}(\psi)$
  - $\neg\neg\mathrm{E} : \prod_{\phi:o} . \mathrm{T}(\neg\neg\phi) \to \mathrm{T}(\phi)$
  - $\neg\vee\mathrm{E}_l : \prod_{\phi,\psi:o} . \mathrm{T}(\neg(\phi \vee \psi)) \to \mathrm{T}(\neg\phi)$
  - $\neg\vee\mathrm{E}_r : \prod_{\phi,\psi:o} . \mathrm{T}(\neg(\phi \vee \psi)) \to \mathrm{T}(\neg\psi)$
  - $\vee\mathrm{E} : \prod_{\phi,\psi,\vartheta:o} . (\mathrm{T}(\phi) \to \mathrm{T}(\vartheta)) \to (\mathrm{T}(\psi) \to \mathrm{T}(\vartheta)) \to (\mathrm{T}(\phi \vee \psi) \to \mathrm{T}(\vartheta))$
  - $\neg\wedge\mathrm{E} : \prod_{\phi,\psi,\vartheta:o} .$

    $(\mathrm{T}(\neg\phi) \to \mathrm{T}(\vartheta)) \to (\mathrm{T}(\neg\psi) \to \mathrm{T}(\vartheta)) \to (\mathrm{T}(\neg(\phi \wedge \psi)) \to \mathrm{T}(\vartheta))$
- Example of a Proof
  - $\Delta : \Pi_{\phi,\psi:o} . \mathrm{T}(\neg\phi \vee \phi) \to (\mathrm{T}(\phi) \to \mathrm{T}(\psi)) \to (\mathrm{T}(\phi) \to \mathrm{T}(\neg\psi)) \to \mathrm{T}(\neg\phi)$

Where the term Δ is defined to be the following.

$$\lambda\phi : o.\lambda\psi : o.\lambda X : \mathrm{T}(\neg\phi \vee \phi).\lambda Y : \mathrm{T}(\phi) \to \mathrm{T}(\psi).\lambda Z : \mathrm{T}(\phi) \to \mathrm{T}(\neg\psi).$$
$$\vee\mathrm{E}(\neg\phi)(\phi)(\neg\phi)(\lambda W : \mathrm{T}(\phi).\neg\mathrm{E}(\psi)(\neg\phi)(Y(W))(Z(W))(\lambda v : \mathrm{T}(\phi).v))$$

Adequacy and Faithfulness Property 1 *The following hold*

$$\phi_1, \ldots, \phi_n \vdash_{\mathrm{KI}} \psi$$

iff there exists a term t such that:

$$p_1 : o, \ldots, p_n : o \vdash_{\Sigma_{\mathrm{KI}}} t : \mathrm{T}(\phi_1) \to \cdots \to \mathrm{T}(\phi_n) \to \mathrm{T}(\psi)$$

Where $p_1, \ldots, p_n$ are the atomic variables occuring in $\phi_i, 1 \leq i \leq n$ and ψ (we use the same symbol for denoting a formula and the corresponding term in ELF). Moreover, there is a compositional bijection between proofs in the natural deduction system and proof terms such as t above.

The above signature is quite similar to that for classical propositional logic. The $\neg I$ is the only missing constant.[1] However, unlike the classical (or intuitionistic) case, the $\to$ of the ELF does not correspond to any connective of the logic (not even a definable one.). Some of the types of the constants above strictly correspond, therefore, to higher order sequents and consequence relations. Thus, $\vee E$ means something like:

$$(\Gamma_1, \phi \vdash^1_{\mathrm{KI}} \vartheta), (\Gamma_2, \psi \vdash^1_{\mathrm{KI}} \vartheta), (\Gamma_3 \vdash^1_{\mathrm{KI}} \phi \vee \psi) \vdash^2_{\mathrm{KI}} (\Gamma_1, \Gamma_2, \Gamma_3 \vdash^1_{\mathrm{KI}} \vartheta)$$

The fact that $\to$ can be used to handle both $\vdash^1_{\mathrm{KI}}$ and $\vdash^2_{\mathrm{KI}}$ is in a perfect accordance to the ELF's paradigm concerning the properties of a well-behaved natural-deduction formalism.

The example of a proof given above is of an important higher-order logical fact about $\vdash_{\mathrm{KI}}$: by adding the excluded-middle as an axiom to the corresponding natural deduction formalism the usual classical introduction rule for negation becomes derivable. Thus such an addition is sufficent for obtaining classical logic.

4.2 First Order Logic with a Choice Operator

4.2.1 The System

In [12] the signature of first order logic was presented in detail, rather than duplicate it here we present a version with an additional binding operator. Thus

[1] Splitting the rules of $\neg$ into separate introduction and elimination rules for its combination with other connective is a known idea also in the context of classical logic, see e.g. [24], p. 66.

the logic we present here illustrates how one handles binding operators in the ELF with three examples. Apart from the quantifiers $\exists$ and $\forall$ we include ϵ, a version of Hilbert's choice operator. If x is a variable of type i, then $x = x$ is a term of type o. We can bind x by λ–abstraction obtaining an object of type $i \to o$, $\lambda x : i.x = x$. The binding operators applied to this give

1. $\epsilon(\lambda x : i.x = x)$, which represents the first–order term $\epsilon x.x = x$.
2. $\exists(\lambda x : i.x = x)$, which represents the first–order formula $\exists x.x = x$.
3. $\forall(\lambda x : i.x = x)$, which represents the first–order formula $\forall x.x = x$.

The only rule concerning the choice operator is the introduction rule:

$$\frac{\exists x \Phi(x)}{\Phi(\epsilon x \Phi(x))}$$

4.2.2 The Signature $\Sigma_{\epsilon 1^\circ}$

This signature encodes the consequence relation of *truth* rather than that of *validity*, see [1]

- Syntactic Categories
  - i : Type
  - o : Type
- Operations
  - $= \quad : i \to i \to o$
  - $\neg : o \to o$
  - $\supset : o \to o \to o$
  - $\epsilon : (i \to o) \to i$
  - $\exists : (i \to o) \to o$
  - $\forall : (i \to o) \to o$
- Judgement
  - $\mathrm{T} : o \to$ Type
- Axioms and Rules
  - $\mathrm{E}_0 : \Pi_{x:i}\,.\ \mathrm{T}(x = x)$

- $\mathrm{E}_1 : \prod_{x,y:i} . \prod_{t:i\to i} . \mathrm{T}(x = y) \to \mathrm{T}(t(x) = t(y))$
- $\mathrm{E}_2 : \prod_{x,y:i} . \prod_{\Phi:i\to o} . \mathrm{T}(x = y) \to \mathrm{T}(\Phi(x)) \to \mathrm{T}(\Phi(y))$
- $\neg\mathrm{I} : \prod_{\phi,\psi:o} . (\mathrm{T}(\phi) \to \mathrm{T}(\psi)) \to (\mathrm{T}(\phi) \to \mathrm{T}(\neg\psi)) \to \mathrm{T}(\neg\phi)$
- $\wedge\mathrm{I} : \prod_{\phi,\psi:o} . \mathrm{T}(\phi) \to \mathrm{T}(\psi) \to \mathrm{T}(\phi \wedge \psi)$
- $\vee\mathrm{I}_l : \prod_{\phi,\psi:o} . \mathrm{T}(\phi) \to \mathrm{T}(\phi \vee \psi)$
- $\vee\mathrm{I}_r : \prod_{\phi,\psi:o} . \mathrm{T}(\psi) \to \mathrm{T}(\phi \vee \psi)$
- $\wedge\mathrm{E}_l : \prod_{\phi,\psi:o} . \mathrm{T}(\phi \wedge \psi) \to \mathrm{T}(\phi)$
- $\wedge\mathrm{E}_r : \prod_{\phi,\psi:o} . \mathrm{T}(\phi \wedge \psi) \to \mathrm{T}(\psi)$
- $\neg\neg\mathrm{E} : \prod_{\phi:o} . \mathrm{T}(\neg\neg\phi) \to \mathrm{T}(\phi)$
- $\vee\mathrm{E} : \prod_{\phi,\psi,\vartheta:o} . (\mathrm{T}(\phi) \to \mathrm{T}(\vartheta)) \to (\mathrm{T}(\psi) \to \mathrm{T}(\vartheta)) \to (\mathrm{T}(\phi \vee \psi) \to \mathrm{T}(\vartheta))$
- $\supset\mathrm{E} : \prod_{\phi_1,\phi_2:o} . \mathrm{T}(\phi_1 \supset \phi_2) \to \mathrm{T}(\phi_1) \to \mathrm{T}(\phi_2)$
- $\supset\mathrm{I} : \prod_{\phi,\psi:o} . (\mathrm{T}(\phi) \to \mathrm{T}(\psi)) \to \mathrm{T}(\phi \supset \psi)$
- $\epsilon\mathrm{I} : \prod_{\Phi:i\to o} . \mathrm{T}(\exists(\Phi)) \to \mathrm{T}(\Phi(\epsilon(\Phi)))$
- $\exists\mathrm{E} : \prod_{\Phi:i\to o} . \prod_{\phi:o} . \mathrm{T}(\exists(\Phi)) \to (\prod_{x:i} . \mathrm{T}(\Phi(x)) \to \mathrm{T}(\phi)) \to \mathrm{T}(\phi)$
- $\exists\mathrm{I} : \prod_{\Phi:i\to o} . \prod_{t:i} . \mathrm{T}(\Phi(t)) \to \mathrm{T}(\exists(\Phi))$
- $\forall\mathrm{E} : \prod_{\Phi:i\to o} . \prod_{t:i} . \mathrm{T}(\forall(\Phi)) \to \Phi(t)$
- $\forall\mathrm{I} : \prod_{\Phi:i\to o} . (\prod_{t:i} . \mathrm{T}(\Phi(t))) \to \mathrm{T}(\forall(\Phi))$

- Example of a Proof

  - $\Delta : \prod_{\Phi:i\to o} . \mathrm{T}(\forall(\Phi)) \to \mathrm{T}(\exists(\Phi))$

In the example above Δ is the following term

$$\lambda\Phi : i \to o \ . \ \lambda\rho : \mathrm{T}(\forall(\Phi)) \ . \ \exists\mathrm{I}(\Phi)(\epsilon(\Phi))(\forall\mathrm{E}(\Phi)(\epsilon(\Phi))(\rho))$$

We should point out that if we removed the choice operator from the signature this example would no longer be provable, unless one explicity added a constant of type i.

Adequacy and Faithfulness Property 2 *We state the adequacy only for a monadic language, the general case follows exactly the same pattern. Letting*

$$\Gamma = \{x_1 : i, \ldots x_n : i, X_1 : i \to o, \ldots X_m : i \to o\}$$

the following hold:

1. $\Gamma \vdash M : i$ *iff* $\Phi_\Gamma(M)$ *is a well formed term of first order logic with a choice operator whose only free individual variables are among* $x_1, \ldots, x_n$ *and whose unary relations are among the* $X_1, \ldots, X_m$.
2. $\Gamma \vdash M : o$ *iff* $\Phi_\Gamma(M)$ *is a well formed formula of first order logic with a choice operator whose only free individual variables are among* $x_1, \ldots, x_n$ *and whose unary relations are among the* $X_1, \ldots, X_m$.
3. $\Gamma \cup \{y_1 : \mathrm{True}(\phi_1), \ldots, y_k : \mathrm{True}(\phi_k)\} \vdash M : \mathrm{True}(\phi)$

iff

$$\Phi_\Gamma(\phi_1), \ldots, \Phi_\Gamma(\phi_k) \vdash \Phi_\Gamma(\phi).$$

where Φ_Γ *is a bijective function*

$$\Phi_\Gamma : \Xi_\Gamma(i) \cup \Xi_\Gamma(o) \to \epsilon 1^\circ$$

to be defined shortly. $\epsilon 1^\circ$ *denotes the collection of terms and formulas of first order logic with a choice operator whose only free individual variables are among* $x_1, \ldots, x_n$ *and whose unary relations are among the* $X_1, \ldots, X_m$. $\Xi_\Gamma(\tau)$ *is the set of long* $\beta\eta$ *normal forms of type* τ *in the context* Γ. *Finally*

$$\Phi_\Gamma(M) = \begin{cases} x & \text{if } M \equiv x \\ \Phi_\Gamma(M') = \Phi_\Gamma(N) & \text{if } M \equiv (M' = N) \\ \epsilon(\Phi_{\Gamma,x:i}(P[x])) & \text{if } M \equiv \epsilon(\lambda x : i.P[x]) \\ \neg\Phi_\Gamma(M') & \text{if } M \equiv \neg M \\ \Phi_\Gamma(M') \supset \Phi_\Gamma(N) & \text{if } M \equiv M' \supset N \\ \forall x.\Phi_{\Gamma,x:i}(M'[x]) & \text{if } M \equiv \forall(\lambda x : i.M[x]) \\ \exists x.\Phi_{\Gamma,x:i}(M'[x]) & \text{if } M \equiv \exists(\lambda x : i.M[x]) \\ X(\Phi_\Gamma(M')) & \text{if } M \equiv X(M'). \end{cases}$$

4.3 Hilbert Style Modal Logics

4.3.1 The System

This case presents a new kind of problem: Standard Hilbert systems for modal logics usually have, apart of axiom schemes, two rules of inference: M.P. and the necessitation rule: from ϕ infer $\Box\phi$. The second one, however, is taken to be only a *rule of proof.* This means that its application to a sentence is permitted only if that sentence was shown to be a logical theorem of the system, but not if it depends on assumptions. It is not the case, therefore, that $\Box\phi$ *follows* from ϕ in such systems. It would not be sound, accordingly, to internalize the necessitation rule by the standard method of introducing a constant Nec (say)

of type $\prod_{\phi:o} \mathrm{T}(\phi) \rightarrow \mathrm{T}(\Box\phi)$ (where T corresponds to the intended consequence relation), since such a constant would force the deducibility of $\Box\phi$ from any set of assumptions which entail ϕ.

The solution to this problem is an example of the power gained by the ELF ability to employ different judgements. The solution, in this case, is to introduce *two* judgements, "True" and "Valid". The first corresponds to the original consequence relation which we have in mind (in which necessitation is only a rule of proof). The second to the one which is obtained by taking both rules as pure rules of derivation. We shall denote the first by $\vdash_t$ and the second by $\vdash_v$.

4.3.2 The Signature $\Sigma_{\mathrm{H}\Box}$

The resulting signature in the particular case of S4 is:

- Syntactic Categories
  - o : Type
- Operations
  - $\neg : o \rightarrow o$
  - $\supset : o \rightarrow o \rightarrow o$
  - $\Box : o \rightarrow o$
- Judgements
  - True : $o \rightarrow$ Type
  - Valid : $o \rightarrow$ Type
- Axioms and Rules
  - C : $\prod_{\phi:o}.\ \mathrm{Valid}(\phi) \rightarrow \mathrm{True}(\phi)$
  - $\mathrm{A}_1 : \prod_{\phi_1,\phi_2:o}.\ \mathrm{Valid}(\phi_1 \supset (\phi_2 \supset \phi_1))$
  - $\mathrm{A}_2 : \prod_{\phi_1,\phi_2,\phi_3:o}.\ \mathrm{Valid}(\phi_1 \supset (\phi_2 \supset \phi_3) \supset (\phi_1 \supset \phi_2) \supset (\phi_1 \supset \phi_3))$
  - $\mathrm{A}_3 : \prod_{\phi_1,\phi_2:o}.\ \mathrm{Valid}((\neg\phi_1 \supset \neg\phi_2) \supset (\phi_2 \supset \phi_1))$
  - $\mathrm{A}_4 : \prod_{\phi:o}.\ \mathrm{Valid}(\Box\phi \supset \phi)$
  - $\mathrm{A}_5 : \prod_{\phi_1,\phi_2:o}.\ \mathrm{Valid}(\Box(\phi_1 \supset \phi_2) \supset (\Box\phi_1 \supset \Box\phi_2))$
  - $\mathrm{A}_6 : \prod_{\phi:o}.\ \mathrm{Valid}(\Box\phi \supset \Box\Box\phi)$
  - $\mathrm{MP}_T : \prod_{\phi_1,\phi_2:o}.\ \mathrm{True}(\phi_1 \supset \phi_2) \rightarrow \mathrm{True}(\phi_1) \rightarrow \mathrm{True}(\phi_2)$
  - $\mathrm{MP}_V : \prod_{\phi_1,\phi_2:o}.\ \mathrm{Valid}(\phi_1 \supset \phi_2) \rightarrow \mathrm{Valid}(\phi_1) \rightarrow \mathrm{Valid}(\phi_2)$

– Nec : $\prod_{\phi:o}$. Valid(ϕ) → Valid($\Box\phi$)

- Example of a Proof

  – $\Delta : \prod_{\phi:o}$ True($\Box(\Box\phi \supset \phi) \supset \Box\phi) \rightarrow$ True(ϕ)

Where Δ is the following term

$$\begin{array}{l}\lambda\phi : o.\lambda x : \mathrm{True}(\Box(\Box\phi \supset \phi) \supset \Box\phi).\\ \quad \mathrm{MP}_T(\Box\phi)(\phi)(\mathrm{C}(\mathrm{A}_4(\phi)))\\ \quad\quad (\mathrm{MP}_T(\Box(\Box\phi \supset \phi))(\Box\phi)(x)\\ \quad\quad\quad (\mathrm{C}(\Box(\Box\phi \supset \phi))(\mathrm{Nec}(\Box\phi \supset \phi)(\mathrm{A}_4(\phi)))))\end{array}$$

Note: $\Box(\Box\phi \supset \phi) \supset \Box\phi$ is the characteristic axiom of GL— the famous modal system for provability in PA. The above is a proof that in S4 any ϕ-instance of this formula actually entails ϕ.

Adequacy and Faithfulness Property 3 *The following hold:*

1. *There is a compositional surjection between proofs in the Hilbert-type system of S4 that $\phi_1, \ldots, \phi_n \vdash_t \psi$ and terms t such that:*

   $$p_1 : o, \ldots, p_n : o \vdash_{\Sigma_{H\Box}} t : \mathrm{True}(\phi_1) \rightarrow \cdots \rightarrow \mathrm{True}(\phi_n) \rightarrow \mathrm{True}(\psi)$$

   Where $p_1, \ldots, p_n$ are the atomic variables of ϕ.

2. *There is a compositional bijection between proofs in the Hilbert-type system of S4 that $\phi_1, \ldots, \phi_n \vdash_v \psi$ and terms t such that:*

   $$p_1 : o, \ldots, p_n : o \vdash_{\Sigma_{H\Box}} t : \mathrm{Valid}(\phi_1) \rightarrow \cdots \rightarrow \mathrm{Valid}(\phi_n) \rightarrow \mathrm{Valid}(\psi)$$

   Where $p_1, \ldots, p_n$ are the atomic variables of ϕ.

The above can be strengthened as follows:

- *There is a compositional surjection between ELF-terms of type:*

  $$\mathrm{Valid}(\phi_1) \rightarrow \cdots \rightarrow \mathrm{Valid}(\phi_n) \rightarrow \mathrm{True}(\psi_1) \rightarrow \cdots \mathrm{True}(\psi_m) \rightarrow \mathrm{True}(\vartheta)$$

  and proofs that $\psi_1, \ldots, \psi_m \vdash_t \vartheta$ in the Hilbert-type system which is obtained by adding $\phi_1, \ldots, \phi_n$ as axioms to S4.

The above method for handling rules of proof is not specific to modal logic. It is, in fact, rather general and the reader is refered to 4.5 for a further example. But in the realm of modal logic the corresponding consequence relations have a natural semantical interpretation in terms of Kripke models:

- $\phi_1, \ldots, \phi_n \vdash_v \psi$ iff ψ is *valid* in any frame (of the relevant class) in which $\phi_1, \ldots, \phi_n$ are all valid (i.e. true in all worlds).

- $\phi_1, \ldots, \phi_n \vdash_t \psi$ iff ψ is *true* in every world (of a frame from the appropriate class in which $\phi_1, \ldots, \phi_n$ are all true.

It is important to note, finally, that the ELF enables us to express and prove logical facts concerning both internalized consequence relations. For example, the fact that a certain ϑ is true in any world in which $\phi_1, \ldots \phi_n$ are all true, provided this world belongs to a frame in which $\psi_1, \ldots, \psi_m$ are all valid, is proved in the ELF by constructing a term of type:

$$\text{Valid}(\psi_1) \to \cdots \to \text{Valid}(\psi_m) \to \text{True}(\phi_1) \to \cdots \to \text{True}(\phi_n) \to \text{True}(\vartheta).$$

4.4 Natural Deduction Style S4

4.4.1 The System

We now give an example of a known natural deduction formalism which is not "well-behaved" according to the ELF standards. It is Prawitz's system for S4 [22]. This system is obtained from the usual natural deduction formulation of classical propositional calculus by the addition of the following two rules:

$$\frac{\phi}{\Box\phi} \qquad \frac{\Box\phi}{\phi}$$

The crucial point about this formalism is that there is a side condition on the application of the first rule (the introduction rule for $\Box$). Prawitz gives two possible versions of this side condition. In the first version all assumptions on which ϕ depends should be modal (i.e. the main connective is $\Box$). In the second these assumptions may be what he calls essentially modal, i.e., formulas which are obtained from modal formulas by arbitrary combinations of disjunction, conjunction and double-negation. In both versions the side condition makes this rule *impure* [1]. This generally means that the coherence which the ELF paradigm expects between the formulation of the rules of a system and the consequence relation represented by it (see section 2) is broken. In the present case, for example the rule has the form

$$\frac{\phi}{\Box\phi},$$

but it is not the case that $\phi \vdash \Box\phi$ (in the sense that there is a proof of $\Box\phi$ from ϕ in the system).

The solution to the problem which is caused by the impurity of the rules for the $\Box$ is solved in the present case by seperating the rules of the system into two groups and introducing two corresponding judgements. The rules of the first group are just the rules of classical logic. These rules induce the usual consequence relation of classical propositional logic. We denote by Taut the corresponding judgement in the ELF internalization of the system. The second group consists of the special rules for $\Box$. We denote by Valid the judgement which corresponds to the whole system, including these two special rules. The constants of the internalization then fall into three groups: those corresponding to pure tautological inferences, those corresponding to the modal rules and those which relate the two sorts of inferences. The resulting signature is:

4.4.2 The Signature $\Sigma_{ND\Box}$

- Syntactic Categories
  - o : Type
- Operations
  - $\bot$: o
  - $\supset : o \to o \to o$
  - $\Box : o \to o$
- Judgements
  - Taut : $o \to$ Type
  - Valid : $o \to$ Type
- Axioms and Rules
  - C : $\prod_{\phi:o}$. Taut(ϕ) $\to$ Valid(ϕ)
  - R : $\prod_{\phi_1,\phi_2:o}$.(Taut(ϕ_1) $\to$ Valid(ϕ_2)) $\to$ (Valid(ϕ_1) $\to$ Valid(ϕ_2))
  - $\supset$I$_V$: $\prod_{\phi_1,\phi_2:o}$. (Valid($\Box\phi_1$) $\to$ Valid(ϕ_2)) $\to$ Valid($\Box\phi_1 \supset \phi_2$)
  - $\bot$ E : $\prod_{\phi:o}$ Taut($\bot$) $\to$ Taut(ϕ)
  - $\neg\neg$E : $\prod_{\phi:o}$ Taut((($\phi \supset \bot$) $\supset \bot$) $\to$ Taut(ϕ)
  - $\supset$I$_T$: $\prod_{\phi_1,\phi_2:o}$. (Taut(ϕ_1) $\to$ Taut(ϕ_2)) $\to$ Taut($\phi_1 \supset \phi_2$)
  - $\supset$E$_T$: $\prod_{\phi_1,\phi_2:o}$. Taut($\phi_1 \supset \phi_2$) $\to$ Taut(ϕ_1) $\to$ Taut(ϕ_2)
  - $\Box$I : $\prod_{\phi:o}$. Valid(ϕ) $\to$ Valid($\Box\phi$)
  - $\Box$E : $\prod_{\phi:o}$. Valid($\Box\phi$) $\to$ Valid(ϕ)

- Example of a Proof

  - $\supset \mathrm{E}_V : \prod_{\phi_1,\phi_2:o} . \ \mathrm{Valid}(\phi_1 \supset \phi_2) \rightarrow \mathrm{Valid}(\phi_1) \rightarrow \mathrm{Valid}(\phi_2)$
  - $\Delta : \prod_{\phi_0:o} \prod_{\phi_1:o} \mathrm{Valid}(\Box(\phi_0 \supset \phi_1) \supset (\Box\phi_0 \supset \Box\phi_1))$

It is straightforward but tedious exercise to construct in the above signature the term $\supset \mathrm{E}_V$ and we omit mentioning explicitly. In the second example we use this term to internalize the natural deduction proof of axiom A_5 of the corresponding Hilbert-type system. Δ is defined to be the following term:

$$\begin{array}{l}\lambda\phi_0 : o.\lambda\phi_1 : o. \supset \mathrm{I}_V(\Box(\phi_0 \supset \phi_1))(\Box\phi_0 \supset \Box\phi_1) \\ \quad (\lambda x : \mathrm{Valid}(\Box(\phi_0 \supset \phi_1)). \\ \quad\quad (\supset \mathrm{I}_V(\phi_0)(\Box\phi_1)(\lambda y : \mathrm{Valid}(\Box\phi_0). \\ \quad\quad\quad \Box\mathrm{I}(\phi_1)(\supset \mathrm{E}_V(\phi_0)(\phi_1)(\Box\mathrm{E}(\phi_0 \supset \phi_1)(x))(\Box\mathrm{E}(\phi_0)(y))))))\end{array}$$

Adequacy and Faithfulness Property 4 *Given a proof of a formula ϕ in Prawitz's system, there is an uniform way of constructing a corresponding term t of the ELF (in the above signature) of type* $\mathrm{Valid}(\phi)$, *in which the only free variables are those corresponding to the atomic variables of ϕ. Moreover, this construction provides a compositional surjection between assumptions free proofs in Prawitz and this sort of ELF terms.*

Some explanations about the role and the meaning of the first three constants of the above signature, are in order:

1. the first two constants, C and R, together enable us to tansfer any purely truth-functional proof of a sequent to a corresponding proof in the internalized system. Formally: using them we can uniformly construct, for each n, a term R_n of type:

   $$\prod_{\phi_1:o} \cdots \prod_{\phi_n:o} (\mathrm{Taut}(\phi_1) \rightarrow \cdots \rightarrow \mathrm{Taut}(\phi_n)) \rightarrow (\mathrm{Valid}(\phi_1) \rightarrow \cdots \rightarrow \mathrm{Valid}(\phi_n))$$

   The term $\supset \mathrm{E}_V$ mentioned above is, for example, immediately obtained from $\supset \mathrm{E}_T$ using R_3. Conversely, C and $\supset \mathrm{E}_T$ suffices for obtaining all the R_n's (which in turn suffices for the adequacy theorem above, but this is done in a roundabout way and so it is a less faithful method [2]).

   The *meaning* of C is obvious. The meaning of R is that if we have a uniform method for extending any proof of ϕ in classical logic to a proof of ψ in Prawitz' system then by the same method we can extend *any* proof of ϕ in that system to a proof of ψ (since applicability of rules in a proof does not depends on what rules were applied above — only on the structure of the formulas involved and the assumptions). R encodes therefore something which is almost an identity function.

2. The role of the remaining constant, $\supset I_V$, is to solve the following problem: applications of the introduction rule for implication can be made in Prawitz's system after applications of the $\Box$-rules. Hence $\supset I_T$ alone is not sufficient for achieving the full power of this rule. $\supset I_V$ directly solves the problem in the basic case in which the application of $\supset$-Introduction is made immediately after an application of $\Box$-introduction. In fact since in this case the discharged formula is necessarily modal, $\supset I_V$ applies. It is possible to prove that this solution of the basic case suffices in general (note that axiom A_4 of the Hilbert system is easily derived using $\Box E$ and $\supset I_V$ and so this rule poses no extra problem![2]). The above proof of A_5 is a good example how this method works.

   Intuitively, $\supset E_V$ corresponds to a deduction theorem about $\vdash_v$ that can be proved either syntactically or by using Kripke models [2].

The explanations concerning $\supset I_V$ should make it obvious how to internalize the second version of Prawitz' system for S4. All one needs to do is to introduce first a new *syntactic* judgement EM (corresponding to the syntactic category of essentially modal formulas) with the obvious characterizing constants. The remaining step is to replace $\supset I_V$ with a constant $\supset I'_V$ of type:

$$\prod_{\phi_1,\phi_2:o} \mathrm{EM}(\phi_1) \to (\mathrm{Valid}(\phi_1) \to \mathrm{Valid}(\phi_2)) \to \mathrm{Valid}(\phi_1 \supset \phi_2)$$

Our final note is concerned with the consequence relations which are defined by Prawitz' system and the above signature. The adequacy theorem above applied to proofs of *theorems* in Prawitz' system. It does not applies to proofs of sequents. As a matter of fact, the "Valid" judgement corresponds exactly to the $\vdash_v$ of the last section: We have that $\phi_1, \ldots, \phi_n \vdash_v \psi$ iff there exist (in the appropriate context) a term t of type $\mathrm{Valid}(\phi_1) \to \cdots \to \mathrm{Valid}(\phi_n) \to \mathrm{Valid}(\psi)$. The consequence relation which is directly defined by Prawitz' system is, on the other hand, $\vdash_t$. In order to fully and faithfully internalize also partial proofs in Prawitz we need to introduce a third judgement: True. The reader may find more details in [2].

4.5 The Classical Lambda Calculus

4.5.1 The System

This example deals with the classical λ-calculus with a full-blown ξ-rule and with the version of λ-calculus where the ξ-rule is taken only as a rule of proof, i.e. only

[2]It is possible, in fact, to use Taut rather then Valid for internalizing $\Box E$, leaving Valid to handle just the single impure rule of Prawitz' system.

in the following form

$$\frac{M \;\text{conv}\; N}{\lambda.M \;\text{conv}\; \lambda.N}$$

4.5.2 The Signature Σ_Λ

- Syntactic Categories
  - o : Type
- Operations
  - $\Lambda : (o \to o) \to o$
  - $\text{App} : o \to o \to o$
- Judgements
  - $\bowtie\colon o \to o \to \text{Type}$
  - $=\colon o \to o \to \text{Type}$
- Axioms and Rules
  - $\text{E}_0 : \prod_{x:o} . \; x \bowtie x$
  - $\text{E}_1 : \prod_{x,y:o} . \; x \bowtie y \to y \bowtie x$
  - $\text{E}_2 : \prod_{x,y,z:o} . \; x \bowtie y \to y \bowtie z \to x \bowtie z$
  - $\text{E}_3 : \prod_{x,y,x',y':o} . \; x \bowtie y \to x' \bowtie y' \to \text{App}(x,x') \bowtie \text{App}(y,y')$
  - $\beta : \prod_{x:o\to o} . \; \prod_{y:o} . \; \text{App}(\Lambda(x),y) \bowtie xy$
  - $\xi : \prod_{x,y:o\to o} . \; (\prod_{z:o} . \; xz \bowtie yz) \to \Lambda(x) \bowtie \Lambda(y)$
  - $\text{Con} : \prod_{x,y:o} . \; x \bowtie y \to x = y$
  - $\text{E}_4 : \prod_{x:o} . \; x = x$
  - $\text{E}_5 : \prod_{x,y:o} . \; x = y \to y = x$
  - $\text{E}_6 : \prod_{x,y,z:o} . \; x = y \to y = z \to x = z$
  - $\text{E}_7 : \prod_{x,y,x',y':o} . \; x = y \to x' = y' \to \text{App}(x,x') = \text{App}(y,y')$
  - $\beta : \prod_{x:o\to o} . \; \prod_{y:o} . \; \text{App}(\Lambda(x),y) = xy$
- Example of a Proof
  - $\Delta : (\prod_{M:o} . \; (\Lambda(\lambda x : o.\text{App}(M,x)) = M) \to$

$$(\prod_{M,N:o} . \; (\prod_{x:o} . \; \text{App}(M,x) = \text{App}(N,x) \to M = N)))$$

The example describes a proof in the ELF that asumming the rule η (i.e. that $\lambda x.Mx = M$ provided $x \notin M$) the rule ext

$$\frac{Mx = Nx}{M = N}$$

can be derived. The term Δ is the following ELF term.

$$\begin{array}{l}\lambda y : \Pi_{M:o}.\ (\Lambda(\lambda x : o.\mathrm{App}(M,x)) = M).\lambda M : o.\lambda N : o.\\ \quad \lambda z : \Pi_{x:o}.\ \mathrm{App}(M,x) = \mathrm{App}(N,x).\mathrm{E}_2(M)(\Lambda(\lambda x : o.\mathrm{App}(M,x)))(N)\\ \quad\quad (\mathrm{E}_1(\Lambda(\lambda x : o.\mathrm{App}(M,x)))(M)(y(M)))(\mathrm{E}_2(\Lambda(\lambda x : o.\mathrm{App}(M,x)))\\ \quad\quad\quad (\Lambda(\lambda x : o.\mathrm{App}(N,x)))(N)(\xi(\lambda x : o.\mathrm{App}(M,x))\\ \quad\quad\quad\quad (\lambda x : o.\mathrm{App}(N,x))(z)(y(N))))\end{array}$$

Adequacy and Faithfulness Property 5 *The following hold:*

1. $x_1 : o, \ldots, x_n : o \vdash M : o$ iff $\Phi_\Gamma(M) \in \Lambda$
2. $x_1 : o, \ldots, x_n : o, \eta_1 : M_1 \bowtie N_1, \ldots, \eta_m : M_m \bowtie N_m \vdash P : M \bowtie N$

iff

$$\Phi_\Gamma(M_1) = \Phi_\Gamma(N_1), \ldots, \Phi_\Gamma(M_m) = \Phi_\Gamma(N_m) \vdash \Phi_\Gamma(M) = \Phi_\Gamma(N)$$

where $M \in \Xi_\Gamma(o)$, $\Xi_\Gamma(o)$ *is the set of normal forms of type* o *in the context* Γ,

$$\Gamma = x_1 : o, \ldots, x_n : o$$

and

$$\Phi_\Gamma : \Xi_\Gamma(o) \longrightarrow \Lambda[x_1, \ldots, x_n]$$

is a bijective function defined as follows

$$\Phi_\Gamma(M) = \begin{cases} x & \text{if } M = x \\ \Phi_\Gamma(M')\Phi_\Gamma(N) & \text{if } M = \mathrm{App}(M', N) \\ \lambda x.\Phi_{\Gamma,x:o}(M'[x]) & \text{if } M = \Lambda(\lambda x.M'[x]) \end{cases}$$

The function Φ_Γ could also be extended to proof terms (such as P above), we shall not do this here. It is interesting to notice that the ξ-rule we have implemented in Σ_Λ corresponds to the following version

$$\frac{M = N}{\lambda x.M = \lambda x.N}$$

provided x does not occur free in any assumption. This is the semantically correct way of understanding the ξ-rule in a natural deduction setting. The introduction in this signature of the judgement $=$ and the connecting rule Con is an instance

of the general method for encoding rules of proof in the ELF. The following fact expresses this more precisely

$$x_1 : o, \ldots, x_n : o, \eta_1 : M_1 = N_1, \ldots, \eta_m : M_m = N_m \vdash P : M = N$$

iff

$$\Phi_\Gamma(M_1) = \Phi_\Gamma(N_1), \ldots, \Phi_\Gamma(M_m) = \Phi_\Gamma(N_m) \vdash_\xi \Phi_\Gamma(M) = \Phi_\Gamma(N)$$

where $\vdash_\xi$ denotes the consequence relation induced by the λ-formal system in which the ξ-rule is only taken as a rule of proof.

4.6 Call-By-Value Lambda Calculus

4.6.1 The System

The call-by-value λ-calculus, usually referred to as the λ_v-calculus, was first introduced and studied in Plotkin [21]. In the usual formulation its syntax is identical to that of the traditional λ-calculus. The crucial difference manifests itself in the proof system. In particular in the formulation of the β-reduction rule.

- $\beta_v \quad (\lambda x \ . \ M)N = M[x := N]$

  - provided that N is a value, i.e. either a variable or an abstraction.

The immediate difficulty in encoding this formal system in the ELF is how to express the syntactic notion of being a variable. This is compounded by the desire within the ELF to have the variables of the ELF stand also for schematic variables ranging over λ_v-terms. The solution to this problem, that is presented here, was inspired by the denotational semantics of the calculus [9]. A typical model is a domain D such that

$$D \vartriangleright \underset{i}{\overset{j}{\rightleftarrows}} [D \to_\perp D].$$

We also require that $i \circ j \not\in [D \to_\perp D]$ while $j \circ i = Id_{[D \to_\perp D]}$. In such a model the variables range over $D - \{\perp\}$. The solution in the ELF is to model syntax using two syntactic categories, v for values and o for expressions. The only bindable type (i.e. a type with a binding operator defined on it) is v. The map $! : v \to o$ can be interpreted as the injection

$$! : D - \{\perp\} \to D.$$

This illustrates the general technique for handling subcategories in ELF.

4.6.2 The Signature Σ_{Λ_v}

- Syntactic Categories
  - o : Type
  - v : Type
- Operations
  - $! : v \to o$
  - $\Lambda_v : (v \to o) \to v$
  - $\mathrm{App} : o \to o \to o$
- Judgement
  - $= \ : o \to o \to \mathrm{Type}$
- Axioms and Rules
  - $\mathrm{E}_0 : \Pi_{x:o} \,.\, x = x$
  - $\mathrm{E}_1 : \Pi_{x,y:o} \,.\, x = y \to y = x$
  - $\mathrm{E}_2 : \Pi_{x,y,z:o} \,.\, x = y \to y = z \to x = z$
  - $\mathrm{E}_3 : \Pi_{x,y,x',y':o} \,.\, x = y \to x' = y' \to \mathrm{App}(x,x') = \mathrm{App}(y,y')$
  - $\beta_v : \Pi_{x:v\to o} \,.\, \Pi_{y:v} \,.\, \mathrm{App}(\Lambda_v(x)!, y!) = xy$
  - $\xi_v : \Pi_{x,y:v\to o} \,.\, (\Pi_{z:v} \,.\, xz = yz) \to \Lambda_v(x)! = \Lambda_v(y)!$
  - $\eta_v : \Pi_{x:v} \,.\, \Lambda_v(\lambda y : v \ . \ \mathrm{App}(x!, y!) = x!$
- Example of a Proof
  - $\lambda x : v \ . \ \beta_v(!)(x) : \Pi_{x:v} \,.\, \mathrm{App}(\Lambda_v(!)!, x!) = x!$

The example of a proof included above demonstrates that

$$\Lambda_v(!)!$$

behaves like the identity (with respect to App) on the image of v, via $!$, in o.

It is worth noting that in this setting the correct version of the η-rule suggests itself more naturally than in the original presentation.

Adequacy and Faithfulness Property 6 *The following hold:*

1. $x_1 : v, \ldots, x_n : v \vdash M : o \quad \text{iff} \quad \Phi_\Gamma(M) \in \Lambda_v$

2. $x_1 : v, \ldots, x_n : v, \eta_1 : M_1 = N_1, \ldots, \eta_m : M_m = N_m \vdash P : M = N$

iff

$$\Phi_\Gamma(M_1) = \Phi_\Gamma(N_1), \ldots, \Phi_\Gamma(M_m) = \Phi_\Gamma(N_m) \vdash \Phi_\Gamma(M) = \Phi_\Gamma(N)$$

where $M \in \Xi_\Gamma(o)$, $\Xi_\Gamma(o)$ *is the set of normal forms of type* o *in the context* Γ,

$$\Gamma = x_1 : v, \ldots, x_n : v$$

and

$$\Phi_\Gamma : \Xi_\Gamma(o) \longrightarrow \Lambda_v[x_1, \ldots, x_n]$$

is a bijective function defined as follows

$$\Phi_\Gamma(M) = \begin{cases} x & \text{if } M = x! \\ \Phi_\Gamma(M')\Phi_\Gamma(N) & \text{if } M = \mathrm{App}(M', N) \\ \lambda x.\Phi_{\Gamma, x:v}(M'[x]) & \text{if } M = \Lambda_v(\lambda x.M'[x])! \end{cases}$$

As before the function Φ_Γ could also be extended to proof terms again we shall not do this here.

4.7 The Lambda I Calculus

4.7.1 The System

In the λ_I-calculus we must deal with another syntactic idiosyncrasy, namely the restriction on λ-abstraction. The set, Λ_I, of terms of the λ_I-calculus is defined as in the classical calculus, except for the abstraction clause. In this case the rule states:

$$M \in \Lambda_I \quad \wedge \quad x \in M \quad \Rightarrow \quad \lambda x \ . \ M \in \Lambda_I$$

The immediate difficulty in formalizing this system in the ELF is how to enforce the binding constructor λ_I to be defined only on *relevant* schemes. For encoding this system we will use an approach similar to the one taken in the case of the λ_v-calculus. In other words we will give a solution inspired by the denotational semantics of the calculus. A typical model, in this case, is a domain D such that

$$D \triangleright \underset{i}{\overset{j}{\rightleftarrows}} [D \to_\perp D].$$

Where we also require that $i \circ j \in [D \to_\perp D]$ and $j \circ i = Id_{[D \to_\perp D]}$, see [9].

4.7.2 The Signature Σ_{Λ_I}

- Syntactic Categories
  - o : Type
- Operations
  - $\bot$: o
  - $\Lambda_I : \prod_{x:o\to o} . \; x(\bot) = \bot \to o$
  - $\mathrm{App} : o \to o \to o$
- Judgement
  - $= \;\; : o \to o \to \mathrm{Type}$
- Axioms and Rules
  - $\mathrm{E}_0 : \prod_{x:o} . \; x = x$
  - $\mathrm{E}_1 : \prod_{x,y:o} . \; x = y \to y = x$
  - $\mathrm{E}_2 : \prod_{x,y,z:o} . \; x = y \to y = z \to x = z$
  - $\mathrm{E}_3 : \prod_{x,y,x',y':o} . \; x = y \to x' = y' \to \mathrm{App}(x,x') = \mathrm{App}(y,y')$
  - $\bot_r$: $\prod_{x:o} . \; \mathrm{App}(x,\bot) = \bot$
  - $\bot_l$: $\prod_{x:o} . \; \mathrm{App}(\bot,x) = \bot$
  - $\bot_\Lambda$: $\bot = \Lambda_I(\lambda x : o \; . \; \bot, \mathrm{E}_0(\bot))$
  - $\beta_I : \prod_{x:o\to o} . \prod_{y:o} . \prod_{t:x(\bot)=\bot} . \; \mathrm{App}(\Lambda_I(x,t),y) = xy$.
  - $\xi_I : \prod_{x,y:o\to o} . \prod_{t_1:x(\bot)=\bot} . \prod_{t_2:y(\bot)=\bot} .$

$$(\prod_{z:o} . \; x(z) = y(z)) \to \Lambda_I(x,t_1) = \Lambda_I(y,t_2)$$

- Example of a Proof
  - $\Delta : \prod_{x:o\to o} . \prod_{t_1,t_2:x(\bot)=\bot} . \; \Lambda_I(x,t_1) = \Lambda_I(x,t_2)$

Where the term Δ is the following:

$\lambda x : o \to o \; . \; \lambda t_1 : x(\bot) = \bot \; . \; \lambda t_2 : x(\bot) = \bot \; . \xi_i(x)(x)(t_1)(t_2)(\lambda z : o \; . \; \mathrm{E}_0(x(z)))$.

Adequacy and Faithfulness Property 7 *The following hold:*

1. $x_1 : o, \ldots, x_n : o \vdash M : o$ iff $\Phi_\Gamma(M) \in \Lambda_I$

2. $x_1 : o, \ldots, x_n : o, \eta_1 : M_1 = N_1, \ldots, \eta_m : M_m = N_m \vdash P : M = N$

iff

$$\Phi_\Gamma(M_1) = \Phi_\Gamma(N_1), \ldots, \Phi_\Gamma(M_m) = \Phi_\Gamma(N_m) \vdash \Phi_\Gamma(M) = \Phi_\Gamma(N)$$

where $M \in \Xi_\Gamma(o)$, $\Xi_\Gamma(o)$ *is the set of normal forms of type* o *in the context* Γ, *in which* $\perp$ *occurs only within the second argument to the* Λ_I*-operator.*

$$\Gamma = x_1 : o, \ldots, x_n : o$$

and

$$\Phi_\Gamma : \Xi_\Gamma(o) \longrightarrow \Lambda_I[x_1, \ldots, x_n]$$

is a surjective function defined as follows

$$\Phi_\Gamma(M) = \begin{cases} x & \text{if } M = x \\ \Phi_\Gamma(M')\Phi_\Gamma(N) & \text{if } M = \mathrm{App}(M', N) \\ \lambda x.\Phi_{\Gamma, x:o}(M'[x]) & \text{if } M = \Lambda_I(\lambda x : o.M'[x], t) \end{cases}$$

4.8 Linear Lambda Calculus

4.8.1 The System

The set Λ_L of terms of the linear lambda calculus is defined as follows:

1. $x \in \Lambda_L$
2. If $M \in \Lambda_L$ and $x \in M$ then $\lambda x.M \in \Lambda_L$
3. If $M, N \in \Lambda_L$ and $FV(M) \cap FV(N) = \emptyset$ then $MN \in \Lambda_L$

In order to encode this system into the ELF one makes use of the idea of actually encoding the notion of linearity of a function. A function $f : X \to Y$, where X and Y are upper semilattices with a least element, is linear iff

1. $f(\perp) = \perp$
2. $f(sup(a, b)) = sup(f(a), f(b))$

We will in fact end up encoding a related calculus Λ_L^* whose syntax is defined as follows:

1. $x \in \Lambda_L^*$
2. If $M \in \Lambda_L^*$ and x occurs free in M exactly once then $\lambda x.M \in \Lambda_L^*$
3. If $M, N \in \Lambda_L^*$ then $MN \in \Lambda_L^*$

This example is based on an idea of Gordon Plotkin and we thank him for allowing us to include it here.

4.8.2 The Signature Σ_{Λ_L}

In the following presentation of the signature we make, for typographical reasons, the following definition. We let

$$L = \lambda x : o \to o. \prod_{z,w:o} . x(z \vee w) = x(z) \vee x(w).$$

- Syntactic Categories
  - o : Type
- Operations
  - $\bot$: o
  - $\vee : o \to o \to o$
  - $\Lambda_L : \prod_{x:o\to o} . x(\bot) = \bot \to L(x) \to o$
  - App : $o \to o \to o$
- Judgement
  - $= \ : o \to o \to$ Type
- Axioms and Rules
  - $\mathrm{E}_0 : \prod_{x:o} . x = x$
  - $\mathrm{E}_1 : \prod_{x,y:o} . x = y \to y = x$
  - $\mathrm{E}_2 : \prod_{x,y,z:o} . x = y \to y = z \to x = z$
  - $\mathrm{E}_3 : \prod_{x,y,x',y':o} . x = y \to x' = y' \to \mathrm{App}(x,x') = \mathrm{App}(y,y')$
  - $\bot_r$: $\prod_{x:o} . \mathrm{App}(x,\bot) = \bot$
  - $\bot_l$: $\prod_{x:o} . \mathrm{App}(\bot,x) = \bot$
  - $\bot_\Lambda$: $\bot = \Lambda_L(\lambda x : o \ . \ \bot, \mathrm{E}_0(\bot), \lambda y : o . \lambda z : o . \vee_{\mathrm{id}} (\bot))$
  - $\vee_{\mathrm{id}} : \prod_{x:o} . x = x \vee x$
  - $\vee_\bot : \prod_{x:o} . x \ \vee \bot = x$
  - $\vee_{\mathrm{sym}} : \prod_{x,y:o} . x \vee y = y \vee x$
  - $\vee_{\mathrm{assoc}} : \prod_{x,y,z:o} . (x \vee y) \vee z = x \vee (y \vee z)$
  - $\vee_= : \prod_{x,y,z:o} . x = y \to (x \vee z) = (y \vee z)$
  - $\vee_l : \prod_{x,y,z:o} . \mathrm{App}(x, y \vee z) = \mathrm{App}(x,y) \vee \mathrm{App}(x,z)$
  - $\vee_r : \prod_{x,y,z:o} . \mathrm{App}(x \vee y, z) = \mathrm{App}(x,z) \vee \mathrm{App}(y,z)$

$$- \vee_\Lambda : \Pi_{x,y:o\to o} \cdot \Pi_{t_1:x(\bot)=\bot} \cdot \Pi_{t_2:y(\bot)=\bot} \cdot \Pi_{t_3:L(x)} \cdot \Pi_{t_4:L(y)} \cdot$$
$$\Pi_{t_5:y(\bot)\vee z(\bot)=\bot} \cdot \Pi_{t_6:L(\lambda z:o.x(z)\vee y(z))} \cdot$$
$$\Lambda_L(x,t_1,t_3) \vee \Lambda_L(y,t_2,t_4) = \Lambda_L(\lambda z:o \ . \ x(z) \vee y(z), t_5, t_6)$$

$$- \beta_L : \Pi_{x:o\to o} \cdot \Pi_{y:o} \cdot \Pi_{t_1:x(\bot)=\bot} \cdot \Pi_{t_2:L(x)} \cdot \mathrm{App}(\Lambda_L(x,t_1,t_2),y) = xy$$

$$- \xi_L : \Pi_{x,y:o\to o} \cdot \Pi_{t_1:x(\bot)=\bot} \cdot \Pi_{t_2:y(\bot)=\bot} \cdot \Pi_{t_3:L(x)} \cdot \Pi_{t_4:L(y)} \cdot$$
$$(\prod_{z:o} \cdot x(z) = y(z)) \to \Lambda_L(x,t_1,t_3) = \Lambda_L(y,t_2,t_4)$$

- Example of a Proof

$$- \Delta : \Pi_{x:o\to o} \cdot \Pi_{t_1,t_2:x(\bot)=\bot} \cdot \Pi_{t_3,t_4:L(x)} \cdot \Lambda_L(x,t_1,t_3) = \Lambda_L(x,t_2,t4)$$

Where the term Δ is the following:

$\lambda x : o \to o.\lambda t_1 : x(\bot) = \bot .\lambda t_2 : x(\bot) = \bot .\lambda t_3 : L(x).\lambda t_4 : L(x)$
$\xi_L(x)(x)(t_1)(t_2)(t_3)(t_4)(\lambda z : o.\mathrm{E}_0(x(z)))$.

Adequacy and Faithfulness Property 8 *The following hold:*

1. $\vdash M : o$ iff $\Phi_\emptyset(M) \in \Lambda_L^\circ$
2. $\eta_1 : M_1 = N_1, \ldots, \eta_m : M_m = N_m \vdash P : M = N$

iff

$$\Phi_\emptyset(M_1) = \Phi_\emptyset(N_1), \ldots, \Phi_\emptyset(M_m) = \Phi_\emptyset(N_m) \vdash \Phi_\emptyset(M) = \Phi_\emptyset(N)$$

where $M \in \Xi_\Gamma(o)$, $\Xi_\Gamma(o)$ is the set of normal forms of type o in the context Γ, in which $\bot$ occurs only within the second argument and $\vee$ occurs only in the third arguement to the Λ_L-operator.

$$\Gamma = x_1 : o, \ldots, x_n : o$$

and

$$\Phi_\Gamma : \Xi_\Gamma(o) \longrightarrow \Lambda_L^*[x_1, \ldots, x_n]$$

is a surjective function defined as follows

$$\Phi_\Gamma(M) = \begin{cases} x & \text{if } M = x \\ \Phi_\Gamma(M')\Phi_\Gamma(N) & \text{if } M = \mathrm{App}(M', N) \\ \lambda x.\Phi_{\Gamma,x:o}(M'[x]) & \text{if } M = \Lambda_L(\lambda x : o.M'[x], t, s) \end{cases}$$

The above signature can be taken as a basis for the ELF encoding of the external consequence relation of the minimal fragment of linear logic [4] [10].

4.9 Hoare's Logic

In this example we give a brief exposition of Hoare's Logic in the language of integers $\{+, 0, 1\}$. A more detailed exposition of this and other solutions can be found in [16]. In a later section we shall deal briefly with another solution that was not dealt with in the afore mentioned paper.

4.9.1 The System

Let L denote the first order language of the integers with equality, the meta-variables x, y, z denote or range over the variables of L, the meta-variables s, t denote or range over the terms or *expressions* of L, the meta-variable e is used to denote a quantifier-free formula or *boolean expression* of L, and, finally, p, q, r denote or range over the formulas or *assertions* of L. Let W denote the least class of while programs satisfying

1. for every variable x and expression t, $x := t \in W$; and
2. if $S_1, S_2 \in W$ then $S_1; S_2 \in W$, and for every boolean expression e of L, we have that $\mathtt{if}(e, S_1, S_2) \in W$ and $\mathtt{while}(e, S_1) \in W$.

The basic formulas of Hoare's logic are objects of the form $\{p\}S\{q\}$ where p, q are assertions and S is a while program. The intuitive meaning of an *asserted program,*

$$\{p\}S\{q\},$$

is as follows: whenever p holds before execution of S and S terminates, then q holds after execution of S. Hoare's logic is a system of formal reasoning about these asserted programs. Its axioms and proof rules are the following.

Axiom 1: Assignment Axiom

$$\{p[t/x]\}x := t\{p\}.$$

Rule 2: Composition Rule

$$\frac{\{p\}S_1\{r\}, \quad \{r\}S_2\{q\}}{\{p\}S_1; S_2\{q\}}$$

Rule 3: If Rule

$$\frac{\{p \wedge e\}S_1\{q\}, \quad \{p \wedge \neg e\}S_2\{q\}}{\{p\}\mathtt{if}(e, S_1, S_2)\{q\}}$$

Rule 4: While Rule

$$\frac{\{p \wedge e\}S\{p\}}{\{p\}\mathtt{while}(e, S)\{p \wedge \neg e\}}$$

The final rule involves some notion of a consequence relation, [1], for the assertion language.

Rule 5: Consequence Rule

$$\frac{p \Rightarrow p_1, \quad \{p_1\}S\{q_1\}, \quad q_1 \Rightarrow q}{\{p\}S\{q\}}$$

Here $p \Rightarrow p_1$ and $q_1 \Rightarrow q$ are assumed to follow from the background first order theory using the proof system for the assertion language. As usual, $p[t/x]$ stands for the result of substituting t for the free occurences of x in p.

There are, at least, three complications one must deal with in internalizing this logic, and we discuss them briefly prior to giving the signature.

1. We must, by necessity, distinguish between the variables of the first order logic, which are simply the variables of the LF, and the variables of the while language. This is because the latter are not substitutive. In particular we could not model $:=$ as an object of type $i \to i \to w$ since this would allow the formation of expressions like $0 := 1$, which are obviously syntactically and semantically ill-formed. The variables of l, i and o are the variables of the LF. The operation ! takes an identifier to its contents. In other words if $x : l$ then $x!$ (which we will write instead of the awkward $!(x)$) is the contents of x. It can thus be thought of as an evaluation mechanism.

2. Since the ELF does not have subtypes we must distinguish between the boolean expressions and the first order formulas. We do this by introducing a new judgment (a syntactic one), $\mathtt{QF}$, on o. Thus the if and while constructs not only take an element of o as an argument but also a proof that they are quantifier free.

3. The most important fact about this system is that the $:=$ operator is an example of a binding operator which does not α-convert. In the Hoare triple

$$\{P(t)\}x := t\{P(x)\}$$

free occurences of x in $P(x)$ are bound by the assignment operator $x := t$, this is not true of those occurences in $P(t)$ nor of the occurences in t in the assignment. α-conversion would not be in the spirit of Hoare's logic, since one wants to reason about the identifier x not some α-conversion of it. This has the consequence that simply modelling the assignment axiom by

$$\mathtt{Ass} : \Pi_{x:l}\Pi_{t:i}\Pi_{p:i\to o} \vdash_h \{p(t)\}x := t\{p(x!)\}$$

would be incorrect, as the following example indicates.

Example 1 *Suppose* $\Gamma = \{y : l\}$ *is the current context and take the following instantiation of the assignment axiom,*

$$\mathtt{ASS}(y)(1)(\lambda u.\neg(y! = u)).$$

This term inhabits the following type

$$\vdash \{\neg(y! = 1)\}y := 1\{\neg(y! = y!)\},$$

which one would have hoped was uninhabitable.

The easiest solution to this problem of formalizing the corrected version of the assignment axiom is to incorporate syntactic notions explicitly into the theory. We do this by adding three new judgements concerning non-interference along the lines of [23].

- $\neq : l \to (l \to \mathrm{TYPE})$
- $\sharp_i : l \to (i \to \mathrm{TYPE})$
- $\sharp_o : l \to (o \to \mathrm{TYPE})$

The intuitive meaning of the judgements can be explained, again using infix notation, as follows:

- $x \neq y$ is interpreted as meaning that x and y denote distinct identifiers or locations. Because we have no constructors or constants of type l, terms of type l must $\beta\eta$ reduce to LF variables.
- $x\sharp_i t$ is interpreted as meaning that no assignment to the location denoted by x effects the value of the term denoted by t. This of course is equivalent to saying that the location or identifier denoted by x does not occur in the term denoted by t.
- $x\sharp_o e$ is interpreted as meaning that no assignment to the location denoted by x effects the value or meaning of the formula denoted by e. Again this is equivalent to saying that the location or identifier denoted by x does not occur freely in the formula denoted by e (Note that it cannot occur bound).

4.9.2 The Signature Σ_{HL}

- Syntactic Categories
  - l : Type

- i : Type
- o : Type
- w : Type
- h : Type

- Operations
  - $! : l \to i$
  - $0 : i$
  - $1 : i$
  - $+ : i \to i \to i$
  - $= \ : i \to i \to o$
  - $\neg : o \to o$
  - $\supset : o \to o \to o$
  - $\forall : (i \to o) \to o$
  - $\ldots := \ldots : l \to i \to w$
  - $\ldots ; \ldots : w \to w \to w$
  - $\text{if} : \prod_{e:o} . \ \mathtt{QF}(e) \to w \to w \to w$
  - $\text{while} : \prod_{e:o} . \ \mathtt{QF}(e) \to w \to w$
  - $\{\ldots\} \ldots \{\ldots\} : o \to w \to o \to h$

- Judgements
  - $\vdash_o : o \to \text{Type}$
  - $\vdash_h : h \to \text{Type}$
  - $\neq \ : l \to l \to \text{Type}$
  - $\sharp_i : l \to i \to \text{Type}$
  - $\sharp_o : l \to o \to \text{Type}$
  - $\mathtt{QF} : o \to \text{Type}$

- Axioms and Rules
  - $\sharp_0 : \prod_{x:l} . \ x \sharp_i 0$
  - $\sharp_1 : \prod_{x,y:l} . \ x \neq y \to x \sharp_i y!$
  - $\sharp_2 : \prod_{x:l} . \ \prod_{t_1,t_2:i} . \ x \sharp_i t_1 \to x \sharp_i t_2 \to x \sharp_i (t_1 + t_2)$

- $\sharp_3 : \Pi_{x:l} \cdot \Pi_{t_1,t_2:i} \cdot x\sharp_i t_1 \rightarrow x\sharp_i t_2 \rightarrow x\sharp_o(t_1 = t_2)$
- $\sharp_4 : \Pi_{x:l} \cdot \Pi_{\phi:o} \cdot x\sharp_o\phi \rightarrow x\sharp_o\neg\phi$
- $\sharp_5 : \Pi_{x:l} \cdot \Pi_{\phi_1,\phi_2:o} \cdot x\sharp_o\phi_1 \rightarrow x\sharp_o\phi_2 \rightarrow x\sharp_o(\phi_1 \supset \phi_2)$
- $\sharp_6 : \Pi_{\Phi:i\rightarrow o} \cdot \Pi_{x,y:l} \cdot (x \neq y \rightarrow x\sharp_o\Phi(y!)) \rightarrow x\sharp_o\forall\Phi$
- $\mathtt{QF}_0 : \Pi_{t_1:i} \ . \ \Pi_{t_2:i}.\mathtt{QF}(t_1 = t_2)$,
- $\mathtt{QF}_2 : \Pi_{\phi_1:o}.\mathtt{QF}(\phi_1) \rightarrow \mathtt{QF}(\neg\phi_1)$
- $\mathtt{QF}_3 : \Pi_{\phi_1:o}\Pi_{\phi_2:o} \quad \mathtt{QF}(\phi_1) \rightarrow \mathtt{QF}(\phi_2) \rightarrow \mathtt{QF}(\phi_1 \supset \phi_2)$
- $\mathrm{Ass} : \Pi_{x:l} \cdot \Pi_{t:i} \cdot \Pi_{\Phi:i\rightarrow o} \cdot x\sharp_o\forall\Phi \rightarrow (\vdash_h \{\Phi(t)\}x := t\{\Phi(x!)\})$
- $\mathrm{Seq} : \Pi_{\phi_0,\phi_1,\phi_2:o}\Pi_{w_1,w_2:w}$

$$\vdash_h \{\phi_0\}w_1\{\phi_1\} \rightarrow \quad \vdash_h \{\phi_1\}w_2\{\phi_2\} \rightarrow \quad \vdash_h \{\phi_0\}w_1; w_2\{\phi_2\}$$

- $\mathrm{If} : \Pi_{\phi:o}\Pi_{\phi_1:o}\Pi_{\phi_2:o}\Pi_{w_1:w}\Pi_{w_2:w}\Pi_{p:\mathtt{QF}(\phi)}$

$$\vdash_h \{\phi_1 \wedge \phi\}w_1\{\phi_2\} \rightarrow \vdash_h \{\phi_1 \wedge \neg\phi\}w_2\{\phi_2\} \rightarrow \vdash_h \{\phi_1\}\mathrm{if}(\phi, p, w_1, w_2)\{\phi_2\}$$

- $\mathrm{While} : \Pi_{\phi:o}\Pi_{\phi_1:o}\Pi_{w_1:w}\Pi_{p:\mathtt{QF}(\phi)}$

$$\vdash_h \{\phi_1 \wedge \phi\}w_1\{\phi_1\} \rightarrow \quad \vdash_h \{\phi_1\}\mathrm{while}(\phi, p, w_1)\{\neg\phi_1\}$$

- $\mathrm{Con} : \Pi_{\phi_1:o}\Pi_{\phi_1':o}\Pi_{\phi_2:o}\Pi_{\phi_2':o}\Pi_{w_1:w}$

$$\vdash_o \phi_1 \Rightarrow \phi_1' \rightarrow \quad \vdash_o \phi_2' \Rightarrow \phi_2 \rightarrow \quad \vdash_h \{\phi_1'\}w_1\{\phi_2'\} \rightarrow \quad \vdash_h \{\phi_1\}w_1\{\phi_2\}$$

Adequacy and Faithfulness Property 9 *Let Γ_n^m be the following context, for m, n integers:*

$$\Gamma_n^m = \{y_0 : i, \ldots, y_m : i, x_0 : l, \ldots, x_n : l, z_{i,j} : x_i \neq x_j, z_{i,j}^* : x_j \neq x_i\}_{0 \leq i \leq j \leq n}.$$

In the above LF signature and in the context Γ_n^m we have the following facts concerning syntax:

- *l : All well formed long $\beta\eta$–normal forms of type l are LF variables of type l, and hence are among the $x_0, \ldots, x_n$.*
- *There are compositional bijections τ_K, for each syntactic category K, which map long $\beta\eta$–normal forms of type K to:*
  - *well formed terms of the assertion language built up from the set of identifiers $\bar{x}$ and the logical variables $\bar{y}$, in the case when $K = i$.*

- *formulas of the assertion language built up from the set of identifiers $\bar{x}$ (which if they occur must occur free) and the logical variables $\bar{y}$, in the case when $K = o$.*
- *while programs of τ built up from the set of identifiers $\bar{x}$ and the logical variables $\bar{y}$ (which do not occur in the left hand side of any assignment statement), in the case when $K = w$.*
- *asserted programs (i.e. Hoare triples) built up from the set of identifiers $\bar{x}$ and the logical variables $\bar{y}$ (here again no variable from $\bar{y}$ can occur in the left hand side of any assignment statement), in the case of $K = h$.*

- *There is a compositional bijection between proofs of a Hoare triple $\{p\}S\{q\}$ from assumptions $r_0, \ldots, r_s$ (in the assertion language) and assumptions $\{p_0\}S_0\{q_0\}, \ldots, \{p_t\}S_t\{q_t\}$ (concerning asserted programs) and well formed $\beta\eta$–normal forms of type*

$$\vdash_h \{p\}S\{q\}$$

in the above signature and in the context Γ where

$$\Gamma = \Gamma_n^m \cup \{w_j : (\vdash_o r_j), v_i : (\vdash_h \{p_i\}S_i\{q_i\})\}_{0 \leq j \leq s, 0 \leq i \leq t},$$

and Γ_n^m is adequate for the syntax of objects involved.

4.10 Two- Register Machine Hoare's Logic

4.10.1 The System

Another approach to Hoare's logic is not to reason about Hoare triples directly but rather deal primarily with functions from state to triples. Explicity we deal with objects obtained from triples by abstracting the program locations. Thus we must restrict our attention to assertions concerning programs built up from a fixed and finite number of such locations. In the abridged signature we present below this number is two.

4.10.2 The Signature Σ_{HL^2}

- Syntactic Categories
  - l : Type
  - i : Type
  - o : Type
  - w : Type
  - h : Type

- Operations
  - $! : l \to i$
  - $0 : i$
  - $+ : i \to i \to i$
  - $= \ : i \to i \to o$
  - $\neg : o \to o$
  - $\supset : o \to o \to o$
  - $\forall : (i \to o) \to o$
  - $\ldots := \ldots : l \to i \to w$
  - $\ldots ; \ldots : w \to w \to w$
  - $\{\ldots\}\ldots\{\ldots\} : o \to w \to o \to h$
- Judgements
  - $\vdash : (l \to l \to h) \to \text{Type}$
- Axioms and Rules
  - $\text{Ass}_1 : \Pi_{t:l\to l\to i} \cdot \Pi_{\Phi:i\to i\to o} \cdot$
    $$\vdash \lambda x : l \ . \ \lambda y : l.\{\Phi(t(x,y),y!)\}x := t(x,y)\{\Phi(x!,y!)\}$$
  - $\text{Ass}_2 : \Pi_{t:l\to l\to i} \cdot \Pi_{\Phi:i\to i\to o} \cdot$
    $$\vdash \lambda y : l \ . \ \lambda x : l.\{\Phi(t(x,y),y!)\}x := t(x,y)\{\Phi(x!,y!)\}$$
  - $\text{Seq} : \Pi_{\phi_0,\phi_1,\phi_2:l\to l\to o} \cdot \Pi_{w_1,w_2:l\to l\to w} \cdot$
    $$(\vdash \lambda x : l \ . \ \lambda y : l.\{\phi_0(x,y)\}w_1(x,y)\{\phi_1(x,y)\}) \to$$
    $$(\vdash \lambda x : l \ . \ \lambda y : l.\{\phi_1(x,y)\}w_2(x,y)\{\phi_2(x,y)\}) \to$$
    $$(\vdash \lambda x : l \ . \ \lambda y. : l.\{\phi_0(x,y)\}w_1(x,y);w_2(x,y)\{\phi_2(x,y)\})$$

We shall not state an adequacy and faithfullness property for the above signature. It follows the usual pattern.

References

[1] Arnon Avron. *Simple Consequence Relations.* Technical Report, Laboratory for the Foundations of Computer Science, Edinburgh University, 1987. In preparation.

[2] Arnon Avron. *Modal logics in the Edinburgh LF.* Technical Report, Laboratory for the Foundations of Computer Science, Edinburgh University, 1987. In preparation.

[3] Arnon Avron. *Gentzenn's methods in three-valued logic.* Technical Report, Laboratory for the Foundations of Computer Science, Edinburgh University, 1987. In preparation.

[4] Arnon Avron. *The semantics and Proof Theory of Linear Logic.* Technical Report, Laboratory for the Foundations of Computer Science, Edinburgh University, 1987. ECS-LFCS-87-27.

[5] Nicolas G. de Bruijn. A survey of the project AUTOMATH. In J. P. Seldin and J. R. Hindley, editors, *To H. B. Curry: Essays in Combinatory Logic, Lambda Calculus, and Formalism*, pages 589–606, Academic Press, 1980.

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

[7] Robert L. Constable, *et. al. Implementing Mathematics with the NuPRL Proof Development System.* Prentice–Hall, Englewood Cliffs, NJ, 1986.

[8] Thierry Coquand. *Une théorie des constructions.* Thèse de Troisième Cycle, Université Paris VII, January 1985.

[9] Mariangiola Dezani, Furio Honsell and Simonetta Ronchi della Rocca. *Models for Theories of Functions Strictly Depending on all their Arguments. Journal of Symbolic Logic* 51:3, 1986. Abstract.

[10] Jean-Yves Girard. Linear Logic. *Theoretical Computer Science.* to appear.

[11] Michael Gordon, Robin Milner, and Christopher Wadsworth. *Edinburgh LCF: A Mechanized Logic of Computation.* Volume 78 of *Lecture Notes in Computer Science*, Springer-Verlag, Heidelberg, 1979.

[12] Robert Harper, Furio Honsell, Gordon Plotkin. *A Framework for Defining Logics.* Proceedings of the Second Annual Conference on Logic in Computer Science, Cornell, 1987.

[13] L. S. Jutting. *Checking Landau's Grundlagen in the AUTOMATH System.* PhD thesis, Eindhoven University, The Netherlands, 1977.

[14] Per Martin-Löf. An intuitionistic theory of types: predicative part. In H. E. Rose and J. C. Shepherdson, editors, *Logic Colloquium, '73*, pages 73–118, North-Holland, Amsterdam, 1973.

[15] Per Martin-Löf. *On the Meanings of the Logical Constants and the Justifications of the Logical Laws.* Technical Report 2, Scuola di Specializzazione in Logica Matematica, Dipartimento di Matematica, Università di Siena, 1985.

[16] Ian A. Mason. *Hoare's Logic in the LF.* Technical Report, Laboratory for the Foundations of Computer Science, Edinburgh University, 1987. In preparation.

[17] Albert Meyer and Mark Reinhold. 'Type' is not a type: preliminary report. In *Proceedings of the 13th ACM Symposium on the Principles of Programming Languages*, 1986.

[18] Bengt Nordström, Kent Petersson, and Jan Smith. *An Introduction to Martin-Löf's Type Theory.* University of Göteborg, Göteborg, Sweden, 1986. Preprint.

[19] Lawrence Paulson. Natural deduction proof as higher–order resolution. *Journal of Logic Programming*, 3:237–258, 1986.

[20] Kent Petersson. *A Programming System for Type Theory.* Technical Report 21, Programming Methodology Group, University of Göteborg/Chalmers Institute of Technology, March 1982.

[21] Gordon Plotkin. *Call–by–name, Call–by–value and the λ–calculus. Theoretical Computer Science*, 1:125–159, 1975.

[22] Dag Prawitz. *Natural Deduction: A Proof-Theoretical Study.* Almquist & Wiksell, Stockholm, 1965.

[23] Reynolds, J.C. *Syntactic Control of Interference.* Conference Record of the Fifth Annual Symposium on Principles of Programming Languages, Tucson, 1978.

[24] Joseph R. Schoenfield. *Mathematical Logic.* Addison–Wesley, Reading, Massachusetts, 1967.