The University of Edinburgh

Department of Computer Science

Representing Logics in Type Theory

by

Philippa Gardner

CST-93-92
(also published as ECS-LFCS-92-227
July 1992

Department of Computer Science
James Clerk Maxwell Building
The King's Buildings
Mayfield Road
Edinburgh EH9 3JZ

Representing Logics in Type Theory
Philippa Gardner

Representing Logics in Type Theory

Philippa Gardner

Doctor of Philosophy
University of Edinburgh
January 1992

(Graduation date July 1992)

To my father

Abstract

Computer Science today contains many examples of logics given by proof systems. Although one intuitively knows how to use these systems and recognise correct derivations, there is no definitive account which captures this intuition. It is therefore natural to seek a framework for representing logics, which unifies the structure common to all logics. We introduce such a framework, called ELF^+ and based on the Edinburgh Logical Framework (ELF). The major advantage of ELF^+ is that it allows us to give, apparently for the first time, *general* definitions of correct representation; such definitions are not possible with ELF since information is lost during encoding. We rectify this deficiency using the extra distinctions between terms provided by the universes of a Pure Type System which yields a simple presentation of the type theory of ELF^+. To do this, we extend these type systems to include signatures and $\beta\eta$-equivalence.

Using the ideas underlying representation in ELF^+, we give a standard presentation of the logics under consideration, based on Martin-Löf's notion of judgements and Aczel's work on Frege structures. This presentation forms a reference point from which to investigate representations in ELF^+; it is not itself a framework since we do not specify a logic using a finite amount of information. Logics which do not fit this pattern are particularly interesting as they are more difficult, if not impossible, to encode.

The syntactic definitions of representation have elegant algebraic formulations which utilise the abstract view of logics as consequence relations. The properties of the ELF^+ entailment relation determine the behaviour of the variables and consequence relations of the logics under consideration. Encodings must preserve this common structure. This motivates the presentation of the logics and their corresponding type theories as strict indexed categories (or split fibrations) so that encodings give rise to indexed functors. Our syntactic notions of correct representation now have simple formulations as indexed isomorphisms, which both confirms that our approach is a natural one and provides the beginnings of an algebraic framework for representing logics.

Acknowledgements

I would like to thank my supervisor, Gordon Plotkin, for many ideas and inspirational discussions, and for his help when it was needed. I am also indebted to John Power for his guidance and moral support, and to Eugenio Moggi and Anne Salvesen for their help in the early stages of this thesis.

My thanks also go to many friends and colleagues, associated with the Laboratory for the Foundations of Computer Science, for their advice, suggestions and friendship: in particular, to Stuart Anderson, Julian Bradfield, George Cleland, Robert Harper, Furio Honsell, Zhaohui Luo, James McKinna, Dale Miller, Randy Pollack and Alistair Sinclair, and to Claire Jones for her advice about LaTeX.

This work was supported by a SERC Research Studentship and, in the finishing stages, by the Laboratory for the Foundations of Computer Science.

Finally, I cannot thank my family enough for their support and encouragement throughout.

Declaration

I declare that this thesis was composed by myself, and the work contained in it is my own except where otherwise stated.

Philippa Gardner

Table of Contents

Chapter 1

Introduction

A wide variety of logics defined from proof systems are used in Computer Science today. Although one intuitively knows how to use these systems and recognise correct derivations, there is no definitive account which captures this intuition. It is therefore natural to seek a framework for representing logics which unifies the structure common to all logics. The framework must explain the notions central to these logics such as binding, discharge of assumptions and context sensitive side-conditions. Moreover, it should be easy to translate a logic to this unified setting and to recognise when this translation results in a representation of the logic. As well as providing insights into the important theoretical question of what a logic is, these goals have an immediate application in the provision of computer-assisted tools for reasoning with various logics.

Type theories have emerged as leading candidates for frameworks. In this thesis we introduce such a type theory for representing logics, called ELF^+ and based on the Edinburgh Logical Framework (ELF). The major advantage of ELF^+ is that it allows us to give precise definitions of representation. Such definitions are not possible with ELF since information is lost during encoding; the adequacy theorems of ELF representations are only applicable to particular encodings and cannot be generalised. Using these definitions, we give examples of 'good' representations, prove that linear and relevant logics [Gir87] [Dun84] cannot be well-represented and show that the representation of Hilbert-style S_4 [Che80] is not as natural as, for example, the representation of first-order logic.

Type theories and logics

Type theories and minimal intuitionistic logics are connected by the well-known *propositions-as-types principle*, sometimes called the Curry–Howard correspondence [CF58] [How80]. With this interpretation a proposition is viewed as a type whose inhabitants correspond to proofs of this proposition. The idea was developed further by de Bruijn [dB70] and Martin-Löf [Mar80], who associated the Π-abstraction of dependent type theories with universal quantification. A comprehensive account of these ideas can be found in [Bar90], which presents various intuitionistic minimal logics as Pure Type Systems [Ber90] [Ter89].

A different interpretation of logics in type theory involves the *judgements-as-types* principle, advocated in the work on the Edinburgh Logical framework (ELF) [HHP89]. It is inspired by Martin-Löf's emphasis of formal systems as calculi for constructing derivations of basic judgements [Mar85]. These judgements are the formulae in first-order logic and the terms $A\,set$, $A = B$, $a \in A$ and $a \in A = B$ in Martin-Löf's type theory. Rules are expressed using two higher-order forms. If J and K are basic judgements, then the *hypothetical* judgement $J \to K$ expresses a form of consequence, that K is provable under the assumption J, and the *general* judgement $\to_x J$ expresses the fact that J is provable uniformly in variable x. The judgements-as-types principle asserts that judgements are identified with types whose objects correspond to derivations in the logic, with the Π-abstraction being used to represent the higher-order forms. Certain type theories are, thus, candidates for providing a general metatheory for various logics defined using proof systems.

The Edinburgh Logical Framework

We give a brief outline of representation in the Edinburgh Logical Framework to illustrate the problems which arise when using this framework. The Edinburgh Logical Framework (ELF) of Harper, Honsell and Plotkin [HHP89] is a type theory, closely related to several of the AUTOMATH languages [dB80]. It is based on a

λ-calculus with first-order dependent types with three levels of terms: objects, types (which classify the objects) and kinds (which classify families of types). Terms are identified up to $\beta\eta$-equivalence; all matters relating to representations of logics are treated up to this equality. The specification of a logic is given by a *signature*, declaring a finite list of constants; ELF together with this signature forms the representing type theory of the logic. The method of finding such representations is informal and so each signature is accompanied by an *adequacy theorem* to show that we do indeed have a representation.

The treatment of the syntax in ELF is inspired by Church [Chu40] and Martin-Löf's theory of arities [NPS90]; binding operators are represented using the λ-abstraction of ELF and higher-order operators. For example, the signature of first-order logic with arithmetic, denoted by Σ_{Fol}, consists of two constant types ι and o whose inhabitants correspond to the arithmetic expressions and formulae respectively. We have constants $+ : \iota \to (\iota \to \iota)$ and $\forall : (\iota \to o) \to o$ for addition and universal quantification respectively. For ELF terms t and u corresponding to arithmetic expressions, the term $+(t)(u)$ represents their sum, and, for an ELF term ϕ corresponding to a formula, the term $\forall(\lambda x{:}\iota.\phi)$ represents the universal quantification.

As has already been mentioned, our representation of the rules of inference focuses on the notion of judgement stressed by Martin-Löf [Mar85]. Logics are represented in ELF by introducing the judgements-as-types principle mentioned earlier. The basic judgements of first-order logic are that propositions (expressed by formulae) are true, sometimes written as ϕ *true* to distinguish the concept of a formula being well-formed from that of it being true. These are represented by types of the form $\mathit{true}(\phi')$, where the constant *true* inhabits the appropriate kind and the object ϕ' is in o. The inhabitants of $\mathit{true}(\phi')$ are identified with the proofs of the formula denoted by ϕ'. The structure of the ELF type system allows for the uniform treatment of higher-order judgements as Π-abstractions so that rules can be represented by constants of the appropriate type. For example, the implication introduction rule of first-order logic is given by

$$\supset I : \Pi\phi, \psi : o.(\mathit{true}(\phi) \to \mathit{true}(\psi)) \to \mathit{true}(\phi \supset \psi).$$

Accompanying each ELF signature specifying a logic is an adequacy theorem to

provide some confirmation that the resulting type theory is a representation of the logic. It gives a correspondence, called an *encoding*, between the syntax and basic judgements (and, for a stronger result, the proofs) of the logic and certain ELF terms such that the consequence relation of the logic and the corresponding part of the ELF entailment relation coincide. For example, the adequacy theorem accompanying the representation of first-order logic with arithmetic provides bijections ε_X and δ_X between the arithmetic expressions and formulae with free variables in X and the $\beta\eta$-equivalence classes of objects in ι and o respectively such that, for formulae $\phi_1, \ldots, \phi_m, \phi$ and a set of variables $X = \{x_1, \ldots, x_n\}$, we have

$$\phi_1, \ldots, \phi_m \vdash_X \phi \text{ if and only if}$$
$$x_1 : \iota, \ldots, x_n : \iota, p_1 : true(\delta_X(\phi_1)), \ldots, p_m : true(\delta_X(\phi_m)) \vdash^{ELF}_{\Sigma_{Fol}} p : true(\delta_X(\phi))$$

where $p_1, \ldots, p_m$ are distinct ELF variables corresponding to proofs of assumptions $\phi_1, \ldots, \phi_m$ and the ELF term p in $true(\delta_X(\phi))$ denotes some proof of ϕ.

Limitations of ELF

ELF represents an important advance in the study of formal systems. However, there are problems with using this type theory as a framework, as is illustrated by the simple encoding of first-order logic outlined above. The adequacy theorem for first-order logic only applies to this particular representation, since it refers to specific constants ι, o and *true* declared in Σ_{Fol}. It cannot be stated generally as information is lost in the encoding owing to the types being used for many purposes. The terms ι, o and $true(\phi)$ for $\phi : o$ are all types, so we cannot distinguish which objects correspond to arithmetic expressions, formulae and proofs without appealing to the particular types they inhabit. The machinery of ELF also gives rise to types which are meaningless from a first-order logic perspective: for example, the type $\Pi x{:}o.\iota$. Extra types, having no general correspondence with the underlying logic, also arise from less direct encodings; in some cases, this results in logics with different consequence relations being specified by the same signature, which is clearly undesirable. In this thesis we introduce a new framework, which allows us to make such distinctions without reference to specific types and to give a precise definition of representations in the type theory.

The new framework ELF$^+$

We define a new framework, called ELF$^+$ and based on ELF, which retains the information lost in the ELF encodings, and so allows for a generalisation of the adequacy theorems of ELF to apply to all signatures specifying logics. This generalisation yields equivalences between logics and their representing type theories. A problem with ELF is that there is not enough distinction between the ELF types to determine, in advance, the part of the entailment relation which corresponds to the consequence relation of the logic, without appealing to the particular encoding under consideration. This distinction is achieved in the new framework ELF$^+$ by splitting the ELF types into three: *sorts*, *types* and *judgements*. The method of representation in the new framework does not differ greatly from that of ELF. The difference becomes apparent in the analysis of the resulting type theories. The interpretation of the consequence relation of the logic in the entailment relation of the representing type theory is defined as a correspondence between the basic judgements and the ELF$^+$ judgements, and between the terms of the logic and the inhabitants of ELF$^+$ sorts, such that the entailment relation gives a sound representation of the consequence relation; such a correspondence is called an *encoding*. An encoding is *adequate* when the correspondence provides an equivalence between the logic and the part of the entailment relation determined by the sorts and judgements; examples of logics which have adequate representations in ELF$^+$ include first- and higher-order logic [Pra65] [Chu40], and Hilbert-style S_4 [Che80]; examples which cannot be represented include linear and relevant logics [Gir87] [Dun84]. A *natural encoding* requires a stronger link between proofs and inhabitants of judgements; our representation of Hilbert-style S_4 is adequate, but not natural.

The syntactic definitions of representations have an elegant algebraic formulation which utilises the abstract view of logics as consequence relations. The properties of the ELF$^+$ entailment relation determine the behaviour of the variables and consequence relations of the logics under consideration. Encodings must preserve this common structure. This motivates the presentation of the logics and their corresponding type theories as strict indexed categories (or split fibrations)

so that encodings give rise to indexed functors. More specifically, a logic, with a consequence relation satisfying certain properties, provides an indexed category whose base presents the terms and whose fibres are defined by the consequence relation. This approach uses ideas from the area of categorical logic (initiated by Lawvere [Law70]) applied in general to a wide class of logics, rather than to particular logics. Using ELF^+, we are also able to present the representing type theory as an indexed category with the sorts providing the base category and the judgements the fibres. Adequate encoding corresponds to indexed isomorphisms. By adapting both indexed categories to incorporate the extra information regarding the proofs and the inhabitants of judgements, natural encodings also yield isomorphisms between indexed categories. As well as giving a simple algebraic formulation of encodings, this approach provides us with the beginnings of an algebraic framework for logics.

Not all logics can be represented in ELF^+ (or in ELF). There are various reasons for this: different behaviours of the logic variables, the consequence relation having properties incompatible with those of the entailment relation, rules with 'awkward' side-conditions. Using the ideas underlying representations in ELF^+, we give a *standard* presentation of logics defined using formal systems, based on Martin-Löf's notion of judgements [Mar85] and Aczel's work on Frege structures [Acz80]. This presentation forms a reference point from which to investigate representations in ELF^+; it is not itself a framework since we do not specify a logic using a finite amount of information. Logics which do not fit this pattern are particularly interesting as they are more difficult, if not impossible, to encode.

The distinction between terms required by the new framework exploits, and was partially inspired by, the techniques of Beradi and Terlouw in extending Barendregt's λ-cube [Bar90] to Pure Type Systems (PTSs) [Ber90] [Ter89]. Both ELF and ELF^+ are presented as PTSs, adapted to allow for signatures and $\beta\eta$-equality, to emphasise the link between the two type theories and provide results for ELF^+ via those for ELF. We use Salvesen's method [Sal89] of incorporating η in ELF to extend the β-equivalence of PTSs. To prove the decidability of ELF^+, which is essential for the reduction of proof-checking to type-checking, the Church–Rosser property (CR) for $\beta\eta$-equivalence is required. This is not known for general PTSs and involves a lengthy and subtle proof for ELF [Sal89]. We

prove CR for ELF$^+$ from the corresponding results for ELF and the untyped λ-calculus, thus avoiding the technical details required in Salvesen's proof [Sal89]. In recent unpublished work [Sal91], Salvesen extends her ideas to include a wide class of PTSs[1]. Her results include the Church–Rosser property, subject reduction and imply the decidability of ELF$^+$.

This thesis introduces and develops the new framework ELF$^+$ in chapters 4, 5 and 6, which form the main part of the thesis. An introduction to PTSs is found in chapter 2. There, the definition of a PTS is adapted to account for signatures and $\beta\eta$-equality on well-typed terms. The standard presentation of logics is given in chapter 3, with substitution results handled in this general setting to make the transfer to the algebraic presentation straightforward. In chapter 4, the new framework is defined and motivated and the method of representation illustrated. An informal account of the encoding definitions is given, with the formal justification presented in chapter 5. Examples and counter-examples illustrate the definitions. The behaviour of the variables and the properties of the consequence relation of encoded logics, determined by the ELF$^+$ entailment relation and emphasised in the logic chapter, motivates the presentation of the logics and their corresponding type theories as indexed categories (split fibrations), given in chapter 6. Adequate and natural encodings correspond to isomorphisms between the appropriate indexed categories. We conclude by summarising our achievements and describing some possible future research resulting from our work.

Related research

The new framework is based on the Edinburgh Logical Framework (ELF) [HHP89], adapted using techniques from Pure Type Systems (PTSs) [Bar90]. The design of ELF was influenced by AUTOMATH [dB80] and by Martin-Löf's work on the foundations of intuitionistic logic [Mar80]. The seminal work on machine-assisted proof was initiated in the late 1960s by de Bruijn, whose goal was to develop a framework for expressing arbitrary mathematical arguments in a notation suitable

[1]Geuvers has recently proved CR for functional, normalising PTSs.

for checking by a machine. His approach was based on representing mathematical texts as terms in a typed λ-calculus, reducing proof checking to type checking. A variety of mathematical theories have been developed and checked, most notably the formalisation of Landau's textbook on Mathematical Analysis [Jut77]. This work has been important in the development of machine-assisted proof, especially the NuPRL system of Constable et al. [Con86] and the Calculus of Constructions of Coquand and Huet [Coq85]. However, this subsequent research differs in spirit from AUTOMATH in that the latter two are concerned only with the formalisation of constructive mathematics, whereas AUTOMATH sought to encompass classical mathematics as well. The ELF project can be viewed as a development of the AUTOMATH ideas that seeks to keep a clear distinction between the object- and meta-level. The ELF approach differs in that it aims to develop a general theory of representations of formal systems.

The work of Martin-Löf [NPS90] influenced the design of ELF and hence ELF$^+$. In particular, his emphasis on the notion of judgement and on its uniform extension to higher-order forms was very important [Mar85]. The system of 'logical types' (as yet unpublished, but see [NPS90]), providing a basis for his intuitionistic set theory, is formally similar to the LF type theory, but the applications are substantially different. In particular, work on ELF is concerned with encoding formal proofs in arbitrary logical systems and is not concerned with specifically intuitionistic problems such as proof normalisation. In contrast, Martin-Löf uses the system of logical types as the foundation for his set theory and does not consider its application to general formal systems. Martin-Löf also separates expressions, by using judgements $A\ set$, $A\ prop$, $A\ true$ for expressions A, in a similar fashion to the distinctions of ELF$^+$-terms with comparable, although not identical, uses. The framework Alf [ACN90] is based on Martin-Löf's ideas.

The extension of Barendregt's λ-cube [Bar90] to PTSs [Ber90] [Ter89] provides a greater range of type theories, one of which forms the basis of ELF$^+$. Barendregt [Bar90] has popularised the notion of PTS and presents various intuitionistic minimal logics as type theories using the propositions-as-types paradigm [CF58] [How80]. This use of type systems to present logics contrasts with the ELF$^+$ approach where the type system is used as a metatheory, rather than to present particular formal systems which happen to fit. The idea of adding signatures to

PTSs appears in Beradi's thesis [Ber90]. Pollack [Pol9-] is currently working on an implementation of PTSs which, therefore, provides an implementation of ELF$^+$, albeit without signatures and η.

There has been much work associated with the ELF project. Our emphasis on consequence relations is motivated by Avron's research [Avr91]. Salvesen [Sal89] proves the Church–Rosser property for the $\beta\eta$-equivalence of ELF which is used in the corresponding result for ELF$^+$. In recent unpublished work, she extends her ideas to functional PTSs with η satisfying strong normalisation. Her work implies the decidability of ELF$^+$. Geuvers has very recently proved the Church–Rosser property for functional, normalising PTSs [Geu91]; throughout this thesis we just refer to Salvesen's result as this was the one known to us when writing. Various examples of formal systems have been encoded in ELF. These include two different variations on Hoare logic [AHMP87] [AHM89], modal logics from K to S4 [AHMP87], various λ-calculi, including λ_I, λ_v, and linear λ-calculus [AHMP87] [AHM89], various type theories, including the LF type system itself, Martin-Löf's type theory, and the Damas-Milner type assignment system [Har90]. Other results, which have not been used directly in this thesis but which are still significant to ELF$^+$, can be found in [Pym90], [Ell90], [PW91] and [Pfe91]. Work is currently under way to produce a 'linear' ELF [MPP92], incorporating ideas from Girard's linear logic [Gir87]. Other current research includes Simpson's study [Sim93] of a semantic analysis of representations in a framework with two universes using Kripke models [Che80], rather than the syntactic analysis studied here. Pfenning's system Elf [Pfe91] is a logic programming implementation of ELF and Pollack's LEGO system [Pol9-] provides another implementation; Pollack's extension of the LEGO system to a wide class of PTSs [Pol9-] provides an imlementation of ELF$^+$.

Paulson's Isabelle system uses ideas similar to ELF in the context of higher-order logic [Pau87]. Nipkow has adapted the Isabelle system to allow for order-sorted polymorphism [Nip91]. Constable and Howe [CH90] demonstrated the use of NuPRL as a logical framework, emphasising the use of the richer type structure of the NuPRL type theory in an encoding. Felty has studied the representation of logics in λ-Prolog, in particular the LF type theory itself [Fel89]. Mendler and Aczel [MA88] are developing the theory of MaThImP, as a system for doing inter-

active mathematics on a machine that is also based on a general theory of logical systems, albeit of a rather different flavour than that considered here. Feferman has proposed a theory of formal systems based on a general system of finitary inductive definitions [Fef89].

Our algebraic account of representations in ELF^+ uses ideas from the area of categorical logic initiated by Lawvere [Law70], but applied to logics in general rather than to particular logics. We concentrate on the abstract view of logics as consequence relations due to Tarski [Tar56], which is also utilised by Harper, Sannella and Tarlecki [HST89] when they introduce a general definition of a logical system as a family of consequence relations indexed by signatures. There are various case studies of logics modelled as categories (see Seely's work on hyperdoctrines [See83] and the references therein, and also [Amb91] for an example of a categorical presentation of a first-order linear logic). There has been much work on presenting dependent type theories categorically: for example, contextual categories [Car78] [Str89], categories with fibrations [Pit89], comprehension categories [Jac91]; see Jacobs [Jac91] for a more complete list. General approaches to modelling logics include the work on institutions [BG90] and model-theoretic logics [BF85]. Algebraic accounts of logics are investigated by Meseguer [Mes89], Aczel [Acz91] and Pym [Pym91], and Mendler and Aczel [MA88] provide an algebraic notion of framework for, in particular, the Logical Theory of Constructions [ACM90].

Chapter 2

Pure Type Systems

Pure Type Systems (PTSs) [Bar90], sometimes called Generalised Type Systems, provide a concise notation for presenting many type systems in a unified way; they originate from the work of Beradi [Ber90] and Terlouw [Ter89] who generalise Barendregt's 'cube of typed λ-calculi' [Bar90]. We define ELF and the new framework, ELF$^+$, using this notation to provide simple presentations which emphasise the link between the two frameworks and provide results for ELF$^+$ via those for ELF. In order to give a precise account of these frameworks however, we must extend the PTS structure to include $\beta\eta$-equivalence and signatures.

2.1 Introduction to Pure Type Systems

A PTS is specified by sets of *universes*, *axioms* and *rules* which determine the syntax and proof system of the type theory. The universes provide the starting point on which the type theory is based, with some universes inhabiting others as indicated by the axioms. For example, ELF has two universes *Type* and *Kind*, with the universe *Type* inhabiting *Kind*. The rules determine which families of terms are allowed; that is, they control the formation of terms of shape $\Pi x{:}A.B$.

2.1.1 DEFINITION A *specification* of a Pure Type System is a triple $(\mathcal{U}, \mathcal{A}, \mathcal{R})$ where

- $\mathcal{U}$ is a set, called the set of *universes*;
- $\mathcal{A} \subseteq \mathcal{U} \times \mathcal{U}$ is the set of *axioms*;
- $\mathcal{R} \subseteq \mathcal{U} \times \mathcal{U} \times \mathcal{U}$ is the set of *rules*.

Remark

1. This definition is a very minor restriction of Barendregt's definition where $\mathcal{U}$ (called the set of sorts in [Bar90]) is a subset of a set of constants $\mathcal{C}$ and the axioms are elements of $\mathcal{C} \times \mathcal{U}$. Barendregt only gives one example which uses this extra expressivity for the axioms and, in example 2.2.2, we argue that this example is misleading and propose an alternative presentation using signatures.

2. A full understanding of the role of the rules is only achieved by studying the proof system derived from the specification $(\mathcal{U}, \mathcal{A}, \mathcal{R})$ (definition 2.2.5). The intuition is that, for rule $(u, v, w) \in \mathcal{R}$, we can form the term $\Pi x{:}A.B$ inhabiting w if A inhabits u and if B inhabits v assuming x inhabits A.

Notation The axiom (u, v) is written as $u : v$ and the rule (u, v, v) is usually abbreviated to (u, v).

Let Var be a countably infinite set of *variables*. It is useful to divide Var into disjoint finite subsets Var^u for each $u \in \mathcal{U}$; that is, $Var = \bigcup_{u \in \mathcal{U}} Var^u$. The members of Var are usually denoted by x, y, z; when the universe is important we write x^u for $u \in \mathcal{U}$.

Let $(\mathcal{U}, \mathcal{A}, \mathcal{R})$ be a specification of a Pure Type System ζ. The set of *preterms* $\mathcal{T}$ is defined by the abstract syntax

$$\mathcal{T} ::= x \mid u \mid \Pi x{:}\mathcal{T}.\mathcal{T} \mid \lambda x{:}\mathcal{T}.\mathcal{T} \mid \mathcal{T}\mathcal{T},$$

where $u \in \mathcal{U}$ and $x \in Var^v$ for some $v \in \mathcal{U}$. We use the letters A, B, M, N to denote preterms. We say that preterm A is a λ-*abstraction* or Π-*abstraction* if A is

of the form $\lambda x{:}A_1.A_2$ or $\Pi x{:}A_1.A_2$ respectively, and let $A \to B$ abbreviate $\Pi x{:}A.B$ when $x \notin fv(B)$. The notions of free variables, substitution and α-conversion are special cases of definitions 3.1.10, 3.1.15 and 3.1.13.

2.1.2 DEFINITION For preterms M and N, M is a *subterm* of N if $M \in Sub(N)$ where $Sub(N)$, the set of subterms of N, is defined by

$$\begin{aligned} Sub(N) &= \{N\} \text{ if } N \text{ is a variable or universe;} \\ Sub(N) &= \{N\} \cup Sub(P) \cup Sub(Q), \\ &\qquad \text{if } N \text{ has one of the shapes } \Pi x{:}P.Q,\ \lambda x{:}P.Q \text{ or } PQ. \end{aligned}$$

The definitional equality on preterms is β-equality, which is defined as one would expect from the usual one-step β-reduction, denoted by $\to_\beta$ [Bar84]. We denote β-reduction by $\rhd_\beta$, the reflexive and transitive closure of $\to_\beta$, and let $=_\beta$ denote the corresponding congruence relation of β-conversion. An important property of β-reduction is the Church–Rosser property [Bar84]; that is, if $A \rhd_\beta B$ and $A \rhd_\beta C$ then $B \rhd_\beta D$ and $C \rhd_\beta D$ for some preterm D. For ζ-preterms A and B, $A : B$ is called a *ζ-assertion*; we refer to A as the first component of the ζ-assertion and often denote ζ-assertions by α. A *ζ-precontext* is a finite sequence, possibly empty, of ζ-assertions whose first components are all variables. For a ζ-precontext $\Gamma = \langle x_1 : A_1, \ldots, x_n : A_n \rangle$, $n \geq 0$, the domain of Γ, $dom(\Gamma)$, is $\{x_1, \ldots, x_n\}$. A precontext Γ *extends* precontext Δ if Γ is Δ, Δ' for some precontext Δ'. A precontext Γ is *contained in* Δ, denoted $\Gamma \subseteq \Delta$, if every $x : A$ in Γ is also in Δ. A *ζ-sequent* is of the form $\Gamma \vdash^\zeta A : B$, where Γ is a ζ-precontext and A and B are ζ-preterms; the relation $\vdash$ is the *entailment relation* of the type theory. We sometimes write $\Gamma \vdash^\zeta A : B : C$ as a shorthand for $\Gamma \vdash^\zeta A : B$ and $\Gamma \vdash^\zeta B : C$. The superscript ζ is omitted when the PTS is apparent.

2.1.3 DEFINITION The type system for the PTS ζ with specification $(\mathcal{U}, \mathcal{A}, \mathcal{R})$ is defined by the following formal system:

AXIOM $\quad \langle\,\rangle \vdash u : v \qquad\qquad u : v \in \mathcal{A}$

START $\quad \dfrac{\Gamma \vdash A : u}{\Gamma, x : A \vdash x : A} \qquad u \in \mathcal{U}, x \in Var^u, x \notin dom(\Gamma)$

$$\textsc{Weak}\quad \frac{\Gamma \vdash A : u \qquad \Gamma \vdash B : C}{\Gamma, x : A \vdash B : C} \qquad u \in \mathcal{U}, x \in Var^u, x \notin dom(\Gamma)$$

$$\Pi\quad \frac{\Gamma \vdash A : u \qquad \Gamma, x : A \vdash B : v}{\Gamma \vdash \Pi x{:}A.B : w} \qquad (u, v, w) \in \mathcal{R}$$

$$\lambda\quad \frac{\Gamma \vdash \Pi x{:}A.B : u \qquad \Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x{:}A.M : \Pi x{:}A.B} \qquad u \in \mathcal{U}$$

$$\textsc{App}\quad \frac{\Gamma \vdash M : \Pi x{:}A.B \qquad \Gamma \vdash N : A}{\Gamma \vdash MN : B[N/x]}$$

$$\textsc{Conv}\quad \frac{\Gamma \vdash A : B \qquad \Gamma \vdash B' : u}{\Gamma \vdash A : B'} \qquad B =_\beta B', u \in \mathcal{U}$$

A ζ-precontext Γ is a *ζ-context* if Γ is empty or there exist ζ-preterms A and B such that $\Gamma \vdash^\zeta A : B$. A ζ-preterm A is a *ζ-term* if $\Gamma \vdash^\zeta A : B$ for precontext Γ and preterm B.

Barendregt uses the abstract symbols $*$, $\Box$ and $\triangle$ to denote universes. When investigating a particular type theory, universes are often labelled more concretely as, for example, *prop*, *set*, *type* and *kind*, to convey some meaning relative to the type theory of interest. This becomes confusing when providing a unified theory since, for example, the universe *type* has different meanings in the Calculus of Constructions [Coq85] and ELF [HHP89]. We use Barendregt's notation in this chapter and revert to names when describing the new framework.

2.1.4 EXAMPLE The Calculus of Constructions [Coq85] can be presented as PTS with specification λC given by

$$\begin{aligned} \mathcal{U} &= \{*, \Box\} \\ \mathcal{A} &= \{* : \Box\} \\ \mathcal{R} &= \{(*,*), (*,\Box), (\Box,*), (\Box,\Box)\} \end{aligned}$$

The $*$ corresponds to *Prop* and $\Box$ corresponds to *Type* in [Bar90]. The subsystem with only the rule $(*,*)$ is simply typed λ-calculus with type variables giving the basic types. The rules $(\Box,*)$, $(*,\Box)$ and $(\Box,\Box)$ add polymorphic types, dependent types and higher-order features respectively. The subsystem with the rules $(*,*)$ and $(*,\Box)$ is the $\lambda\Pi$-calculus on which ELF is based (in this case $*$ corresponds to *Type* and $\Box$ to Kind in [HHP89]). The system with the rules $(*,*)$ and $(\Box,*)$ is essentially Girard's system F [Gir72], and adding the rule $(\Box,\Box)$ to system F corresponds to $F\omega$ [Gir72]. In this way we obtain a collection of eight type systems all containing simply typed λ-calculus and all contained in the Calculus of Constructions. This collection is known as the 'cube of λ-calculi' [Bar90].

2.1.5 EXAMPLE PTSs are also used to present many-sorted minimal intuitionistic logics. For example, Geuvers [Geu90] defines a higher-order logic based on Church's presentation [Chu40] as the PTS with specification

$$\begin{array}{lll} \lambda Hol & \mathcal{U} = & \{*,\Box,\triangle\} \\ & \mathcal{A} = & \{*:\Box,\Box:\triangle\} \\ & \mathcal{R} = & \{(*,*),(\Box,*),(\Box,\Box)\} \end{array}$$

With this definition, objects of $\Box$ correspond to the domains of higher-order logic; in particular, the domain $*$, which contains the formulae, is distinguished. The universe $\triangle$ is a starting universe declared so that domains other than $*$ can be given. Functional and predicate domains and the objects therein are formed using the rule $(\Box,\Box)$. Rule $(*,*)$ provides the logical implication and $(\Box,*)$ provides the quantification, with the corresponding introduction and elimination rules given by the λ- and APP rules of the PTS.

2.1.6 EXAMPLE Not all PTSs are normalising; for example, the PTS specified by

$$\begin{array}{ll} \mathcal{U} = & \{*\} \\ \mathcal{A} = & \{*:*\} \\ \mathcal{R} = & \{(*,*)\} \end{array}$$

is an inconsistent system in the sense that every type is inhabited [Bar90].

The following results, taken from [GN91] unless otherwise stated, give the elementary properties of the entailment relation of an arbitrary PTS specified by

$(\mathcal{U}, \mathcal{A}, \mathcal{R})$. They have important repercussions when using a PTS to represent a logic since the properties of the entailment relation and the consequence relation must be compatible (as discussed in section 3.2).

The first lemma, adapted from a result in [GN91], states that we can only use variables that we declare.

2.1.7 LEMMA [Free variable lemma] Suppose $\Gamma \vdash B : C$ for $\Gamma = \langle x_1{:}A_1, \ldots, x_n{:}A_n \rangle$. Then

1. the $x_1, \ldots, x_n$ are distinct;
2. $fv(B), fv(C) \subseteq \{x_1, \ldots, x_n\}$;
3. any derivation of $\Gamma \vdash B : C$ has as subderivation $x_1{:}A_1, \ldots, x_{i-1}{:}A_{i-1} \vdash A_i : u_i$ for some $u_i \in \mathcal{U}$ and any $i \in \{1, \ldots, n\}$.

Proof By induction on the derivation of $\Gamma \vdash B : C$. □

2.1.8 COROLLARY A precontext $\Gamma = \langle x_1{:}A_1, \ldots, x_n{:}A_n \rangle$ is a context if and only if $x_1{:}A_1, \ldots, x_{i-1}{:}A_{i-1} \vdash A_i : u_i$ for some $u_i \in \mathcal{U}$ and for each $i \in \{1, \ldots, n\}$, and the $x_1, \ldots, x_n$ are distinct.

The next few lemmas show that contexts behave as expected.

2.1.9 LEMMA Let Γ be a ζ-context. Then

1. $\Gamma \vdash u : v$ for all $u : v \in \mathcal{A}$;
2. $\Gamma \vdash x : A$ whenever $x : A$ is in Γ.

Proof By assumption $\Gamma \vdash B : C$ for preterms B and C. The result follows by induction on the derivation of $\Gamma \vdash B : C$. □

2.1.10 LEMMA [Substitution] If $\Gamma, x : A, \Gamma' \vdash B : C$ and $\Gamma \vdash M : A$ then $\Gamma, \Gamma'[M/x] \vdash B[M/x] : C[M/x]$.

Proof By induction on the derivation of $\Gamma, x : A, \Gamma' \vdash B : C$. We consider two cases; the others are trivial or similar.

Case 1. The last line is obtained from the START rule and the proof is split into two subcases.

Subcase 1. If Γ' is $\Gamma'', y : C$ and B is y, and the last line in the derivation is

$$\frac{\Gamma, x : A, \Gamma'' \vdash C : u}{\Gamma, x : A, \Gamma'', y : C \vdash y : C}$$

for $u \in \mathcal{U}, y \in Var^u$ and $y \notin dom(\Gamma, x : A, \Gamma'')$. By induction, it follows that $\Gamma, \Gamma''[M/x] \vdash C[M/x] : u$ and $\Gamma, \Gamma''[M/x], y : C[M/x] \vdash y : C[M/x]$ using the START rule. Since $y \neq x$, we have $\Gamma, \Gamma'[M/x] \vdash y[M/x] : C[M/x]$.

Subcase 2. If Γ' is the empty context, B is x, C is A and the last line in the derivation is

$$\frac{\Gamma \vdash A : u}{\Gamma, x : A \vdash x : A} \qquad u \in \mathcal{U}, x \in Var^u, x \notin dom(\Gamma).$$

By the free variable lemma, $fv(A) \subseteq dom(\Gamma)$, so $A[M/x]$ is A and $\Gamma \vdash x[M/x] : A[M/x]$ by the premise.

Case 2. The last line in the derivation is obtained from the APP rule where B is NN' and C is $C'[N/y]$ and the line is

$$\frac{\Gamma, x : A, \Gamma'' \vdash N : \Pi y{:}A'.C' \qquad \Gamma, x : A, \Gamma'' \vdash N' : A'}{\Gamma, x : A, \Gamma'' \vdash NN' : C'[N'/y]}$$

By renaming if necessary, we may assume $x \neq y$. Using the induction hypothesis, we have $\Gamma, \Gamma''[M/x] \vdash N[M/x] : \Pi y{:}A'[M/x].C'[M/x]$ and $\Gamma, \Gamma''[M/x] \vdash N'[M/x] : A'[M/x]$. So, using the APP rule, it follows that $\Gamma, \Gamma''[M/x] \vdash (NN')[M/x] : C'[M/x][N'[M/x]/y]$, and, using the substitution results (proposition 3.1.15), we obtain the entailment $\Gamma, \Gamma''[M/x] \vdash (NN')[M/x] : (C'[N'/y])[M/x]$. □

Next we generalise the substitution lemma to allow for simultaneous substitution, which will be used extensively in chapter 6.

2.1.11 LEMMA [Generalised substitution lemma] The entailments $\Gamma, x_1{:}A_1, \ldots, x_n{:}A_n, \Delta \vdash \alpha$, for assertion α, and $\Gamma \vdash t_i : A_i[t_1, \ldots, t_{i-1}/x_1, \ldots, x_{i-1}]$, for $i \in \{1, \ldots, n\}$ imply $\Gamma, \Delta[\overline{t}/\overline{x}] \vdash \alpha[\overline{t}/\overline{x}]$.

Proof By many uses of the substitution lemma, we have $\Gamma, \Delta[t_1/x_1] \ldots [t_n/x_n] \vdash \alpha[t_1/x_1] \ldots [t_n/x_n]$. Using the free variable lemma, we know that $fv(t_i) \subseteq dom(\Gamma)$ for each $i \in \{1, \ldots, n\}$ and the sets $\{x_1, \ldots, x_n\}$ and $dom(\Gamma)$ are disjoint. Hence, by the substitution results in proposition 3.1.15, we have $\Gamma, \Delta[\overline{t}/\overline{x}] \vdash \alpha[\overline{t}/\overline{x}]$. □

2.1.12 LEMMA [Thinning] If $\Gamma \vdash A : B$ and $\Gamma \subseteq \Gamma'$ for context Γ', then $\Gamma' \vdash A : B$.

Proof By induction on the derivation of $\Gamma \vdash A : B$ (care must be taken in the Π-rule to avoid variable clashes, since $x \notin dom(\Gamma)$ does not guarantee $x \notin dom(\Gamma')$). □

Remark This lemma is sometimes called the weakening lemma, but we do not use this terminology to avoid confusion with the weakening rule. It shows that postulating more assumptions does not invalidate the provable results. This affects the logics we are able to represent in a type theory; for example, we cannot represent systems for non-monotonic reasoning.

2.1.13 LEMMA [Generation] Let $\Gamma \vdash A : B$.

1. If A is $u \in \mathcal{U}$ then $u : v \in \mathcal{A}$ and $B =_\beta v$ for some $v \in \mathcal{U}$.

2. If A is variable x^u then $x : B'$ is in Γ and $\Gamma \vdash B' : u$ and $B =_\beta B'$ for some $u \in \mathcal{U}$.

3. If A is $\Pi x{:}A_1.A_2$ then $\Gamma \vdash A_1 : u$ and $\Gamma, x : A_1 \vdash A_2 : v$ and $B =_\beta w$ for some $(u, v, w) \in \mathcal{R}$.

4. If A is $A_1 A_2$ then $\Gamma \vdash A_1 : \Pi x{:}B_1.B_2$ and $\Gamma \vdash A_2 : B_1$ and $B =_\beta B_2[A_2/x]$ for preterms B_1 and B_2.

5. If A is $\lambda x{:}A_1.A_2$ then $\Gamma \vdash \Pi x{:}A_1.B' : u$ for $u \in \mathcal{U}$ and $\Gamma, x : A_1 \vdash A_2 : B'$ and $B =_\beta \Pi x{:}A_1.B'$.

Proof The proof involves inspecting the derivation of $\Gamma \vdash A : B$. Call the AXIOM, START, Π-, λ- and APP rules the formation rules. We can follow the branch of the derivation until we get to a formation rule; the only other rules are the WEAK and

CONV rules, which do not affect A. In each case, the conclusion of the formation rule is $\Gamma' \vdash A : B'$ where Γ is an extension of Γ' and $B =_\beta B'$. The result follows by inspection of the rule used, together with the thinning lemma (lemma 2.1.12). □

A corollary of the generation lemma shows that, although there can be infinitely many levels of inhabitation (that is, $A_1 : A_2 : A_3 \ldots$), we soon get to the universes with inhabitation given by the axioms. We distinguish the universes which do not inhabit terms; these provide the starting point with which to build the type theory.

2.1.14 DEFINITION Let ζ be the PTS specified by $(\mathcal{U}, \mathcal{A}, \mathcal{R})$. An element u of $\mathcal{U}$ is a *top universe* if u is not the first component of an axiom.

2.1.15 LEMMA Let u be a top universe. Then $\Gamma \not\vdash u : A$ for any preterm A and context Γ.

The following result is a more precise statement of the corresponding one in [Bar90], which does not give the notion of top universe.

2.1.16 COROLLARY [to lemma 2.1.13] The entailment $\Gamma \vdash A : B$ implies that there is a $u \in \mathcal{U}$ such that B is u and u is a top universe, or $\Gamma \vdash B : u$.

Proof Prove by induction on the derivation of $\Gamma \vdash A : B$. We look at two cases. The other cases are trivial or similar.

Case 1. $\Gamma \vdash A : B$ is $\Gamma \vdash \Pi x{:}A'.B' : w$ and the last rule in the derivation is

$$\frac{\Gamma \vdash A' : u \qquad \Gamma, x : A' \vdash B : v}{\Gamma \vdash \Pi x{:}A'.B' : w} \qquad (u, v, w) \in \mathcal{R}$$

Either w is a top universe or $w : w' \in \mathcal{A}$ for some $w' \in \mathcal{U}$.

Case 2. $\Gamma \vdash A : B$ is $\Gamma \vdash MN : B'[N/x]$ and the last line in the derivation is

$$\frac{\Gamma \vdash M : \Pi x{:}A'.B' \qquad \Gamma \vdash N : A'}{\Gamma \vdash MN : B'[N/x]}$$

By the induction hypothesis, $\Gamma \vdash \Pi x{:}A'.B' : w$ for $w \in \mathcal{U}$. Using the generation lemma, we have $(u, v, w) \in \mathcal{R}$ such that $\Gamma \vdash A' : u$ and $\Gamma, x : A' \vdash B' : v$. By the substitution lemma, $\Gamma \vdash B'[N/x] : v$. □

2.1.17 LEMMA [Permutation] If $\Gamma, x : A, y : B, \Gamma' \vdash C : D$ and $\Gamma \vdash B : u$ for some $u \in \mathcal{U}$, then $\Gamma, y : B, x : A, \Gamma' \vdash C : D$.

Proof By induction on the derivation of $\Gamma, x : A, y : B, \Gamma'' \vdash C : D$ using the thinning lemma (lemma 2.1.12). □

2.1.18 DEFINITION Let Γ, Γ' be ζ-precontexts of the form $\langle x_1{:}A_1, \ldots, x_n{:}A_n\rangle$ and $\langle x_1{:}B_1, \ldots, x_n{:}B_n\rangle$ respectively. Then $\Gamma \to_\beta \Gamma'$ if $A_i \to_\beta B_i$, for some $i \in \{1, \ldots, n\}$, and A_j is B_j, for all $i \neq j$ with $j \in \{1, \ldots, n\}$. Also $\Gamma \rhd_\beta \Gamma'$ if $A_i \rhd_\beta B_i$ and $\Gamma =_\beta \Gamma'$ if $A_i =_\beta B_i$, for all $i \in \{1, \ldots, n\}$.

2.1.19 LEMMA [subject reduction] If $\Gamma \vdash A : B$ and $A \rhd_\beta A'$ then $\Gamma \vdash A' : B$. If $\Gamma \vdash A : B$ and $\Gamma \rhd_\beta \Gamma'$ then $\Gamma' \vdash A : B$.

Proof It is enough to prove the results for the one-step β-reduction: that is,

$$\Gamma \vdash A : B \text{ and } A \to_\beta A' \text{ implies } \Gamma \vdash A' : B; \tag{2.1}$$

$$\Gamma \vdash A : B \text{ and } \Gamma \to_\beta \Gamma' \text{ imples } \Gamma' \vdash A : B. \tag{2.2}$$

These are proved simultaneously by induction on the derivation of $\Gamma \vdash A : B$. We consider two possibilities. The other cases are easy or similar.

Case 1. The last applied rule is the Π-rule where A is $\Pi x{:}C.D$ and B is w and the last line in the derivation is

$$\frac{\Gamma \vdash C : u \qquad \Gamma, x : C \vdash D : v}{\Gamma \vdash \Pi x{:}C.D : w} \qquad (u, v, w) \in \mathcal{R}.$$

Then 2.1 and 2.2 follow from the induction hypothesis (for 2.2).

Case 2. The last applied rule is the APP rule where A is MN and B is $D[N/x]$ and the last line in the derivation is

$$\frac{\Gamma \vdash M : \Pi x{:}C.D \qquad \Gamma \vdash N : C}{\Gamma \vdash MN : D[N/x]}$$

The second result follows directly from the induction hypothesis. The first is proved by looking at cases. If A' is $M'N$ where $M \to_\beta M'$ then by the induction hypothesis $\Gamma \vdash M' : \Pi x{:}C.D$ and so $\Gamma \vdash M'N : D[N/x]$. If A' is MN' where $N \to_\beta N'$ then by the induction hypothesis $\Gamma \vdash N' : C$ and $\Gamma \vdash MN' : D[N'/x]$. Since $D[N/x] =_\beta D[N'/x]$, we have $\Gamma \vdash MN' : D[N/x]$ using the CONV rule. If M is of the form $\lambda x{:}C'.M'$ and $MN \to_\beta M'[N/x]$ then, by the generation lemma, there exists a ζ-preterm D' and $(u, v, w) \in \mathcal{R}$ such that $\Gamma \vdash C' : u$, $\Gamma, x : C' \vdash M' : D'$ and $\Gamma, x : C' \vdash D' : v$. We know that $\Pi x{:}C.D =_\beta \Pi x{:}C'.D'$ implies $D =_\beta D'$. By the substitution lemma $\Gamma \vdash M'[N/x] : D'[N/x]$ and so $\Gamma \vdash: M'[N/x] : D[N/x]$ since $D[N/x] =_\beta D'[N/x]$. □

The following result is proved by Jutting [Bar91], extending Luo's proof for the Extended Calculus of Constructions [Luo90]. His proof for arbitrary PTSs is involved and so is not given here. It asserts that PTS judgements are, in Martin-Löf's terms, 'analytic judgements', since the derived validity of a judgement $\Gamma \vdash \alpha$ depends only on the variables that actually occur in α [Mar85].

2.1.20 LEMMA [Strengthening] If $\Gamma, x : A, \Gamma' \vdash B : C$ then $\Gamma, \Gamma' \vdash B : C$ provided $x \notin dom(\Gamma') \cup fv(B) \cup fv(C)$.

2.1.21 DEFINITION A PTS with specification $(\mathcal{U}, \mathcal{A}, \mathcal{R})$ is *functional* when the sets $\mathcal{A}$ and $\mathcal{R}$ satisfy

1. $u : v, u : v' \in \mathcal{A}$ implies $v = v'$;
2. $(u, v, w), (u, v, w') \in \mathcal{R}$ implies $w = w'$.

2.1.22 LEMMA [Unicity of types] Let ζ be a functional PTS. If $\Gamma \vdash A : B$ and $\Gamma \vdash A : B'$ then $B =_\beta B'$.

Proof By induction on the structure of A. We look at two cases; the others are trivial or similar. If A is $u \in \mathcal{U}$ then, by the generation lemma, B and B' are v and v' respectively for $v, v' \in \mathcal{U}$. Since ζ is functional, we have $v = v'$. If A is $\Pi x{:}A'.A''$ then, again by the generation lemma, there exists $(u, v, w), (u', v', w') \in \mathcal{R}$ such that the following hold: $\Gamma \vdash A' : u, \Gamma \vdash A' : u', \Gamma, x : A' \vdash A'' : v, \Gamma, x : A' \vdash A'' : v'$,

B is w and B' is w'. By the induction hypothesis, $u = u'$ and $v = v'$ and so $w = w'$ since ζ is functional. □

Remark This result obviously does not hold for arbitrary PTSs since, for example, one can declare $u : v$ and $u : v'$ in $\mathcal{A}$.

2.1.23 COROLLARY Let ζ be a functional PTS. If $\Gamma \vdash A : B$ and $B \rhd_\beta B'$ then $\Gamma \vdash A : B'$.

Proof Using the corollary to the generation lemma (corollary 2.1.16), we know that B is u and u is a top universe, or $\Gamma \vdash B : u$. If B is u then B and B' are identical. If $\Gamma \vdash B : u$ then $\Gamma \vdash B' : u$ by the unicity of types lemma and, using the CONV rule, $\Gamma \vdash A : B$. □

2.1.1 PTS morphism

A simple comparison of PTSs is given in terms of a map between specifications; that is, a map between the sets of universes which preserves the axioms and rules. In particular, we use this map to obtain results for ELF^+ via those for ELF.

2.1.24 DEFINITION Let ζ and ζ' be PTSs specified by $(\mathcal{U}, \mathcal{A}, \mathcal{R})$ and $(\mathcal{U}', \mathcal{A}', \mathcal{R}')$ respectively. A *PTS morphism* from ζ to ζ' is a mapping $f : \mathcal{U} \to \mathcal{U}'$ which preserves the axioms and rules; that is,

1. if $u : v \in \mathcal{A}$ then $f(u) : f(v) \in \mathcal{A}'$;
2. if $(u, v, w) \in \mathcal{R}$ then $(f(u), f(v), f(w)) \in \mathcal{R}'$.

To extend a PTS morphism $f : \zeta \to \zeta'$, with ζ specified by $(\mathcal{U}, \mathcal{A}, \mathcal{R})$, to a map between preterms, choose injective maps from Var^u to $Var^{f(u)}$, for $u \in \mathcal{U}$, and then define by induction on the structure of preterms. Again, the map can be extended to a map from precontexts of ζ to precontexts of ζ'. The extensions of f to the preterms and precontexts of ζ will also be called f. The following lemma shows that a PTS morphism $f : \zeta \to \zeta'$ provides a sound interpretation of ζ in ζ'.

2.1.25 LEMMA If f is a PTS morphism from ζ to ζ' then

$$\Gamma \vdash^\zeta A : B \text{ implies } f(\Gamma) \vdash^{\zeta'} f(A) : f(B).$$

Proof By induction on the length of derivation of $\Gamma \vdash^{\varsigma} A : B$. □

2.1.26 EXAMPLE There is a simple PTS morphism from the specification of $\lambda\Pi$ (example 2.1.4) to that for the Calculus of Constructions (example 2.1.4) given by the identity function on the universes, since the rules for $\lambda\Pi$ are contained in those for the Calculus of Constructions.

2.1.27 EXAMPLE Consider the following two specifications:

$$\mathcal{U} = \{*_1, *_2, \Box\} \qquad \mathcal{U}' = \{*, \Box\}$$
$$\mathcal{A} = \{*_1 : \Box\} \qquad \mathcal{A}' = \{* : \Box\}$$
$$\mathcal{R} = \{(*_1, *_1, *_2), (*_1, *_2, *_2)\} \qquad \mathcal{R}' = \{(*, \Box)\}$$

There is a PTS morphism from $(\mathcal{U}, \mathcal{A}, \mathcal{R})$ to $(\mathcal{U}', \mathcal{A}', \mathcal{R}')$, the specification of simply typed λ-calculus, given by the forgetful maps $\Box \longmapsto \Box$ and $*_i \longmapsto *$ for $i \in \{1, 2\}$.

2.2 Pure Type Systems with signatures

In ELF, the signatures, declaring constants, and the contexts, declaring variables, have very different uses: signatures specify logics; contexts, amongst other things, correspond to assumptions in the representing type theory. Beradi first proposed the idea of adding signatures to PTSs [Ber90]. In [Geu90] signatures are defined as special contexts, namely fixed contexts where variable discharge does not occur. However, in ELF signatures are not special contexts; the formation of the signature constants is stronger than that of the context variables. We give a simple notion of signature within a PTS which yields a precise presentation of ELF and the new framework ELF$^+$ (up to β-equality) using the PTS notation. We also give an alternative presentation of simply typed λ-calculus with a finite number of base types using a signature to declare those base types. This differs from Barendregt's presentation, where base types are treated as universes which results, rather confusingly, in different PTSs for simply typed λ-calculi with different base types. Finally, we propose a more general notion of signature, where the formation of the signature is separate from the main proof system, to allow greater flexibility between constant and variable declarations. For example, we obtain a more

natural presentation of various many-sorted minimal intuitionistic logics with the 'structural rules', providing the function and predicate symbols, separate from the 'logical rules' for implication and universal quantification.

2.2.1 EXAMPLE The presentation of $\lambda\Pi$ as a PTS is specified by

$$\begin{aligned}\mathcal{U} &= \{*, \Box\}\\ \mathcal{A} &= \{* : \Box\}\\ \mathcal{R} &= \{(*,*), (* : \Box)\}\end{aligned}$$

In [Geu90], ELF is defined using this PTS with the signatures given as fixed, initial contexts. However, ELF is a type theory in which signatures are not special contexts; a constant may be declared in *Type*, but not a variable.

2.2.2 EXAMPLE The second example is simply typed λ-calculus [Bar84], which illustrates our reasons for working with a slightly restricted definition of PTS and our misgivings about the original definition. This is given by a set of types, denoted by $\mathcal{T}$, defined inductively as follows:

- $A_1, \ldots, A_n \in \mathcal{T}$ (called the base types);
- if $\sigma, \tau \in \mathcal{T}$ then $(\sigma \to \tau) \in \mathcal{T}$ (called the functional types).

Given disjoint, countably infinite sets of variables Var^σ, for each $\sigma \in \mathcal{T}$, the set of typed λ-terms is $\bigcup_{\sigma\in\mathcal{T}} \Lambda_\sigma$, where Λ_σ is defined by

- $x^\sigma \in \Lambda_\sigma$;
- $M \in \Lambda_{\tau\to\sigma}$ and $N \in \Lambda_\tau$ imply $MN \in \Lambda_\sigma$;
- $M \in \Lambda_\tau$ and variable $x \in \Lambda_\sigma$ imply $\lambda x.M \in \Lambda_{\sigma\to\tau}$.

This system is presented in [Bar90] by the PTS

$$\begin{aligned}\mathcal{U} &= \{*, \Box\}\\ \mathcal{A} &= \{* : \Box\}\\ \mathcal{R} &= \{(*,*)\}\end{aligned}$$

where $*$ corresponds to $\mathcal{T}$. In this PTS the types are given by variables in Var^*, whereas simply typed λ-calculus has a finite set of base types and no variable types. Because of this, Barendregt [Bar90] proposes the PTS

$$\begin{aligned}\mathcal{U} &= \{*\}\\ \mathcal{A} &= \{A_1 : *, \ldots, A_n : *\}\\ \mathcal{R} &= \{(*,*)\}\end{aligned}$$

where, in his formulation, the axioms are defined as elements of $\mathcal{C} \times \mathcal{U}$, for a set $\mathcal{C}$ containing $\mathcal{U}$, rather than elements of $\mathcal{U} \times \mathcal{U}$. A significant advantage of PTSs is that they unify the presentation of type theories in a compact notation which emphasises the structure of the type theories. This structure is obscured in the above example since two simply typed λ-calculi with different base types are expressed as different PTSs. The base types have been over-emphasised by treating them as universes. Instead, an alternative presentation will be proposed using signatures to declare the base types, so that these two calculi with different base types are expressed in terms of one PTS, but with different signatures.

2.2.3 EXAMPLE We are also not satisfied with the presentation of certain minimal intuitionistic logics as PTSs. For example, many-sorted predicate logic is represented in [Bar90] by the PTS

$$\begin{aligned}\mathcal{U} &= \{*^s, *^p, *^f, \Box^s, \Box^p\}\\ \mathcal{A} &= \{*^s : \Box^s, *^p : \Box^p\}\\ \mathcal{R} &= \{(*^s, *^s, *^f), (*^s, *^f), (*^s, \Box^p), (*^s, *^p), (*^p, *^p)\}\end{aligned}$$

Elements of $*^s$ correspond to basic sorts and those of $*^p$ to the formulae. The $\Box^s$ and $\Box^p$ are top universes which allow us to define such elements. Elements of the universe $*^f$ can be interpreted as higher-order sorts whose inhabitants are the function symbols of the logic. The inhabitants of $*^s$ are formed from the rules $(*^s, *^s, *^f)$ and $(*^s, *^f)$, and the predicate domains, whose inhabitants are the predicate symbols, are given by $(*^s, \Box^p)$. The 'logical' rules $(*^s, *^p)$ and $(*^p, *^p)$ form the implication and universal quantification with the accompanying introduction and elimination rules given by the λ- and APP rules. This gives an unnatural presentation of minimal intuitionistic logic since the 'syntactic' rules and 'logical' rules are mixed, and again we require a finite set of base sorts, rather than variable sorts.

First, we give a simple method for incorporating signatures into the PTS notation which is enough to give a precise account of ELF and ELF$^+$. Later, we propose a more general formulation which allows for greater distinction between the formation of signatures and contexts.

We specify a PTS with signatures using a quadruple of sets of universes, variable universes, axioms and rules. As in the original PTS presentation [Bar90], we distinguish the universes from which variables are declared; we may declare variables in term A if A inhabits a variable universe, whereas constants may be declared in terms inhabiting any universe.

2.2.4 DEFINITION A *specification of a PTS with signatures* is a quadruple of the form $(\mathcal{U}, \mathcal{V}, \mathcal{A}, \mathcal{R})$, where

- $\mathcal{U}$ is a set, called the set of *universes*;
- $\mathcal{V} \subseteq \mathcal{U}$, called the set of *variable universes*;
- $\mathcal{A} \subseteq \mathcal{U} \times \mathcal{U}$, called the set of *axioms*;
- $\mathcal{R} \subseteq \mathcal{V} \times \mathcal{U} \times \mathcal{U}$, called the set of *rules*.

The set of preterms, $\mathcal{T}$, of a PTS with signatures, given by specification $(\mathcal{U}, \mathcal{V}, \mathcal{A}, \mathcal{R})$ is defined using countably infinite sets of variables Var and constants $Const$ using the abstract syntax

$$\mathcal{T} ::= x \mid u \mid a \mid \Pi x{:}\mathcal{T}.\mathcal{T} \mid \lambda x{:}\mathcal{T}.\mathcal{T} \mid \mathcal{T}\mathcal{T},$$

where u is a universe, $x \in Var$ and $a \in Const$. Again it is useful to divide the sets Var and $Const$ into disjoint infinite subsets Var^v and $Const^u$ for $v \in \mathcal{V}$ and $u \in \mathcal{U}$. Arbitrary variables and constants are denoted by x, y, z and a, b, c respectively. The ζ-preterms, ζ-assertions and ζ-contexts are defined as before. A ζ-presignature is a finite sequence of ζ-assertions whose first components are all constants. A ζ-sequent is of the form $\Gamma \vdash^{\zeta}_{\Sigma} A : B$ for presignature Σ, precontext Γ and preterms A and B. We omit the superscript ζ when the PTS with signatures is apparent.

The method of declaring constants is based on the standard approach used, for example, in the type theory defining ELF [HHP89].

2.2.5 DEFINITION The *PTS with signatures*, specified by $(\mathcal{U}, \mathcal{V}, \mathcal{A}, \mathcal{R})$ is defined by the following proof system:

AXIOM $$\langle\,\rangle \vdash_\Sigma u : v \qquad u : v \in \mathcal{A}$$

SIGNATURE $$\frac{\langle\,\rangle \vdash_\Sigma A : u}{\langle\,\rangle \vdash_{\Sigma,a:A} a : A} \qquad u \in \mathcal{U}, a \in Const^u, a \notin dom(\Sigma)$$

$$\frac{\langle\,\rangle \vdash_\Sigma A : u \quad \langle\,\rangle \vdash_\Sigma B : C}{\langle\,\rangle \vdash_{\Sigma,a:A} B : C} \qquad u \in \mathcal{U}, a \in Const^u, a \notin dom(\Sigma)$$

CONTEXT $$\frac{\Gamma \vdash_\Sigma A : v}{\Gamma, x : A \vdash_\Sigma x : A} \qquad v \in \mathcal{V}, x \in Var^v, x \notin dom(\Gamma)$$

$$\frac{\Gamma \vdash_\Sigma A : v \quad \Gamma \vdash_\Sigma B : C}{\Gamma, x : A \vdash_\Sigma B : C} \qquad v \in \mathcal{V}, x \in Var^v, x \notin dom(\Gamma)$$

Π -RULE $$\frac{\Gamma \vdash_\Sigma A : u \quad \Gamma, x : A \vdash_\Sigma B : v}{\Gamma \vdash_\Sigma \Pi x{:}A.B : w} \qquad (u, v, w) \in \mathcal{R}$$

λ -RULE $$\frac{\Gamma \vdash_\Sigma \Pi x{:}A.B : u \quad \Gamma, x : A \vdash_\Sigma M : B}{\Gamma \vdash_\Sigma \lambda x{:}A.M : \Pi x{:}A.B} \qquad u \in \mathcal{U}$$

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

CONV $$\frac{\Gamma \vdash_\Sigma A : B \quad \Gamma \vdash_\Sigma B' : u}{\Gamma \vdash_\Sigma A : B'} \qquad B =_\beta B', u \in \mathcal{U}$$

A ζ-*signature* is a ζ-presignature such that $\Gamma \vdash^{\zeta}_{\Sigma} A : B$ for some precontext Γ and preterms A and B. A ζ-*context over signature* Σ is a ζ-precontext such that $\Gamma \vdash^{\zeta}_{\Sigma} A : B$ for preterms A and B. A *PTS with signature* Σ, denoted by (ζ, Σ), is a PTS with signatures, denoted by ζ, such that Σ is a ζ-signature and the sequents of interest are of the form $\Gamma \vdash^{\zeta}_{\Sigma} A : B$ for some precontext Γ and preterms A and B.

Remark An alternative system is to replace the signature rules by the rules

$$\frac{\Gamma \vdash_{\Sigma} A : u}{\Gamma \vdash_{\Sigma, a:A} a : A} \qquad u \in \mathcal{U}, a \in Const^{u}, a \notin dom(\Sigma)$$

and

$$\frac{\Gamma \vdash_{\Sigma} A : u \qquad \Gamma \vdash_{\Sigma} B : C}{\Gamma \vdash_{\Sigma, a:A} B : C} \qquad u \in \mathcal{U}, a \in Const^{u}, a \notin dom(\Sigma).$$

This allows the signature to be extended at arbitrary points in a derivation rather than considering it, once formed, as fixed. We do not use this approach as both logics and ELF have fixed signatures, although it may be more appropriate to have flexible signatures when using type theories as theorem provers.

One can obtain a simple connection between PTSs with and without signatures by viewing signatures as initial contexts, as in Geuvers' paper [Geu90]. This link gives us easy access to results for PTSs with signatures via those for PTSs.

2.2.6 DEFINITION Let ζ be a PTS with signatures specified by $(\mathcal{U}, \mathcal{V}, \mathcal{A}, \mathcal{R})$. The *fundamental PTS for* ζ, denoted by ζ_f, is the PTS with specification $(\mathcal{U}, \mathcal{A}, \mathcal{R})$.

To obtain the connection between the PTS with signatures and the fundamental PTS, choose a bijection $f : Var^{\zeta} \cup Const^{\zeta} \to Var^{\zeta_f}$ satisfying

$$f(x) \in Var^{v} \text{ for } x \in Var^{v} \text{ and } v \in \mathcal{V};$$
$$f(c) \in Var^{u} \text{ for } c \in Const^{u} \text{ and } u \in \mathcal{U}.$$

This is easily extended to preterms, precontexts and presignatures to obtain the following result.

2.2.7 LEMMA Let ζ be a PTS with signatures and ζ_f its fundamental PTS. Then $\Gamma \vdash^{\zeta}_{\Sigma} A : B$ if and only if $f(\Sigma), f(\Gamma) \vdash^{\zeta_f} f(A) : f(B)$.

Proof Easy induction on derivations. □

As a consequence of this lemma, all the results given in section 2.1 hold for PTSs with signatures.

2.2.8 EXAMPLE The framework ELF with β-equality is presented as the PTS with signatures specified by

$$\begin{aligned} \zeta_{LF} \qquad \mathcal{U} &= \{*, \Box\} \\ \mathcal{V} &= \{*\} \\ \mathcal{A} &= \{* : \Box\} \\ \mathcal{R} &= \{(*,*),(*,\Box)\} \end{aligned}$$

This is equivalent to the original presentation given in appendix A.

2.2.9 EXAMPLE Simply typed λ-calculus is given by the PTS with signatures, $\zeta_{\lambda^{\rightarrow}}$, specified by

$$\begin{aligned} \zeta_{\lambda^{\rightarrow}} \qquad \mathcal{U} &= \{*, \Box\} \\ \mathcal{V} &= \{*\} \\ \mathcal{A} &= \{* : \Box\} \\ \mathcal{R} &= \{(*,*)\}. \end{aligned}$$

Simply typed λ-calculus with one base type 0 is $\zeta_{\lambda^{\rightarrow}}$ with signature $\{0 : *\}$ for $0 \in Const^{\Box}$, and simply typed λ-calculus with two base types 0 and $0'$ is $\zeta_{\lambda^{\rightarrow}}$ with signature $\{0 : *, 0' : *\}$ for $0, 0' \in Const^{\Box}$. We may also define constants in the base types; for example, the signature $\{0 : *, a : 0\}$ for $0 \in Const^{\Box}$ and $a \in Const^{*}$ declares a base sort 0 inhabited by base constant a. There is no restriction on the constants we are able to declare, although this restriction can be achieved by having no connection between $\mathcal{U}$ and $\mathcal{V}$ and defining the sets of axioms and rules as $\mathcal{A} \subseteq (\mathcal{U} \cup \mathcal{V}) \times (\mathcal{U} \cup \mathcal{V})$ and $\mathcal{R} \subseteq \mathcal{V} \times (\mathcal{U} \cup \mathcal{V}) \times (\mathcal{U} \cup \mathcal{V})$ respectively.

2.2.10 EXAMPLE The PTS with signature presentation of many-sorted minimal intuitionistic predicative logic is

$$\begin{aligned} \zeta_{\lambda PRED} \qquad \mathcal{U} &= \{*^s, *^p, *^f, \Box^s, \Box^p\} \\ \mathcal{V} &= \{*^s, *^p\} \\ \mathcal{A} &= \{*^s : \Box^s, *^p : \Box^p\} \\ \mathcal{R} &= \{(*^s,*^s,*^f),(*^s,*^f),(*^s,\Box^p),(*^s,*^p),(*^p,*^p)\} \end{aligned}$$

with only variables corresponding to terms and proofs of formulae permitted. First-order logic with arithmetic, given in chapter 3, is presented as $\lambda PRED$ with signature $\Sigma = \{i : *^s, 0 : i, succ : i \to i, + : i \to i \to i, =: i \to i \to *^p\}$, where $i, 0, succ, +, = \in Const$. Notice that there is a natural separation into the 'syntactic' rules, $(*^s, *^s, *^f)$, $(*^s, *^f)$ and $(*^s, \Box^p)$, used in the formation of the function and predicate symbols, and the 'logical' rules, $(*^s, *^p)$ and $(*^p, *^p)$, which give the formation, introduction and elimination rules for implication and universal quantification. This motivates an alternative approach of adding signatures.

We discuss a possible alternative method for adding signatures, where the formation of signatures is separate from the main proof system. This method incorporates ideas mentioned in Beradi's thesis [Ber90] and is based on discussions with Geuvers and Pollack. The motivation for this approach is illustrated by the above example of the presentation of many-sorted predicate logic, where the rules fall naturally into 'syntactic' and 'logical' rules. We would also like to present, for example, first-order logic with Peano arithmetic, which includes the induction schema

$$\phi(0) \text{ and } \forall x.(\phi(x) \supset \phi(succ(x))) \text{ implies } \forall x.\phi(x).$$

The alternative definition of PTS with signatures has a similar specification to the previous definition except that we separate the rules into two classes: constant and variable rules. The idea is that a signature is a context from a PTS defined using the constant rules; once the signature is formed, the constant rules are redundant. This clarifies the intuition that constants are special variables which cannot be discharged. The *specification* of a PTS with signatures is now a quintuple $(\mathcal{U}, \mathcal{V}, \mathcal{A}, \mathcal{R}_{\mathcal{U}}, \mathcal{R}_{\mathcal{V}})$, where

- $\mathcal{U}$ is a set, called the set of *universes*;
- $\mathcal{V} \subseteq \mathcal{U}$, called the set of *variable universes*;
- $\mathcal{A} \subseteq \mathcal{U} \times \mathcal{U}$, called the set of *axioms*;
- $\mathcal{R}_{\mathcal{U}} \subseteq \mathcal{U} \times \mathcal{U} \times \mathcal{U}$, called the set of *constant rules*;
- $\mathcal{R}_{\mathcal{V}} \subseteq \mathcal{V} \times \mathcal{U} \times \mathcal{U}$, called the set of *variable rules*.

The intuition is that a signature is a context from the PTS specified by $(\mathcal{U}, \mathcal{A}, \mathcal{R}_{\mathcal{U}})$ and the main proof system is essentially defined using $\mathcal{V}$, $\mathcal{A}$ and $\mathcal{R}_{\mathcal{V}}$. The PTS with signature Σ, given by specification $(\mathcal{U}, \mathcal{V}, \mathcal{A}, \mathcal{R}_{\mathcal{U}}, \mathcal{R}_{\mathcal{V}})$, is defined by a context Σ from the PTS ζ_Σ specified by $(\mathcal{U}, \mathcal{A}, \mathcal{R}_{\mathcal{U}})$, and the proof system given as follows:

AXIOM $\quad \langle\,\rangle \vdash^{\zeta}_{\Sigma} u : v \qquad u : v \in \mathcal{A}$

SIG $\quad \langle\,\rangle \vdash^{\zeta}_{\Sigma} a : A \qquad \Sigma \vdash^{\zeta_\Sigma} a : A, a \in dom(\Sigma)$

START $\quad \dfrac{\Gamma \vdash^{\zeta}_{\Sigma} A : v}{\Gamma, x : A \vdash^{\zeta}_{\Sigma} x : A} \qquad v \in \mathcal{V}, x \in Var^v, x \notin dom(\Gamma)$

WEAK $\quad \dfrac{\Gamma \vdash^{\zeta}_{\Sigma} A : v \qquad \Gamma \vdash^{\zeta}_{\Sigma} B : C}{\Gamma, x : A \vdash^{\zeta}_{\Sigma} B : C} \qquad v \in \mathcal{V}, x \in Var^v, x \notin dom(\Gamma)$

$\Pi \quad \dfrac{\Gamma \vdash^{\zeta}_{\Sigma} A : u \qquad \Gamma, x : A \vdash^{\zeta}_{\Sigma} B : v}{\Gamma \vdash^{\zeta}_{\Sigma} \Pi x{:}A.B : w} \qquad (u, v, w) \in \mathcal{R}_{\mathcal{V}}$

$\lambda \quad \dfrac{\Gamma \vdash^{\zeta}_{\Sigma} \Pi x{:}A.B : u \qquad \Gamma, x : A \vdash^{\zeta}_{\Sigma} M : B}{\Gamma \vdash^{\zeta}_{\Sigma} \lambda x{:}A.M : \Pi x{:}A.B} \qquad u \in \mathcal{U}$

APP $\quad \dfrac{\Gamma \vdash^{\zeta}_{\Sigma} M : \Pi x{:}A.B \qquad \Gamma \vdash^{\zeta}_{\Sigma} N : A}{\Gamma \vdash^{\zeta}_{\Sigma} MN : B[N/x]}$

$$\textsc{Conv} \qquad \frac{\Gamma \vdash A : B \qquad \Gamma \vdash B' : u}{\Gamma \vdash A : B'} \qquad B =_\beta B', u \in \mathcal{U}$$

Remark The side-condition, $\Sigma \vdash^{\zeta_\Sigma} a : A, a \in dom(\Sigma)$, for the SIG rule results in $\langle\,\rangle \vdash^{\zeta}_{\Sigma} a : A$ whenever $a : A' \in \Sigma$, $\Sigma \vdash^{\zeta_\Sigma} A' : u$ for $u \in \mathcal{U}$ and $A =_\beta A'$; this would not hold for the more restrictive condition $a : A \in \Sigma$.

Remark The results for this system are left for future work since, in this thesis, the simple notion in definition 2.2.5 is enough for our purposes. For the moment, just notice that the corresponding result to corollary 2.1.16, namely,

$$\Gamma \vdash_\Sigma A : B \text{ implies } B \text{ is a top universe or } \Gamma \vdash_\Sigma B : u \text{ for universe } u,$$

does not hold. For example, let ζ be a PTS with signature Σ specified by $(\mathcal{U}, \mathcal{V}, \mathcal{A}, \mathcal{R}_\mathcal{U}, \mathcal{R}_\mathcal{V})$ and assume $a : \Pi x{:}A.B \in \Sigma$, where the formation of the Π-abstraction involves a rule in $\mathcal{R}_\mathcal{U}$ which is not in $\mathcal{R}_\mathcal{V}$. We have $\langle\,\rangle \vdash^{\zeta}_{\Sigma} a : \Pi x{:}A.B$, but not $\langle\,\rangle \vdash_\Sigma \Pi x{:}A.B : u$ for some universe u. Therefore, we cannot use the method of corollary 2.1.23 to prove $\Gamma \vdash_\Sigma A : B$ and $B \rhd_\beta C$ implies $\Gamma \vdash_\Sigma A : C$.

2.2.11 EXAMPLE We return to the presentation of minimal intuitionistic first-order logic which is now given by

$$\zeta'_{\lambda PRED} \qquad \begin{aligned} \mathcal{U} &= \{*^s, *^p, *^f, \Box^s, \Box^p\} \\ \mathcal{V} &= \{*^s, *^p\} \\ \mathcal{A} &= \{*^s : \Box^s, *^p : \Box^p\} \\ \mathcal{R}_\mathcal{U} &= \{(*^s, *^s, *^f), (*^s, *^f), (*^s, \Box^p)\} \\ \mathcal{R}_\mathcal{V} &= \{(*^s, *^p), (*^p, *^p)\} \end{aligned}$$

First-order logic with arithmetic, given in chapter 3, is presented as $\zeta'_{\lambda PRED}$ with signature $\Sigma = \{i : *^s, 0 : i, succ : i \to i, + : i \to i \to i, =: i \to i \to *^p\}$, where $i, 0, succ, +, =$ are in $Const$.

2.2.12 EXAMPLE The constant rules in the above example are extended to give a presentation of first-order logic with Peano arithmetic in this setting:

$$\begin{aligned}
\zeta_{\lambda PRED+PA} \qquad \mathcal{U} &= \{*^s, *^p, *^f, \Box^s, \Box^p\} \\
\mathcal{V} &= \{*^s, *^p\} \\
\mathcal{A} &= \{*^s : \Box^s, *^p : \Box^p\} \\
\mathcal{R}_{\mathcal{U}} &= \{(*^s, *^s, *^f), (*^s, *^f), (*^s, \Box^p), (*^s, *^p), (*^p, *^p), (\Box^p, *^p, \Box^p)\} \\
\mathcal{R}_{\mathcal{V}} &= \{(*^s, *^p), (*^p, *^p)\}
\end{aligned}$$

The signature for first-order logic with Peano arithmetic would include $\{\iota : *^s, 0 : \iota, succ : \iota \to \iota, ind : \Pi\phi{:}\iota \to *^p.(\phi(0) \to (\Pi x{:}\iota.\phi(x) \to \phi(succ(x)))) \to \Pi x{:}\iota.\phi(x)\}$.

Remark This method of presenting the induction schema for Peano arithmetic is used in Beradi's thesis. An alternative approach is to add the induction schema to the proof system, rather than the signature.

2.2.13 EXAMPLE In [HHP89], it is stated that the ELF type system has the proof-theoretic strength of simply typed λ-calculus. This is illustrated by viewing ELF as the PTS with signatures

$$\begin{aligned}
\zeta'_{LF} \qquad \mathcal{U} &= \{*, \Box\} \\
\mathcal{V} &= \{*\} \\
\mathcal{A} &= \{*{:}\Box\} \\
\mathcal{R}_{\mathcal{U}} &= \{(*, *), (*, \Box)\} \\
\mathcal{R}_{\mathcal{V}} &= \{(*, *)\}
\end{aligned}$$

which is motivated by a result of Geuvers for ELF [Geu90] which implies that, once the signature has been formed, the $(*, \Box)$ rule is redundant.

We do not continue investigating this view of signatures since incorporating the rules of signature formation within the proof system of the PTS with signatures is perfectly satisfactory for our purposes.

2.3 Pure Type Systems with $\beta\eta$-equivalence

The framework ELF is a type theory with $\beta\eta$-equivalence: all matters relating to encodings of logics are treated up to this equality. We must therefore extend the definition of a PTS to include η-conversion. This is not straightforward. The Church–Rosser property for PTSs with β holds for the usual notion of β-reduction on the preterms. This property is lost for arbitrary preterms when the η-reduction,

$$\lambda x{:}A.Bx \to B \text{ if } x \notin fv(B),$$

is added. For example, the term $\lambda x{:}A.(\lambda x{:}B.M)x$ reduces via η to $\lambda x{:}B.M$ and by β to $\lambda x{:}A.M$, which has a common reduct if and only if A and B do. Let $\to_{\beta\eta}$ denote the one-step reduction, defined by $\to_\beta$ together with the above reduction, whose reflexive and transitive closure is $\rhd_{\beta\eta}$. Salvesen [Sal89] adds $\beta\eta$-equality to the ELF type system using equality judgements, independent of the notion of reduction, which restricts the equality to ELF terms. She shows that the Church–Rosser property holds for terms by a very technical and delicate argument. We follow her approach and add η to the proof systems of PTSs using the equality judgements. The results, corresponding to those for the standard definition, are not known in general. In recent unpublished work [Sal91], Salvesen proves these results for functional PTSs with η satisfying strong normalisation, which, in particular, implies the decidability of the new framework ELF$^+$.

The set of preterms of a PTS is extended to include preterms of the form $A = B$ for preterms A and B. The proof system for PTSs with η is obtained by removing the CONV rule in the standard proof system for PTSs (definition 2.1.3), replacing it by a conversion rule containing an equality judgement of the form $M = N : A$ and adding rules for inferring the equality judgement.

2.3.1 DEFINITION Let $(\mathcal{U}, \mathcal{A}, \mathcal{R})$ be a specification of a PTS (definition 2.1.1). The *PTS with* η, specified by $(\mathcal{U}, \mathcal{A}, \mathcal{R})$, is defined by the proof system given in table 2.1 and table 2.2.

The notions of ζ-context and ζ-term are similar to those for standard PTSs.

AXIOM	$\langle\,\rangle \vdash u : v$	$u : v \in \mathcal{A}$
START	$\dfrac{\Gamma \vdash A : u}{\Gamma, x : A \vdash x : A}$	$u \in \mathcal{U}, x \in Var^u, x \notin dom(\Gamma)$
WEAK	$\dfrac{\Gamma \vdash A : u \quad \Gamma \vdash B : C}{\Gamma, x : A \vdash B : C}$	$u \in \mathcal{U}, x \in Var^u, x \notin dom(\Gamma)$
Π	$\dfrac{\Gamma \vdash A : u \quad \Gamma, x : A \vdash B : v}{\Gamma \vdash \Pi x{:}A.B : w}$	$(u, v, w) \in \mathcal{R}$
λ	$\dfrac{\Gamma \vdash \Pi x{:}A.B{:}u \quad \Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x{:}A.M : \Pi x{:}A.B}$	$u \in \mathcal{U}$
APP	$\dfrac{\Gamma \vdash M : \Pi x{:}A.B \quad \Gamma \vdash N : A}{\Gamma \vdash MN : B[N/x]}$	
CONV	$\dfrac{\Gamma \vdash A : B \quad \Gamma \vdash B = C : u}{\Gamma \vdash A : C}$	$u \in \mathcal{U}$

Table 2.1: The proof system for a Pure Type System with η specified by $(\mathcal{U}, \mathcal{A}, \mathcal{R})$.

REFL
$$\frac{\Gamma \vdash A : B}{\Gamma \vdash A = A : B}$$

SYMM
$$\frac{\Gamma \vdash A = A' : B}{\Gamma \vdash A' = A : B}$$

TRANS
$$\frac{\Gamma \vdash A = A' : B \qquad \Gamma \vdash A' = A'' : B}{\Gamma \vdash A = A'' : B}$$

Π-EQ
$$\frac{\Gamma \vdash A = A' : u \qquad \Gamma, x : A \vdash B = B' : v}{\Gamma \vdash \Pi x{:}A.B = \Pi x{:}A'.B' : w} \qquad (u, v, w) \in \mathcal{R}$$

λ -EQ
$$\frac{\Gamma \vdash \Pi x{:}A.B : u \qquad \Gamma \vdash A = A' : v \qquad \Gamma, x : A \vdash M = M' : B}{\Gamma \vdash \lambda x{:}A.M = \lambda x{:}A'.M' : \Pi x{:}A.B}$$
for $u, v \in \mathcal{U}$

APP-EQ
$$\frac{\Gamma \vdash M = M' : \Pi x{:}A.B \qquad \Gamma \vdash N = N' : A}{\Gamma \vdash MN = M'N' : B[N/x]}$$

CONV-EQ
$$\frac{\Gamma \vdash A = A' : B \qquad \Gamma \vdash B = C : u}{\Gamma \vdash A = A' : C} \qquad u \in \mathcal{U}$$

BETA
$$\frac{\Gamma \vdash C : A \qquad \Gamma \vdash \Pi x{:}A.D : u \qquad \Gamma, x : A \vdash B : D}{\Gamma \vdash (\lambda x{:}A.B)C = B[C/x] : D[C/x]} \qquad u \in \mathcal{U}$$

ETA
$$\frac{\Gamma \vdash B : \Pi x{:}A.C}{\Gamma \vdash \lambda x{:}A.(Bx) = B : \Pi x{:}A.C} \qquad x \notin fv(B)$$

Table 2.2: The proof system for a Pure Type System with η specified by $(\mathcal{U}, \mathcal{A}, \mathcal{R})$ (continued).

The following basic results give a few elementary properties of the entailment relation of an arbitrary PTS with η specified by $(\mathcal{U}, \mathcal{A}, \mathcal{R})$. These are proved by trivial adaptation of the proofs of the corresponding results for the standard PTS with β-conversion and so are not given here. Throughout we use the notation $\Gamma \vdash \alpha$ to denote $\Gamma \vdash A : B$ or $\Gamma \vdash A = B : C$ for preterms A, B and C.

2.3.2 LEMMA [Free variable lemma] Suppose $\Gamma \vdash B : C$ for $\Gamma = \langle x_1{:}A_1, \ldots, x_n{:}A_n \rangle$. Then

1. the $x_1, \ldots, x_n$ are distinct;
2. $fv(B), fv(C) \subseteq \{x_1, \ldots, x_n\}$;
3. a derivation $\Gamma \vdash B : C$ has subderivation $x_1{:}A_1, \ldots, x_{i-1}{:}A_{i-1} \vdash A_i : u_i$ for $u_i \in \mathcal{U}$ and each $i \in \{1, \ldots, n\}$.

2.3.3 LEMMA [Start Lemma] Let Γ be a context. Then

1. $\Gamma \vdash u : v$ for $u : v \in \mathcal{A}$;
2. $\Gamma \vdash x : A$ for all $x : A \in \Gamma$.

2.3.4 LEMMA [Substitution]
If $\Gamma, x : A, \Gamma' \vdash \alpha$ and $\Gamma \vdash M : A$ then $\Gamma, \Gamma'[M/x] \vdash \alpha[M/x]$.

2.3.5 LEMMA [Generalised substitution] If $\Gamma, x_1{:}A_1, \ldots, x_n{:}A_n, \Delta \vdash \alpha$ and, for $i \in \{1, \ldots, n\}$, $\Gamma \vdash t_i : A_i[t_1, \ldots, t_{i-1}/x_1, \ldots, x_{i-1}]$ then $\Gamma, \Delta[\overline{t}/\overline{x}] \vdash \alpha[\overline{t}/\overline{x}]$.

2.3.6 LEMMA [Thinning] If $\Gamma \vdash \alpha$ and $\Gamma \subseteq \Gamma'$ for context Γ' then $\Gamma' \vdash \alpha$.

2.3.7 LEMMA [Permutation] If $\Gamma, x : A, y : B, \Gamma' \vdash \alpha$ and $\Gamma \vdash B : u$ for some $u \in \mathcal{U}$ then $\Gamma, y : B, x : A, \Gamma' \vdash \alpha$.

2.3.8 LEMMA [Weak Generation] Let $\Gamma \vdash A : B$.

1. If A is u then $u : v \in \mathcal{A}$ for some $u \in \mathcal{U}$.
2. If A is x then $x : C \in \Gamma$ for $\Gamma \vdash C : u$ and $u \in \mathcal{U}$.

3. If A is $\Pi x{:}A_1.A_2$ then $\Gamma \vdash A_1 : u$ and $\Gamma, x : A_1 \vdash A_2 : v$ and $(u, v, w) \in \mathcal{R}$ for some $w \in \mathcal{U}$.

4. If A is $\lambda x{:}A_1.A_2$ then $\Gamma, x : A_1 \vdash A_2 : C$ for some term C and $\Gamma \vdash \Pi x{:}A_1.C : u$ for $u \in \mathcal{U}$.

5. If A is A_1A_2 then $\Gamma \vdash A_1 : \Pi x{:}C_1.C_2$ and $\Gamma \vdash A_2 : C_1$ for some terms C_1 and C_2.

Remark The generation lemma for Pure Type Systems with β-equality is stronger as it also gives information regarding B using the transitivity rule for β-equality. The analogous result cannot be proved for $\beta\eta$-equality using the same technique, as is illustrated by the following segment of a proof tree:

$$\dfrac{\dfrac{\vdots}{\Gamma \vdash A : B \quad \Gamma \vdash B = B_1 : u}}{\dfrac{\Gamma \vdash A : B_1 \quad \Gamma \vdash B_1 = B_2 : v}{\Gamma \vdash A : B_2}}$$

Here, $\Gamma \vdash B_1 = B_2 : u$ does not follow using the transitivity rule as u and v need not be the same universe.

2.3.9 LEMMA Let u be a top universe. Then $\Gamma \not\vdash u : A$ for any preterm A and context Γ.

Remark An obvious question to ask is whether the system with β-reduction given on the preterms of the PTS and the system with the equality judgement giving β are equivalent. Coquand [HP91] points out that this is a non-trivial question and proves they are equivalent concepts for a type system containing ELF.

Remark In recent unpublished work [Sal91], Salvesen shows that functional PTSs with η and strong normalisation satisfy the Church–Rosser property and subject reduction. From this, unicity of types, generation and strengthening hold. In section 4.2.3, we give an alternative proof of the Church–Rosser property for ELF$^+$ which avoids the technicalities required in Salvesen's proof.

The sets of preterms for PTSs with and without η are identical since the differences occur in the proof systems, not the specifications. Also, the definition of PTS morphism between PTSs with η is, therefore, similar to the one for standard PTSs (definition 2.1.24). The following result, stating that the translation gives a sound interpretation of the domain PTS in the image PTS, is used to derive results for ELF^+ from those of ELF.

2.3.10 LEMMA Let f be a PTS morphism from ζ_1 to ζ_2, where ζ_1 and ζ_2 are two PTSs with η. Then

1. $\Gamma \vdash^{\zeta_1} A : B$ implies $f(\Gamma) \vdash^{\zeta_2} f(A) : f(B)$;
2. $\Gamma \vdash^{\zeta_1} A = B : C$ implies $f(\Gamma) \vdash^{\zeta_2} f(A) = f(B) : f(C)$.

Proof This is proved by simultaneous induction on the derivation of $\Gamma \vdash^{\zeta_1} \alpha$ for α of the form $A : B$ or $A = B : C$. □

We obtain strong normalisation for ELF^+ via the corresponding result for ELF using the next lemma, and so we are able to apply Salvesen's results to the new framework.

2.3.11 LEMMA Let f be a PTS morphism from ζ_1 to ζ_2, where ζ_1 and ζ_2 are PTSs with η. If ζ_2 is strongly normalising then so is ζ_1.

Proof This follows from the fact that f is a map preserving β- and η-redexes. □

The next result is unsurprising, but is stated here as it will be used later in our proof of the Church–Rosser property for ELF^+.

2.3.12 LEMMA Let f be a PTS morphism from ζ_1 to ζ_2, where ζ_1 and ζ_2 are PTSs with η, and let A and B be two preterms. Then $f(A) = f(B)$ implies that either A and B are identical or the sets of subterms of A and B (definition 2.1.2) contain universes.

The PTS presentation of a type theory is defined using a set of preterms determined by the specification. Jutting [Bar91] shows that each preterm of a PTS in Barendregt's λ-cube can be assigned a unique level such that $\Gamma \vdash A : B$ implies that the level of B is one more than the level of A. The intuition for this

is illustrated by two presentations of ELF, one based on the Pure Type System $\lambda\Pi$ (example 2.2.8) and the other on the original presentation with three sorts of syntax—objects, families and kinds (appendix A). The $\lambda\Pi$-preterms of level 0 correspond to objects in the original presentation, preterms of level 1 to the families, those of level 2 to the kinds, and the preterm of level 3 to a universe in which the kinds live. We extend this notion of levels to a certain class of PTSs with η by reversing the ordering on levels so that arbitrarily many universes can be accommodated. This result is also used later.

2.3.13 DEFINITION Let P denote the set of preterms of a PTS with η specified by $(\mathcal{U}, \mathcal{A}, \mathcal{R})$, and let $\mathcal{T}$ denote the set of preterms. The *level relation* $\# \subseteq \mathcal{T} \times \omega$ is defined inductively as follows:

1. if u is a top universe (definition 2.1.14) then $\#(u, 0)$;
2. if $u : v \in \mathcal{A}$ and $\#(v, n)$ then $\#(u, n+1)$;
3. $\#(u, n)$ and $x \in Var^u$ for $u \in \mathcal{U}$ imply $\#(x, n+2)$;
4. $\#(B, n)$ implies $\#(\lambda x{:}A.B, n)$ and $\#(\Pi x{:}A.B, n)$ and $\#(BA, n)$.

Notation We say the preterm A has level n if $\#(A, n)$.

It is not always the case that PTSs have top universes: for example, the PTS specified in example 2.1.6 has one universe $*$ and axiom $* : *$. The level relation can, therefore, be empty. We restrict our attention to the PTSs whose universes *all* have levels. We also impose the restriction that the universes have unique levels.

2.3.14 LEMMA Let ζ be a PTS with η whose universes all have unique levels. Then,

1. the preterms all have unique levels;
2. if variable x and term B have the same level then A and $A[B/x]$ have the same level.

Proof Easy by induction on the structure of preterms. □

Remark The uniqueness is required as the following example shows. Let $\mathcal{U} = \{u, v, w\}$ and $\mathcal{A} = \{v : u, w : u, v : w\}$. Then we have $\#(u, 0)$, $\#(v, 1)$, $\#(w, 1)$ and $\#(v, 2)$, which imply $\#(x^v, 3)$, $\#(x^v, 4)$ and $\#(y^w, 3)$, but not $\#(y^w, 4)$.

We show, for a certain class of PTSs, that $\Gamma \vdash A : B$ and $\#(A, n+1)$ imply $\#(B, n)$ for some $n \geq 0$. This is achieved, using the technique due to Jutting [Bar91], by proving a stronger result whose formulation requires the following definition.

2.3.15 DEFINITION Let A, B and C be preterms of a PTS with η whose universes all have unique levels. Then

1. $A : B$ is *ok* if $\#(A, n+1)$ and $\#(B, n)$ for some $n \geq 0$;
2. $A = B : C$ is *ok* if $\#(A, n+1)$ and $\#(B, n+1)$ and $\#(C, n)$ for some $n \geq 0$;
3. for α of the form $A : B$ or $A = B : C$, α is *hereditarily ok* if α is ok and all substatements $y : P$ (occurring just after a symbol 'λ' or 'Π') in α are ok;
4. Γ is *hereditarily ok* if Γ is $x_1{:}A_1, \ldots, x_n{:}A_n$ and $x_i : A_i$ is hereditarily ok for each $i \in \{1, \ldots, n\}$.

More restrictions on PTSs are required if we are to show that $\Gamma \vdash A : B$ implies $A : B$ is ok. By definition, the preterm $\Pi x{:}A.B$ has the same level as B. It is therefore necessary to impose the restriction that, for every rule (u, v, w) in the specification of a PTS, the universes v and w have the same level. We call such PTSs *even*.

2.3.16 LEMMA Let ζ be an even PTS with η specified by $(\mathcal{U}, \mathcal{A}, \mathcal{R})$. Then $\Gamma \vdash \alpha$ implies that α and Γ are hereditarily ok, where α is of the form $A : B$ or $A = B : C$ for preterms A, B and C.

Proof An easy induction on the derivation of $\Gamma \vdash \alpha$. Lemma 2.3.14 is used for the cases where the last line in the derivation is an instance of the APP or APP-EQ

rules. The extra condition on the rules in the specification of ζ is required for the PI and PI-EQ rules. □

Remark Jutting's proof of this result for the type systems in Barendregt's λ-cube uses lemmas specific to the λ-cube for the case where the last line in the derivation is the CONV rule. It is not clear how to generalise his method for PTSs with β-equality defined on the preterms. These lemmas are not required for $\beta\eta$-equality given by incorporating equality judgements into the proof system.

2.3.17 COROLLARY [to lemma 2.3.16] Let ζ be an even PTS with η specified by $(\mathcal{U}, \mathcal{A}, \mathcal{R})$. Then:

1. $\Gamma \vdash^{\zeta} A : B$ implies $\#(A, n+1)$ and $\#(B, n)$ for some $n \geq 0$;
2. $\Gamma \vdash^{\zeta} A = B : C$ implies $\#(A, n+1)$ and $\#(B, n+1)$ and $\#(C, n)$ for some $n \geq 0$.

The adaptation of PTSs with η to include signatures follows similar lines to the one for PTSs given in section 2.2. The specification of a PTS with signatures and η is the same as the one in definition 2.2.4. The proof system is obtained by amalgamating definitions 2.1.3 and 2.2 in the obvious way. For completeness, we present the full proof system for a PTS with signatures and η in tables 2.3 and 2.4, since we use this presentation to define the new framework ELF$^+$. The results for PTSs with η lift to PTSs with signatures and η.

AXIOM	$\langle\,\rangle \vdash_\Sigma u : v$	$u : v \in \mathcal{A}$
SIGNATURE	$\dfrac{\langle\,\rangle \vdash_\Sigma A : u}{\langle\,\rangle \vdash_{\Sigma,a:A} a : A}$	$u \in \mathcal{U}, a \in Const^u, a \notin dom(\Sigma)$
	$\dfrac{\langle\,\rangle \vdash_\Sigma A : u \quad \langle\,\rangle \vdash_\Sigma B : C}{\langle\,\rangle \vdash_{\Sigma,a:A} B : C}$	$u \in \mathcal{U}, a \in Const^u, a \notin dom(\Sigma)$
CONTEXT	$\dfrac{\Gamma \vdash_\Sigma A : v}{\Gamma, x : A \vdash_\Sigma x : A}$	$v \in \mathcal{V}, x \in Var^v, x \notin dom(\Gamma)$
	$\dfrac{\Gamma \vdash_\Sigma A : v \quad \Gamma \vdash_\Sigma B : C}{\Gamma, x : A \vdash_\Sigma B : C}$	$v \in \mathcal{V}, x \in Var^v, x \notin dom(\Gamma)$
Π -RULE	$\dfrac{\Gamma \vdash_\Sigma A : u \quad \Gamma, x : A \vdash_\Sigma B : v}{\Gamma \vdash_\Sigma \Pi x{:}A.B : w}$	$(u, v, w) \in \mathcal{R}$
λ -RULE	$\dfrac{\Gamma \vdash_\Sigma \Pi x{:}A.B : u \quad \Gamma, x : A \vdash_\Sigma M : B}{\Gamma \vdash_\Sigma \lambda x{:}A.M : \Pi x{:}A.B}$	$u \in \mathcal{U}$
APP	$\dfrac{\Gamma \vdash_\Sigma M : \Pi x{:}A.B \quad \Gamma \vdash_\Sigma N : A}{\Gamma \vdash_\Sigma MN : B[N/x]}$	
CONV	$\dfrac{\Gamma \vdash_\Sigma A : B \quad \Gamma \vdash_\Sigma B = B' : u}{\Gamma \vdash_\Sigma A : B'}$	$u \in \mathcal{U}$

Table 2.3: The proof system for a PTS with signatures and η specified by $(\mathcal{U}, \mathcal{V}, \mathcal{A}, \mathcal{R})$.

REFL
$$\frac{\Gamma \vdash_\Sigma A : B}{\Gamma \vdash_\Sigma A = A : B}$$

SYMM
$$\frac{\Gamma \vdash_\Sigma A = A' : B}{\Gamma \vdash_\Sigma A' = A : B}$$

TRANS
$$\frac{\Gamma \vdash_\Sigma A = A' : B \qquad \Gamma \vdash_\Sigma A' = A'' : B}{\Gamma \vdash_\Sigma A = A'' : B}$$

Π-EQ
$$\frac{\Gamma \vdash_\Sigma A = A' : u \qquad \Gamma, x : A \vdash_\Sigma B = B' : v}{\Gamma \vdash_\Sigma \Pi x{:}A.B = \Pi x{:}A'.B' : w} \qquad (u, v, w) \in \mathcal{R}$$

λ -EQ
$$\frac{\Gamma \vdash_\Sigma \Pi x{:}A.B : u \qquad \Gamma \vdash_\Sigma A = A' : v \qquad \Gamma, x : A \vdash_\Sigma M = M' : B}{\Gamma \vdash_\Sigma \lambda x{:}A.M = \lambda x{:}A'.M' : \Pi x{:}A.B}$$

for $u \in \mathcal{U}$ and $v \in \mathcal{V}$

APP-EQ
$$\frac{\Gamma \vdash_\Sigma M = M' : \Pi x{:}A.B \qquad \Gamma \vdash_\Sigma N = N' : A}{\Gamma \vdash_\Sigma MN = M'N' : B[N/x]}$$

CONV-EQ
$$\frac{\Gamma \vdash_\Sigma A = A' : B \qquad \Gamma \vdash_\Sigma B = C : u}{\Gamma \vdash_\Sigma A = A' : C} \qquad u \in \mathcal{U}$$

BETA
$$\frac{\Gamma \vdash_\Sigma C : A \qquad \Gamma \vdash_\Sigma \Pi x{:}A.D : u \qquad \Gamma, x : A \vdash_\Sigma B : D}{\Gamma \vdash_\Sigma (\lambda x{:}A.B)C = B[C/x] : D[C/x]} \qquad u \in \mathcal{U}$$

ETA
$$\frac{\Gamma \vdash_\Sigma B : \Pi x{:}A.C}{\Gamma \vdash_\Sigma \lambda x{:}A.(Bx) = B : \Pi x{:}A.C} \qquad x \notin fv(B)$$

Table 2.4: The proof system for a PTS with signatures and η specified by $(\mathcal{U}, \mathcal{V}, \mathcal{A}, \mathcal{R})$ (continued).

Chapter 3

Logical Systems

The purpose of this thesis is to provide a new framework, ELF$^+$, for representing a wide variety of logical systems: that is, logics described using proof systems. This framework necessarily takes a particular approach to syntax and rules, which may differ from the approach in the original presentation of such systems. Before we introduce ELF$^+$, we therefore give a *standard* presentation of the logics under consideration, using ideas underlying representations in ELF and ELF$^+$. This presentation does not constitute a framework since we do not specify a logic using a finite amount of information; as we shall see, the syntax and rules of a logic have a finitary specification in ELF$^+$. Instead, it forms a reference point from which to describe representations in ELF$^+$. Logics whose presentations do not fit this pattern are particularly interesting as they are more difficult, if not impossible, to represent.

In this thesis, logics consist of syntax, judgements and a proof system acting on the judgements. The syntax of a logic is a set of expressions, partitioned by a set of syntactic classes. For example, the expressions of first-order logic fall into two syntactic classes, namely terms and formulae. This partitioning of expressions is also natural in the presentation of , for example, elements of the datatypes *int*, *list* and *bool*. The formation of expressions is based on Aczel's work on Frege structures [Acz80], inspired by Martin-Löf's theory of arities [NPS90]. With representations in ELF$^+$, certain variables of the framework correspond to variables of the underlying logic so that substitution in the metatheory dictates the behaviour of variables in the logic. We define substitution for logics presented in this standard form. This allows for a smooth translation to an algebraic presentation of these logics as described in chapter 6.

Proof systems are viewed as calculi for constructing derivations of certain combinations of expressions, identified by the judgements. In first-order logic, the formulae are the judgements, sometimes written $\phi\, true$ for formula ϕ to distinguish the concept of a formula being true from that of it being well-formed. The judgements of Martin-Löf's type theory [Mar85] are $A\, set$, $a \in A$, $A = B$ and $a \in A = B$, where A, B, a and b are expressions of the type theory. A formal description of the proof systems under consideration is given, from which the derivations and consequence relation are defined. This description includes an account of the assumptions and free variables in derivations since these must be declared explicitly in ELF$^+$; in particular, we account for the discharge of assumptions and variables local to derivations. This last point may be unfamiliar since a full account of the behaviour of variables is rarely given. It is illustrated by first-order logic where the truth of $\forall x.\phi$ does not depend on the variable x, since we treat $\forall x.\phi$ and $\forall y.\phi[y/x]$ as syntactically equivalent, where y is not free in ϕ. Not all logics can be represented in ELF$^+$ since their consequence relations may have properties incompatible with the ELF$^+$ entailment relation; examples include systems for non-monotonic reasoning. These restrictions are highlighted in our presentation.

As is to be expected in research of this generality, many of the ideas discussed here can be found in the literature. The novelty of our approach lies in the combination of ideas which give a better understanding of representations in ELF$^+$ and indicate the potential difficulty with finding such representations.

3.1 Syntax and judgements

In this section, we give general definitions for the formation of the syntactic classes, with those classes containing variables distinguished, expressions and judgements, and define substitution and related definitions at this general level. This exposition is helpful when explaining representations of logics in ELF and ELF$^+$, and allows for a smooth translation to the algebraic presentation of logics described in chapter 6. The reader may prefer to skip this section and refer to it for specific examples when required.

We describe a collection of logics whose syntactic classes, expressions and judgements are defined using a quadruple of sets of function symbols, referred to as the *alphabet* of the logic:

- a set of *class* symbols C;
- a subset $C' \subseteq C$, which distinguishes the syntactic classes containing variables;
- a set of *expression* symbols E;
- a set of *judgement* symbols J.

The set of class symbols is a finite set of function symbols with arities given by the natural numbers. For example, the set of class symbols for first-order logic is $\{t^0, f^0\}$, consisting of two nullary function symbols which denote the classes of terms and formulae respectively; for higher-order logic it is $\{\iota^0, o^0, \Rightarrow^2\}$, where ι and o are nullary function symbols and $\Rightarrow$ is a binary symbol. The class symbols which form the syntactic classes containing variables are distinguished; in first-order logic, the syntactic class of terms contains variables, whereas the class of formulae does not.

The formation of the expressions is governed by arities formed from the syntactic classes.

3.1.1 DEFINITION Let (C, C', E, J) be the alphabet of a logic. The *set of syntactic classes* for this logic is defined inductively as follows:

1. the nullary class symbols are syntactic classes;

2. given an n-ary class symbol f in C , with $n \geq 1$, and syntactic classes $c_1, \ldots, c_n$, then $f(c_1, \ldots, c_n)$ is a syntactic class.

The *syntactic classes containing variables* are those syntactic classes formed solely from elements of C'.

The formation of the expressions is governed by the expression symbols whose arities are formed from the syntactic classes. These arities can be viewed as simple types; this view is inspired by Martin-Löf's theory of arities [NPS90] and follows Aczel's work on Frege structures [Acz80], modified to retain the partitioning of expressions via syntactic classes. Each arity has the form $(\alpha_1, \ldots, \alpha_n) \to c$ for $n \geq 0$, where $\alpha_1, \ldots, \alpha_n$ are arities (called the domain arities) and c is a syntactic class. Associated with each arity is a level; for $n = 0$ the level is 0 and, for $n > 0$, the level is $l+1$, where l is the maximum level of $\alpha_1, \ldots, \alpha_n$. The set of expression symbols is a countable set of function symbols, with each symbol accompanied by an arity indicating the application and binding power of that function symbol. For example, the set of expression symbols for first-order arithmetic can be given as

$$\{0^t, succ^{t\to t}, +^{(t,t)\to t}, =^{(t,t)\to f}, \supset^{(f,f)\to f}, \forall^{(t\to f)\to f}, \exists^{(t\to f)\to f}\}.$$

The fact that $\forall$ and $\exists$ are binding operators is indicated by the domain arity of level 1. For our purposes, the arities can be limited to those of level not exceeding 2, since we require only variables of the logic to occur bound. The set of expression symbols need not be finite. For example, the set for higher-order logic is

$$\bigcup_{\alpha,\beta}\{app_{\alpha,\beta}^{(\alpha\Rightarrow\beta,\alpha)\to\beta}, \lambda_{\alpha,\beta}^{(\alpha\to\beta)\to(\alpha\Rightarrow\beta)}, =_{\alpha}^{(\alpha\Rightarrow\alpha\Rightarrow o)}, \supset^{(o\Rightarrow o\Rightarrow o)}, \forall_{\alpha}^{(\alpha\Rightarrow o)\Rightarrow o}, \exists_{\alpha}^{(\alpha\Rightarrow o)\Rightarrow o}\},$$

where α and β are syntactic classes.

The formal definition of the set of expressions is as one would expect. The expressions are generated from countably infinite sets of variables, one, denoted by Var^c, for each syntactic class c containing variables.

3.1.2 DEFINITION Let (C, C', E, J) be the alphabet of a logic. The *set of expressions* for this logic is defined inductively as follows:

1. if x is a variable in Var^c for syntactic class c formed from C', then x is an expression with arity c;

2. if $f \in E$ with arity $(\alpha_1, \ldots, \alpha_n) \to c$ of level ≤ 2, and $e_1, \ldots, e_n$ are expressions of arity $\alpha_1, \ldots, \alpha_n$ respectively, then $f(e_1, \ldots, e_n)$ is an expression with arity c;

3. if e is an expression with arity c of level 0, and $x_1, \ldots, x_n$ are distinct variables with arities $\alpha_1, \ldots, \alpha_n$, each of level 0, then $(x_1, \ldots, x_n)e$ is an expression with arity $(\alpha_1, \ldots, \alpha_n) \to c$.

The *logic expressions* are those expressions with arity of level 0, and the *term expressions* are those logic expressions inhabiting syntactic classes containing variables.

Remark In the last clause $x_1, \ldots, x_n$ bind any free occurrences in expression e. We do not distinguish α-equivalent expressions, by which we mean expressions equivalent up to renaming of bound variables. We delay the formal definitions of these notions until after the judgements have been defined.

Notation Let e be an expression with arity c of level 0. We say that e *inhabits* c. We employ infix and other notational devices as appropriate. For example, we write $\phi \wedge \psi$ rather than $\wedge(\phi, \psi)$ and $\exists x.\phi$ rather than $\exists((x)\phi)$. For convenience, we sometimes call the variables of the logic the *logic variables.*

Notice that the expression symbols with arities of level > 0 are not expressions of the logic. For example, in first-order logic with arithmetic, the symbols *succ*, $\supset$ and $\forall$ are not expressions. Essentially, we have constructed the $\beta\eta$-long normal forms of terms of second-order λ-calculus, since $(x_1, \ldots, x_n)e$ conveys the fact that the $x_1, \ldots, x_n$ are bound in e, as does $\lambda x_1 \ldots \lambda x_n.e$. This analogy is slightly misleading, however, as our notation does not assume any particular behaviour of the variables except binding and α-conversion: there is no notion of β-equivalence, as we cannot form β-redexes, and η-equivalence is superfluous as we only abstract

expressions which, by definition, have been fully applied; for example, we have $\forall((x)x = y)$, but not $\forall(= y)$.

We have given a general description of the expressions of a logic. From this, the judgements identify the combinations of logic expressions which are actually used in the proof system. For example, in first-order logic the judgements are the formulae, sometimes written $\phi\, true$ to separate the notion of a formula being true from a formula being well-formed. The judgements of Martin-Löf's type theory [NPS90] are $A\, set$, $A = B$, $a \in A$ and $a = b \in A$. In general, we define the judgements using a countable set of *judgement symbols* with accompanying arities of the form $(\alpha_1, \ldots, \alpha_n)$ for $n \geq 1$ where each α_i is a syntactic class. The set of judgement symbols for first-order logic is $\{true^{(f)}\}$, while that for Martin-Löf's type theory is

$$\{set^{(exp)}, \in^{(exp,exp)}, Equals^{(exp,exp)}, equals^{(exp,exp,exp)}\},$$

where *exp* is the syntactic class of expressions. We do not insist that the set of judgement symbols be finite since we can envisage judgements indexed by syntactic classes (for example, equality judgements).

3.1.3 DEFINITION Let (C, C', E, J) be the alphabet of a logic. The *set of judgements* for this logic is

$$\{j(e_1, \ldots, e_n) : j^{(\alpha_1, \ldots, \alpha_n)} \in J \text{ for syntactic classes } \alpha_1, \ldots, \alpha_n \text{ and}$$

$$\text{logic expressions } e_1, \ldots, e_n \text{ inhabiting } \alpha_1, \ldots, \alpha_n \text{ respectively}\}.$$

Remark These judgements are sometimes referred to as *basic* judgements, to distinguish them from Martin-Löf's higher-order judgements mentioned above and discussed in the next section.

3.1.4 EXAMPLE We have already seen that the terms and formulae of first-order arithmetic are generated by the alphabet

$$\begin{array}{ll} C & \{t^0, f^0\}; \\ C' & \{t^0\}; \\ E & \{0^t, succ^{t\to t}, +^{(t,t)\to t}, =^{(t,t)\to f}, \supset^{(f,f)\to f}, \forall^{(t\to f)\to f}, \exists^{(t\to f)\to f}\}; \\ J & \{true^f\}. \end{array}$$

3.1.5 EXAMPLE Simply typed λ-calculus is sometimes presented informally as

$$\sigma ::= \iota \,|\, o \,|\, \sigma \Rightarrow \sigma;$$
$$M ::= x \,|\, \lambda x{:}\sigma.M \,|\, MN,$$

with judgements of the form $M : \sigma$. Our presentation uses the alphabet

C $\quad \{type^0, term^0\}$;
C' $\quad \{term^0\}$;
E $\quad \{\iota^{type}, o^{type}, \Rightarrow^{(type,type)\to type}, app^{(term,term)\to term}, \lambda^{(type,term\to term)\to term}\}$;
J $\quad \{colon^{(term,type)}\}$.

3.1.6 EXAMPLE Another presentation of the syntax of Church's simply typed λ-calculus involves one syntactic class of expressions with no distinction between the types and terms. This is precisely the view taken by Barendregt et al. [Bar90] in their uniform presentation of Church's type theories as Pure Type Systems (PTSs). The expressions (or preterms) and judgements of the PTS with specification $(\mathcal{U}, \mathcal{A}, \mathcal{R})$ (see chapter 2) are determined by the alphabet

C $\quad \{exp^0\}$;
C' $\quad \{exp^0\}$;
E $\quad \{u^{exp}, \Pi^{(exp,exp\to exp)\to exp}, \lambda^{(exp,exp\to exp)\to exp}, app^{(exp,exp)\to exp} : u \in \mathcal{U}\}$;
J $\quad \{colon^{(exp,exp)}\}$.

Remark An alternative set of judgement symbols for PTSs is $\{inhab^{(exp)}\}$, where the inhabitation of an expression is significant: that is, expression B is inhabited if $\Gamma \vdash A : B$ for arbitrary expression A and context Γ. This emphasis on inhabitation is used in the analysis of representations of consequence relations in ELF$^+$.

3.1.7 EXAMPLE The syntax and judgements of higher-order logic, whose expressions are the well-typed terms of simply typed λ-calculus, can be given by

C $\quad \{\iota^0, o^0, \Rightarrow^2\}$;
C' $\quad \{\iota^0, o^0, \Rightarrow^2\}$;
E $\quad \bigcup_{\alpha,\beta}\{app^{(\alpha\Rightarrow\beta,\alpha)\to\beta}_{\alpha,\beta}, \lambda^{(\alpha\to\beta)\to(\alpha\Rightarrow\beta)}_{\alpha,\beta}, =^{\alpha\Rightarrow\alpha\Rightarrow o}_{\alpha}, \supset^{o\Rightarrow o\Rightarrow o}, \forall^{(\alpha\Rightarrow o)\Rightarrow o}_{\alpha}, \exists^{(\alpha\Rightarrow o)\Rightarrow o}_{\alpha}\}$,
where α and β range over the syntactic classes;
J $\quad \{true^{(o)}\}$.

(In Church's formulation of higher-order logic [Chu40], one λ-abstraction is used, rather than infinitely many indexed by the syntactic classes.)

3.1.8 EXAMPLE In the previous example, the set of expression symbols is defined by indexing over the syntactic classes. A more complicated indexing occurs when we present the syntax of primitive recursive functions with syntactic class *exp*:

C $\quad \{exp^0\}$;

C' $\quad \{exp^0\}$;

E $\quad \{zero^{exp}, succ^{exp \to exp}\} \cup \{p_{ik}^{exp_k \to exp} : i, k \in \aleph_0, 0 \leq i < k\}$,

where exp_k denotes $\overbrace{(exp, \ldots, exp)}^{k}$ for $k > 0$ and

$Const_k$ denotes an infinite set of function symbols of arity $exp_k \to exp$;

J $\quad \{=^{(exp,exp)}\}$.

3.1.9 EXAMPLE Inevitably there are logics whose usual syntactic presentation does not coincide with the above format. Part of the work of encoding a logic in ELF$^+$ involves providing a presentation which matches the one given, although this is not necessarily possible in all cases. For example, in the λ_I-calculus [Bar84] and the linear lambda calculus (λ_L-calculus) [AHM89] are examples where the application of function symbols is restricted, a mechanism which is not allowed in the above presentation. The λ-abstraction for the λ_I-calculus is

$$\lambda x.M \text{ is an expression if } M \text{ is an expression and } x \in fv(M),$$

and that for the λ_L-calculus is

$\lambda x.M$ is an expression if M is an expression and variable x occurs free once in M.

We now formalise the behaviour of variables in the expressions and judgements of a logic. It seems acceptable to have common notions of free and bound variables and α-conversion. A common behaviour of variables of the logic is not so apparent. For example, in first-order logic the free variables may be regarded as place holders for term expressions of the same syntactic class. In the π-calculus [MPW89], variables are just names which are substituted for other names, while in Hoare logic [Apt81] variables play two roles: a variable denotes both a location and the

value within that location. Also, certain logics used in situation theory [Acz90] have a component-wise substitution; that is, for expression $f(a_1, \ldots, a_n)$, each expression a_i is replaced by expression b_i to obtain $f(b_1, \ldots, b_n)$.

When representing logics in ELF$^+$, the logic variables are identified with certain variables of the type theory. This determines a particular behaviour of the logic variables which we now formalise. It is given at this general level to facilitate the translation from the syntactic presentation of logics described here to the algebraic presentation described in chapter 6.

Remark In the concluding chapter on future research (chapter 7), we propose a new notion of framework which does not rely on a common behaviour of the logic variables; the aim is capture binding and α-conversion at the logic level and substitution at the schematic (or metatheoretic) level.

The free variables are defined as usual, with $(x_1, \ldots, x_n)e$ denoting the fact that the $x_1, \ldots, x_n$ are bound variables in e.

3.1.10 DEFINITION The *set of free variables* of an expression or judgement, a, of a logic, denoted by $fv(a)$, is defined inductively as follows:

1. if a is a variable then $fv(a) = \{a\}$;
2. if a is $f(e_1, \ldots, e_n)$, where f is an expression or judgement symbol and $n \geq 0$, then $fv(a) = \bigcup_{i=1}^{n} fv(e_i)$;
3. if a is the expression $(x_1, \ldots, x_n)e$ with $n > 0$ then $fv(a) = fv(e) - \{x_1, \ldots, x_n\}$.

We now give the definition of simultaneous substitution for expressions and judgements as it is used extensively in chapter 6. The substitution of one expression for a variable is an instance of this general definition. For more details see [Sto88].

3.1.11 DEFINITION [Simultaneous substitution] Let $t_1, \ldots, t_n$ be expressions of a logic and let $x_1, \ldots, x_n$ be distinct variables, with $n > 0$, such that, for each $i \in \{1, \ldots, n\}$, the x_i and t_i belong to the same syntactic class. Also, let a denote either an expression or a judgement. Define $a[t_1, \ldots, t_n/x_1, \ldots, x_n]$, written more concisely as $a[\overline{t}/\overline{x}]$, by induction on the structure of a as follows:

1. if a is the variable x_i then $x_i[\overline{t}/\overline{x}] = x_i$;

2. if a is the variable y, for $y \notin \{x_1, \ldots, x_n\}$, then $y[\overline{t}/\overline{x}] = y$;

3. if a is $f(e_1, \ldots, e_m)$, where f is either an expression or judgement symbol and $m \geq 0$, then $f(e_1, \ldots, e_m)[\overline{t}/\overline{x}] = f(e_1[\overline{t}/\overline{x}], \ldots, e_m[\overline{t}/\overline{x}])$;

4. if a is the expression $(y_1, \ldots, y_m)e$, with $m > 0$, then $((y_1, \ldots, y_m)e)[\overline{t}/\overline{x}] = ((z_1, \ldots, z_m)e[z_1, \ldots, z_m/y_1, \ldots, y_m][\overline{t}/\overline{x}])$, where the $z_1, \ldots, z_m$ are distinct variables such that, for all $j \in \{1, \ldots, m\}$, variable z_j is not contained in $\{x_1, \ldots, x_n\} \cup \bigcup_{i=1}^{n} fv(t_i) \cup fv(e)$.

In 4, the $z_1, \ldots, z_m$ are chosen to be the first variables from the appropriate syntactic classes satisfying $z_j \notin \{x_1, \ldots, x_n\} \cup \bigcup_{i=1}^{n} fv(t_i) \cup fv(e) \cup \{z_1, \ldots, z_{j-1}\}$ using some standard enumeration of the variables of the logic.

Notation We assume that the notation $a[\overline{t}/\overline{x}]$ implies that $\overline{t}$ and $\overline{x}$ denote finite sequences of expressions and distinct variables of the same length n, for $n > 0$, such that, for each $i \in \{1, \ldots, n\}$, the t_i and x_i belong to the same syntactic class. We say that $[\overline{t}/\overline{x}]$ is a *substitution.* It is a *renaming substitution* if $\overline{t}$ is a finite sequence of distinct variables.

3.1.12 DEFINITION M is a *subexpression* of N if $M \in Sub(N)$, where $Sub(N)$, the set of subexpressions of N, is defined inductively by

$$Sub(x) = \{x\};$$
$$Sub(f(e_1, \ldots, e_n)) = \{f(e_1, \ldots, e_n)\} \cup \bigcup_{i=1}^{n} Sub(e_i);$$
$$Sub((x_1, \ldots, x_n)e) = \{(x_1, \ldots, x_n)e\} \cup Sub(e).$$

3.1.13 DEFINITION

1. A *change of bound variables* in M is the replacement of the subexpression $(x_1, \ldots, x_n)N$ by $(y_1, \ldots, y_n)N[\overline{y}/\overline{x}]$, where $n > 0$ and the y_i are distinct variables which do not occur in N.

2. M is *α-congruent* to N, denoted by $M =_\alpha N$, if N results from M by a sequence of changes of bound variables.

The following proposition is trivial to prove by structural induction on expressions and judgements.

3.1.14 PROPOSITION Let a be a judgement of a logic or an expression with arity α. Then $a[\overline{t}/\overline{x}]$ is also a judgement or an expression with the same arity α.

A more detailed analysis of simultaneous substitution can be found in [Sto88]. Here, we just list those results that will be required later in this thesis.

3.1.15 PROPOSITION Let a be an expression or judgement of a logic. Then, for $n \geq 0$,

1. $a[\overline{x}/\overline{x}] =_\alpha a$;
2. $a[\overline{t}/\overline{x}] =_\alpha a$ if $x_i \notin fv(a)$ for all $i \in \{1, \ldots, n\}$, where $\overline{x}$ denotes the variables $x_1, \ldots, x_n$;
3. $a[\overline{y}/\overline{x}][\overline{z}/\overline{y}] =_\alpha a[\overline{z}/\overline{x}]$ if $y_i \notin fv(a)$ for all $i \in \{1, \ldots, n\}$, where $\overline{y}$ denotes the variables $y_1, \ldots, y_n$;
4. $a[\overline{t}/\overline{x}][\overline{s}/\overline{y}] =_\alpha a[\overline{s}/\overline{y}][\overline{t}[\overline{s}/\overline{y}]/\overline{x}]$ if $x_i \notin fv(\overline{s})$, where $\overline{x}$ denotes the variables $x_1, \ldots, x_n$;
5. $a[\overline{t}/\overline{x}][\overline{s}/\overline{y}] =_\alpha a[\overline{t}[\overline{s}/\overline{y}]/\overline{x}]$ provided $y_i \in fv(a)$ implies $y_i \in \{x_1, \ldots, x_n\}$, where $\overline{y}$ denotes the variables $y_1, \ldots, y_n$ and $i \in \{1, \ldots, n\}$;
6. $a[t_1/x_1] \ldots [t_n/x_n] =_\alpha a[\overline{t}/\overline{x}]$ provided the x_i are distinct and $x_j \notin \bigcup_{i=1}^n fv(t_i)$ for all $j \in \{1, \ldots, n\}$.

Remark Part 5 is used extensively in chapter 6 for the categorical presentation of logics and their representing type theories.

3.2 Proof systems and consequence relations

In this section, we give a formal account of proof systems, from which we define the derivations and consequence relation of a logic, and which highlights the restrictions imposed by requiring consequence relations to be compatible with the ELF$^+$ entailment relation; in particular, we concentrate on natural deduction systems. We account for assumptions and free variables in a derivation since these concepts must be explicitly declared in ELF$^+$; this involves explaining the notions of discharge of assumptions and variables local to derivations. The last point is illustrated by first-order logic: the truth of a formula $\forall x.\phi$ does not rely on x since $\forall x.\phi$ and $\forall y.\phi[y/x]$, for y not free in ϕ, are syntactically equivalent. We also link these ideas to Martin-Löf's notion of higher-order judgements (also called hypothetico-general judgements in [Mar85]), written in the form $J_1, \ldots, J_m \rightarrow_{x_1,\ldots,x_n} J$ to indicate that basic judgement J is provable, generally in $x_1, \ldots, x_n$, from assumptions $J_1, \ldots, J_m$; proof systems are represented in ELF$^+$ by regarding rules as tuples of basic and higher-order judgements. Throughout this section, we assume that a logic is based on an arbitrary alphabet which defines the sets of syntactic classes, logic expressions and basic judgements.

Assumptions and free variables are declared explicitly in ELF$^+$ and so must be taken into account here; derivations in a proof system are therefore based on the notion of sequent.

3.2.1 DEFINITION A *sequent* of a logic has the form $\Gamma \Rightarrow_X J$, where Γ is a finite set of judgements, J is a judgement and X is a finite set of logic variables such that $fv(\Gamma) \cup fv(J) \subseteq X$, where $fv(\Gamma)$ is $\bigcup_{J_i \in \Gamma} fv(J_i)$.

Notation For a sequent $\Gamma \Rightarrow_X J$, the set of judgements Γ is the set of *assumptions* for the sequent. We often use the notation $\Gamma, J_1, \ldots, J_n \Rightarrow_{X,x_1,\ldots,x_m} J$ to denote a sequent with the set of assumptions $\Gamma \cup \{J_1, \ldots, J_n\}$ and set of free variables $X \cup \{x_1, \ldots, x_m\}$.

Remark We have restricted ourselves to *sets* of assumptions. This is a necessary property if we are to represent logics in ELF$^+$, although it excludes, for example, various linear logics [Gir87] where assumptions can only be used once. Regarding

derivations as trees, it is perhaps more natural to regard the collection of assumptions as a 'multiset': that is, a collection of judgements in which the number of times each element occurs is significant, but the order of elements is not. We cannot represent a multiset of assumptions using a context in a type theory since we have unrestricted use of declared variables in this context. An alternative approach is to concentrate on ordered sets, that is, collections of distinct judgements where the order is important. However, this complicates matters and the need for it seems to be rare. The permutation lemma (lemma 2.1.17) for PTSs ensures that we do not encounter problems with representing sets of assumptions using contexts.

3.2.2 DEFINITION A *rule* is a set of $(n+1)$-tuples of sequents where $n \geq 0$.

Rules are typically written in the form

$$\frac{seq_1 \ldots seq_n}{seq}$$

where schematic variables and side-conditions may be employed. We call such a presentation the *schematic form* of the rule, $seq_1 \ldots seq_n$ are the *premises* and seq the *conclusion.*

3.2.3 DEFINITION An *instance* of a rule is an element of that rule.

3.2.4 EXAMPLE The $\supset I$-rule of natural deduction-style first-order logic [Pra65] can be written as

$$\supset I \qquad \frac{\Gamma, \phi\, true \Rightarrow_X \psi\, true}{\Gamma \Rightarrow_X (\phi \supset \psi)\, true}$$

This notation indicates that, for particular instantiations ϕ' and ψ' of the schematic variables ϕ and ψ, the formula, $\phi' \supset \psi'$, is true using assumptions Γ' if ψ' is true using assumptions $\Gamma' \cup \{\phi'\, true\}$. The discharge of assumptions is reflected by insisting that $\phi'\, true$ occurs in the assumptions of the premise; $\phi'\, true$ need not occur in the assumptions of the conclusion.

3.2.5 EXAMPLE The $\forall I$-rule

$$\forall I \qquad \frac{\Gamma \Rightarrow_{X,x} \phi\, true}{\Gamma \Rightarrow_X \forall x.\phi\, true}$$

denotes that $\forall x.\phi'$ is true using assumptions Γ' and free variables in X if ϕ' is true using the same assumptions and free variables in $X \cup \{x\}$. Variable x is local to the proof of $\phi'\, true$ from Γ' since it has no importance in the quantification $\forall x.\phi'$: the formulae $\forall x.\phi'$ and $\forall x.\phi'[y/x]$, for y not free in ϕ', are syntactically equivalent by α-conversion.

These two examples illustrate our treatment of the discharge of assumptions and the binding of variables in derivations. An alternative view of rules, and one that underpins representations in ELF$^+$, is to emphasise these concepts of discharge and variable binding using Martin-Löf's higher-order judgements. In this view, rules are $(n+1)$-tuples of basic and higher-order judgements. The *higher-order* judgements are of the form $J_1, \ldots, J_m \rightarrow_{x_1,\ldots,x_n} J$ for $n, m \geq 0$, where $J_1, \ldots, J_m, J$ are basic judgements. (When n and m are 0 we omit the arrow). They indicate that basic judgement J is provable, generally in $x_1, \ldots, x_n$, from assumptions containing $J_1, \ldots, J_m$. Two particular kinds of higher-order judgements are often highlighted: the *hypothetical*, $J \rightarrow K$, and the *general*, $\rightarrow_X J$.

Remark To contrast the two concepts of rule, observe that in definition 3.2.2, the assumptions and free variables are explicitly given whereas, using Martin-Löf's higher-order judgements, just the information regarding discharge and variable binding is present. We use the first approach to provide the formal definitions of derivation and consequence relation, since we concentrate in this thesis on representing standard consequence relations, defined from basic judgements, in ELF$^+$. The second approach is used to describe the representation of rules in ELF$^+$; in the chapter on future research (chapter 7), we propose investigating a higher-order consequence relation, defined from Martin-Löf's basic and higher-order judgements, to help analyse the representations of derivations in ELF$^+$.

3.2.6 DEFINITION A *formal system* is a finite set of rules.

Remark We define proof systems as formal systems satisfying the cut condition, which states that derivations can be combined: given derivations of $\Gamma \Rightarrow_X J$ and

$\Delta, J \Rightarrow_X K$, we obtain a derivation of $\Gamma, \Delta \Rightarrow_X K$. A precise definition will be given once the notion of derivation has been defined.

We view derivations as trees of sequents whose formation is governed by the rules. An alternative view is that derivations are sequences of sequents, where each sequent is obtained from its predecessors by the application of a rule. The first approach gives more information regarding the assumptions used at each stage and so is more appropriate here.

3.2.7 DEFINITION Let P be a formal system. A *derivation* in P of sequent $\Gamma \Rightarrow_X J$ is defined inductively as follows:

1. $(\Gamma \Rightarrow_X J)$ if $J \in \Gamma$;

2. if Π_i is a derivation of $\Gamma_i \Rightarrow_{X_i} J_i$ for $i \in \{1, \ldots, n\}$ and $(\Gamma_1 \Rightarrow_{X_1} J_1, \ldots, \Gamma_n \Rightarrow_{X_n} J_n, \Gamma \Rightarrow_X J)$ is an instance of a rule in P, then $(\Pi_1, \ldots, \Pi_n \,|\, \Gamma \Rightarrow_X J)$ is a derivation.

Notation When $n = 0$, the vertical bar is omitted. We often write $(\Pi_1, \ldots, \Pi_n \mid \Gamma \Rightarrow_X J)$ in the form

$$\frac{\Pi_1 \ \ldots \ \Pi_n}{\Gamma \Rightarrow_X J}$$

omitting the horizontal line when n is 0.

3.2.8 DEFINITION Let P be a formal system and let $\Gamma \Rightarrow_X J$ have derivation Π. The *set of free variables* of Π is X.

3.2.9 EXAMPLE The set of free variables in a derivation of $\Gamma \Rightarrow_X J$ does not depend solely on the free variables in the assumptions Γ and conclusion J, as the following derivation illustrates. Let ϕ be a formula in first-order logic with $fv(\phi) = \{x\}$. The derivation

$$\dfrac{\dfrac{\{\forall x.\phi\} \Rightarrow_{\{z\}} \forall x.\phi}{\{\forall x.\phi\} \Rightarrow_{\{z\}} \phi[z/x]}}{\{\forall x.\phi\} \Rightarrow_{\{z\}} \exists z.\phi[z/x]}$$

requires the free variable z for the second line.

3.2.10 EXAMPLE The distinction between free variables is important. For example, consider the derivation

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma,\phi,\forall x.\phi \Rightarrow_{X,x} \forall x.\phi}}{\Gamma,\phi,\forall x.\phi \Rightarrow_{X,x} \phi[z/x]}}{\Gamma,\phi,\forall x.\phi \Rightarrow_{X,x} \exists z.\phi[z/x]}}{\dfrac{\Gamma \Rightarrow_X \exists x.\phi \quad \Gamma,\phi \Rightarrow_{X,x} \forall x.\phi \supset \exists z.\phi[z/x]}{\Gamma \Rightarrow_X \forall x.\phi \supset \exists z.\phi[z/x]}\ \exists E}$$

When $z \neq x$ then z must be in X, but when $z = x$ this is not necessary.

3.2.11 DEFINITION Let P be a formal system. The *consequence relation* of P, written $\Gamma \vdash_X J$ for a set of judgements Γ, judgement J and set of logic variables X, is defined by

$$\Gamma \vdash_X J \text{ if and only if a derivation of } \Gamma \Rightarrow_X J \text{ exists in } P.$$

Remark We concentrate here on analysing the representation of this standard consequence relation in ELF^+. Later, in section 5.2, we investigate a consequence relation with explicit reference to proofs and extend the analysis of representations accordingly. This gives an indication of the feasibility of mimicking derivations in a logic using its ELF^+ representation. An alternative approach when studying representations of derivations is to explore consequence relations based on sequents. The concluding chapter on future research explains this further, together with a schematic notion of consequence relation.

Remark Our approach differs from Avron's abstract view of consequence relations [Avr91], which need not be defined from proof systems and which do not give explicit account of the variables.

Remark Harper, Sannella and Tarlecki [HST89] define a logical system as a family of consequence relations (using Avron's definition) indexed over sets of expression symbols (which they call the signatures) of the logic, and study translations between consequence relations given by translations between signatures.

As has already been mentioned, we view the cut condition as a fundamental property of the proof system of a logic.

3.2.12 DEFINITION A *proof system* is a formal system whose consequence relation satisfies the cut condition

CUT $\quad \Gamma \vdash_X J$ and $\Delta, J \vdash_X K$ imply $\Gamma, \Delta \vdash_X K$.

Remark The analogous property of the ELF$^+$ entailment relation is given by the substitution lemma (lemma 2.1.10).

We have already imposed some restriction on the proof systems we consider, motivated by the properties of the entailment relation of PTSs. Further restrictions are required by the properties stated in the thinning lemma (lemma 2.1.12) and the generalised substitution lemma (lemma 2.1.11).

3.2.13 DEFINITION A consequence relation is *intuitionistic* if it satisfies

1. (weakening) $\Gamma \vdash_X J$ implies $\Delta \vdash_X J$ for $\Gamma \subseteq \Delta$;
2. (substitution) $\Gamma \vdash_X J$ implies $\Gamma[\overline{t}/\overline{x}] \vdash_{X/\{\overline{x}\} \cup fv(\overline{t})} J[\overline{t}/\overline{x}]$, where if $\overline{t}$ is $t_1, \ldots, t_n$ then $fv(\overline{t})$ is $\bigcup_{i=1}^{n} fv(t_i)$.

Remark Observe that $\Gamma \vdash_X J$ implies $\Gamma \vdash_Y J$ for $X \subseteq Y$, by condition 2.

Finally, we distinguish a particular style of proof system, called a *natural deduction system*, based on rules expressed using the schematic form mentioned earlier. All proof systems with intuitionistic consequence relations have equivalent formulations (in the sense that their consequence relations coincide) to systems using this style; it is with respect to these formulations that ELF$^+$ representations are given.

A *natural deduction rule* is a rule constructed from the schematic form

$$\frac{seq_1 \ldots seq_n}{seq} \qquad \text{side-condition}$$

where seq is $\Gamma \Rightarrow_X J$ and each seq_i, for $i \in \{1, \ldots, n\}$, has the shape $\Gamma, J_1, \ldots, J_n \Rightarrow_{X, x_1, \ldots, x_m} J'$ for $n, m \geq 0$. A *natural deduction system* is a proof system consisting of natural deduction rules and (possibly) the structural rules

WEAK $\quad \dfrac{\Gamma \Rightarrow_X J}{\Gamma, \Delta \Rightarrow_X J}$

CUT
$$\frac{\Gamma \Rightarrow_X J \qquad \Delta, J \Rightarrow_X K}{\Gamma, \Delta \Rightarrow_X K}$$

SUBS
$$\frac{\Gamma \Rightarrow_X J}{\Gamma[\overline{t}/\overline{x}] \Rightarrow_{X/\{\overline{x}\} \cup fv(\overline{t})} J[\overline{t}/\overline{x}]}$$

Remark Natural deduction systems do not necessarily lead to intuitionistic consequence relations. Weakening, cut and closure under substitution may not hold due to 'awkward' side-conditions limiting the assumptions. For example, the presentation of call-by-value λ-calculus in [Plo74] consists of the usual untyped λ-terms and rules including the restricted β-rule

$$(\lambda x.A)B \quad = \quad A[B/x] \qquad \text{if } B \text{ is a value,}$$

where a value is an expression which is not an application. As Plotkin points out in [Plo74], the free variables should range over values and not arbitrary terms. Otherwise, for $M = (\lambda x.\lambda x.x)(x)$, $N = \lambda x.x$ and $L = (\lambda x.x)(\lambda x.x)$, we have $M = N$, by the β-equality, but not $M[L/x] = N[L/x]$, since $N[L/x]$ is in normal form and $M[L/x]$ cannot be reduced. In [AHM89], the calculus is represented in ELF by limiting substitution to values.

3.2.14 EXAMPLE A fragment of the natural deduction system for first-order logic is

$\supset I$
$$\frac{\Gamma, \phi\, true \Rightarrow_X \psi\, true}{\Gamma \Rightarrow_X (\phi \supset \psi)\, true}$$

$\supset E$
$$\frac{\Gamma \Rightarrow_X (\phi \supset \psi)\, true \quad \Gamma \Rightarrow_X \phi\, true}{\Gamma \Rightarrow_X \psi\, true}$$

$\forall I$
$$\frac{\Gamma \Rightarrow_{X,x} \phi\, true}{\Gamma \Rightarrow_X \forall x.\phi\, true}$$

$\forall E$
$$\frac{\Gamma \Rightarrow_X \forall x.\phi\, true}{\Gamma \Rightarrow_X \phi[t/x]\, true}$$

$\exists I$
$$\frac{\Gamma \Rightarrow_X \phi[t/x]\, true}{\Gamma \Rightarrow_X \exists x.\phi\, true}$$

$$\exists E \qquad \frac{\Gamma \Rightarrow_X \exists x.\phi\, true \quad \Gamma, \phi\, true \Rightarrow_{X,x} \psi\, true}{\Gamma \Rightarrow_X \psi\, true}$$

There is a direct encoding of first-order logic in ELF^+ (example 5.2.7) which preserves the structure of derivations. Natural deduction systems with no side-conditions are easy to encode in ELF^+, once the syntax has been represented. We refer to the logic with just the rules $\supset I$ and $\supset E$ as natural deduction-style propositional logic.

3.2.15 EXAMPLE The natural deduction system of S_4 [Pra65] consists of the rules for natural deduction-style propositional logic and the rules

$$\Box I \qquad \frac{\Gamma \Rightarrow_X \phi\, true}{\Gamma \Rightarrow_X \Box\phi\, true} \qquad \Gamma \text{ consists of modalities,}$$

$$\Box E \qquad \frac{\Gamma \Rightarrow_X \Box\phi\, true}{\Gamma \Rightarrow_X \phi\, true}$$

where a modality is a formula beginning with $\Box$. This logic is represented in ELF^+ by an indirect encoding which is adequate (definition 5.1.4), but not natural (definition 5.2.3). It is usually the case that natural deduction rules with side-conditions which limits the assumptions in the premises are more difficult to represent.

3.2.16 EXAMPLE It is standard practice to concentrate on theoremhood when using Hilbert systems. We follow Avron's approach [Avr91] for incorporating assumptions into these systems and describe Hilbert systems as special cases of natural deduction systems with the natural deduction rules restricted to those of the form

$$\frac{\Gamma \Rightarrow_X J_1 \ldots \Gamma \Rightarrow_X J_n}{\Gamma \Rightarrow_X J} \qquad \text{side-condition;}$$

that is, no discharge of assumptions or binding of variables occur in these systems.

Remark As Avron cautions, regarding Hilbert systems as special natural deduction systems makes the frequent problem of finding a natural deduction system for

a consequence relation defined from a Hilbert system redundant. A deeper analysis of the rules and connectives is required to highlight the differences between these types of systems. This is beyond the scope of this thesis.

The Hilbert system of propositional logic [Ham80] can be formulated as the following natural deduction system:

$$A_1 \qquad \Gamma \Rightarrow_X (\phi \supset (\psi \supset \phi))\, true$$

$$A_2 \qquad \Gamma \Rightarrow_X ((\phi \supset (\psi \supset \theta)) \supset (\phi \supset \psi) \supset (\phi \supset \theta))\, true$$

$$\text{MP} \qquad \frac{\Gamma \Rightarrow_X (\phi \supset \psi)\, true \qquad \Gamma \Rightarrow_X \phi\, true}{\Gamma \Rightarrow_X \psi\, true}$$

where, in this case, X denotes a set of propositional variables. This system has a direct representation in ELF^+ using a fragment of the signature given in example 5.1.12 to represent Hilbert-style S_4. This natural deduction system is extended to give a proof system for Hilbert-style S_4 [Che80] by incorporating the rules

$$A_3 \qquad \Gamma \Rightarrow_X (\Box\phi \supset \phi)\, true$$

$$A_4 \qquad \Gamma \Rightarrow_X (\Box(\phi \supset \psi) \supset (\Box\phi \supset \Box\psi))\, true$$

$$A_5 \qquad \Gamma \Rightarrow_X (\Box\phi \supset \Box\Box\phi)\, true$$

and either the rule

$$\text{NEC} \qquad \frac{\Gamma \Rightarrow_X \phi}{\Gamma \Rightarrow_X \Box\phi} \qquad \phi \text{ is a theorem,}$$

which results in a natural deduction system with an undecidable side-condition, or the rule

$$\text{NEC} \qquad \frac{\emptyset \Rightarrow_X \phi}{\Gamma \Rightarrow_X \Box\phi}$$

which is not a natural deduction rule. In examples 5.1.12 and 5.2.10, we indicate that the representation of this presentation of Hilbert-style S_4 in ELF^+ is not a direct encoding, as the behaviour of the assumptions would suggest.

Chapter 4

The Framework ELF$^+$

We propose a new framework, ELF$^+$, as a type theory for representing logics. It is based on ELF [HHP89] and follows the same pattern of representation in that a logic is specified by an ELF$^+$ signature which gives rise to a correspondence between the logic and the resulting type theory. With ELF$^+$, it is possible to give a general definition of this correspondence with 'good' representations providing equivalences between logics and their representing type theories. This extends the *adequacy theorems* [HHP89] for ELF encodings which are only applicable to particular representations and cannot be generalised. The main point is that the terms in the ELF universe *Type* have many uses: they represent the basic judgements and syntactic classes, they are extra terms given by the machinery of ELF and they are extra terms required in an encoding. This means that information is lost during representation and so it is impossible to identify, from the representing type theory, the part of the entailment relation which corresponds to the consequence relation of the underlying logic. ELF$^+$ has three universes in place of *Type*, which allows for more distinction between terms and enables us to give general definitions of the equivalences we seek.

Before we introduce the new framework, we discuss the representation of first-order logic in ELF given in [HHP89]. This simple encoding illustrates the problems with ELF, and also serves as an introduction to representations in ELF$^+$ since many of the ideas apply. Readers familiar with ELF may skip to section 4.1.3, which motivates the need for a new framework. In section 4.2, we introduce the type theory ELF$^+$ presented as a Pure Type System with signatures and

η. Recent unpublished results of Salvesen [Sal91],[1] which extend her work on incorporating η in ELF to functional PTSs satisfying strong normalisation, imply that the system is decidable. We give an alternative proof of the Church–Rosser property for ELF$^+$, a key result for showing decidability, based on the results for ELF [Sal89]. Section 4.2.2 shows that our new framework ELF$^+$ overcomes the problems identified in section 4.1.3. In the following chapter, we provide the formal justification for the new framework.

4.1 Representation in ELF

In both frameworks, denoted indiscriminately by ELF$^{(+)}$, a logic is specified by a *signature*. The logic expressions, judgements and proofs of the logic are all represented by terms with the type checking rules enforcing the well-formedness conditions; in particular, proof checking is reduced to type checking. The variables of the logic are identified with certain variables of the type theory and the binding operators are represented using λ-abstraction, inspired by Church [Chu40] and Martin-Löf's system of arities [NPS90]. The advantage of this approach is that in many cases it enables the machinery associated with handling binding operators (such as α-conversion and capture-avoiding substitution) to be shifted to the metatheory, rather than be repeated for each presentation. Of course, only binding operators that behave similarly to λ-abstraction can be represented in this way. Systems with non-standard variable binding have not been fully investigated; the representation of Hoare logic in ELF [Apt81] involves a complicated specification and it is not clear whether the π-calculus [MPW89] can be represented in ELF$^{(+)}$.

Our representation of the rules and proofs focuses on the notion of judgements stressed by Martin-Löf [NPS90] and described in chapter 3. Proof systems are viewed as calculi for constructing derivations of basic judgements and rules are given by schemata formed using Martin-Löf's higher-order judgements. Basic judgements are represented by terms inhabiting a universe (*Type* in the case of

[1] In very recent work [Geu91], Herman has proved the Church–Rosser property for functional, normalising PTSs

ELF using the 'judgements-as-types' principle) whose inhabitants correspond to proofs. The structure of the type system of the two frameworks allows for a uniform treatment of the higher-order judgements as Π-abstractions so that rules can be represented as constants of the appropriate type.

We present ELF as a PTS with signatures and η (also given in section 2.3). The advantage of the PTS notation is that it provides a simple description of the framework which is easy to understand (contrast this presentation with the original presentation in appendix A) and which emphasises the differences and similarities between ELF and the new framework ELF$^+$.

4.1.1 DEFINITION The framework ELF is the PTS with signatures and η (example 2.2.8) given by the specification

$$\begin{aligned}\mathcal{U} &= \{Type, Kind\}\\ \mathcal{V} &= \{Type\}\\ \mathcal{A} &= \{Type : Kind\}\\ \mathcal{R} &= \{(Type, Type), (Type, Kind)\}\end{aligned}$$

We now proceed to describe the encoding of first-order logic in ELF, specified by Σ_{Fol}, assuming the language of expressions is that of arithmetic (examples 3.1.4 and 3.2.14). It will be clear that the method applies to any first-order signature. Many of the ideas discussed here also apply to the new framework, as representation in ELF$^+$ is similar to representation in ELF.

4.1.1 Representation of first-order logic in ELF

The representation of the syntax of first-order arithmetic is straightforward. The syntactic classes of the first-order terms and formulae are represented in ELF by the two constants

$$\begin{aligned}\iota &: Type\\ o &: Type\end{aligned}$$

whose inhabitants correspond to individuals and formulae. These inhabitants are formed by introducing a constant for each expression symbol of the logic:

$$\begin{aligned} 0 &: \iota \\ succ &: \iota \to \iota \\ + &: \iota \to \iota \to \iota \\ = &: \iota \to \iota \to o \\ \supset &: o \to o \to o \\ \forall &: (\iota \to o) \to o \end{aligned}$$

In Σ_{Fol}, there is no declaration of terms to denote variables; variables of first-order logic are identified with certain ELF variables. Thus, for example, the ELF term $+(succ(x'))(0)$ in a context declaring $x' : \iota$ represents the open expression $succ(x) + 0$. This means that we can use the λ-abstraction of ELF to give the binding operators so that, for example, the universal quantifier is handled by a constant whose domain is of function type $(\iota \to o)$; the formula $\forall x.x = x$ is represented by the term $\forall(\lambda x{:}\iota.\ =(x)(x))$. This allows us to avoid explicitly formalising the machinery associated with binding operators in each individual case.

As emphasised in chapter 3, we view proof systems as calculi for generating derivations of basic judgements. The ELF approach for representing these judgements is based on the 'judgement-as-types' principle where the basic and higher-order judgements (discussed in chapter 3) correspond to objects of the universe *Type*; the inhabitants of these objects correspond to proofs. In Σ_{Fol}, the basic judgements are given by the constant

$$true \quad : \quad o \to Type$$

so that, for term ϕ' in o corresponding to formula ϕ, the judgement $true(\phi')$ corresponds to $\phi\, true$.

The method for representing proofs as ELF terms inhabiting judgements relies on the *uniform* representation of the higher-order judgements. Recall that, if J and K are basic judgements, then the hypothetical judgement $J \to K$ expresses a form of consequence, that K is provable under the assumption J, and the general judgement $\to_{x^\sigma} J$ expresses the fact that J is provable generally in x for variable x from syntactic class σ. The hypothetical judgement is represented in ELF by

$J' \to K'$ for terms J' and K' corresponding to the basic judgements J and K, and the general judgement by $\Pi x'{:}\sigma'.J'$ for terms J, σ' and x' corresponding to the basic judgement J, the syntactic class σ and the variable x. More generally, we represent the hypothetico-general judgement $J_1, \ldots, J_n \to_{x_1^{\sigma_1}, \ldots, x_m^{\sigma_m}} K$ for $n, m \geq 0$ by a term of the form $\Pi x_1{:}A_1 \ldots \Pi x_m{:}A_m.J_1' \to \ldots \to J_n' \to K'$.

For the purpose of encoding, we regard rules in their schematic form. With direct encodings, one constant is declared for each rule in the proof system; this is the case for first-order logic and higher-order logic, but is not so for Hilbert-style S_4 (see examples 5.1.10 and 5.1.12). For the moment, we concentrate on representing the rules for first-order logic, given in example 3.2.14, which are given by the schematic form:

$$\text{R} \qquad \frac{J_1 \quad \ldots \quad J_n}{J}$$

for schematic judgements $J_1, \ldots, J_n$ and J. This schematic form is represented by a constant R' inhabiting a type

$$\Pi y_1{:}B_1 \ldots \Pi y_m{:}B_m.J_1' \to \ldots \to J_n' \to J',$$

where the $J_1', \ldots, J_n', J'$ are terms corresponding to the schematic judgements and the $y_1, \ldots, y_m$ close the term: that is, $fv(J') \cup \bigcup_{i=1}^{n} fv(J_i') = \{y_1, \ldots, y_m\}$. An instance of the above rule corresponds to a term $R'(a_1) \ldots (a_m)$ inhabiting $(J_1' \to \ldots \to J_n' \to J')[a_1/y_1] \ldots [a_m/y_m]$ for a_i inhabiting B_i in the appropriate context.

We give a detailed account of the representation of the natural deduction system for first-order logic. The rule for implication elimination has two basic schematic judgements for its premises with two schematic variables denoting formulae. Its constant in Σ_{Fol} is

$$\supset E \quad : \quad \Pi\phi, \psi{:}o.true(\supset(\phi)(\psi)) \to true(\phi) \to true(\psi)$$

The term $\supset E(\phi')(\psi')(p)(q)$, where ϕ' and ψ' correspond to formulae ϕ and ψ and p and q to proofs of $\phi \supset \psi\, true$ and $\phi\, true$ respectively, inhabits the judgement $true(\psi')$ and represents a proof of $\psi\, true$. The introduction rule for implication is similarly schematic in two formulae; this time its premise is viewed as a hypothetical judgement. Its corresponding constant is

$$\supset I \quad : \quad \Pi\phi, \psi{:}o.(true(\phi) \to true(\psi)) \to true(\phi \supset \psi)$$

using infix notation for constant $\supset$. A term of the form $\supset I(\phi')(\psi')(\lambda p{:}true(\phi').q)$ inhabiting $true(\phi' \supset \psi')$ corresponds to a proof of $\phi \supset \psi\, true$, where the λ-abstraction provides a 'proof' of the hypothetical judgement $\phi\, true \rightarrow \psi\, true$: that is, a function which takes a proof of $\phi\, true$ and gives a proof of $\psi\, true$.

We have seen that binding operators are represented using the λ-abstraction with, for example, the formulae $\forall x.\phi$ given by $\forall(\lambda x{:}\iota.\phi')$. In the representation of the rules for universal and existential quantification, the schematic formula is given by the term $F : \iota \rightarrow o$, so that substitution in the logic is transferred to β-reduction in the type theory. The constant for the $\forall E$-rule is

$$\forall E \quad : \quad \Pi F{:}\iota \rightarrow o.\Pi t{:}\iota.true(\forall(F)) \rightarrow true(Ft)$$

If F is $\lambda x{:}\iota.\phi'$ and p inhabits $true(\forall(\lambda x{:}\iota.\phi'))$ then $\forall E(\lambda x{:}\iota.\phi')(0)(p)$ inhabits $true((\lambda x{:}\iota.\phi')(0))$, which β-reduces to $true(\phi'[0/x])$

The declaration of the constant corresponding to the universal introduction rule relies on the uniform treatment of general judgements in ELF:

$$\forall I \quad : \quad \Pi F{:}\iota \rightarrow o.(\Pi x{:}\iota.true(Fx)) \rightarrow true(\forall(\lambda x{:}\iota.Fx))$$

For term $p : true(\phi')$ in context $\Gamma, x : \iota$, we have $\forall I(\lambda x{:}\iota.\phi')(q)$ inhabiting $true(\forall(\lambda x{:}\iota.\phi')$ in context Γ where q is $\lambda x{:}\iota.p$ in $\Pi x{:}\iota.true((\lambda x{:}\iota.\phi')(x))$; here q corresponds to a 'proof' of the general judgement $\rightarrow_x \phi\, true$: that is, a function which, given any term t, provides a proof of $\phi[t/x]\, true$.

The specification of the existential introduction rule follows from the ideas already mentioned:

$$\exists I \quad : \quad \Pi F{:}\iota \rightarrow o.\Pi t{:}\iota.true(Ft) \rightarrow true(\exists F)$$

The existential elimination rule has both discharge and variable-occurrence conditions:

$$\exists E \quad : \quad \Pi F{:}\iota \rightarrow o.\Pi \psi{:}o.true(\exists F) \rightarrow (\Pi x{:}\iota.true(Fx) \rightarrow true(\psi)) \rightarrow true(\psi)$$

The side-condition for the $\exists E$-rule is a matter of scoping: since ψ is bound outside the scope of x, no instance of ψ can have x free, as required.

Remark Not all specifications of the rules are so straightforward since the application of the rules may depend on 'awkward' side-conditions. For example, the encoding of Hilbert-style S_4 (example 5.1.12) requires extra constants to represent the consequence relation of the logic. This is discussed further in the next chapter.

4.1.2 Adequacy theorem

Accompanying each specification of a logic in ELF is an adequacy theorem which identifies the part of the entailment relation which corresponds to the consequence relation. All matters relating to representations of logics in ELF are treated up to $\beta\eta$-equivalence; in particular, we use the $\beta\eta$-long normal forms (section 4.2.4) as representations of these equivalence classes.

Notation Let A be an ELF term. We write $A^{\beta\eta}_{\Gamma}$ to denote the set

$$\{t : \Gamma \vdash_{\Sigma} t : A \text{ and } t \text{ is in } \beta\eta\text{-long normal form with respect to } (\Sigma;\Gamma)\}.$$

For each sequence of variables X, let $T(X)$ and $F(X)$ denote the sets of terms and formulae with free variables in X.

The adequacy theorem for Σ_{Fol} uses the evident correspondence between the arithmetic expressions of first-order logic and inhabitants of ι, and between formulae and terms in o. This correspondence is given by two functions $\xi_X : T(X) \to \iota^{\beta\eta}_{\Gamma_X}$ and $\delta_X : F(X) \to o^{\beta\eta}_{\Gamma_X}$, where if X is $\langle x_1^{term}, \ldots, x_n^{term}\rangle$ then Γ_X is $\langle x_1' : \iota, \ldots, x_n' : \iota\rangle$, for bijection $(\_)' : Var^{Log} \to Var^{Type}$. These functions are defined inductively on the structure of the logic expressions as follows:

$$\begin{aligned}
\xi_X(x) &= x' && x \in X\\
\xi_X(0) &= 0\\
\xi_X(succ(t)) &= succ(\xi_X(t))\\
\xi_X(t+s) &= +(\xi_X(t))(\xi_X(s))\\
\delta_X(t=s) &= =(\xi_X(s))(\xi_X(t))\\
\delta_X(\phi \supset \psi) &= \supset(\delta_X(\phi))(\delta_X(\psi))\\
\delta_X(\forall x.\phi) &= \forall(\lambda x' : \iota.\delta_{X,x}(\phi))\\
\delta_X(\exists x.\phi) &= \exists(\lambda x' : \iota.\delta_{X,x}(\phi))
\end{aligned}$$

4.1.2 THEOREM [Adequacy theorem for first-order logic] For each sequence of variables $X = \langle x_1^{term}, \ldots, x_n^{term} \rangle$, the functions ξ_X and δ_X are bijections satisfying:

1. ξ_X and δ_X are *compositional*: that is, for term expressions $t \in T(Y)$ and $s_1, \ldots, s_n \in T(X)$ and formula $\phi \in F(Y)$,

$$\xi_X(t[\overline{s}/\overline{x}]) = \xi_Y(t)[\overline{\xi_X(s)}/\overline{\xi_Y(x)}]$$
$$\delta_X(\phi[\overline{s}/\overline{x}]) = \delta_Y(\phi)[\overline{\xi_X(s)}/\overline{\xi_Y(x)}].$$

2. for sequences of logic variables $\langle x_1, \ldots, x_n \rangle$ and formulae $\langle \phi_1, \ldots, \phi_m \rangle$,

$$\phi_1\, true, \ldots, \phi_m\, true \vdash_{\{x_1,\ldots,x_n\}} \phi\, true \text{ if and only if}$$

$$x_1' : \iota, \ldots, x_n' : \iota, p_1 : true(\delta_X(\phi_1)), \ldots, p_m : true(\delta_X(\phi_m)) \vdash_{\Sigma_{Fol}} \_ : true(\delta_X(\phi)),$$

where $\_ : true(\delta_X(\phi))$ denotes the inhabitation of $true(\delta_X(\phi))$.

Proof In [HHP89] and also implicit in the proof of theorem 5.1.3. □

Remark In [HHP89], Harper, Honsell and Plotkin give a stronger result for their adequacy theorem for first-order logic which gives a correspondence between the structure of proofs in the logic and the structure of their representing terms in ELF. Ideally, we aim to mimic derivations in a logic using its representation in ELF. This stronger result gives some measure of the feasibility of this goal. For the moment, we concentrate on the standard consequence relation for natural deduction systems 3.2.13. In section 5.2, we define interpretations in ELF$^+$ which allow for these stronger correspondences. To distinguish the two standards of representation, we call a representation *adequate* when the representation of the standard consequence relation is analysed, and *natural* when information regarding the structure of proofs is also required.

Remark Condition 2 is given for arbitrary ELF terms inhabiting $true(\delta_X(\phi))$, rather than specific terms, since, for the moment, we focus on representing the standard consequence relation for first-order logic.

4.1.3 EXAMPLE To illustrate the interpretation of first-order logic in (ELF, Σ_{Fol}), consider the derivation

$$\dfrac{\dfrac{\{\phi, \psi\} \Rightarrow_\emptyset \phi}{\{\phi\} \Rightarrow_\emptyset \psi \supset \phi}}{\emptyset \Rightarrow_\emptyset \phi \supset (\psi \supset \phi)}$$

This corresponds to the ELF term

$$\supset I_{\delta_{(\,)}(\phi),\delta_{(\,)}(\psi\supset\phi)}(\lambda p{:}true(\delta_{(\,)}(\phi)).\ \supset I_{\delta_{(\,)}(\psi),\delta_{(\,)}(\phi)}(\lambda q{:}true(\delta_{(\,)}(\psi)).p)),$$

where, for legibility, we write the terms representing formulae as subscripts. It is easy to verify that this term inhabits $true(\delta_{(\,)}(\phi \supset (\psi \supset \phi)))$ in the empty context.

4.1.4 EXAMPLE The entailments of ELF account for free ELF variables in the same way that sequents keep track of logic variables. The ELF term corresponding to the derivation for first-order logic in example 3.2.9 is

$$\exists I(\lambda x'{:}\iota.\delta_{X,x}(\phi))(z')(\forall E(\lambda x'{:}\iota.\delta_{X,x}(\phi))(p)),$$

which inhabits $true(\delta_X(\exists z.\phi[z/x]))$ in context $z' : \iota, p : true(\delta_X(\forall x.\phi))$. Notice that the ELF variable z' occurs free in the term, just as logic variable z is free in the derivation.

4.1.3 Problems with ELF

The adequacy theorem (theorem 4.1.2) for the ELF representation of first-order logic only applies to this particular representation since it identifies the part of the entailment relation, which corresponds to the consequence relation, by appealing to specific constants in Σ_{Fol}. We seek a general identification which results in equivalences between logics and their representing type theories. Such a definition is not possible in ELF since information is lost during representation as the universe *Type* serves many purposes.

1. Both the basic judgements of a logic and the syntactic classes are represented in ELF by inhabitants of *Type*. For example, in (ELF, Σ_{Fol}) we have ι, o and $true(\phi)$ for $\phi : o$ in *Type*. We identify terms of the form $true(\phi)$ with the basic judgements and variables of type ι with the variables of the logic, but this information cannot be given, except by appealing to specific constants in Σ_{Fol}.

2. It is not unusual for extra constants to be required in the representation of a logic. The encoding of first-order logic is a simple representation which does not illustrate this. The encoding of higher-order logic [HHP89] [AHM89] uses

extra constants to represent the syntax, and the encoding of Hilbert-style S_4 [AHM89] uses extra constants to represent the consequence relation. We do not go into details here. The representations of these two logics are given for ELF$^+$ in examples 5.1.10 and 5.1.12. The second example illustrates an instance of representations in ELF where two different logics (that is, logics having different consequence relations) have the same specification; the distinction is made apparent in the adequacy theorem. In ELF$^+$, logics with different consequence relations have different specifications.

3. There are other inhabitants of *Type*, arising from the machinery of ELF, which have no meaning in the encoded logic. For example, in (ELF, Σ_{Fol}) the term $\Pi x{:}o.\iota$ has no correspondence in the logic.

Remark Some Π-abstractions using the rule $(Type, Type, Type)$ do have an interpretation in the underlying logic. The term $\iota \to \iota$ contains terms representing unary expression symbols: for example, $succ : \iota \to \iota$ corresponding to the expression symbol *succ*. It also contains the term $+(t')$, which has no direct link as we only consider complete expressions: that is, $t+s$ rather than $t+\_$. We concentrate on the basic terms inhabiting *Type*, the terms which are not Π-abstractions, since these are used to interpret the consequence relation of the logic.

These points show that the framework ELF does not distinguish the terms representing the basic judgements, the ELF variables corresponding to the logic variables and the extra terms, resulting from the encoding or machinery of ELF, without specific reference to the representation under consideration. We propose a new framework which retains these distinctions.

4.2 The type theory ELF$^+$

In this section we introduce the PTS with signatures and η which defines ELF$^+$. The decidability of the framework follows from recent unpublished work by Salvesen [Sal91] which generalises her results for ELF [Sal89] to functional PTSs with η satisfying strong normalisation. We give an alternative proof of the Church–Rosser property (CR), a key result in proving decidability which is based on the res-

ult for ELF [Sal89] and utilises the similarities and differences between the two frameworks. It involves two translations: one to ELF, which preserves the structure but which loses the universe distinction, and the other to the untyped λ-calculus [Bar84], which retains the universe information but loses the typing in the λ-abstractions. Together they provide CR for ELF$^+$ from the corresponding results for ELF and the untyped λ-calculus. All matters relating to representations in ELF$^+$ are treated up to $\beta\eta$-equivalence. We therefore define the $\beta\eta$-long normal forms, extending a definition found in [Hue75] for the simply typed λ-calculus, to provide the natural witnessing terms for our purposes.

4.2.1 Definition of the type theory

We have illustrated, using the simple encoding of first-order logic, that information is lost during representation in ELF. This is due to the lack of distinction between the terms corresponding to the basic judgements, those corresponding to the syntactic classes and the extra terms given by the encoding or machinery of the type theory. The new framework ELF$^+$ gives more distinction between the terms using three universes, called *Sort*, *Type* and *Judge*, in place of the one ELF universe *Type*. The connection between a logic and its representing type theory is as follows:

- basic judgements correspond to inhabitants of *Judge*;
- syntactic classes containing variables are represented by inhabitants of *Sorts*;
- variables of the logic are identified with sort variables (variables in Var^{Sort}).

The universe *Type* consists of terms which have no particular link with the logic. The part of the entailment relation determined by the inhabitants of *Sort* and *Judge* should, therefore, correspond to the consequence relation of the underlying logic.

4.2.1 DEFINITION The framework ELF$^+$ is defined by the PTS with signatures and η, specified by $(\mathcal{U}, \mathcal{V}, \mathcal{A}, \mathcal{R})$:

$$\begin{aligned}
\mathcal{U} &= \{Sort, Type, Judge, Kind\} \\
\mathcal{V} &= \{Sort, Type, Judge\} \\
\mathcal{A} &= \{Sort : Kind, Type : Kind, Judge : Kind\} \\
\mathcal{R} &= \{(Sort, Kind), (Type, Kind)\} \cup \{(s_1, s_2, Type) : s_1, s_2 \in \mathcal{V}\}
\end{aligned}$$

Notation An ELF$^+$ term A is a *kind* if $\Gamma \vdash_{\Sigma}^{ELF^+} A : Kind$ for some context Γ and signature Σ. Similarly, a term A is a *sort*, *type* or *judgement* if it inhabits the appropriate universe with respect to some context and signature.

The idea of splitting the universe *Type* of ELF into three motivates the choice of $\mathcal{U}$, $\mathcal{V}$ and $\mathcal{A}$. Some explanation of the rules of ELF$^+$ is necessary. The rules $(Sort, Kind)$ and $(Type, Kind)$ allow us to represent judgements dependent on syntactic classes. Just as the ELF rule $(Type, Kind)$ is used to declare the constant *true* in $o \to Type$ in the ELF representation of first-order logic, we use these ELF$^+$ rules to declare the constants which provide the basic judgements of the logic. It will be shown that the ELF$^+$ terms representing syntactic classes can inhabit *Sort* or *Type*, but not *Judge*, since the intention is for the inhabitants of *Judge* to correspond to the basic judgements of the represented logic. We therefore see no reason to include the rule $(Judge, Kind)$.

The Π-abstractions of sorts, types and judgements, given by the rules of the form $(s_1, s_2, Type)$ for $s_1, s_2 \in \mathcal{V}$, all inhabit *Type*. The motivation for this is that we view Π-abstraction as part of the machinery of the metatheory, rather than as having a direct correspondence in the object logic. This differs from Barendregt's method of representing certain minimal intuitionistic logics as Pure Type Systems [Bar90], in which propositions are treated as types and Π-abstraction represents the universal quantification.

Remark A result of the above choice of rules is that the basic judgements correspond to terms inhabiting *Judge*, whereas the higher-order judgements are represented by Π-abstractions (just as in the ELF case) which inhabit *Type*. This is because we define the consequence relation of a logic from basic judgements. An alternative approach is to view a higher-order consequence relation as fundamental (see [Avr89]), in which case the higher-order judgements are important

and must be distinguished from the other ELF$^+$ terms inhabiting *Type*. One possibility is to replace the rule $(v, Judge, Type)$ by $(v, Judge, Judge)$, or perhaps even $(v, Type, New)$ for some new universe *New*, for $v \in \mathcal{V}$. In the first case the basic judgements are distinguished as those terms in *Judge* which are not Π-abstractions, and the higher-order judgements by the Π-abstractions $\Pi x{:}A.J$ and $\Pi p{:}J.K$ for J and K in *Judge* and A in *Sort*. However, there are terms of this universe which have no meaning in the underlying logic and so this approach needs to be explored further. For the moment we work with the more standard definition of the consequence relation.

Remark Another choice of framework would be to distinguish just two universes, *Extra* and *Judge* say, with *Judge* playing the role of *Judge* in ELF$^+$ and *Extra* playing the roles of *Sort* and *Type*. Again we have the option to represent the higher-order judgements in either universe. General analysis of representations in these frameworks, adapted from the definitions in chapter 5, is possible, although information regarding which terms correspond to syntax and which have no correspondence with the underlying logic is lost.

Remark It is not clear whether $(Judge, Sort, Type)$ and $(Judge, Type, Type)$ should be included. The rule $(Judge, Sort, Type)$ allows for syntax to be dependent on judgements. This is not allowed in the logics described in chapter 3, although a natural example of a logic where this might occur is intuitionistic first-order logic extended by the choice operator. The idea is that, given a proof p of $\exists x.\phi(x)\,true$, we obtain a term t, dependent on the proof of $\exists x.\phi(x)\,true$, such that the judgement $\phi(t)\,true$ holds. Also, there are examples of logics represented in ELF$^+$ (see example 5.1.15), whose specifications have been adapted from ELF encodings, which, although their syntax does not depend on proofs of judgements, still require these rules for their representation.

4.2.2 Representation in ELF$^+$

Representations of logics in ELF$^+$ are similar to those in ELF; the main contribution of ELF$^+$ is in the *analysis* of representation rather than the method of specification. The difference lies in the choice of universe that a constant inhabits which depends on the intended use of that constant. We give two examples

to illustrate the differences. The first represents first-order logic and shows that the problems highlighted in section 4.1.3 with the ELF representation have been solved. The second encodes higher-order logic and illustrates the method for dealing with extra constants required by an encoding. These examples illustrate that we can recognise the ELF$^+$ terms corresponding to the term expressions and basic judgements of the represented logic without appealing to that logic. The formal analysis is deferred until the next chapter where we also discuss more examples.

4.2.2 EXAMPLE In the specification of first-order logic in ELF$^+$, also denoted by Σ_{Fol}, the two constants corresponding to the syntactic classes are

$$\begin{aligned} \iota &: \mathit{Sort} \\ o &: \mathit{Type} \end{aligned}$$

The different universes they inhabit indicate the varying roles that ι and o play. The ELF$^+$ terms in sort ι correspond to the term expressions of the logic; the ELF$^+$ terms in type o to well-formed formulae which are of no real interest in themselves, but which are necessary to form the basic judgements. This is mirrored in the encoding since the basic judgements are formed by the constant

$$\mathit{true} \quad : \quad o \to \mathit{Judge}$$

The typing indicates that ELF$^+$ terms in sort ι correspond to term expressions and ELF$^+$ judgements of the form $\mathit{true}(\phi)$ for ϕ in o to basic judgements. Of course, in this particular case there is a link between the inhabitants of o and the formulae, but this is not a general concept, whereas a link between the ELF$^+$ judgements and the basic judgements is.

Remark Notice the similarity between our approach of separating the ELF$^+$ terms corresponding to the term expressions, formulae and basic judgements and Martin-Löf's intuitionistic type theory [Mar85] with the judgements $A\ set$, $A\ prop$ and $A\ true$. A full comparison of ELF$^+$ and Martin-Löf's type theory is left for future research.

As before, arithmetic expressions are represented by declaring a constant for each expression symbol:

$$\begin{array}{rcll} 0 & : & \iota & : \mathit{Sort} \\ \mathit{succ} & : & \iota \to \iota & : \mathit{Type} \\ + & : & \iota \to \iota \to \iota & : \mathit{Type} \\ & & & \\ = & : & \iota \to \iota \to o & : \mathit{Type} \\ \supset & : & o \to o \to o & : \mathit{Type} \\ \forall & : & (\iota \to o) \to o & : \mathit{Type} \end{array}$$

The constant 0 inhabits a sort, since it corresponds to a term expression, whereas other constants, corresponding to higher-order expression symbols, all inhabit *Type*. Just as the higher-order expression symbols are constructs for the logic expressions, these constants form the inhabitants of ι and o. The ELF$^+$ terms $+$ and $+(t)$ for $t : \iota$ both inhabit types, whereas $+(t)(s)$, for $s : \iota$, is in sort ι and so corresponds to a term expression.

The rules for first-order logic are represented in a similar way to the ELF encoding; we just give one constant corresponding to the $\supset I$-rule

$$\supset I \quad : \quad \Pi\phi,\psi{:}o.(\mathit{true}(\phi) \to \mathit{true}(\psi)) \to \mathit{true}(\phi \supset \psi) \quad : \quad \mathit{Type}$$

We distinguish between rules and proofs; rules correspond to terms inhabiting *Type* and proof to terms inhabiting *Judge*.

The problems illustrated by the ELF representation of first-order logic in section 4.1.3 have been solved. We can now give a general identification of the ELF$^+$ terms, which correspond to the basic judgements of the logic since they inhabit *Judge*. We can also identify the ELF$^+$ terms which represent the term expressions since they inhabit sorts; in particular, the sort variables correspond to the variables of the logic. The next example shows the method for coping with extra constants required in an encoding.

4.2.3 EXAMPLE The representation of first-order logic is simple and direct. Not all encodings in ELF$^+$ (or ELF for that matter) are so easy. This is illustrated by the representation of the syntax of higher-order logic which is based on simply

typed λ-calculus:

$$\begin{array}{rl} \text{domains} & \alpha ::= \iota \mid o \mid \alpha \Rightarrow \alpha; \\ \text{terms} & t ::= x^\alpha \mid (\lambda x^\alpha . t^\beta)^{\alpha \Rightarrow \beta} \mid (t^{\alpha \Rightarrow \beta} s^\alpha)^\beta \end{array}$$

The domains, viewed as syntactic classes, cannot be represented directly as there are infinitely many of them. In Σ_{Hol}, the signature specifying higher-order logic in ELF$^+$, we have the constants

$$\begin{array}{rcl} \mathit{dom} & : & \mathit{Type} \\ \iota & : & \mathit{dom} \\ o & : & \mathit{dom} \\ \Rightarrow & : & \mathit{dom} \to \mathit{dom} \to \mathit{dom} \end{array}$$

where each class symbol corresponds to a constant to give an obvious link between the domains and the terms in *dom*. We associate to each inhabitant of *dom* a term, identified with the objects of that domain, given by the constant

$$\mathit{obj} \quad : \quad \mathit{dom} \to \mathit{Sort}$$

For each $\alpha : \mathit{dom}$, it is the term $\mathit{obj}(\alpha)$ which represents a domain of higher-order logic, rather than α itself, since inhabitants of $\mathit{obj}(\alpha)$ correspond to the expressions of the logic. Thus, $\mathit{obj}(\alpha)$ is a sort and term α in *dom* is considered an extra constant required in the encoding as the universes suggest. This demonstrates the standard technique of using the *Type* universe to represent extra constants. The representation of Hilbert-style S_4 in example 5.1.12 requires extra constants to represent the consequence relation of the logic and also uses this method. The complete signature Σ_{Hol} is given in example 5.1.10.

Remark Notice that, in the representations of first-order logic and higher-order logic in ELF$^+$, the term corresponding to the syntactic class of formulae is the type o in the first and sort $\mathit{obj}(o)$ in the second. The former distinguishes between the first-order terms and formulae, whereas the latter treats a formula as any other term expression. This mirrors precisely the behaviour of formulae in first-order and higher-order logic.

4.2.3 Results

The decidability of ELF$^+$ follows from very recent work of Salvesen [Sal91] which extends her results for incorporating η in ELF [Sal89] to functional PTSs satisfying

strong normalisation. We give an alternative proof of the Church–Rosser property (CR), based on Salvesen's results for ELF [Sal89], which utilises the similarities and differences between the two frameworks. This involves two translations; one is to ELF, which loses the distinction between universes, and the other is to the untyped λ-calculus [Bar84], losing the type information in the λ-abstraction. Together, they give CR for ELF$^+$ from the corresponding properties for ELF and the untyped λ-calculus.

Detailed proofs of the following lemmas are given for ELF in [Sal89].

4.2.4 LEMMA Let $(\mathcal{U}, \mathcal{V}, \mathcal{A}, \mathcal{R})$ be the specification of either ELF or ELF$^+$ in definition 4.1.1 or 4.2.1.

1. $\Gamma \vdash_\Sigma A = B : C$ implies $\Gamma \vdash_\Sigma A : C$ and $\Gamma \vdash_\Sigma B : C$.
2. $\Gamma \vdash_\Sigma A : B$ implies B is *Kind* or $\Gamma \vdash_\Sigma B : u$ for $u \in \mathcal{U}$.
3. $\Gamma, x : A \vdash_\Sigma \alpha$ and $\Gamma \vdash_\Sigma A = A' : v$ for $v \in \mathcal{V}$ implies $\Gamma, x : A' \vdash_\Sigma \alpha$, where α denotes an ELF$^{(+)}$ assertion.
4. [instance of the substitution of equalities lemma (lemma 4.2.5)] Let $\Gamma \vdash_\Sigma A : v$, for $v \in \mathcal{V}$, and $\Gamma \vdash_\Sigma N = N' : A$. Then
   (a) $\Gamma, x : A \vdash_\Sigma B = B' : v$ for $v \in \mathcal{V}$ implies $\Gamma \vdash_\Sigma B[N/x] = B'[N'/x] : v$;
   (b) $\Gamma, x : A \vdash_\Sigma B : \mathit{Kind}$ implies $\Gamma \vdash_\Sigma B[N/x] = B[N'/x] : \mathit{Kind}$.

Proof (sketch) The result is proved simultaneously by induction on the derivation of $\Gamma \vdash_\Sigma \alpha$ for ELF$^{(+)}$ assertion α. Part 4 is required to prove part 1 when the last line in the derivation uses the APP-EQ rule. It has an unsatisfactory proof which relies on the properties of the set of rules $\mathcal{R}$. Part 4a is proved from observing that $\Gamma \vdash_\Sigma (\lambda x{:}A.B)N = (\lambda x{:}A.B)N' : u$ and $\Gamma \vdash_\Sigma (\lambda x{:}A.B)N = B[N/x] : u$ and $\Gamma \vdash_\Sigma (\lambda x{:}A.B)N' = B[N'/x] : u$ where the formation of the λ-abstractions relies on the properties of set $\mathcal{R}$. Part 4b is proved by noting that B has the form $\Pi x_1{:}B_1 \ldots \Pi x_n{:}B_n.w$ for $w \in \mathcal{V}$. Then, by induction on $n \geq 0$, one shows that

$$\Gamma, x_1 : B_1[N/x], \ldots, x_k : B_k[N/x] \vdash_\Sigma B_{k+1}[N/x] = B_{k+1}[N'/x] : v_k$$

for $k \in \{1, \ldots, n\}$ and $v_k \in \mathcal{V}$. The rest of the proof is easy and is left to the reader. □

4.2.5 LEMMA [substitution of equalities] Let $(\mathcal{U}, \mathcal{V}, \mathcal{A}, \mathcal{R})$ be the specification of either ELF or ELF$^+$ found in definition 4.1.1 or 4.2.1. Then

$$\Gamma, x : A, \Gamma' \vdash_\Sigma B = B' : C \text{ and } \Gamma \vdash_\Sigma N = N' : A \text{ implies}$$

$$\Gamma, \Gamma'[N/x] \vdash_\Sigma B[N/x] = B'[N'/x] : C[N/x].$$

Proof The proof follows from lemma 4.2.4. □

A technical lemma is now given which identifies some constraints on the assertions which hold in ELF$^{(+)}$. The results for ELF are used in our proof of CR for ELF$^+$.

4.2.6 LEMMA Let $(\mathcal{U}, \mathcal{V}, \mathcal{A}, \mathcal{R})$ be the specification of either ELF or ELF$^+$ in definition 4.1.1 or 4.2.1.

1. $\Gamma \not\vdash_\Sigma \lambda x{:}A.B : \mathit{Kind}$;
2. $\Gamma \not\vdash_\Sigma AB : \mathit{Kind}$;
3. $\Gamma \vdash_\Sigma A : B$ and $\#(A, 2)$ or $\#(A, 3)$ implies A does not contain a universe.

Proof Part 1 is proved by assuming that $\Gamma \vdash_\Sigma \lambda x{:}A.B : \mathit{Kind}$ and proving contradiction. The preterm Kind is certainly not a Π-abstraction and so, by inspecting the possible derivation trees, it follows that $\Delta \vdash_\Sigma C = \mathit{Kind} : v$ for some context $\Delta \subseteq \Gamma$, some preterm C and $v \in \mathcal{U}$. By lemma 4.2.4 $\Delta \vdash_\Sigma \mathit{Kind} : v$ which contradicts lemma 2.3.9.

Result 2 is also proved by showing that $\Gamma \vdash_\Sigma AB : \mathit{Kind}$ gives a contradiction. Again by looking at the possible derivation trees, we must either prove that $\Delta \vdash_\Sigma C = \mathit{Kind} : v$ for some term C, context $\Delta \subseteq \Gamma$ and $v \in \mathcal{U}$, which using the above argument gives a contradiction, or that the last line in the derivation is an instance of the APP rule. In this case $\Gamma \vdash_\Sigma A : \Pi x{:}C.\mathit{Kind}$ for some term C. By lemma 4.2.4, $\Gamma \vdash_\Sigma \Pi x{:}C.\mathit{Kind} : u$ and so, using the weak generation lemma (lemma 2.3.8), $\Gamma, x : C \vdash_\Sigma \mathit{Kind} : w$ for $w \in \mathcal{U}$ which contradicts lemma 2.3.9.

Result 3 is proved by structural induction on A. We consider two cases; the others are similar or trivial. When A is of the form $\Pi x{:}A_1.A_2$ then, by the weak generation lemma (lemma 2.3.8), $\Gamma \vdash_\Sigma A_1 : u$ and $\Gamma, x : A_1 \vdash_\Sigma A_2 : v$ and $(u, v, w) \in \mathcal{R}$.

So $\#(u,1)$, by inspecting $\mathcal{A}$ and $\mathcal{R}$, and, by corollary 2.3.17, $\#(A_1,2)$. By definition of the level relation $\#$, we have $\#(\Pi x{:}A_1.A_2,n)$ implies $\#(A_2,n)$. Using the induction hypothesis A_1 and A_2 contain no universes. If A is A_1A_2 then, by the weak generation lemma, $\Gamma \vdash_\Sigma A_1 : \Pi x{:}B_1.B_2$ and $\Gamma \vdash_\Sigma A_2 : B_1$ for preterms B_1 and B_2. We have $\#(A_1,n)$ when $\#(A_1A_2,n)$ and so, by the induction hypothesis, A_1 does not contain a universe. By lemma 4.2.4, $\Gamma \vdash_\Sigma \Pi x{:}B_1.B_2 : w$ for $w \in \mathcal{U}$. Using the weak generation lemma, $\Gamma \vdash_\Sigma B_1 : u'$ and $\Gamma, x : B_1 \vdash_\Sigma B_2 : v'$ for $(u',v',w') \in \mathcal{R}$. So $\#(u',1)$ and $\#(B_1,2)$ using corollary 2.3.17. Hence, $\#(A_2,3)$ and so A_2 does not contain a universe, by the induction hypothesis. □

Remark It is also the case that 1 and 2 hold for any universe $u \in \mathcal{U}$, not just the top universe *Kind*. The proof of this, however, involves the Church–Rosser property.

We now give the two translations, used in the proof of CR for ELF$^+$; one is from ELF$^+$ to ELF and the other is from ELF$^+$ to the untyped λ-calculus [Bar84].

The PTS morphism f from ELF$^+$ to ELF is given by

$$\begin{array}{lcl} \mathit{Sort} & \longmapsto & \mathit{Type} \\ \mathit{Type} & \longmapsto & \mathit{Type} \\ \mathit{Judge} & \longmapsto & \mathit{Type} \\ \mathit{Kind} & \longmapsto & \mathit{Kind} \end{array}$$

and extended to the preterms, precontexts and presignatures, as described in section 2.1.1. This provides a sound interpretation of ELFgts1.tex $^+$ in ELF by lemma 2.3.12: that is,

$$\Gamma \vdash^{\mathit{ELF}^+}_{\Sigma} \alpha \text{ implies } f(\Gamma) \vdash^{\mathit{ELF}}_{f(\Sigma)} f(\alpha),$$

where α is an ELF$^+$ assertion.

The translation from the preterms of a PTS to the untyped λ-terms, extended with the constants $\mathcal{U} \cup \{\Pi\} \cup \mathit{Const}$, where $\mathcal{U}$ is the set of universes for the PTS, Π is the constant used to translate the Π-abstraction and *Const* is the set of constants for ELF$^+$, is defined inductively on the structure of the preterms:

$$\begin{array}{lcll} u^\circ & = & u, & \in \mathcal{U} \\ x^\circ & = & x, & \text{for variable } x \end{array}$$

$$
\begin{array}{rcll}
a^\circ & = & a, & \text{for constant } a \\
(\Pi x{:}A.B)^\circ & = & \Pi(A^\circ)(\lambda x.B^\circ) & \\
(\lambda x{:}A.B)^\circ & = & \lambda x.B^\circ & \\
(AB)^\circ & = & A^\circ B^\circ &
\end{array}
$$

It is easy to show that this translation preserves the equality between terms.

4.2.7 LEMMA Let ζ be a PTS with signatures and η. Then $\Gamma \vdash^{\zeta}_{\Sigma} A = B : C$ implies $A^\circ =_{\beta\eta} B^\circ$, where $=_{\beta\eta}$ denotes the $\beta\eta$-equality of the untyped λ-terms [Bar84].

The next lemma is important for our proof of the Church–Rosser property for ELF$^+$. The PTS morphism f from ELF$^+$ to ELF loses the information about the universes. Therefore, ELF$^+$ terms, whose images using f are equal, may not necessarily be identical; for example, $f(Type) = f(Sort)$. However, terms, whose images using f and $(\_)^\circ$ are equal, must be identical.

4.2.8 LEMMA Let Σ be a signature of ELF$^+$. For ELF$^+$ preterms D and E, if $f(D) = f(E)$ and $\Gamma \vdash^{ELF}_{f(\Sigma)} f(D) : A$, for some preterm A and context Γ, and $D^\circ =_{\beta\eta} E^\circ$ then D and E are identical.

Proof The proof follows by induction on the structure of D and E (which is the same since $f(D) = f(E)$). If D and E are universes then $D^\circ =_{\beta\eta} E^\circ$ implies D° and E° are identical, using the Church–Rosser property for the untyped λ-calculus. Since $(\_)^\circ$ preserves the universes, we know that D and E are identical. The cases when D and E are constants or variables are similar. For the λ-abstraction and application cases, we use the equality $f(D) = f(E)$. We know that $f(D) = f(E)$, and so to show that D and E are identical it is enough, by lemma 2.3.12, to show that D and E do not contain universes. By lemma 4.2.6, $\Gamma \not\vdash^{ELF}_{f(\Sigma)} f(D) : Kind$ so we have $\#(D, 2)$ or $\#(D, 3)$ using corollary 2.3.17. Using lemma 4.2.6, this means that $f(D)$ does not contain a universe. Therefore, D and E do not contain universes. When D and E are Π-abstractions, of the form $\Pi x{:}D_1.D_2$ and $\Pi x{:}E_1.E_2$ respectively, we require all the premises. We have $\Gamma \vdash^{ELF}_{f(\Sigma)} \Pi x : f(D_1).f(D_2) : u$ for $u \in \{Type, Kind\}$ and, by the weak generation lemma (lemma 2.3.8), $\Gamma \vdash^{ELF}_{f(\Sigma)} f(D_1) : Type$. Hence, $\#(f(D_1), 2)$ and, by lemma 4.2.6, $f(D_1)$ does not contain a universe. By lemma 2.3.12, D_1 and E_1 are

identical. We now turn to the equality $\Pi(D_1^\circ)(\lambda x.D_2^\circ) =_{\beta\eta} \Pi(E_1^\circ)(\lambda x.E_2^\circ)$. Using CR for the untyped λ-calculus and observing that, since Π is a constant, that reductions preserve the outermost application structure, we have $\lambda x.D_2^\circ =_{\beta\eta} \lambda x.E_2^\circ$. Again by CR, we know that $\lambda x.D_2^\circ$ and $\lambda x.E_2^\circ$ have a common reduct. By delaying η-reduction using the outermost λ-abstraction (if it is used) to last, it follows that $D_2^\circ =_{\beta\eta} E_2^\circ$. We also have $f(D_2) = f(E_2)$, by definition of f, and $\Gamma, x : f(D_1) \vdash^{ELF}_{f(\Sigma)} f(D_2) : u$ by the weak generation lemma. Using the induction hypothesis, it follows that D_2 and E_2, and therefore $\Pi x D_1.D_2$ and $\Pi x : E_1.E_2$, are identical. □

We are now in a position to prove Church–Rosser for ELF$^+$.

4.2.9 THEOREM [Church–Rosser for ELF$^+$] $\Gamma \vdash^{ELF^+}_{\Sigma} A : B$ and $A \rhd A'$ and $A \rhd A''$ implies $A' \rhd A'''$ and $A'' \rhd A'''$ for preterms A', A'' and A'''.

Proof The PTS morphism f from ELF$^+$ to ELF, sending $Sort \mapsto Type$, $Type \mapsto Type$, $Judge \mapsto Type$ and $Kind \mapsto Kind$, preserves the β- and η-redexes and so results in $f(\Gamma) \vdash^{ELF}_{f(\Sigma)} f(A) : f(B)$, $f(A) \rhd f(A')$ and $f(A) \rhd f(A'')$. Using Church–Rosser and subject reduction for ELF [Sal89], we have $f(A') \rhd C$ and $f(A'') \rhd C$ for some ELF term C. Since f preserves β- and η-redexes, we can use the same reduction paths to obtain $A' \rhd D$ and $A'' \rhd E$ for ELF$^+$ terms D and E, such that $f(D) = f(E) = C$. The map $(\_)^\circ$ from the ELF$^+$ preterms to the untyped λ-terms also preserves β- and η-redexes so that $A^\circ \rhd^\circ_{\beta\eta} D^\circ$ and $A^\circ \rhd^\circ_{\beta\eta} E^\circ$, where $\rhd^\circ_{\beta\eta}$ denotes $\beta\eta$-reduction in the untyped λ-calculus, and so $D^\circ =_{\beta\eta} E^\circ$ using the Church–Rosser property for the untyped λ-calculus. By lemma 4.2.7, D and E are identical. □

Recall that strong normalisation holds for ELF$^+$ by lemma 2.3.11 since ELF is strongly normalising [HHP89]. We appeal to Salvesen's results [Sal91] for subject reduction, unicity of types, generation and strengthening for ELF$^+$.

4.2.4 $\beta\eta$-long normal forms

Throughout this section we assume that we are dealing with a functional PTS with signatures and η satisfying strong normalisation and rely on Salvesen's res-

ults [Sal91][1]; in particular, we assume CR and subject reduction. We investigate the equivalence classes of terms with respect to contexts and signatures given by the equality judgement: that is, if $\Gamma \vdash_\Sigma A = B : C$ then A and B are in the same equivalence class with respect to $(\Sigma;\Gamma)$. There are two standard ways of choosing witnesses for these equivalence classes. One approach selects the *$\beta\eta$-normal form* which is the unique term containing no $\beta\eta$-redexes. The other approach identifies the *$\beta\eta$-long normal form* and is of more relevance to us. The intuition is that the terms in $\beta\eta$-long normal form with respect to some signature and context are fully-applied. For example, in the ELF$^+$ representation of first-order logic, we associate the formula $\forall x.y = x$ with the ELF$^+$ term $\forall(\lambda x{:}\iota.\, =(y)(x))$ in context $y : \iota$, rather than the term $\forall(=(y))$. The constant $=: \iota \to \iota \to o$ in Σ_{Fol} is fully-applied in the first term, but not in the second. The $\beta\eta$-long normal forms are presented in [Hue75] for the simply typed λ-calculus. In [HHP89], Harper, Honsell and Plotkin give a description of the so called *canonical forms*, which correspond to our $\beta\eta$-long normal forms. It is impossible for them to show that each well-formed term is equal to a unique canonical term since they only have β-equality. Felty [Fel89] and Augustsson, Coquand and Nordström [ACN90] provide systems which generate just these canonical forms.

4.2.10 DEFINITION Let ζ be a functional PTS with signatures and η satisfying strong normalisation and let A be a preterm. Then,

1. A is in *$\beta\eta$-normal form* if it has no subterm of the form $(\lambda x{:}B_1.B_2)(C)$ (called a β-redex) or $\lambda x{:}B.Cx$ for $x \notin fv(C)$ (called an η-redex);

2. A has a *$\beta\eta$-normal form with respect to* $(\Sigma;\Gamma)$ if $\Gamma \vdash^\zeta_\Sigma A = B : C$ and B is in $\beta\eta$-normal form.

The $\beta\eta$-normal forms provide witnesses to the equivalence classes defined by the equality relation. They are found using the decidable reduction relation $\to_{\beta\eta}$.

---

[1] In [Geu91], Geuvers shows CR for functional, normalising PTSs with η.

4.2.11 LEMMA Let ζ be a functional PTS with signatures and η satisfying strong normalisation and let $\Gamma \vdash_\Sigma A : B$. Then,

1. A has an unique $\beta\eta$-normal form with respect to $(\Sigma; \Gamma)$.
2. if $\Delta \vdash^\zeta_\Sigma A : C$ then the $\beta\eta$-normal forms of A with respect to $(\Sigma; \Gamma)$ and $(\Sigma; \Delta)$ are the same.

Proof By strong normalisation, the Church–Rosser property and subject reduction. □

Remark This lemma shows that a term A has unique $\beta\eta$-normal form irrespective of the term it inhabits with respect to a particular context and signature. In future, we refer to the $\beta\eta$-normal form of A with respect to $(\Sigma; \Gamma)$ as the $\beta\eta$-normal form of A.

Remark In [HHP89], the canonical forms, which correspond to our $\beta\eta$-long normal forms, are defined using the unique β-normal forms. We use the $\beta\eta$-normal forms since they are unique using our stronger equality, whereas the β-normal forms are not.

The analysis of the shape of the $\beta\eta$-normal forms is required in order to construct the $\beta\eta$-long normal forms. First, a technical lemma is given which extends lemma 4.2.6. This uses the Church–Rosser property and so was not proved earlier.

4.2.12 LEMMA Let ζ be a functional PTS with signatures and η satisfying strong normalisation. Then $\Gamma \not\vdash^\zeta_\Sigma \lambda x{:}A.B : u$ for all $u \in \mathcal{U}$.

Proof Assume the contrary. Using the generation lemma, it follows that $\Gamma \vdash^\zeta_\Sigma u = \Pi x{:}C.D : v$, for $v \in \mathcal{U}$ and preterms C and D, which is impossible using strong normalisation and the Church–Rosser property. □

4.2.13 LEMMA Let A be in $\beta\eta$-normal form. The subterms of A (definition 2.1.2) are also in $\beta\eta$-normal form.

Proof By definition. □

4.2.14 LEMMA Let ζ be a functional PTS with signatures and η satisfying strong normalisation.

1. A term A in $\beta\eta$-normal form has shape

$$\lambda x{:}A_1.\ldots\lambda x_n{:}A_n.\Pi y_1{:}B_1.\ldots\Pi y_m{:}B_m.@M_1\ldots M_k$$

for $n, m, k \geq 0$, where @ is a variable, constant or universe and the As, Bs and Ms are in $\beta\eta$-normal form, and where

   (a) A_i is not a λ-abstraction for each $i \in \{1,\ldots,n\}$;

   (b) B_j is not a λ-abstraction for each $j \in \{1,\ldots,m\}$.

2. Let $\Sigma = \langle a_1{:}A_1,\ldots,a_n{:}A_n\rangle$ and $\Gamma = \langle x_1 : A_{n+1},\ldots,x_m : A_{n+m}\rangle$ for $n, m \geq 0$. The $\beta\eta$-normal form of A_k with respect to $(\Sigma;\Gamma)$, for $k \in \{1,\ldots,n+m\}$, is not a λ-abstraction.

Proof Part 1 is proved by induction on the structure of A. The non-trivial cases are when A is a Π-abstraction or an application. If A is $\Pi x{:}C_1.C_2$ it is enough, since C_2 is in $\beta\eta$-normal form, to show that C_2 is not a λ-abstraction. Using the weak generation lemma (lemma 2.3.8), we have $\Gamma, x : C_1 \vdash^{\zeta}_{\Sigma} C_2 : u$ for universe u and context Γ and so, for this case, the result holds by lemma 4.2.12. For A of the form C_1C_2, we show that C_1 is not a λ- or Π- abstraction. The former is immediate since C_1C_2 cannot be a β-redex. By the generation lemma, we know that $\Gamma \vdash^{\zeta}_{\Sigma} C_1 : \Pi x{:}D_1.D_2$ for preterms D_1 and D_2. If C_1 is a Π-abstraction, then, by the generation lemma, we have $\Gamma' \vdash_{\Sigma} u = \Pi x : D_1.D_2 : v$ for universes u and v. By CR and strong normalisation, since the Π-structure is preserved by $\beta\eta$-reduction, it follows that u must be a Π-abstraction, which by CR is not the case.

For part 1a, we know that $\Gamma, x_1{:}A_1,\ldots,x_{i-1}{:}A_{i-1} \vdash^{\zeta}_{\Sigma} A_i : u_i$, for $u_i \in \mathcal{U}$ and $i \in \{1,\ldots,n\}$, using the weak generation lemma. From lemma 4.2.12, A_i is not a λ-abstraction. Parts 1b and 2 are proved in a similar way. □

The $\beta\eta$-normal forms of terms distinguish elements from the equivalence classes of terms given by the equality judgements. Another way of choosing witnesses to these classes is to use the *$\beta\eta$-long normal forms*, defined using the $\beta\eta$-normal

forms. This involves the concept of a constant or variable being fully-applied: for example, in the ELF$^+$ encoding of first-order logic the term $x : \iota \vdash^{ELF^+}_{\Sigma_{Fol}} +(x) : \iota \to \iota$ is not fully-applied, whereas $x : \iota \vdash^{ELF^+}_{\Sigma_{Fol}} +(x)(x) : \iota$ is.

4.2.15 DEFINITION Let $\Gamma \vdash_\Sigma A : B$. Then,

1. the *arity* of a universe in A with respect to $(\Sigma; \Gamma)$ is 0;
2. the *arity* of free variable x or constant a in A with respect to $(\Sigma; \Gamma)$ is the number of Πs in the prefix of C', where $x : C \in \Gamma$ or $a : C \in \Sigma$ and C' is the $\beta\eta$-normal form of C;
3. the *arity* of bound variable y in A with respect to $(\Sigma; \Gamma)$ is the number of Πs in the prefix of D', where D is the term attached to the binding occurrence of y and D' is the $\beta\eta$-normal form of D.

Remark The definition is well-defined using the free variable lemma (lemma 2.3.2), lemma 4.2.12 and lemma 4.2.14.

4.2.16 DEFINITION Let $\Gamma \vdash_\Sigma A : B$.

1. The term A is in *$\beta\eta$-long normal form with respect to* $(\Sigma; \Gamma)$ if it has shape

$$\lambda x_1{:}A_1 \ldots \lambda x_n{:}A_n.\Pi y_1{:}B_1 \ldots \Pi y_m{:}B_m.@M_1, \ldots, M_k$$

for $n, m, k \geq 0$, where

   (a) the arity of @ with respect to $(\Sigma; \Gamma, x_1{:}A_1, \ldots, x_n{:}A_n, y_1{:}B_1, \ldots, y_m{:}B_m)$ is k;

   (b) each A_i, for $i \in \{1, \ldots, n\}$, is in $\beta\eta$-long normal form with respect to $(\Sigma; \Gamma, x_1{:}A_1, \ldots, x_{i-1}{:}A_{i-1})$;

   (c) each B_j, for $j \in \{1, \ldots, m\}$, is in $\beta\eta$-long normal form with respect to $(\Sigma; \Gamma, x_1{:}A_1, \ldots, x_n{:}A_n, y_1{:}B_1, \ldots, y_{j-1}{:}B_{j-1})$;

   (d) the M_l for $l \in \{1, \ldots, k\}$ are in $\beta\eta$-long normal form with respect to $(\Sigma; \Gamma, x_1{:}A_1, \ldots, x_n{:}A_n, y_1{:}B_1, \ldots, y_m{:}B_m)$.

2. The term A has a *$\beta\eta$-long normal form with respect to* $(\Sigma; \Gamma)$ if, for preterm C, $\Gamma \vdash_\Sigma A = C : B$ and C is in $\beta\eta$-long normal form with respect to $(\Sigma; \Gamma)$.

Remark Notice that the definition of terms in $\beta\eta$-long normal form depends on the context and signature since, for example, given $x : A \vdash_\Sigma x : A$ for constant A declared in Σ, the variable x is in $\beta\eta$-long normal form with respect to the appropriate signature and context, whereas this is not the case for x in $x : A \to A \vdash_\Sigma x : A \to A$.

Just as in [HHP89], we use the $\beta\eta$-long normal forms, rather than equivalences, of ELF$^+$ terms in our analysis of ELF$^+$ representations. Given $\Gamma \vdash_\Sigma A : B$, we state an algorithm for constructing the $\beta\eta$-long normal form of A with respect to $(\Sigma; \Gamma)$. The proof of termination for this algorithm is sketched here and proved in [Gar92]. Dowek [Dow91] has independently shown termination using a different proof of essentially the same algorithm given for the Calculus of Construction [Coq85] and its subtheories.

4.2.17 DEFINITION Let $\Gamma \vdash_\Sigma A : B$. The *pseudo-long form of A with respect to* $(\Sigma; \Gamma)$ is constructed by first finding the $\beta\eta$-normal form of A, which has shape $\lambda x_1 : A_1 \ldots \lambda x_n : A_n . \Pi y_1 : B_1 \ldots \Pi y_m : B_m . @M_1 \ldots M_k$ for $n, m, k \geq 0$, where $@$ is a universe, constant or variable, and then proceeding as follows.

1. If $@$ is a universe go to 4.

2. If $@$ is a free variable or constant then replace $@M_1 \ldots M_k$ by

$$\lambda w_{k+1}{:}D_{k+1} \ldots \lambda w_p{:}D_p . @M_1 \ldots M_k w_{k+1} \ldots w_p,$$

for $p \geq k$, where

   (a) $@ : C$ is declared in Γ or Σ;

   (b) the $\beta\eta$-normal form of C is $\Pi x_1{:}C_1 \ldots \Pi x_p{:}C_p . \varepsilon N_1 \ldots N_q$;

   (c) each D_l, for $l \in \{k+1, \ldots, p\}$, is $C_l[M_1/w_1] \ldots [M_k/w_k]$;

   (d) the $w_{k+1} \ldots w_p$ are distinct variables which do not occur in $fv(A) \cup \{x_1, \ldots, x_n\} \cup \{y_1, \ldots, y_m\} \cup dom(\Gamma)$.

   Now go to 4.

3. If $@$ is a bound variable then replace $@M_1 \ldots M_k$ by

$$\lambda w_{k+1}{:}D_{k+1} \ldots \lambda w_p{:}D_p . @M_1 \ldots M_k w_{k+1} \ldots w_p,$$

for $p \geq k$, where $@ : C$ is one of the $x : A_i$ or $y : B_j$ for $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, m\}$, and conditions 2b, 2c and 2d hold. Now go to 4.

4. If n, m, k and p are 0 then stop;
   otherwise do the following:

   (a) for each $i \in \{1, \ldots, n\}$, replace A_i by its pseudo-long form with respect to $(\Sigma; \Gamma, x_1{:}A_1, \ldots, x_{i-1}{:}A_{i-1})$;

   (b) for each $j \in \{1, \ldots, m\}$, replace B_j by its pseudo-long form with respect to $(\Sigma; \Gamma, x_1{:}A_1, \ldots, x_n{:}A_n, y_1{:}B_1, \ldots, y_{j-1}{:}B_{j-1})$;

   (c) for each $l \in \{1, \ldots, k\}$, replace M_l by its pseudo-long form with respect to $(\Sigma; \Gamma, x_1{:}A_1, \ldots, x_n{:}A_n, y_1{:}B_1, \ldots, y_m{:}B_m)$;

   (d) for $r \in \{k+1, \ldots, p\}$, replace D_r by its pseudo-long form with respect to $(\Sigma; \Gamma, x_1{:}A_1, \ldots, x_n{:}A_n, y_1{:}B_1, \ldots, y_m{:}B_m, w_{k+1} : D_{k+1}, \ldots w_{r-1} : D_{r-1})$;

   (e) for $r \in \{k+1, \ldots, p\}$, replace w_r by its pseudo-long form with respect to $(\Sigma; \Gamma, x_1{:}A_1, \ldots, x_n{:}A_n, y_1{:}B_1, \ldots, y_m{:}B_m, w_{k+1} : D_{k+1}, \ldots w_r : D_r)$.

Remark It is not obvious that the above algorithm terminates since, for each $l \in \{k+1, \ldots, p\}$, the term D_l in parts 2 and 3 may contain β-redexes. One proof of termination [Gar92] (suggested by Plotkin) is to use an equivalent algorithm which first produces terms that are fully-applied (for example given $\Gamma \vdash_\Sigma @M_1 \ldots M_k$, the term $@M_1 \ldots M_k$ is fully-applied with respect to $(\Sigma; \Gamma)$ if the arity of $@$ is k) and then β-reduces; the algorithm for providing the fully-applied terms is preserved under substitution and the fully-applied condition is preserved by β-reduction. From this proof of termination, the uniqueness of the $\beta\eta$-long normal forms can be shown; we assume uniqueness for the remainder of this thesis.

Chapter 5

Adequate and Natural Encodings

The formal justification of the new framework ELF$^+$ is presented in this chapter. We show that ELF$^+$ permits general definitions of representation; the syntactic versions of these definitions are given here, and in the following chapter the simple algebraic formulations are presented.

Initially, we focus our attention on the consequence relations of formal systems. In the first section, we define the notion of an *adequate encoding* of an arbitrary logic represented in ELF$^+$, which characterises the representation of a consequence relation in the entailment relation of the representing type theory. We give examples of adequate encodings, and prove that linear and relevant logics [Gir87] [Dun84] cannot be represented in ELF$^+$.

The adequacy theorem accompanying the ELF representation of first-order logic [HHP89] also links the structure of derivations in the logic with the structure of certain ELF terms corresponding to the derivations. This gives some indication that the proof system of the logic can be mimicked by its representation in ELF$^+$. We provide a general definition of this correspondence, called a *natural* encoding. Using the same method as in [HHP89], this involves extending the syntax of the logic to incorporate expressions for proofs and adapting the proof system accordingly, which results in a consequence relation with an explicit account of the proof expressions. It is not clear how to perform this extension in general. We illustrate the method using first-order logic which provides a natural encoding in ELF$^+$. We also prove that the encoding of Hilbert-style S_4 [Che80] in ELF, although adequate, is not natural.

5.1 Representation of consequence relations

We characterise the representation of the consequence relation of a logic (definition 3.2.13) in ELF$^+$. This characterisation takes two parts; we first define an *encoding*, which gives a sound interpretation of the consequence relation in the entailment relation, and then an *adequate* encoding which states when this results in an equivalence. With representations in ELF it is not possible to define the basic notion of an encoding since, in some cases (example 5.1.15), a single signature is used to specify logics with different consequence relations.

5.1.1 Encodings

The definition of an encoding is given for an arbitrary logic specified by an ELF$^+$ signature. It provides a correspondence between the syntax and judgements of the logic and the ELF$^+$ terms in $\beta\eta$-long normal form (justified and defined in section 4.2.4). This correspondence identifies variables of the represented logic with sort variables, preserves substitution and gives a sound interpretation of the consequence relation in the entailment relation. Some care must be taken with identifying variables of the logic with sort variables. Each variable of the logic inhabits a unique syntactic class, whereas the corresponding information in the type theory is determined by the context, and so varies. We therefore define encodings using functions indexed by sequences of variables of the logic. The following notation is used throughout.

Notation Let LOG be an arbitrary logic represented in ELF$^+$ by Σ_{Log}. We distinguish the following sets of ELF$^+$ terms:

$$\begin{aligned} term_\Gamma &= \{t: \text{ for some preterm } A,\ \Gamma \vdash_{\Sigma_{Log}} t : A\}; \\ sort_\Gamma &= \{c : \Gamma \vdash_{\Sigma_{Log}} c : Sort\}; \\ texp_\Gamma &= \{s: \text{ for some } c \in sort_\Gamma, \Gamma \vdash_{\Sigma_{Log}} s : c\}; \\ judge_\Gamma &= \{j : \Gamma \vdash_{\Sigma_{Log}} j : Judge\}. \end{aligned}$$

The correspondence between the encoded logic and its representing type theory is given using the $\beta\eta$-long normal forms and so we distinguish the set

$$term_\Gamma^{\beta\eta} = \{t : t \in term_\Gamma \text{ and } t \text{ is in } \beta\eta\text{-long normal form w.r.t. } (\Sigma_{Log}; \Gamma)\}$$

and, similarly, the sets $sort_\Gamma^{\beta\eta}$, $texp_\Gamma^{\beta\eta}$ and $judge_\Gamma^{\beta\eta}$. We also require, for each preterm A,

$$A_\Gamma^{\beta\eta} = \{t : \Gamma \vdash_{\Sigma_{Log}} t : A \text{ and } t \text{ is in } \beta\eta\text{-long normal form w.r.t } (\Sigma_{Log}; \Gamma)\}.$$

Recall that $\mathcal{T}$ denotes the set of preterms of (ELF$^+$, Σ_{Log}) and Var^{Sort} and Var^{Judge} denote the sets of sort variables and judgement variables respectively: that is, if $\Gamma \vdash^{ELF^+}_{\Sigma_{Log}} x : A : Sort$ then x is a sort variable, and similarly for the judgement variables.

The set of syntactic classes containing term expressions, the set of term expressions and the set of judgements of a logic LOG are denoted by S_{LOG}, T_{LOG} and J_{LOG} respectively. For X a finite sequence of distinct logic variables, $T_{LOG}(X)$ and $J_{LOG}(X)$ denote the subsets whose members contain free variables in X. We omit the subscripts when the particular logic is apparent.

5.1.1 DEFINITION Let LOG be an arbitrary logic specified in ELF$^+$ by Σ_{Log}. An *encoding* of LOG in (ELF$^+$, Σ_{Log}) is a triple (η, ξ, δ) where $\eta : S \to \mathcal{T}$ is a function satisfying, for all $c \in S$,

$$\langle\,\rangle \vdash_{\Sigma_{Log}} \eta(c) : Sort,$$

such that $\eta(c)$ is in $\beta\eta$-long normal form with respect to $(\Sigma_{Log}; \langle\,\rangle)$.

Both ξ and δ are families of functions $\xi_X : T(X) \to \mathcal{T}$ and $\delta_X : J(X) \to \mathcal{T}$, for finite sequences of logic variables $X = \langle x_1^{\sigma_1}, \ldots, x_n^{\sigma_n}\rangle$ and for which there are distinguished bijections $f^c : Var^c \to Var^{Sort}$ for each $c \in S$, such that:

1. $\xi_X(x) = f^c(x)$ for x^c in X;

2. for each term expression t from syntactic class σ and judgement j, both with free variables in the sequence $X = \langle x_1^{\sigma_1}, \ldots, x_n^{\sigma_n}\rangle$, we have

$$\Gamma_X \vdash_{\Sigma_{Log}} \xi_X(t) : \eta(\sigma);$$
$$\Gamma_X \vdash_{\Sigma_{Log}} \delta_X(j) : Judge,$$

where Γ_X is $\langle \xi_X(x_1) : \eta(\sigma_1), \ldots, \xi_X(x_n) : \eta(\sigma_n) \rangle$ and $\xi_X(t)$ and $\delta_X(j)$ are in $\beta\eta$-long normal form with respect to $(\Sigma_{Log}; \Gamma_X)$;

3. the ξ_X and δ_X are *compositional*: that is, for term expressions $t \in T(Y)$ and $s_1, \ldots, s_n \in T(X)$ and judgement $j \in J(Y)$,

$$\xi_X(t[\overline{s}/\overline{x}]) = \xi_Y(t)[\overline{\xi_X(s)}/\overline{\xi_Y(x)}]$$
$$\delta_X(j[\overline{s}/\overline{x}]) = \delta_Y(j)[\overline{\xi_X(s)}/\overline{\xi_Y(x)}];$$

4. the interpretation is *sound*: that is, for sequences $X = \langle x_1^{\sigma_1}, \ldots, x_n^{\sigma_n} \rangle$ and $\langle j_1, \ldots, j_m \rangle$ of variables and judgements of the logic respectively,

$$\{j_1, \ldots, j_m\} \vdash_{\{x_1, \ldots, x_n\}} j \text{ implies}$$

$$\Gamma_X, p_1 : \delta_X(j_1), \ldots, p_m : \delta_X(j_m) \vdash_{\Sigma_{Log}} \_ : \delta_X(j),$$

where Γ_X is $\langle \varepsilon_X(x_1) : \eta(\sigma_1), \ldots, \xi_X(x_n) : \eta(\sigma_n) \rangle$, the $p_1, \ldots, p_m$ are distinct variables in Var^{Judge} and $\_ : \delta_X(j)$ denotes the inhabitation of ELF$^+$ term $\delta_X(j)$.

Remark The encoding definition depends on certain properties of the logics under consideration. The definition of syntactic classes (definition 3.1.1) does not depend on the variables of the logic so the image of η is contained in $sort^{\beta\eta}_{\langle\rangle}$. We also assume that the term expressions and the judgements (definitions 3.1.2 and 3.1.3) do not contain information regarding proof; this is mirrored in the encoding since, for each sequence of variables X, the images of ξ_X and δ_X are contained in $term^{\beta\eta}_{\Gamma_X}$ and $judge^{\beta\eta}_{\Gamma_X}$ for context of sorts Γ_X.

Remark An alternative approach is to define encodings without the indexing of variables and transfer the bookkeeping of variables to the definition of adequacy, where it is essential. Our method gives simpler definitions.

Remark In general, the $\beta\eta$-long normal forms are not preserved by substitution: for example, if $y : A, x : A \to A \vdash_\Sigma \Pi z : B.xM : u$ and $y : A \vdash_\Sigma (\lambda y{:}A.y) : A \to A$ then $y : A \vdash_\Sigma \Pi z : B[(\lambda y{:}A.y)/x].(\lambda y{:}A.y)M[(\lambda y{:}A.y)/x] : u$ which contains a β-redex. The compositional definition is well-defined in part 3 since λ-abstractions are not substituted, as substitutions are restricted to inhabitants of sorts.

Remark In the above correspondence we do not link derivations in the logic and inhabitants of ELF$^+$ judgements since the standard consequence relation of the logic contains no information about the derivations. The proof information is therefore disregarded in the entailment relation: that is, we are interested in the inhabitation of ELF$^+$ judgements, rather than particular ELF$^+$ terms.

Notation Let (η, ξ, δ) be an encoding of a logic in ELF$^+$. For each sequence $X = \langle x_1^{\sigma_1}, \ldots, x_n^{\sigma_n} \rangle$ of variables of the logic, we let Γ_X denote the contexts of sorts $\langle \xi_X(x_1) : \eta(\sigma_1), \ldots, \xi_X(x_n) : \eta(\sigma_n) \rangle$. We write $\xi_X : T(X) \to tex p_{\Gamma_X}^{\beta\eta}$ and $\delta_X : J(X) \to judge_{\Gamma_X}^{\beta\eta}$ to denote the functions extensionally equal to ξ_X and δ_X, but with the more precise ranges. These are well-defined by condition 2 in definition 5.1.1. We also write $\eta : S \to sort_{\langle\rangle}^{\beta\eta}$. These functions play a central role in the definition of an adequate encoding (definition 5.1.4).

5.1.2 PROPOSITION Let (η, ξ, δ) be an encoding of a logic in ELF$^+$ using Σ_{Log} and let X and Y be sequences of variables of the logic. If $t \in T(X)$ and $t \in T(Y)$ then $\xi_X(t) = \xi_Y(t)$. Similarly, if $j \in J(X)$ and $j \in J(Y)$ then $\delta_X(j) = \delta_Y(j)$.

Proof Follows from the compositional property (part 3 of definition 5.1.1). □

Ideally, the correspondence between a logic and its representation in a framework should be immediately apparent, although it is not clear that this goal is compatible with the aim of representing a wide variety of logics. With ELF$^+$, the link between the logic and the representing type theory is usually obvious, although some work must be done to show that it satisfies the conditions required in definition 5.1.1. We give an encoding of first-order logic in ELF$^+$ specified by Σ_{Fol} (section 4.2.2); other examples of encodings can be found in 5.1.10 and 5.1.12. In the case of Σ_{Fol}, the proof that we indeed have an encoding is similar to part of the proof of the adequacy theorem accompanying the representations of first-order logic in ELF [HHP89]; this is to be expected as our definitions make the intuition behind the adequacy theorems precise.

5.1.3 THEOREM The ELF$^+$ signature Σ_{Fol} gives an encoding of first-order logic in ELF$^+$.

Proof The syntactic class of terms, denoted by *term*, is represented in ELF^{+} by ι. The function $\eta : S \to sort^{\beta\eta}_{\langle\rangle}$ is therefore:

$$\eta(term) \;=\; \iota$$

For each sequence $X = \langle x_1, \ldots, x_n \rangle$ of variables of first-order logic (we omit the superscripts as there is only one syntactic class containing variables), the function $\xi_X : T(X) \to tex p^{\beta\eta}_{\Gamma_X}$ is defined inductively on the structure of $t \in T(X)$ as follows:

$$\begin{aligned} \xi_X(x) &= x' && x \in X \\ \xi_X(0) &= 0 \\ \xi_X(succ(t)) &= succ(\xi_X(t)) \\ \xi_X(t+s) &= +(\xi_X(t))(\xi_X(s)) \end{aligned}$$

where $(\_)'$ denotes a bijection from Var^{Log} to Var^{Sort} and Γ_X is $\langle x'_1 : \iota, \ldots, x'_n : \iota \rangle$. The function ξ_X is evidently well-defined and total. Compositionality for ξ_X is shown to hold by a straightforward structural induction on first-order term expressions.

Similarly, for each sequence of variables $X = \langle x_1, \ldots, x_n \rangle$, the function $\delta_X : J(X) \to judge^{\beta\eta}_{\Gamma_X}$ is given by $\delta_X(\phi\ true) = true(\gamma_X(\phi))$ for formula ϕ, where $\gamma_X : F(X) \to o^{\beta\eta}_{\Gamma_X}$, with $F(X)$ denoting the set of formulae with free variables in X, is defined inductively as follows:

$$\begin{aligned} \gamma_X(t = s) &= =(\xi_X(t))(\xi_X(s)) \\ \gamma_X(\phi \supset \psi) &= \supset(\gamma_X(\phi))(\gamma_X(\psi)) \\ \gamma_X(\forall x.\phi) &= \forall(\lambda x' : \iota.\gamma_{X,x}(\phi)) \\ \gamma_X(\exists x.\phi) &= \exists(\lambda x' : \iota.\gamma_{X,x}(\phi)) \end{aligned}$$

where X, x denotes $X \cup \{x\}$. It is easily shown that γ_X is well-defined and total. Hence, δ_X is a well-defined, total function satisfying part 2 of definition 5.1.1. The compositional property for δ_X is shown by proving the equivalent property for γ_X, which proceeds by straightforward structural induction on the formulae of first-order logic.

All that remains to do is show that the above correspondence provides a sound interpretation of the consequence relation of the logic in the entailment

relation of (ELF$^+$, Σ_{Fol}); in this case, for finite sequences $X = \langle x_1, \ldots, x_n \rangle$ and $\langle \phi_1\, true, \ldots, \phi_n\, true \rangle$ of variables and basic judgements of the logic respectively, we must show

$$\{\phi_1\, true, \ldots, \phi_m\, true\} \vdash_{\{x_1, \ldots, x_n\}} \phi\, true \text{ implies}$$

$$\Gamma_X, p_1 : \delta_X(\phi_1\, true), \ldots, p_m : \delta_X(\phi_m\, true) \vdash \_ : \delta_X(\phi\, true),$$

where Γ_X is $\langle x'_1 : \iota, \ldots, x'_n : \iota \rangle$ and $\_ : \delta_X(\phi\, true)$ denotes the inhabitation of $\delta_X(\phi\, true)$. The proof follows by induction on the derivation of

$$\{\phi_1\, true, \ldots, \phi_m\, true\} \Rightarrow_{\{x_1, \ldots, x_n\}} \phi\, true.$$

□

5.1.2 Adequate encodings

An *adequate* encoding defines an exact correspondence between a consequence relation of a logic and its representation in ELF$^+$. This is important since it not only states that we get a sound and complete interpretation of the consequence relation in the entailment relation, but also that we can recover the logic from the representing type theory since no information has been lost during encoding. We show that logics which have adequate encodings in ELF$^+$ have intuitionistic consequence relations (definition 3.2.13).

5.1.4 DEFINITION An encoding (η, ξ, δ) is *adequate* when

1. $\eta : S \to sort^{\beta\eta}_{\langle\rangle}$ is a bijection;
2. for each finite sequence $X = \langle x_1^{\sigma_1}, \ldots, x_n^{\sigma_n} \rangle$ of variables, the functions $\xi_X : T(X) \to texp^{\beta\eta}_{\Gamma_X}$ and $\delta_X : J(X) \to judge^{\beta\eta}_{\Gamma_X}$ are bijections;
3. the interpretation is *complete*; that is, for sequences $X = \langle x_1^{\sigma_1}, \ldots, x_n^{\sigma_n} \rangle$ and $\langle j_1, \ldots, j_m \rangle$ of variables and judgements of the logic respectively,

$$\Gamma_X, p_1 : \delta_X(j_1), \ldots, p_m : \delta_X(j_m) \vdash_{\Sigma_{Log}} \_ : \delta_X(j) \text{ implies}$$
$$\{j_1, \ldots, j_m\} \vdash_{\{x_1, \ldots, x_n\}} j,$$

where the $p_1, \ldots, p_m$ are distinct variables in Var^{Judge} and $\_ : \delta_X(j)$ denotes the inhabitation of ELF$^+$ term $\delta_X(j)$.

Remark A weaker notion results if we only require that the functions are injective and that the completeness condition (part 3) holds; we call such an encoding *weakly adequate.* For example, the representation of first-order logic (section 4.2.2) provides a weakly adequate representation of propositional logic. We do not concentrate on this definition since it is important to be able to discern from the type theory that part of the entailment relation which corresponds to the consequence relation of the underlying logic. This is very important when one wishes to investigate, for example, proof search and general tactics for representations in ELF$^+$, topics which are beyond the scope of this thesis, but which are studied in [PW91], [Schm83] and elsewhere.

Remark Weaker notions of encoding require investigation. In chapter 3, we emphasise that ELF$^+$ has a certain approach to syntax, which need not be the same as the approach in the original presentation of the represented logic. We therefore gave a standard presentation which can be viewed as a transitional stage between logics and their representations in ELF$^+$. For example, the representation of Hilbert-style S_4 (example 5.1.12) uses the transitional logic $\mathcal{L}_{new}$. One possible avenue to explore is an encoding consisting of two maps; one from the transitional logic to the original logic and the other from the same domain to the representation in ELF$^+$.

Notation Let (η, ξ, δ) be an adequate encoding. For each context of sorts $\Gamma_S = \langle x_1{:}A_1, \ldots, x_n{:}A_n \rangle$ in $\beta\eta$-long normal form, let X_{Γ_S} denote the sequence of variables

$$\langle g^{\eta^{-1}(A_1)}(x_1)^{\eta^{-1}(A_1)}, \ldots, g^{\eta^{-1}(A_n)}(x_n)^{\eta^{-1}(A_n)} \rangle,$$

where $g^c : Var^{Sort} \to Var^c$ is the inverse of the function $f^c : Var^c \to Var^{Sort}$ given by the encoding. We write $\xi'_{\Gamma_S} : texp^{\beta\eta}_{\Gamma_S} \to T(X_{\Gamma_S})$ and $\delta'_{\Gamma_S} : judge^{\beta\eta}_{\Gamma_S} \to J(X_{\Gamma_S})$ for the inverse of functions $\xi_{X_{\Gamma_S}}$ and $\delta_{X_{\Gamma_S}}$ respectively.

5.1.5 DEFINITION Let LOG be a logic specified in ELF$^+$ by Σ_{Log}. The logic is *adequately represented* in ELF$^+$ if there is an adequate encoding of LOG in (ELF^+, Σ_{Log}).

5.1.6 PROPOSITION Let (η, ξ, δ) be an adequate encoding. The functions $\xi'_{\Gamma_S} : texp^{\beta\eta}_{\Gamma_S} \to T(X_{\Gamma_S})$ and $\delta'_{\Gamma_S} : judge^{\beta\eta}_{\Gamma_S} \to J(X_{\Gamma_S})$ are compositional: that

is, for $t \in sort^{\beta\eta}_{\Delta_S}$ and $j \in judge^{\beta\eta}_{\Delta_S}$ and term expressions $s_1, \ldots, s_n \in sort^{\beta\eta}_{\Gamma_S}$, where $\Delta_S = \langle x_1{:}A_1, \ldots, x_n{:}A_n\rangle$ and $\Gamma_S \vdash_{\Sigma_{Log}} s_i : A_i[s_1, \ldots, s_{i-1}/x_1, \ldots, x_{i-1}]$, we have

$$\xi'_{\Gamma_S}(t[\overline{s}/\overline{x}]) = \xi'_{\Delta_S}(t)[\overline{\xi'_{\Gamma_S}(s)}/\overline{\xi'_{\Delta_S}(x)}];$$
$$\delta'_{\Gamma_S}(j[\overline{s}/\overline{x}]) = \delta'_{\Delta_S}(j)[\overline{\xi'_{\Gamma_S}(s)}/\overline{\xi'_{\Delta_S}(x)}].$$

Proof Follows from the compositionality of ε_X and δ_X, for each set of variables X. □

It is intuitively clear that, for a logic to be well-represented in a framework, the metatheory of the logic and the framework must be compatible. We are at last able to capture this intuition as the following theorem shows.

5.1.7 THEOREM Logics which are adequately represented in ELF$^+$ have intuitionistic consequence relations (definition 3.2.13).

Proof Let LOG be a logic which has an adequate encoding denoted by (η, ξ, δ) in (ELF$^+$, Σ_{Log}) and assume that we have $\{J_1, \ldots, J_n\} \vdash^{Log}_{\{x_1^{\sigma_1}, \ldots, x_n^{\sigma_n}\}} J$ and $\{J, K_1, \ldots, K_m\} \vdash^{Log}_{\{x_1^{\sigma_1}, \ldots, x_n^{\sigma_n}\}} K$. Then $\Gamma_X, p_1 : \delta_X(J_1), \ldots, p_n : \delta_X(J_n) \vdash_{\Sigma_{Log}} \pi : \delta_X(J)$ and $\Gamma_X, q : \delta_X(J), q_1 : \delta_X(K_1), \ldots, q_m : \delta_X(K_m) \vdash_{\Sigma_{Log}} \pi' : \delta_X(K)$ where Γ_X is $\langle \varepsilon_X(x_1) : \eta(\sigma_1), \ldots, \varepsilon_X(x_n) : \eta(\sigma_n)\rangle$. By the substitution lemma (lemma 2.1.10), the thinning lemma (lemma 2.1.12) and renaming variables if necessary we have $\Gamma_X, p_1 : \delta_X(J_1), \ldots, p_n : \delta_X(J_n), q_1 : \delta_X(K_1), \ldots, q_m : \delta_X(K_m) \vdash_{\Sigma_{Log}} \pi'[\pi/q] : \delta_X(K)$. Since (η, ξ, δ) is an adequate encoding, we have $\{J_1, \ldots, J_n, K_1, \ldots, K_m\} \vdash^{Log}_X K$. Hence, LOG satisfies the cut condition in definition 3.2.12. In a similar fashion, it is easy to show that LOG also satisfies the weakening and substitution conditions in definition 3.2.13. □

5.1.8 COROLLARY There are no adequate representations of linear and relevant logics [Gir87] [Dun84] in ELF$^+$.

Proof The consequence relations of these logics are not intuitionistic since they do not satisfy the weakening condition in definition 3.2.13. □

5.1.9 THEOREM The encoding of first-order logic in (ELF$^+$, Σ_{Fol}) defined in the proof of theorem 5.1.3 is adequate.

Proof The function $\eta : S \to sort^{\beta\eta}_{\langle\rangle}$ is obviously a bijection. For each context of sorts Γ_S in $\beta\eta$-long normal form, we define $\varepsilon'_{\Gamma_S} : sort^{\beta\eta}_{\Gamma_S} \to T(X_{\Gamma_S})$ by induction on the structure of the terms in $sort^{\beta\eta}_{\Gamma_S}$ (in this case, terms inhabiting ι) as follows:

$$\begin{aligned}
\varepsilon'_{\Gamma_S}(x) &= g(x) && x \in dom(\Gamma_S)\\
\varepsilon'_{\Gamma_S}(0) &= 0\\
\varepsilon'_{\Gamma_S}(succ(t)) &= succ(\varepsilon'_{\Gamma_S}(t))\\
\varepsilon'_{\Gamma_S}(+(t)(s)) &= \varepsilon'_{\Gamma_S}(t) + \varepsilon'_{\Gamma_S}(s)
\end{aligned}$$

where $g : Var^{Log} \to Var^{term}$ is inverse to $(\ )' : Var^{Log} \to Var^{Sort}$ given in the proof of theorem 5.1.3. The function $\varepsilon_X : T(X) \to sort^{\beta\eta}_{\Gamma_X}$, for each sequence of variables X, is a bijection since ε'_{Γ_X} is its right inverse.

Similarly we have, for each context of sorts Γ_S in $\beta\eta$-long normal form, the function $\delta'_{\Gamma_S} : judge^{\beta\eta}_{\Gamma_S} \to J(X_{\Gamma_S})$. This is given by $\delta'_{\Gamma_S}(true(\phi)) = \gamma'_{\Gamma_S}(\phi)\,true$, where $\gamma'_{\Gamma_S} : o^{\beta\eta}_{\Gamma_S} \to F(X_{\Gamma_S})$ is the inverse of $\gamma_{X_{\Gamma_S}} : F(X_{\Gamma_S}) \to o^{\beta\eta}_{\Gamma_S}$ defined in the proof of theorem 5.1.3.

Finally we show, for $J_1, \ldots, J_m, J \in judge^{\beta\eta}_{\Gamma_S}$ with Γ_S a context of sorts in $\beta\eta$-long normal form, that

$$\Gamma_S, p_1{:}J_1, \ldots, p_m{:}J_m \vdash^{ELF^+}_{\Sigma_{Fol}} \Pi : J \text{ implies } \{\delta'_{\Gamma_S}(J_1), \ldots, \delta'_{\Gamma_S}(J_m)\} \vdash_{\varepsilon'_{\Gamma_S}(\Gamma_S)} \delta'_{\Gamma_S}(J),$$

where Π is in $\beta\eta$-long normal form with respect to $(\Sigma_{Fol}; \Gamma_S, p_1{:}J_1, \ldots, p_m{:}J_m)$ and $\varepsilon'_{\Gamma_S}(\Gamma_S)$ denotes the set $\{\varepsilon'_{\Gamma_S}(x) : x \in dom(\Gamma_S)\}$.

This is proved by induction on the structure of Π: we just look at one case. Let Δ denote $\Gamma_S, p_1{:}J_1, \ldots, p_m{:}J_m$, so that $\Delta \vdash_{\Sigma_{Fol}} \Pi : J$, and assume Π is of the form $\forall I(\lambda x{:}\iota.\theta)(\lambda x{:}\iota.q)$. Using the generation lemma and renaming variables if necessary, $\Delta, x : \iota \vdash_{\Sigma_{Fol}} q : true(\theta)$ and J is $true(\forall(\lambda x{:}\iota.\theta))$ by the uniqueness of $\beta\eta$-long normal forms. By the induction hypothesis, and using the permutation lemma to obtain $\Delta, x : \iota$ in the right form, it follows that

$$\{\delta'_{\Gamma_S, x:\iota}(J_1), \ldots, \delta'_{\Gamma_S, x:\iota}(J_m)\} \vdash_Y \delta'_{\Gamma_S, x:\iota}(true(\theta)),$$

where Y is $\{\varepsilon'_{\Gamma_S,x:\iota}(y) : y \in dom(\Gamma_S, x:\iota)\}$, and so we have a derivation of the corresponding sequent $\{\delta'_{\Gamma_S,x:\iota}(J_1),\ldots,\delta'_{\Gamma_S,x:\iota}(J_m)\} \Rightarrow_Y \gamma'_{\Gamma_S,x:\iota}(\theta)\,true$. Using the $\forall I$-rule of first-order logic, we infer

$$\{\delta'_{\Gamma_S,x:\iota}(J_1),\ldots,\delta'_{\Gamma_S,x:\iota}(J_m)\} \Rightarrow_{Y/\{x'\}} \forall x'.\gamma'_{\Gamma_S,x:\iota}(\theta)\,true,$$

where x' is $\xi'_{\Gamma_S,x:\iota}(x)$. By proposition 5.1.2, it follows that

$$\{\delta'_{\Gamma_S}(J_1),\ldots,\delta'_{\Gamma_S}(J_m)\} \vdash_{\xi'_{\Gamma_S}(\Gamma_S)} \delta'_{\Gamma_S}(J).$$

□

5.1.3 More examples

5.1.10 EXAMPLE [Higher-order logic] The representation of higher-order logic in ELF$^+$ gives an example of an adequate encoding which requires extra constants to express the syntax of the logic in the type theory. There are many ways of presenting higher-order logic (see for example [Chu40], [And71], [Sch77], [Tak75]). We encode the version given in [HHP89], which follows Church in using the simply typed λ-calculus [Mit91] to form the syntax of the logics with the simple types being treated as syntactic classes (often called the domains). The syntax and basic judgements are defined using the alphabet notation introduced in chapter 3:

$$\begin{aligned}
C &= \{i^0, o^0, \Rightarrow^2\}\\
C' &= C\\
E &= \{0^\iota, succ^{\iota\Rightarrow\iota}, +^{\iota\Rightarrow\iota\Rightarrow\iota}, \supset^{o\Rightarrow o\Rightarrow o}\} \cup \bigcup_\alpha \{=_\alpha^{\alpha\Rightarrow\alpha\Rightarrow o}, \forall_\alpha^{(\alpha\Rightarrow o)\Rightarrow o}\} \cup\\
&\qquad \bigcup_{\alpha,\beta} \{app_{\alpha,\beta}^{(\alpha\Rightarrow\beta,\alpha)\to\beta}, \lambda_{\alpha,\beta}^{(\alpha\to\beta)\to(\alpha\Rightarrow\beta)}\}\\
J &= \{true^{(o)}\}
\end{aligned}$$

To illustrate the representation, we consider the following selection of rules in the natural deduction system for higher-order logic:

$$\forall I \qquad \frac{\Gamma \Rightarrow_{X,x^\sigma} \phi\,true}{\Gamma \Rightarrow_X \forall_\sigma(\lambda_{\sigma,o}x.\phi)\,true}$$

$$\forall E \qquad \frac{\Gamma \Rightarrow_X \forall_\sigma(e)\,true}{\Gamma \Rightarrow_X app_{\alpha,o}(e)(e')\,true}$$

EQ
$$\frac{\Gamma \Rightarrow_X \phi\, true \quad \Gamma \Rightarrow_X (\phi =_o \psi)\, true}{\Gamma \Rightarrow_X \psi\, true}$$

LAM
$$\frac{\Gamma \Rightarrow_{X,x} (e =_\tau e')\, true}{\Gamma \Rightarrow_X (\lambda_{\sigma,\tau} x.e =_{\sigma\Rightarrow\tau} \lambda_{\sigma,\tau} x.e')\, true}$$

β
$$\Gamma \Rightarrow_X (app_{\sigma,\tau}(\lambda_{\sigma,\tau} e)(x) =_\tau e[e'/x])\, true$$

η
$$\Gamma \Rightarrow_X (\lambda_{\sigma,\tau} x.(app_{\sigma,\tau}(e)(x)) =_{\sigma\Rightarrow\tau} e)\, true \qquad x \notin fv(e)$$

The representation of the domains has already been discussed in example 4.2.3. They are specified in ELF$^+$ by the following declarations in the signature Σ_{Hol}, adapted from the specification of higher-order logic in ELF [AHM89]:

$$\begin{aligned} dom &: Type \\ \iota &: dom \\ o &: dom \\ \Rightarrow &: dom \to dom \to dom \\ \\ obj &: dom \to Sort \end{aligned}$$

The remaining part of the specification is similar to the ELF representation of first-order logic. The expressions are specified by a set of constants, one for each expression symbol of higher-order logic:

$$\begin{aligned} 0 &: obj(\iota) \\ succ &: obj(\iota \Rightarrow \iota) \\ + &: obj(\iota \Rightarrow \iota \Rightarrow \iota) \\ \supset &: obj(o \Rightarrow o \Rightarrow o) \\ = &: \Pi s{:}\, dom.obj(s \Rightarrow s \Rightarrow o) \\ \forall &: \Pi s{:}\, dom.obj((s \Rightarrow o) \Rightarrow o) \\ \Lambda &: \Pi s{:}\, dom.\Pi t{:}\, dom.(obj(s) \to obj(t)) \to obj(s \Rightarrow t) \\ app &: \Pi s{:}\, dom.\Pi t{:}\, dom.obj(s \Rightarrow t) \to obj(s) \to obj(t) \end{aligned}$$

Notice that the representation of the quantifier and equality, both indexed by the domains, makes use of the dependent terms in an essential way, just as in the ELF representation of higher-order logic. The abstraction operator of higher-order logic is specified by the constant Λ to avoid confusion with the λ-abstraction of the type theory, and the constant $\Rightarrow$ is used as an infix operator.

Using the same approach as in the encoding of first-order logic, we declare a constant to form the truth judgements of the logic:

$$true \quad : \quad obj(o) \to Judge$$

The inference rules are specified using techniques similar to those for first-order logic. As a notational expedient, we make use of the following 'externalisation' of the equality constant:

$$\approx \equiv \lambda s{:}\, dom.\lambda x{:}\, obj(s), \lambda y : obj(s).app_{s,o}(app_{s,s\Rightarrow o}(=_s x)(y)$$

which inhabits

$$\Pi s{:}\, dom.obj(s) \to obj(s) \to obj(o)$$

and which we write in infix form. Again we have one constant for each rule (we write arguments to applications as subscripts to enhance readability):

$$\forall I \quad : \quad \Pi s{:}\, dom.\Pi F{:}\, obj(s{\Rightarrow}o).(\Pi x{:}\, obj(s).true(app_{s,o}\, F\, x)) \to true(app_{s\Rightarrow o,o} \forall_s F)$$

$$\forall E \quad : \quad \Pi s{:}\, dom.\Pi F{:}\, obj(s{\Rightarrow}o).\Pi x{:}\, obj(s).true(app_{(s\Rightarrow o),o} \forall_s F) \to true(app_{s,o}\, F\, x)$$

$$eq \quad : \quad \Pi\phi{:}obj(o).\Pi\psi{:}obj(o).true(\phi) \to true(\phi \approx_o \psi) \to true(\psi)$$

$$lam \quad : \quad \Pi s,t{:}dom.\Pi f,g{:}obj(s){\to}obj(t).(\Pi x{:}obj(s).true(f\, x \approx_t g\, x)) \to$$
$$true(\Lambda_{s,t}\, \lambda x{:}\, obj(s).fx \approx_{s\Rightarrow t} \Lambda_{s,t}\, \lambda x{:}\, obj(s).gx)$$

$$\beta \quad : \quad \Pi s,t{:}\, dom.\Pi f{:}\, obj(s){\to}obj(t).\Pi x{:}\, obj(s).$$
$$true(app_{s,t}\, (\Lambda_{s,t}\, (\lambda x{:}obj(s).fx))\, x \approx_t f\, x)$$

$$\eta \quad : \quad \Pi s,t{:}\, dom.\Pi f{:}\, obj(s{\Rightarrow}t).true(\Lambda_{s,t}\, (\lambda x{:}\, obj(s).app_{s,t}\, f\, x) \approx_{(s\Rightarrow t)} f)$$

5.1.11 THEOREM The signature Σ_{Hol} provides an adequate encoding of higher-order logic in ELF$^+$.

Proof The mapping $\eta : S \to sort^{\beta\eta}_{\langle\rangle}$ is given, for each domain σ, by $\eta(\sigma) = obj(\beta(\sigma))$, where $\beta : S \to dom^{\beta\eta}_{\langle\rangle}$ is defined by

$$\beta(\iota) \quad = \quad \iota$$

$$\begin{aligned}\beta(o) &= o\\ \beta(\sigma \Rightarrow \sigma') &= \Rightarrow(\beta(\sigma))(\beta(\sigma'))\end{aligned}$$

The link between the term expressions and inhabitants of sorts is given, for each finite sequence X of variables of higher-order logic, by $\xi_X : T(X) \to texp^{\beta\eta}_{\Gamma_X}$ as follows:

$$\begin{aligned}\xi_X(x^\sigma) &= f^\sigma(x) && x^\sigma \in X\\ \xi_X(0) &= 0\\ \xi_X(succ) &= succ\\ \xi_X(+) &= +\\ \xi_X(\supset) &= \supset\\ \xi_X(\forall_\sigma) &= \forall(\beta(\sigma)) && \sigma \in S\\ \xi_X(=_\sigma) &= =(\beta(\sigma)) && \sigma \in S\\ \xi_X((\lambda_\sigma x.e)^{\sigma\to\tau}) &= \Lambda(\beta(\sigma))(\beta(\tau))(\lambda f^\sigma(x){:}\eta(\sigma).\xi_{X,x}(e^\tau)) && \sigma,\tau \in S\\ \xi_X((e^{\sigma\to\tau}d^\sigma)^\tau) &= app(\beta(\sigma))(\beta(\tau))(\xi_X(e^{\sigma\to\tau}))(\xi_X(d^\sigma)) && \sigma,\tau \in S\end{aligned}$$

and $f^\sigma : Var^\sigma \to Var^{Sort}$ is a bijection for each $\sigma \in S$. The connection between the judgements of the logic and inhabitants of $Judge$ is given, for each finite sequence of variables X by $\delta_X : J(X) \to judge^{\beta\eta}_{\Gamma_X}$, where $\delta_X(\phi) = true(\xi_X(\phi))$. From these mappings we obtain an encoding.

The function $\eta : S \to class^{\beta\eta}_{\langle\rangle}$ is a bijection since $\beta : S \to dom^{\beta\eta}_{\langle\rangle}$ is. To show that $(\eta, \varepsilon, \delta)$ provides an adequate encoding, we define functions $\varepsilon'_{\Gamma_S} : sort^{\beta\eta}_{\Gamma_S} \to T(X_{\Gamma_S})$ and $\delta'_{\Gamma_S} : judge^{\beta\eta}_{\Gamma_S} \to J(X_{\Gamma_S})$. Let $g^c : Var^{Sort} \to Var^c$ denote the inverse functions of f^c for each syntactic class c. Then, for a context of sorts Γ_S in $\beta\eta$-long normal form, we define ε'_{Γ_S} by induction on the structure of terms inhabiting sorts in this context as follows:

$$\begin{aligned}\varepsilon'_{\Gamma_S}(x) &= g^{\eta^{-1}(obj(A))}(x) && x : obj(A) \in \Gamma_S\\ \varepsilon'_{\Gamma_S}(0) &= 0\\ \varepsilon'_{\Gamma_S}(succ) &= succ\\ \varepsilon'_{\Gamma_S}(+) &= +\\ \varepsilon'_{\Gamma_S}(\supset) &= \supset\\ \varepsilon'_{\Gamma_S}(\forall(A)) &= \forall_{\beta^{-1}(A)}\end{aligned}$$

$$\begin{aligned}
\varepsilon'_{\Gamma_S}(=(A)) &= =_{\beta^{-1}(A)} \\
\varepsilon'_{\Gamma_S}(\Lambda(A)(B)(\lambda x{:}obj(A).e)) &= \lambda_{\beta^{-1}(A)}(g^{\eta^{-1}(obj(A))}(x)).\varepsilon'_{\Gamma_S,x:obj(A)}(e) \\
\varepsilon'_{\Gamma_S}(app(A)(B)(e)(f)) &= \varepsilon'_{\Gamma_S}(e)\varepsilon'_{\Gamma_S}(f).
\end{aligned}$$

The function $\delta'_{\Gamma_S} : judge^{\beta\eta}_{\Gamma_S} \to J(X_{\Gamma_S})$ is given by $\delta'_{\Gamma_S}(true(\phi)) = \varepsilon'_{\Gamma_S}(\phi)\,true$. It is routine to show that ε'_{Γ_X} and δ'_{Γ_X} are inverse to ξ_X and δ_X for each finite sequence of variables X and that the completion condition in definition 5.1.4 is satisfied. □

5.1.12 EXAMPLE [Hilbert-style S_4] The representation of Hilbert-style S_4 (example 3.2.16) is an example where an extra constant is used to represent the consequence relation of the logic in ELF$^+$. The difficulty lies with the NEC-rule presented, for example, as:

$$\text{NEC} \qquad \frac{\emptyset \Rightarrow_X \phi\,true}{\Gamma \Rightarrow_X \Box\phi\,true}$$

This rule cannot be represented directly by the standard method of declaring a constant *nec* inhabiting $\Pi\phi{:}o.true(\phi) \to true(\Box\phi)$ since such a constant would force the inhabitation of $true(\Box\phi)$ in *any* context entailing $true(\phi)$. The solution due to Avron [AHM89] centres on a logic, denoted by $\mathcal{L}_{new}$, with the same syntax as S_4, judgements of the form $\phi\,true$ and $\phi\,valid$ and the proof system in table 5.1. The consequence relation of $\mathcal{L}_{new}$ (denoted by $\vdash^{new}$) restricted to the truth judgements is the same as the consequence relation, $\vdash^{orig}$, of Hilbert-style S_4 .

5.1.13 THEOREM Let $\phi_1, \ldots, \phi_n, \phi$ be formulae of Hilbert-style S_4 and X a finite set of logic variables (in this case denoting formulae). Then

$\{\phi_1\,true, \ldots, \phi_n\,true\} \vdash^{orig}_X \phi\,true$ if and only if $\{\phi_1\,true, \ldots, \phi_n\,true\} \vdash^{new}_X \phi\,true$

Proof See [Avr86]. □

In Avron's approach, Hilbert-style S_4 is represented in ELF by encoding $\mathcal{L}_{new}$ and then, in the accompanying adequacy theorem, limiting the correspondence to those ELF terms representing the truth judgements. An important advantage of ELF$^+$ is that the specification of Hilbert-style S_4 is different from the specification of $\mathcal{L}_{new}$. The difference occurs in the universes in which the terms corresponding

A_1	$(\phi \supset (\psi \supset \phi))\, valid$
A_2	$((\phi \supset (\psi \supset \theta)) \supset (\phi \supset \psi) \supset (\phi \supset \theta))\, valid$
A_3	$(\Box\phi \supset \phi)\, valid$
A_4	$(\Box(\phi \supset \psi) \supset (\Box\phi \supset \Box\psi))\, valid$
A_5	$(\Box\phi \supset \Box\Box\phi)\, valid$
MP_v	$\dfrac{(\phi \supset \psi)\, valid \quad \phi\, valid}{\psi\, valid}$
TRUTH	$\dfrac{\phi\, valid}{\phi\, true}$
NEC	$\dfrac{\phi\, valid}{\Box\phi\, valid}$
MP_t	$\dfrac{(\phi \supset \psi)\, true \quad \phi\, true}{\psi\, true}$

Table 5.1: The new logic $\mathcal{L}_{new}$.

to $\phi\,true$ and $\phi\,valid$ inhabit. We declare the constants $true : o \to Judge$ and $valid : o \to Type$ which indicate that the terms of the form $true(\phi)$ correspond to the basic judgements of Hilbert-style S_4 and the terms of the form $valid(\phi)$ are extra terms given by the encoding. (In the representation of $\mathcal{L}_{new}$, the constants *true* and *valid* both inhabit $o \to Judge$.) The full specification of Hilbert-style S_4, denoted by Σ_{Mod}, is as follows:

$$
\begin{array}{lcl}
o & : & Sort \\
\\
\supset & : & o \to o \to o \\
\Box & : & o \to o \\
\\
true & : & o \to Judge \\
valid & : & o \to Type \\
\\
C & : & \Pi\phi{:}o.valid(\phi) \to true(\phi) \\
A1 & : & \Pi\phi,\psi{:}o.valid(\phi \supset (\psi \supset \phi)) \\
A2 & : & \Pi\phi,\psi,\theta{:}o.valid((\phi \supset (\psi \supset \theta)) \to ((\phi \supset \psi) \to (\phi \supset \theta))) \\
A3 & : & \Pi\phi{:}o.valid(\Box\phi \supset \phi) \\
A4 & : & \Pi\phi,\psi{:}o.valid(\Box(\phi \supset \psi) \supset (\Box\phi \supset \Box\psi)) \\
A5 & : & \Pi\phi{:}o.valid(\Box\phi \supset \Box\Box\phi) \\
MP_V & : & \Pi\phi,\psi{:}o.valid(\phi) \to valid(\phi \supset \psi) \to valid(\psi) \\
Nec & : & \Pi\phi{:}o.valid(\phi) \to valid(\Box\phi) \\
MP_T & : & \Pi\phi,\psi{:}o.true(\phi) \to true(\phi \supset \psi) \to true(\psi)
\end{array}
$$

5.1.14 THEOREM The signature Σ_{Mod} provides an adequate representation of Hilbert-style S_4 in ELF^+.

Proof The correspondence between the syntax and judgements of Hilbert-style S_4 and terms in (ELF^+, Σ_{Mod}) is easy. We have $\eta : S \to sort^{\beta\eta}_{()}$ given by

$$\eta(form) = o$$

and, for X a finite sequence of logic variables (denoting formulae in this case), the maps $\xi_X : T(X) \to texp^{\beta\eta}_{\Gamma_S}$ and $\delta_X : J(X) \to judge^{\beta\eta}_{\Gamma_S}$ are defined inductively by

$$
\begin{array}{rcll}
\xi_X(\phi) & = & \phi', & \phi \in X \\
\xi_X(\phi \supset \psi) & = & \supset(\xi_X(\phi))(\xi_X(\psi)) & \\
\xi_X(\Box\phi) & = & \Box(\xi_X(\phi)) &
\end{array}
$$

for bijection $(\_)' : Var^{Log} \to Var^{Sort}$ and

$$\delta_X(\phi\, true) \quad = \quad true(\xi_X(\phi)).$$

We must show, for sequences of variables $X = \langle x_1^{\sigma_1}, \ldots, x_n^{\sigma_n} \rangle$ and of judgements $\langle \phi\, true, \ldots, \phi_n\, true \rangle$, that

$$\{\phi_1\, true, \ldots, \phi_m\, true\} \vdash_{\{x_1, \ldots, x_n\}} \phi\, true \text{ implies}$$

$$\Gamma_X, p_1 : \delta_X(\phi_1\, true), \ldots, p_m : \delta_X(\phi_m\, true) \vdash_{\Sigma_{Mod}} \_ : \delta_X(\phi\, true),$$

for distinct variables $p_1, \ldots, p_m$ in Var^{Judge}, where $\_ : \delta(\phi\, true)$ denotes the inhabitation of $\delta_X(\phi\, true)$. This fact follows from theorem 5.1.13, together with a correspondence between $\mathcal{L}_{new}$ and (ELF$^+$, Σ_{Mod}) which is given by η and, for each finite sequence of variables X, by ξ_X and $v_X : J_{new}(X) \to T$, where $J_{new}(X)$ denotes the set of basic judgements in $\mathcal{L}_{new}$ with free variables in X. The function v_X is defined by

$$\begin{aligned} v_X(\phi\, true) &= \delta_X(\phi\, true) \\ v_X(\phi\, valid) &= valid(\xi_X(\phi) \end{aligned}$$

and, for sequences of variables $X = \langle x_1, \ldots, x_n \rangle$ and judgements $\langle J_1, \ldots, J_m \rangle$ in $\mathcal{L}_{new}$, satisfies

$$\{J_1, \ldots, J_m\} \vdash_X^{new} J \text{ implies } \Gamma_X, q_1 : v_X(J_1), \ldots, q_m : v_X(J_m) \vdash_{\Sigma_{Mod}} \_ : v_X(J),$$

where $q_1, \ldots, q_m$ are distinct variables from $Var^{Type} \cup Var^{Judge}$ and, as usual, $\_ : v_X(J)$ denotes the inhabitation of $v_X(J)$. It is easy to show that (η, ξ, δ) is an adequate encoding. □

5.1.15 EXAMPLE [λ_I-calculus] The λ_I-calculus [Bar84] is an example of a logic which has been represented in ELF [AHM89], but whose adapted signature for ELF$^+$ does not give an adequate encoding. In this calculus the λ-abstraction is restricted to

$$m \in \Lambda_I \text{ and } x \in fv(M) \text{ implies } \lambda x.M \in \Lambda_I,$$

where Λ_I denotes the set of term expressions. It has already been noted that the general principle for encoding logics in $ELF^{(+)}$ is to avoid declaring a term

whose objects correspond to logic variables; the variables of the encoded logic are identified with the sort variables. The difficulty therefore in encoding the λ_I-calculus is to identify which expressions have x as a free variable whilst still having the sort variables stand for logic variables. The method is inspired by the denotational semantics of the calculus and involves adding an extra constant $\bot\colon o$ which limits the application of the constant representing the λ-abstraction. The signature specifying the λ_I-calculus, denoted by Σ_I, is:

$$
\begin{array}{lcl}
exp & : & Sort \\
 & & \\
\bot & : & exp \\
\lambda_I & : & \Pi x{:}exp \to exp.x(\bot) = \bot \to exp \\
app & : & exp \to exp \to exp \\
 & & \\
= & : & exp \to exp \to Judge \\
 & & \\
E_0 & : & \Pi x{:}exp.x = x \\
E_1 & : & \Pi x, y{:}exp.x = y \to y = x \\
E_2 & : & \Pi x, y, z{:}exp.(x = y \to y = z \to x = z) \\
E_3 & : & \Pi x, y, x', y'{:}exp.x = y \to x' = y' \to (app(x, x') = app(y, y')) \\
\bot_r & : & \Pi x{:}exp.app(x, \bot) = \bot \\
\bot_l & : & \Pi x{:}exp.app(\bot, x) = \bot \\
\bot_\lambda & : & \bot = \lambda_I(\lambda x{:}exp.\bot, E_0(\bot)) \\
\beta_I & : & \Pi x{:}exp \to exp.\Pi y{:}exp.\Pi t{:}x(\bot) = \bot.app(\lambda_I(x, t), y) = xy
\end{array}
$$

Remark The signature Σ_I provides an example which makes use of the rule $(Judge, Sort, Type)$ of the type theory to form the term in which the constant λ_I resides even though, in λ_I, the syntax does not depend on proofs. It is not clear whether this rule, plus $(Judge, Type, Type)$, should in fact be included. This example does not clarify this point since this signature does not give an adequate encoding, as our next theorem states.

5.1.16 THEOREM The signtature Σ_I does not provide an adequate encoding of the λ_I-calculus in ELF^+.

Proof We suppose $(\eta, \varepsilon, \delta)$ is such an adequate encoding and show that this results in a contradiction. Since (η, ξ, δ) is adequate, there is a function

$\varepsilon'_{\langle\rangle} : sort^{\beta\eta}_{\langle\rangle} \to T(\emptyset)$ such that $\varepsilon'_{\langle\rangle} \circ \varepsilon_\emptyset = id_{T(\emptyset)}$ and $\varepsilon_\emptyset \circ \varepsilon'_{\langle\rangle} = id_{sort^{\beta\eta}_{\langle\rangle}}$. We show that all possible choices of $\varepsilon'_{\langle\rangle}(\bot)$ give a contradiction. If $\varepsilon'_{\langle\rangle}(\bot)$ is a variable, x say, then $\varepsilon_\emptyset(x) = \bot$ which does not satisfy the definition of $\varepsilon_\emptyset$. If $\varepsilon'_\emptyset(\bot)$ is MN for $M, N \in \Lambda_I$, then $\bot = \varepsilon_\emptyset(MN) = \varepsilon(Mx)[N/x]$ for $x \notin fv(M)$ by compositionality of ε and so $\varepsilon_\emptyset(Mx) = \bot$. Also, using the same argument for $y \notin fv(M)$, we have $\varepsilon_\emptyset(My) = \bot$, which contradicts the injectivity of ε. A similar argument applies when $\varepsilon'_{\langle\rangle}(\bot)$ is a λ-abstraction. □

Remark It is conceivable that there is an encoding from the λ_I-calculus to (ELF$^+$, Σ_I) which sends $\lambda x.M \in \Lambda_I$ to $\lambda_I(\lambda x{:}exp.M')(p)$ for ELF$^+$ term M' representing M and p some chosen inhabitant of term $(\lambda x{:}exp.M')(\bot) = \bot$. Adding an extra constant $\bot$ to the calculus is unlikely to provide an adequate encoding since the construction of ELF$^+$ terms using constant Λ_I depends on infinitely many terms inhabiting $x(\bot) = \bot$ for $x : exp \to exp$.

Remark An alternative approach to representing the λ_I-calculus is to transfer the restriction of the λ-abstraction to the rules for the equality judgement; that is, to declare λ_I in $(exp \to exp) \to exp$ and use a constant dir in $(exp \to exp) \to Type$, with the intuition that $dir(\lambda x : exp.e)$ is inhabited if x occurs free in e, to restrict the inhabitation of the equality judgement. Further research on the representation of side-conditions, exploiting the universe $Type$, is required.

5.2 Representation of proofs

Adequate encodings characterise representations of consequence relations in ELF$^+$. These encodings can be strengthened to give a stronger correspondence which also links derivations in a logic with terms inhabiting judgements in the representing type theory: we call such encodings *complete*. A complete encoding is *natural* when this stronger correspondence also yields an equivalence. A natural encoding captures the intuition that, for example, the representation of first-order logic is more direct than the representation of Hilbert-style S_4, and gives some measure of when a proof system can be mimicked by its representation in ELF$^+$. The full statements of the adequacy theorems for ELF [HHP89] are thus generalised. Following the approach in [HHP89], the syntax of a logic must be extended to give

an explicit account of derivations, and the proof system adapted accordingly. It is not clear how to do this in general. For the moment, we illustrate the putative method by example. In particular, we adapt first-order logic to give an account of proofs and show that Σ_{Fol} provides a natural encoding of first-order logic in ELF^+. We also show that the signature Σ_{Mod} gives an adequate encoding of Hilbert-style S_4 which is not natural, and prove that logics with natural representations must have consequence relations that are preserved under substitution of proofs.

5.2.1 Proof expressions

In this section, the syntax of first-order logic is extended to include a class of proof expressions, adapted from the definition in [HHP89], in order to give an explicit account of derivations in the consequence relation of first-order logic. The idea is that the derivation of, for example, $(\phi \supset \psi)\,true$, using the $\supset I$-rule in the last line, is denoted by the proof expression $\supset I(\phi)(\psi)((p)q)$ where q is a proof of $\psi\,true$ depending on the proof variable p which denotes a proof of $\phi\,true$. The discharge of assumption ϕ corresponds to binding the proof variable p. The conclusion $(\phi \supset \psi)\,true$ gives enough information to infer the formulae ϕ and ψ, although this is not so in general as the $\exists E$-rule illustrates. Thus the proof expressions themselves must carry information regarding the formulae used. Following the approach advocated in chapter 3, these expressions are generated from a set of *proof symbols*, denoted by pf, which have a similar behaviour to the expression and judgement symbols and which are accompanied by arities, where the set of arities of first-order logic is adapted to incorporate the class of proof expressions. The proof symbols for first-order logic are as follows:

$$\supset I^{(f,\,f,\,pf \to pf) \to pf}$$
$$\supset E^{(f,\,f,\,pf,\,pf) \to pf}$$
$$\forall I^{(t \to f,\,t \to pf) \to pf}$$
$$\forall E^{(t \to f,\,t,\,pf) \to pf}$$
$$\exists I^{(t \to f,\,t,\,pf) \to pf}$$
$$\exists E^{(t \to f,\,f,\,pf,\,(t,\,pf) \to pf) \to pf}$$

There is a proof symbol for each rule of first-order logic, given in example 3.2.14, whose arity indicates the schematic variables used in the rule, the proofs assumed

and the required binding of logic and proof variables. The proof expressions are defined using a countably infinite set of proof variables, distinct from the logic and schematic variables and denoted by Var^{Proof}, in a similar way to the logic expressions; for example, the proof expression $\forall I((x)\phi)((x)p)$ is formed from the proof symbol $\forall^{(t\to f,t\to pf)\to pf}$, the term variable x, the formula ϕ and the proof expression p. The notions of substitution and α-conversion are similar to those in definitions 3.1.11 and 3.1.13. Rules are adapted to sets of $(n+1)$-tuples of proof sequents of the form $\Gamma \Rightarrow_X p : J$, where X a set of variables of the logic, Γ is a set of proof assumptions of the form $\{p_1{:}J_1, \ldots, p_m{:}J_m\}$, with $J_1, \ldots, J_m$ basic judgements with free variables in X and $p_1, \ldots, p_m$ proof variables, J is a basic judgement with free variables in X, and p is a proof expression with free variables in X and free proof variables in $\{p_1, \ldots, p_m\}$.

Not all proof expressions denote valid derivations of a logic, just as not all preterms of a PTS are well-formed. The proof system of first-order logic (example 3.2.14) is adapted to incorporate these proof expressions and identify the ones which are *valid*; see table 5.2.

A complete encoding maps valid proof expressions to terms inhabiting judgements. We assume that the definitions of derivations and consequence relation with proofs are obvious adaptations of definitions 3.2.7 and 3.2.11 respectively. Notice that the consequence relation with proofs for first-order logic satisfies the following:

weakening $\Gamma \vdash_X p : J$ and $\Gamma \subseteq \Delta$ implies $\Delta \vdash_X p : J$;

substitution of variables $\Gamma \vdash_X p : J$ implies $\Gamma[\overline{t}/\overline{x}] \vdash_{X/\{\overline{x}\}\cup fv(\overline{t})} p[\overline{t}/\overline{x}] : J[\overline{t}/\overline{x}]$, where if Γ is $\{p_1{:}J_1, \ldots, p_m{:}J_m\}$ then $\Gamma[\overline{t}/\overline{x}]$ is $\{p_1 : J_1[\overline{t}/\overline{x}], \ldots, p_m : J_m[\overline{t}/\overline{x}]\}$ and $fv(\overline{t}) = \bigcup_{i=1}^{n} fv(t_i)$ for $\overline{t} = \langle t_1, \ldots, t_n \rangle$;

substitution of proof variables $\Gamma \vdash_X \Pi_i : J_i$ and $\Delta, p_1{:}J_1, \ldots, p_m{:}J_m \vdash_X \Sigma : K$ implies $\Gamma, \Delta \vdash_X \Sigma[\overline{\Pi}/\overline{p}] : K[\overline{\Pi}/\overline{p}]$ (renaming variables to avoid conflicting proof variables if necessary).

$$\supset I \qquad \frac{\Gamma, p : \phi\, true \Rightarrow_X q : \psi\, true}{\Gamma \Rightarrow_X \supset I(\phi, \psi, (p)q) : (\phi \supset \psi)\, true}$$

$$\supset E \qquad \frac{\Gamma \Rightarrow_X p : (\phi \supset \psi)\, true \quad \Gamma \Rightarrow_X q : \phi\, true}{\Gamma \Rightarrow_X \supset E(\phi, \psi, p, q) : \psi\, true}$$

$$\forall I \qquad \frac{\Gamma \Rightarrow_{X,x} p : \phi\, true}{\Gamma \Rightarrow_X \forall I((x)\phi, (x)p) : \forall x.\phi\, true}$$

$$\forall E \qquad \frac{\Gamma \Rightarrow_X p : \forall x.\phi\, true}{\Gamma \Rightarrow_X \forall E((x)\phi, t, p) : \phi[t/x]\, true}$$

$$\exists I \qquad \frac{\Gamma \Rightarrow_X p : \phi[t/x]\, true}{\Gamma \Rightarrow_X \exists E((x)\phi\, true, t, p) : \exists x.\phi\, true}$$

$$\exists E \qquad \frac{\Gamma \Rightarrow_X q : \exists x.\phi\, true \quad \Gamma, p : \phi\, true \Rightarrow_{X,x} r : \psi\, true}{\Gamma \Rightarrow_X \exists E((x)\phi, \psi, q, (x, p)r) : \psi\, true}$$

Table 5.2: Proof system of first-order logic with proof expressions.

5.2.2 Natural encodings

We are now in a position to give a stronger correspondence between first-order logic and the representing type theory (ELF$^+$,Σ_{Fol}), by linking the valid proof expressions with ELF$^+$ terms inhabiting judgements. The definition of encoding is extended to that of a *complete* encoding which gives a sound interpretation of the consequence relation with proofs in the entailment relation; a complete encoding is *natural* when it provides an exact link between the consequence relations with proofs and the corresponding part of the ELF$^+$ entailment relation. These concepts are defined at the general level for an arbitrary logic with proof expressions since, although we do not have a general method for constructing proof expressions, we are able to characterise their behaviour.

A *logic with proof expressions* consists of syntax extended to incorporate a class of proof expressions constructed from a countably infinite set of proof variables. These proof expressions have the usual notions of substitution and α-conversion applied to both logic and proof variables. The *consequence relation with proofs* is defined as a relation of the form $\Delta \vdash_X p : j$ where X is a set of variables of the logic, $\Delta = \{p_1{:}j_1, \ldots, p_m{:}j_m\}$ is a set of *proof assumptions*, with each j_i for $i \in \{1, \ldots n\}$ a basic judgement with free variables in X and the $p_1, \ldots, p_m$ distinct proof variables, j is a basic judgement with free variables in X and p is a proof expression with free variables in X and free proof variables in $\{p_1, \ldots, p_m\}$. The proof expression p is said to be *valid*. As in the definition of the standard consequence relation (definition 3.2.11), we impose certain conditions on the consequence relation with proofs to ensure its compatibility with the entailment relation of ELF$^+$. An *intuitionistic consequence relation with proofs* satisfies the weakening condition and is closed under substitution of variables and proof variables (properties listed above for first-order logic).

Notation Let Σ_{Log} be the specification of a logic with proofs in ELF$^+$. We distinguish the following sets of ELF$^+$ terms:

$$proof_\Gamma = \{p\text{: for some } j \in judge_\Gamma,\ \Gamma \vdash_{\Sigma_{Log}} p : j\}$$
$$proof_\Gamma^{\beta\eta} = \{p\text{: } p \in proof_\Gamma \text{ and } p \text{ is in } \beta\eta\text{-long normal form w.r.t. } (\Sigma_{Log}; \Gamma)\}.$$

We also use P_{LOG} to denote the set of proof expressions in the logic with subset $P_{LOG}(X, \Delta)$ containing those proof expressions with free variables in X and free

proof variables in Δ, for finite sequences of variables X and proof assumptions Δ. Let $VP_{LOG}(X, \Delta)$ denote the subset $P_{LOG}(X, \Delta)$ consisting of valid proof expressions. We omit the subscript when the logic is apparent.

5.2.1 DEFINITION Let LOG be a logic specified in ELF^+ by Σ_{Log}. A *complete* encoding of LOG in (ELF^+, Σ_{Log}) is a quadruple $(\eta, \xi, \delta, \chi)$ such that (η, ξ, δ) is an encoding and, for each finite sequence of variables $X = \langle x_1^{\sigma_1}, \ldots, x_n^{\sigma_n} \rangle$ and of proof assumptions $\Delta = \langle p_1{:}j_1, \ldots, p_m{:}j_m \rangle$, and some given standard bijection $h : Var^{Proof} \to Var^{Judge}$, the function $\chi_{X;\Delta} : P(X, \Delta) \to \mathcal{T}$ satisfies:

1. $\chi_{X;\Delta}(p) = h(p)$ for p declared in Δ;

2. for basic judgement j of the logic and proof expression p,

$$\{p_1{:}j_1, \ldots, p_m{:}j_m\} \vdash^{Log}_{\{x_1,\ldots,x_n\}} p : j \text{ implies } \Gamma_X, \Gamma_\Delta \vdash^{ELF^+}_{\Sigma_{Log}} \chi_{X;\Delta}(p) : \delta_X(j),$$

   where context Γ_X is $\langle \xi_X(x_1) : \eta(\sigma_1), \ldots, \xi_X(x_n) : \eta(\sigma_n) \rangle$ and precontext Γ_Δ is $\langle \chi_{X;\Delta}(p_1) : \delta_X(j_1), \ldots \chi_{X;\Delta}(p_m) : \delta_X(j_m) \rangle$;

3. the $\chi_{X;\Delta}$ are *compositional*; that is, for proof expressions $\Pi \in P(Y, \Theta)$ and $\Sigma_1, \ldots, \Sigma_m \in P(X, \Delta)$ and term expressions $t_1, \ldots, t_n \in T(X)$, we have

$$\chi_{X;\Delta}(\Pi[\overline{t}, \overline{\Sigma}/\overline{x}, \overline{p}]) = \chi_{Y;\Theta}(\Pi)[\overline{\xi_X(t)}, \overline{\chi_{X;\Delta}(\Sigma)}/\overline{\xi_Y(x)}, \overline{\chi_{Y;\Theta}(p)}].$$

Notation Let $(\eta, \xi, \delta, \chi)$ be a complete encoding and let $X = \langle x_1^{\sigma_1}, \ldots, x_n^{\sigma_n} \rangle$ be a finite sequence of variables and $\Delta = \langle p_1{:}j_1, \ldots, p_m{:}j_m \rangle$ a finite sequence of proof assumptions. We let Γ_X, Γ_Δ denote the context

$$\langle \xi_X(x_1) : \eta(\sigma_1), \ldots, \xi_X(x_n) : \eta(\sigma_n), \chi_{X;\Delta}(p_1) : \delta_X(j_1), \ldots, \chi_{X;\Delta}(p_m) : \delta_X(j_m) \rangle$$

in $\beta\eta$-long normal form where, by definition, Γ_X is a context of sorts and Γ_Δ is a precontext of judgements. In (ELF^+, Σ_{Log}), the valid proof expressions are represented by terms inhabiting judgements; the proof expressions as a whole are not represented. We therefore restrict $\chi_{X;\Delta}$ to the valid proof expressions: the function $\chi_{X;\Delta} : VP(X, \Delta) \to proof^{\beta\eta}_{\Gamma_X, \Gamma_\Delta}$ is the function extensionally equal to $\chi_{X;\Delta}$ restricted to the domain $VP(X, \Delta)$.

5.2.2 PROPOSITION Let $(\eta, \xi, \delta, \chi)$ be a complete encoding and let X, Y be finite sequences of variables of the logic and Δ, Θ be finite sequences of proof assumptions. If p is in $P(X, \Delta)$ and $P(Y, \Theta)$ then $\chi_{X;\Delta}(p) = \chi_{Y;\Theta}(p)$.

Proof By the compositional condition given in part 3. □

5.2.3 DEFINITION A complete encoding $(\eta, \xi, \delta, \chi)$ is *natural* if

1. (η, ξ, δ) is an adequate encoding;
2. for finite sequences of variables X and proof assumptions Δ, the function $\chi_{X;\Delta} : VP(X, \Delta) \to proof^{\beta\eta}_{\Gamma_X, \Gamma_\Delta}$ is a bijection;
3. the interpretation is *complete*: that is, for sequences $X = \langle x_1^{\sigma_1}, \ldots, x_n^{\sigma_n} \rangle$ and $\langle p_1{:}j_1, \ldots, p_m{:}j_m \rangle$ of logic variables and proof assumptions respectively,

$$\Gamma_X, \Gamma_\Delta \vdash_{\Sigma_{Log}} \chi_{X;\Delta}(p) : \delta_X(j) \text{ implies } \{p_1{:}j_1, \ldots, p_m{:}j_m\} \vdash^{Log}_{\{x_1^{\sigma_1}, \ldots, x_n^{\sigma_n}\}} p : j.$$

5.2.4 DEFINITION Let LOG be a logic specified in ELF$^+$ by Σ_{Log}. We say that the representation is *natural* if there is a natural encoding from LOG to (ELF^+, Σ_{Log}).

5.2.5 THEOREM Logics whose representations in ELF$^+$ are natural have intuitionistic consequence relations with proofs.

Proof This follows in a similar fashion to the proof of theorem 5.1.7 using the generalised substitution lemma (lemma 2.1.11) and the thinning lemma (lemma 2.1.12). □

5.2.6 EXAMPLE Natural deduction-style S_4 [Pra65] (example 3.2.15) does not have a natural representation in ELF$^+$ since, although its consequence relation is intuitionistic, its consequence relation with proofs is not preserved under substitution of proof expressions. This is explained (without resorting to the technical detail of proof expressions) using the following two derivations:

$$\frac{\Box\phi \Rightarrow_X \Box\phi}{\Box\phi \Rightarrow_X \Box\Box\phi}\ Nec$$

$$\frac{\phi \supset \Box\phi, \phi \Rightarrow_X \phi \supset \Box\phi \qquad \phi \supset \Box\phi, \phi \Rightarrow_X \phi}{\phi \supset \Box\phi, \phi \Rightarrow_X \Box\phi}\ MP$$

Substituting the second derivation for the premise of the first, we obtain

$$\frac{\dfrac{\phi \supset \Box\phi, \phi \Rightarrow_X \phi \supset \Box\phi \qquad \phi \supset \Box\phi, \phi \Rightarrow_X \phi}{\phi \supset \Box\phi, \phi \Rightarrow_X \Box\phi}\ MP}{\phi \supset \Box\phi, \phi \Rightarrow_X \Box\Box\phi}\ ?$$

which is not a derivation since the last line is not an instance of the $\Box$I-rule.

5.2.7 THEOREM The signature Σ_{Fol} provides a natural encoding of first-order logic in ELF^+.

Proof In the proof of theorem 5.1.3, we provide a triple (η, ξ, δ) which is an adequate encoding of first-order logic in ELF^+. We extend it to a natural encoding by defining, for each sequence of variables X and proof assumptions Δ, the function $\chi_{X;\Delta} : P(X, \Delta) \to T$, as follows:

$$\chi_{X;\Delta}(p) = p'' \qquad p \in \Delta$$

$$\chi_{X;\Delta}(\supset I(\phi)(\psi)((p)q)) = \supset I(\gamma_X(\phi))(\gamma_X(\psi))(\lambda p'':\delta_X(\phi\ true).\chi_{X;\Delta}(p))$$

$$\chi_{X;\Delta}(\supset E(\phi)(\psi)(p)(q)) = \supset E(\gamma_X(\phi))(\gamma_X(\psi))(\chi_{X;\Delta}(p))(\chi_{X;\Delta}(q))$$

$$\chi_{X;\Delta}(\forall I((x)\phi)((x)p) = \forall I(\lambda x':\iota.\gamma_{X,x}(\phi))(\lambda x':\iota.\chi_{X,x;\Delta}(p))$$

$$\chi_{X;\Delta}(\forall E((x)\phi)(t)(p)) = \forall E(\lambda x':\iota.\gamma_{X,x}(\phi)))(\xi_X(t))(\chi_{X;\Delta}(p))$$

$$\chi_{X;\Delta}(\exists I((x)\phi)(t)(p)) = \exists I(\lambda x':\iota.\gamma_{X,x}(\phi))(\xi_X(t))(\chi_{X;\Delta}(p))$$

$$\chi_{X;\Delta}(\exists E((x)\phi)(\psi)(p)((x,r)(q)))) =$$
$$\exists E(\lambda x':\iota.\gamma_{X,x}(\phi))(\gamma(\psi))(\chi_{X;\Delta}(p))(\lambda x':\iota\lambda r'':\delta_{X,x}(\phi\, true).\chi_{X,x;\Delta,r}(q))$$

where bijection $(\_)' : Var^{Log} \to Var^{Sort}$ and the functions $\gamma_X : F(X) \to o^{\beta\eta}_{\Gamma_X}$, for each sequence of variables X, are defined in the proof of theorem 5.1.3 and $(\_)'' : Var^{Proof} \to Var^{Judge}$ is a bijection. The functions $\chi_{X;\Delta}$ satisfy the conditions required to make $(\eta, \xi, \delta, \chi)$ a natural encoding. The details are left for the reader. $\Box$

5.2.8 EXAMPLE [Higher-order logic] By proceeding analogously to the first-order case, one can give a language of proof expressions for higher-order logic and a formal system for deriving valid proof expressions to provide a complete encoding of higher-order logic in ELF^+ which is natural.

5.2.9 THEOREM The signature Σ_{Hol} provides a natural encoding of higher-order logic in ELF^+.

Proof Extend the adequate encoding given in the proof of theorem 5.1.11. $\Box$

5.2.10 EXAMPLE [Hilbert-style S_4] We show that the representation of Hilbert-style S_4 given in example 5.1.12 is not natural. We extend the logic to incorporate

proof expressions in a similar way to the extension of first-order logic given in section 5.2.1. The proof expressions are defined using the following grammar:

$$\pi ::= p \mid A1(\phi)(\psi) \mid A2(\phi)(\psi)(\theta) \mid A3(\phi) \mid A4(\phi)(\psi) \mid A5(\phi) \mid MP(\phi)(\psi)(\pi)(\pi') \mid Nec(\phi)(\pi)$$

and the proof system adapted accordingly.

5.2.11 THEOREM The representation of Hilbert-style S_4 in ELF^+, given in example 5.1.12, is not natural.

Proof (sketch) Assume to the contrary that there is a natural encoding $(\eta, \xi, \delta, \chi)$ of Hilbert-style S_4 in (ELF^+, Σ_{Mod}); that is, there are bijections $\chi_{X;\Delta} : VP(X, \Delta) \rightarrow proof^{\beta\eta}_{\Gamma_X,\Gamma_\Delta}$ for each set of logic variables X and proof variables Δ. By analysing the possible expressions in $VP(X, \Delta)$ and terms in $proof^{\beta\eta}_{\Gamma_X,\Gamma_\Delta}$, we observe that there are eight possible shapes for the valid proof expressions and nine possible shapes for the inhabitants of judgements. Since $\chi_{X;\Delta}$ satisfies the compositionality property of definition 5.2.1, we know that $\chi_{X;\Delta}$ cannot be a bijection. □

5.2.12 EXAMPLE It is well-known that that natural deduction-style propositional logic (example 3.2.14) and Hilbert-style propositional logic (example 3.2.16) have the same basic consequence relation. The signature Σ_{Nat}(the obvious fragment of Σ_{Fol} (example 4.2.2)) thus provides an adequate encoding of Hilbert-style propositional logic in ELF^+, although it is not natural as one would expect.

5.2.13 THEOREM The signature Σ_{Nat} does not provide a natural encoding of Hilbert-style propositional logic.

Proof Similar analysis to the proof of theorem 5.2.11. □

Remark It is well-known that there is a strong link between the proofs of Hilbert-style and natural deduction-style propositional logic. The rules for the Hilbert-style logic are derived rules in the natural deduction presentation. The deduction theorem shows that the $\supset I$-rule in the natural deduction system is not derivable in Hilbert-style propositional logic. Instead, Schönfinkel's abstraction algorithm [HS88] provides a mapping from derivations in the natural deduction propositional logic to the derivations in the Hilbert-style presentation. Harper and Pfenning using the system Elf [Pfe91], a logic programming implementation

of ELF, study tactics for interpreting one logic in another via their representations in ELF; in particular, they utilise Schönfinkel's algorithm to provide an interpretation of natural deduction-style propositional logic in Hilbert-style propositional logic.

Remark Our approach for reasoning about representations of derivations, based on the analysis in [HHP89], is not completely satisfactory. Although the natural encoding definition (definition 5.2.3) is general for logics with consequence relations with proofs, we do not have a methodology for giving an explicit account of proofs. An alternative approach is to study the notion of a consequence relation of sequents as discussed in the chapter on future work (chapter 7).

Chapter 6

Encodings Expressed as Indexed Functors

We have emphasised the similarity in structure between the logics of interest and their representing type theories, due to the fact that the behaviour of the variables and the consequence relation of the logic is determined by the properties of the ELF^+ entailment relation (theorems 5.1.7 and 5.2.5). Encodings preserve this common structure. This motivates an algebraic presentation of the logics and their representing type theories as strict indexed categories [PS78] (or split fibrations [Ben85]) so that encodings become indexed functors between them: that is, structure preserving maps rather than functions satisfying a list of syntactic conditions. More specifically, a logic with an intuitionistic consequence relation provides a (strict) indexed category, whose base gives the term expressions and whose fibres are defined by the consequence relation. This approach uses ideas from the area of categorical logic (initiated by Lawvere [Law70]), but generalised to a wide class of logics. Using ELF^+ we are also able to view the representing type theory as a (strict) indexed category with the sorts providing the base category and the judgements the fibres. Adequate encodings then correspond to certain indexed isomorphisms. By adapting both indexed categories to incorporate the extra information regarding the proofs and inhabitants of judgements, natural encodings also yield isomorphisms between indexed categories.

The results in chapters 2 and 3 allow for a smooth transition from the syntactic definitions of the previous chapter to the algebraic presentations given here.

6.1 Indexing of categories

The usual categorical structures for presenting logics and dependent type theories (see the section, 'Related research', in the introduction) employ a categorical notion of indexing. In order to understand the indexing of categories, it is instructive to look at the indexing of sets, which can be described in two ways. The first is as a set of indexed categories, $\{X_i\}_{i\in I}$, also written as $X : I \to \mathbf{Set}$ where **Set** denotes the class of all sets. The second is as a map $f : Y \to I$ where I is again the indexing set; the indexed sets are then given by the fibres $f^{-1}(\{i\})$. The categorical counterpart of the first approach is given by indexed categories [PS78], and that of the second by fibrations [Ben85]. There are obvious translations between the two approaches for sets (the analogous categorical concept is called the 'Grothendieck construction'). It is sometimes preferable, for technical reasons, to use the fibration approach since, extending the set analogy, the map f is defined in set-theoretical terms whereas the map X is not; its range is the class of all sets. We choose the indexed category approach since, for our purposes, it is more natural to present a logic by first considering the syntax, which provides the indexing, and then the consequence relation. We concentrate on the definitions necessary for this chapter; an introduction to category theory is given by MacLane [Mac88] and a clear exposition of fibrations and (strict) indexed categories is found in [BW90] and [Jac91].

6.1.1 DEFINITION Let C be a category. A *strict indexed category* is a functor $F : C^{op} \to Cat$ where Cat is the category of small categories. The category C is the *base* category and, for $c \in obj(C)$, the *fibre* over c is the category $F(c)$.

Remark A more general notion of indexed category as a *pseudo-functor* $F: C^{op} \to Cat$ requires isomorphisms $F(id_C) \simeq id_{Cat}$ and $F(u \circ v) \simeq F(u) \circ F(v)$ with certain coherent conditions (see Paré and Schumacher [PS78]). For our purposes, the indexed categories are always strict and so, in future, whenever we refer to indexed categories we assume that they are strict.

6.1.2 DEFINITION Let $F : A^{op} \to Cat$ and $G : B^{op} \to Cat$ be indexed categories. An *indexed functor* from F to G is a pair (σ_{base}, σ) consisting of a functor

$\sigma_{base} : A \to B$ (called the *base* functor) and a natural transformation $\sigma : F \to G \circ \sigma_{base}^{op}$.

Certain properties of functors and natural transformations lead to conditions on indexed functors which are used in the analysis of encodings. An *isomorphism* $F : C \to B$ of categories is a functor F from C to B which is a bijection on objects and arrows. An equivalent definition is that there exists a functor $G : B \to C$ such that $F \circ G = id_B$ and $G \circ F = id_C$. Given functors $F, G : A \to B$, a *natural isomorphism* $\alpha : F \to G$ is a natural transformation in which every component α_a, for $a \in obj(A)$, is an isomorphism. The inverses of the α_a are the components of a natural isomorphism $\alpha^{-1} : G \to F$, which we call the *inverse natural transformation* for α. We write $F \simeq G$ when such a natural isomorphism exists.

6.1.3 DEFINITION An *indexed isomorphism* is an indexed functor whose base functor is an isomorphism and whose natural transformation is a natural isomorphism.

6.2 Logics as indexed categories

We provide a methodology for presenting logics with intuitionistic consequence relations as indexed categories, where the term expressions provide the base category and the consequence relation the fibres. Theorem 5.1.7 states that logics with adequate representations in ELF^+ must have intuitionistic consequence relations; concentrating on logics with such consequence relations is therefore not a restriction. This approach is based on the algebraic characterisation of particular logics and their models (initiated by Lawvere [Law70]), but generalised to a wide class of logics. It concentrates on the abstract view of logics as consequence relations (due to Tarski [Tar56]) and provides the rudiments of an algebraic framework for such logics.

For a logic with an intuitionistic consequence relation, the base category is determined by the term expressions.

6.2.1 PROPOSITION Let LOG denote an arbitrary logic with an intuitionistic consequence relation. Then the following defines a category:

objects finite sequences of distinct logic variables;

morphisms finite tuples of term expressions $(t_1, \ldots, t_n) : X \to Y = \langle y_1, \ldots, y_n \rangle$ such that, for each $i \in \{1, \ldots, n\}$, the t_i and y_i inhabit the same syntactic class;

composition if $(t_1, \ldots, t_n) : X \to Y = \langle y_1, \ldots, y_n \rangle$ and $(s_1, \ldots, s_m) : Y \to Z$ then $(s_1, \ldots, s_m) \circ (t_1, \ldots, t_n)$ is $(s_1[\overline{t}/\overline{y}], \ldots s_m[\overline{t}/\overline{y}]) : X \to Z$;

identity $(x_1, \ldots, x_n) : X \to X = \langle x_1, \ldots, x_n \rangle$.

Proof Use proposition 3.1.14 and the substitution results (proposition 3.1.15) to show that A is a category. □

6.2.2 DEFINITION Let LOG denote a logic with an intuitionistic consequence relation. The category defined in proposition 6.2.1 is the *term category* for LOG.

Remark An analysis of the structure induced by the expression symbols is not given as this structure is not present in the type theory (typically we represent these symbols by constants in the signature). This analysis should be possible using the information provided by the accompanying arities (section 3.1).

Remark We have chosen to define the objects of the base category as sequences of variables to give an easy correspondence with the contexts of sorts. An alternative is to use *sets* of variables and either define an equivalence on the context of sorts using the permutation lemma or work with some standard enumeration of logic variables to determine the corresponding context. Both approaches introduce extra complication when studying encodings as indexed functors. For the same reason, we also present the assumptions as finite sequences of judgements.

6.2.3 PROPOSITION Let LOG denote a logic with an intuitionistic consequence relation. Then the following defines an indexed category $\mathcal{L} : A^{op} \to Cat$. The base category A is the term category for LOG and, for each $X = \langle x_1^{\sigma_1}, \ldots, x_n^{\sigma_n} \rangle$ in $obj(A)$, the fibre $\mathcal{L}(X)$ is:

objects finite sequences of judgements with free variables in X;

morphisms $\langle j_1,\ldots,j_m\rangle \to \langle k_1,\ldots,k_p\rangle$ whenever $\{j_1,\ldots,j_m\} \vdash^{LOG}_{\{x_1,\ldots,x_n\}} k_i$ for $i \in \{1,\ldots,p\}$.

For each morphism $(t_1,\ldots,t_n) : Y \to X = \langle x_1,\ldots,x_n\rangle$ in A, the functor $\mathcal{L}((t_1,\ldots,t_n)^{op} : X \to Y) = (\overline{t})^* : \mathcal{L}(X) \to \mathcal{L}(Y)$ is as follows:

$(\overline{t})^*(\langle j_1,\ldots,j_m\rangle) = \langle j_1[\overline{t}/\overline{x}],\ldots,j_m[\overline{t}/\overline{x}]\rangle$;
$(\overline{t})^*(\langle j_1,\ldots,j_m\rangle \to \langle k_1,\ldots,k_p\rangle) = \langle j_1[\overline{t}/\overline{x}],\ldots,j_m[\overline{t}/\overline{x}]\rangle \to \langle k_1[\overline{t}/\overline{x}],\ldots,k_p[\overline{t}/\overline{x}]\rangle$.

Proof The fibre $\mathcal{L}(X)$, for each $X \in obj(A)$, is a preorder since the logic has a consequence relation which is closed under cut (definition 3.2.12). By proposition 3.1.14, we know that $(t_1,\ldots,t_n)^*$ is well-defined. It is a functor from the consequence relation preserving substitution (definition 3.2.13) and the substitution results (proposition 3.1.15). The functor $\mathcal{L} : A^{op} \to Cat$ is therefore well-defined. It preserves identity and composition using the substitution results (proposition 3.1.15). □

Remark Notice that the proof of this proposition requires that the consequence relation preserves substitution (proposition 3.2.13) and satisfies the following cut condition:

$$\text{if } \{j_1,\ldots,j_n\} \vdash_X k_i \text{ for } i \in \{1,\ldots,m\} \text{ and } \{k_1,\ldots,k_m\} \vdash_X l \text{ then } \{j_1,\ldots,j_n\} \vdash_X l.$$

The weakening property for intuitionistic consequence relations and the fact that these relations are defined on sets is not fundamental to our presentation of logics as indexed categories. We concentrate on these logics since they must have this property to be adequately represented in ELF^+ (theorem 5.1.7).

6.2.4 DEFINITION Let LOG be a logic with an intuitionistic consequence relation. The indexed category defined in proposition 6.2.3 is the *indexed category determined by* LOG.

Remark The standard motivation for presenting logics categorically is to explain the structure of particular logics: that is, to give a categorical account of the behaviour of the connectives and quantifiers. It is well-known, for example, that the universal quantification of intuitionistic first-order logic is right adjoint to

substitution. This analysis concerns the specific properties of logics and, although interesting, is not of fundamental concern to us.

6.3 ELF^+ as an indexed category

We do not take the standard categorical approach to representing type theories. Our presentation is motivated by the use of the type theory as a framework for representing logics. It has already been emphasised that not all of the entailment relation of the representing type theory corresponds to the consequence relation of the underlying logic and, therefore, the whole theory is not presented as an indexed category. Instead, we take advantage of the separation of ELF^+ terms into sorts, types and judgements to provide an indexed category whose base category is given by the inhabitants of sorts and whose fibres are given by the judgements. We concentrate on ELF^+ terms in $\beta\eta$-long normal form since the definitions of adequate and natural encodings are given with respect to these forms. An alternative is to present the categorical structure up to $\beta\eta$-equivalence.

The base category of the indexed category determined by $(\mathrm{ELF}^+, \Sigma_{Log})$ is defined using the inhabitants of sorts.

6.3.1 PROPOSITION Let $(\mathrm{ELF}^+, \Sigma_{Log})$ be the type theory representing a logic. The following defines a category B:

objects contexts of sorts in $\beta\eta$-long normal form;

morphisms $\Gamma_S \to \Delta_S = \langle x_1{:}A_1, \ldots, x_n{:}A_n\rangle$ are finite tuples $(t_1, \ldots, t_n)$ of ELF^+ terms, such that $\Gamma_S \vdash_{\Sigma_{LOG}} t_i : A_i[t_1, \ldots, t_{i-1}/x_1, \ldots, x_{i-1}]$ for $i \in \{1, \ldots, n\}$ and each t_i is in $\beta\eta$-long normal form with respect to $(\Sigma_{Log}; \Gamma_S)$;

composition for morphisms $(t_1, \ldots, t_n) : \Gamma_S \to \Delta_S = \langle x_1{:}A_1, \ldots, x_n{:}A_n\rangle$ and $(s_1, \ldots, s_m) : \Delta_S \to \Theta_S$, their composite $(s_1, \ldots, s_m) \circ (t_1, \ldots, t_n)$ is $(s_1[\overline{t}/\overline{x}], \ldots, s_m[\overline{t}/\overline{x}]) : \Gamma_S \to \Theta_S$;

identity $(x_1, \ldots, x_n) : \Delta_S \to \Delta_S = \langle x_1{:}A_1, \ldots, x_n{:}A_n\rangle$.

Proof To determine that composition is well-defined, we must show that $\Gamma \vdash_{\Sigma_{Log}} t_i : A_i[t_1, \ldots, t_{i-1}/x_1, \ldots, x_{i-1}]$, $i \in \{1, \ldots, n\}$, and $x_1{:}A_1, \ldots, x_n{:}A_n \vdash_{\Sigma_{Log}} \alpha$ implies $\Gamma \vdash_{\Sigma_{Log}} \alpha[\bar{t}/\bar{x}]$ where α is an ELF$^+$ assertion. By the thinning lemma (lemma 2.3.6) and renaming variables if necessary, $\Gamma, x_1{:}A_1, \ldots, x_n{:}A_n \vdash_{\Sigma_{Log}} \alpha$ and so, by the generalised substitution lemma (lemma 2.3.5) and, observing that substitution of sorts preserves $\beta\eta$-long normal forms, the result follows. Using the start lemma (lemma 2.3.3), the identity is well-defined. The definition of simultaneous substitution, together with the accompanying results (proposition 3.1.15), give the identity and associative laws. □

6.3.2 DEFINITION Let (ELF$^+$, Σ_{Log}) be the type theory representing a logic. The indexed category defined in proposition 6.3.1 is the *sort category* of (ELF$^+$, Σ_{Log}) .

Remark We have given no account of context extension: that is, given a context Γ and an entailment $\Gamma \vdash A : u$ for universe u containing variables, the extension is $\Gamma, x : A$. This analysis is not essential for our purposes and is omitted since, although there has been much research in this area (see [Jac91] and the references therein), there is no definitive account of context extension. It is also beyond the scope of this thesis to give a full explanation of the structure arising from Π-abstraction.

Using the sort category we define the indexed category determined by (ELF$^+$,Σ_{Log}) which presents the subtheory corresponding to the encoded logic.

6.3.3 PROPOSITION Let (ELF$^+$, Σ_{Log}) be the type theory representing an arbitrary logic. Then the following defines an indexed category $\mathcal{E} : B^{op} \to Cat$. The base category is the sort category of (ELF$^+$, Σ_{Log}), and, for each $\Gamma_S \in obj(B)$, the fibre $\mathcal{E}(\Gamma_S)$ is the preorder category given by:

objects finite sequences of judgements $J_1, \ldots, J_n$ with $J_i \in judge^{\beta\eta}_{\Gamma_S}$ for $i \in \{1, \ldots, n\}$;

morphisms $\langle J_1, \ldots, J_n \rangle \to \langle K_1, \ldots, K_m \rangle$ when $\Gamma_S, p_1{:}J_1, \ldots, p_n{:}J_n \vdash_{\Sigma_{Log}} \_ : K_j$ for $j \in \{1, \ldots, m\}$, where $\_ : K_j$ denotes the inhabitation of judgement K_j.

For each morphism $(t_1,\ldots,t_n) : \Delta_S \to \Gamma_S = \langle x_1{:}A_1,\ldots,x_n{:}A_n\rangle$ in B, the functor $\mathcal{E} : ((t_1,\ldots,t_n)^{op} : \Gamma_S \to \Delta_S) = (\overline{t})^* : \mathcal{E}(\Gamma_S) \to \mathcal{E}(\Delta_S)$ is given by

$(\overline{t})^*(\langle J_1,\ldots,J_m\rangle) = \langle J_1[\overline{t}/\overline{x}],\ldots,J_m[\overline{t}/\overline{x}]\rangle$;
$(\overline{t})^*(\langle J_1,\ldots,J_m\rangle \to \langle K_1,\ldots,K_l\rangle) = \langle J_1[\overline{t}/\overline{x}],\ldots,J_m[\overline{t}/\overline{x}]\rangle \to \langle K_1[\overline{t}/\overline{x}],\ldots,K_l[\overline{t}/\overline{x}]\rangle$

Proof The fibres are seen to be preorders using the start lemma (lemma 2.3.3), the generalised substitution lemma (lemma 2.3.5) and the substitution results (lemma 3.1.15). For each morphism $(t_1,\ldots,t_n) : \Delta_S \to \Gamma_S$, $(\overline{t})^*$ provides a well-defined functor using the generalised substitution lemma and the start lemma. Finally, $\mathcal{E}$ is a functor, again by the generalised substitution lemma. □

6.3.4 DEFINITION Let (ELF$^+$, Σ_{Log}) denote the type theory representing an arbitrary logic. The *indexed category determined by (ELF$^+$, Σ_{Log})* is the indexed category defined in proposition 6.3.3.

Remark Since adequate encodings focus on representing the consequence relations of logics, the fibres of the indexed category determined by the representing type theory are preorders: that is, we concentrate on inhabitation of judgements rather than the terms inhabiting judgements. To study complete encodings, these indexed categories must be adapted to incorporate the extra information given by the inhabiting terms (definition 6.5.4).

6.4 Adequate encodings give indexed isomorphisms

We are now in a position to show that the syntactic definitions given in chapter 5 correspond to simple categorical concepts; more specifically, encodings give rise to indexed functors such that adequate encodings correspond to indexed isomorphisms.

Notation Let (η,ξ,δ) be an encoding of a logic in ELF$^+$ with distinguished bijections $f^c : Var^c \to Var^{Sort}$ for each syntactic class c containing variables. Recall that when $X = \langle x_1^{\sigma_1},\ldots,x_n^{\sigma_n}\rangle$ is a finite sequence of variables of the logic we let Γ_X denote $\langle f^{\sigma_1}(x_1) : \eta(\sigma_1),\ldots,f^{\sigma_n}(x_n) : \eta(\sigma_n)\rangle$.

Also, if (η, ξ, δ) is an adequate encoding and $g^c : Var^{Sort} \to Var^c$ is the inverse of f^c for each c, then, for $\Gamma_S = \langle x_1{:}A_1, \ldots, x_n{:}A_n\rangle$ a context of sorts in $\beta\eta$-long normal form, we let X_{Γ_S} denote $\langle g^{\eta^{-1}(A_1)}(x_1)^{\eta^{-1}(A_n)}, \ldots, g^{\eta^{-1}(A_n)}(x_n)^{\eta^{-1}(A_n)}\rangle$, where η^{-1} is the inverse of $\eta : S \to sort^{\beta\eta}_{\langle\rangle}$.

6.4.1 THEOREM Let (η, ξ, δ) be an encoding of a logic LOG with an intuitionistic consequence relation in ELF$^+$, and let the indexed categories determined by LOG and (ELF$^+$, Σ_{Log}) be $\mathcal{L} : A^{op} \to Cat$ and $\mathcal{E} : B^{op} \to Cat$ respectively. We can define an indexed functor, denoted by $(e_{base}, e) : \mathcal{L} \to \mathcal{E}$, consisting of base functor $e_{base} : A \to B$ and natural transformation $e : \mathcal{L} \to \mathcal{E} \circ e_{base}$, where

$e_{base}(\langle x_1^{\sigma_1}, \ldots, x_n^{\sigma_n}\rangle) = \langle \xi_X(x_1) : \eta(\sigma_1), \ldots, \xi_X(x_n) : \eta(\sigma_n)\rangle$ for $X = \langle x_1^{\sigma_1}, \ldots, x_n^{\sigma_n}\rangle$;
$e_{base}((t_1, \ldots, t_n) : X \to Y) = (\xi_X(t_1), \ldots, \xi_X(t_n)) : e_{base}(X) \to e_{base}(Y)$,

and, for each $X \in obj(A)$,

$e_X(\langle J_1, \ldots, J_n\rangle) = \langle \delta_X(J_1), \ldots, \delta_X(J_n)\rangle$;
$e_X(\langle J_1, \ldots, J_n\rangle \to \langle K_1, \ldots, K_m\rangle) = \langle \delta_X(J_1), \ldots, \delta_X(J_n)\rangle \to \langle \delta_X(K_1), \ldots, \delta_X(K_m)\rangle$.

Proof e_{base} is well-defined since the encoding maps distinct logic variables to distinct sort variables and term expressions to inhabitants of sorts satisfying the properties stated in condition 2 of definition 5.1.4. Composition is preserved since ξ_X is compositional for each $X = \langle x_1^{\sigma_1}, \ldots, x_n^{\sigma_n}\rangle$ and the identity is obviously preserved. For each $X \in obj(A)$, we have a well-defined functor e_X since the encoding gives a sound interpretation of the consequence relation in the entailment relation and the identity and composition are obviously preserved because $\mathcal{L}(X)$ and $\mathcal{E} \circ e_{base}(X)$ are preorders. These functors give a natural transformation $e : \mathcal{L} \to \mathcal{E} \circ e_{base}$ since the functions $\delta_X : J(X) \to judge^{\beta\eta}_{\Gamma_X}$ are compositional. □

6.4.2 DEFINITION Let (η, ξ, δ) be an encoding of an arbitrary logic LOG in ELF$^+$. The *indexed functor determined by* (η, ξ, δ) is the indexed functor defined in theorem 6.4.1.

The converse of theorem 6.4.1 does not hold; that is, not all indexed functors give rise to encodings. For example, there is no guarantee that an indexed functor preserves the ordering, or even the length, of tuples. We believe that a more detailed analysis of the structure of these indexed categories (in particular, the

categorical interpretation of sequences and contexts) will yield a two-way correspondence. We are able to deduce, however, that the indexed functor determined by an encoding is an indexed isomorphism if and only if the encoding is adequate. This strong correspondence is feasible since we are dealing with a particular indexed functor, given by the encoding, which preserves the ordering and length of tuples. The preservation of structure gives rise to the following lemma, used to link adequacy with isomorphisms.

6.4.3 LEMMA Let (η, ξ, δ) be an encoding of LOG in ELF$^+$ by Σ_{Log} such that the indexed categories determined by LOG and (ELF$^+$, Σ_{Log}) are $\mathcal{L} : A^{op} \to Cat$ and $\mathcal{E} : B^{op} \to Cat$ respectively. Let the indexed functor $(e_{base}, e) : \mathcal{L} \to \mathcal{E}$ determined by the encoding be an indexed isomorphism with inverse $(f_{base}, f) : \mathcal{E} \to \mathcal{L}$.

1. Given

$$f_{base}((\overline{u}, t, \overline{v}) : \Gamma_S \to \Delta_S) = (\overline{w}, t', \overline{v}') : f_{base}(\Gamma_S) \to f_{base}(\Delta_S);$$
$$f_{base}((\overline{x}, t, \overline{y}) : \Gamma'_S \to \Delta'_S) = (\overline{x}', t'', \overline{y}') : f_{base}(\Gamma'_S) \to f_{base}(\Delta'_S),$$

   where $(\overline{u}, t, \overline{v})$ and $(\overline{x}, t, \overline{y})$ denote two arbitrary morphisms in B containing ELF$^+$ term t, we have

   (a) the lengths of Γ_S and $f_{base}(\Gamma_S)$, and of Δ_S and $f_{base}(\Delta_S)$ are the same;

   (b) the lengths of $\overline{u}$ and $\overline{u}'$ and of $\overline{v}$ and $\overline{v}'$ are the same;

   (c) $t' = t''$.

2. For each $\Gamma_S \in obj(B)$, given

$$f_{\Gamma_S}(\langle \overline{j}, J, \overline{k} \rangle) = \langle \overline{j}', J', \overline{k}' \rangle;$$
$$f_{\Gamma_S}(\langle \overline{p}, J, \overline{q} \rangle) = \langle \overline{p}', J'', \overline{q}' \rangle,$$

   where $\langle \overline{j}, J, \overline{k} \rangle$ and $\langle \overline{p}, J, \overline{q} \rangle$ denote two arbitrary objects of $\mathcal{E}(\Gamma_S)$ containing $J \in judge^{\beta\eta}_{\Gamma_S}$, we have:

   (a) the lengths of $\overline{j}$ and $\overline{j}'$ and of $\overline{k}$ and $\overline{k}'$ are the same;

   (b) $J' = J''$.

Proof By the definition of (e_{base}, e) we know that the functor e_{base} and, for all $X \in obj(A)$, the functors e_X preserve order and length of sequences and tuples.

This yields parts 1a, 1b and 2a. Parts 1c and 2b follow from (f_{base}, f) being inverse to (e_{base}, e). □

We are now in a position to show that adequate encodings correspond to indexed isomorphisms.

6.4.4 THEOREM Let (η, ξ, δ) be an encoding of an logic LOG with an intuitionistic consequence relation in ELF$^+$ and let $(e_{base}, e) : \mathcal{L} \to \mathcal{E}$ be the indexed functor determined by (η, ξ, δ), where $\mathcal{L} : A^{op} \to Cat$ and $\mathcal{E} : B^{op} \to Cat$. Then (η, ξ, δ) is adequate if and only if (e_{base}, e) is an indexed isomorphism.

Proof Since (η, ξ, δ) is an adequate encoding, we know that there are functions $\xi'_{\Gamma_S} : texp^{\beta\eta}_{\Gamma_S} \to T(X_{\Gamma_S})$ and $\delta'_{\Gamma_S} : judge^{\beta\eta}_{\Gamma_S} \to J(X_{\Gamma_S})$, for each context of sorts Γ_S in $\beta\eta$-long normal form, which are inverse to $\xi_{X_{\Gamma_S}}$ and $\delta_{X_{\Gamma_S}}$. We use these functors to provide an indexed functor $(f_{base}, f) : \mathcal{E} \to \mathcal{L}$ which is the inverse indexed functor of (e_{base}, e).

The base functor f_{base} is given as follows:

$$f_{base}(\langle x_1{:}A_1, \ldots, x_n{:}A_n\rangle) = \langle \xi'_{\Gamma_S}(x_1)^{\eta^{-1}(A_1)}, \ldots, \xi'_{\Gamma_S}(x_n)^{\eta^{-1}(A_n)}\rangle,$$
$$\text{where } \Gamma_S \text{ is } \langle x_1{:}A_1, \ldots, x_n{:}A_n\rangle;$$
$$f_{base}((t_1, \ldots, t_m) : \Gamma_S \to \Delta_S) = (\xi'_{\Gamma_S}(t_1), \ldots, \xi'_{\Gamma_S}(t_m)) : f_{base}(\Gamma_S) \to f_{base}(\Delta_S),$$

and, for each context of sorts Γ_S in $\beta\eta$-long normal form, the natural transformation $f : \mathcal{E} \to \mathcal{L} \circ f_{base}$ is defined, for each $\Gamma_S \in obj(B)$, by

$$f_{\Gamma_S}(\langle J_1, \ldots, J_m\rangle) = \langle \delta'_{\Gamma_S}(J_1), \ldots, \delta'_{\Gamma_S}(J_m)\rangle;$$
$$f_{\Gamma_S}(\langle J_1, \ldots, J_m\rangle \to \langle K_1, \ldots, K_p\rangle) = \langle \delta'_{\Gamma_S}(J_1), \ldots, \delta'_{\Gamma_S}(J_m)\rangle \to \langle \delta'_{\Gamma_S}(K_1), \ldots, \delta'_{\Gamma_S}(K_p)\rangle.$$

That (f_{base}, f) provides an indexed functor from $\mathcal{E}$ to $\mathcal{L}$ follows from the conditions satisfied by η^{-1}, ξ'_{Γ_S} and δ'_{Γ_S} for $\Gamma_S \in obj(B)$. In particular, we have that f_{base} preserves compositionality and that the f_{Γ_S}, for each $\Gamma_S \in obj(B)$, form the components of a natural transformation since ξ'_{Γ_S} and δ'_{Γ} preserve substitution (see proposition 5.1.6): that is, given $s \in sort^{\beta\eta}_{\Delta_S}$, $j \in judge^{\beta\eta}_{\Delta_S}$, where $\Delta_S = \langle x_1{:}A_1, \ldots, x_n{:}A_n\rangle$ is a context of sorts in $\beta\eta$-long normal form and $\Gamma_S \vdash t_i : A_i[t_1, \ldots, t_{i-1}/x_1, \ldots, x_{i-1}]$, we have

$$\xi'_{\Gamma_S}(s[\overline{t}/\overline{x}]) = \xi'_{\Delta_S}(s)[\overline{\xi'_{\Gamma_S}(t)}/\overline{\varepsilon'_{\Delta_S}(x)}];$$
$$\delta'_{\Gamma_S}(j[\overline{t}/\overline{x}]) = \delta'_{\Delta_S}(j)[\overline{\xi'_{\Gamma_S}(t)}/\overline{\varepsilon'_{\Delta_S}(x)}].$$

The indexed functor (f_{base}, f) is inverse to (e_{base}, e) since, for each $X \in obj(A)$, the functions η^{-1}, ξ'_{Γ_X} and δ'_{Γ_X} are inverse to η, ξ_X and δ_X respectively.

We now show that whenever (e_{base}, e) is an indexed isomorphism then (η, ξ, δ) is an adequate encoding. This relies on lemma 6.4.3. Let $(f_{base}, f) : \mathcal{E} \to \mathcal{L}$ be the inverse indexed functor of (e_{base}, e). Define $\eta' : sort^{\beta\eta}_{\langle\rangle} \to S$ and, for each $\Gamma_S \in obj(\Gamma_S)$, $\xi'_{\Gamma_S} : texp^{\beta\eta}_{\Gamma_S} \to T$ and $\delta'_{\Gamma_S} : judge^{\beta\eta}_{\Gamma_S} \to J$ as follows:

- $\eta'(A) = \sigma$ for each $A \in sort^{\beta\eta}_{\langle\rangle}$, where $f_{base}(\langle x : A\rangle) = \langle y^{\sigma}\rangle$;
- $\xi'_{\Gamma_S}(t) = t'$ for each $t \in sort^{\beta\eta}_{\Gamma_S}$, where $f_{base}(\langle x_1, \ldots, x_n, t\rangle : \Gamma_S \to \Gamma_S, x : A) = \langle y_1, \ldots, y_n, t'\rangle : f_{base}(\Gamma_S) \to f_{base}(\Gamma_S, x : A)$;
- $\delta'_{\Gamma_S}(j) = j'$ for each $j \in judge^{\beta\eta}_{\Gamma_S}$, where $f_{\Gamma_S}(\langle j\rangle) = \langle j'\rangle$.

The function η' is well-defined by lemma 6.4.3, the definition of (e_{base}, e) and the equality $e_{base} \circ f_{base} = id_B$. It is the inverse of η since f_{base} is the inverse functor of e_{base}. For each $\Gamma_S \in obj(B)$, the function $\xi'_{\Gamma_S} : texp^{\beta\eta}_{\Gamma_S} \to T$ is well-defined by lemma 6.4.3 and the fact that the objects are contexts in $\beta\eta$-long normal form. The equality $e_{base} \circ f_{base} = id_B$ implies, for each $\Gamma_S \in obj(B)$, that $\xi'_{\Gamma_S}(t) \subseteq T(X_{\Gamma_S})$ where, if Γ_S is $\langle x_1{:}A_1, \ldots, x_n{:}A_n\rangle$, then X_{Γ_S} is $\langle \xi'_{\Gamma_S}(x_1)^{\eta'(A_1)}, \ldots, \xi'_{\Gamma_S}(x_n)^{\eta'(A_n)}\rangle$; we write $\xi'_{\Gamma_S}(t) : texp^{\beta\eta}_{\Gamma_S} \to T(X_{\Gamma_S})$. The function $\xi'_{\Gamma_X} : texp^{\beta\eta}_{\Gamma_X} \to T(X)$ is inverse to $\xi_X : T(X) \to texp^{\beta\eta}_{\Gamma_X}$, for each $X \in obj(A)$, since f_{base} is inverse to e_{base}. We use a similar argument, this time appealing to the fact that functor $f_{\Gamma_X} : \mathcal{E}(\Gamma_X) \to \mathcal{L}(f_{base}(\Gamma_X))$ is inverse to functor $e_X : \mathcal{L}(X) \to \mathcal{E}(e_{base}(X))$, for all $X \in obj(A)$, to show that the function $\delta'_{\Gamma_X} : judge^{\beta\eta}_{\Gamma_X} \to J(X)$ is inverse to $\delta_X : J(X) \to judge^{\beta\eta}_{\Gamma_X}$. Finally, the result

$$\Gamma_S, p_1{:}j_1, \ldots, p_m{:}j_m \vdash^{ELF^+}_{\Sigma_{Log}} \_ : j : Judge \text{ implies}$$
$$\{\delta'_{\Gamma_S}(j_1), \ldots, \delta'_{\Gamma_S}(j_m)\} \vdash_{X_{\Gamma_S}} \delta'_{\Gamma_S}(j),$$

where if Γ_S is $\langle x_1{:}A_1, \ldots, x_n{:}A_n\rangle$ then X_{Γ_S} is $\langle \xi'_{\Gamma_S}(x_1)^{\eta'(A_1)}, \ldots, \xi'_{\Gamma_S}(x_n)^{\eta'(A_n)}\rangle$, follows from lemma 6.4.3 and the fact that (f_{base}, f) is an indexed functor. □

6.5 Natural encodings give indexed isomorphisms

The categorical presentation of adequate encodings does not require an explicit account of the derivations of a logic or the terms inhabiting ELF^+ judgements. We now include this information regarding proofs in order to express complete encodings as indexed functors, with naturality again corresponding to isomorphism.

The definition of the complete indexed category for LOG, which includes the proof information, has the term category of LOG for its base. Its fibres are defined using the proof expressions rather than just the consequence relation.

6.5.1 PROPOSITION Let LOG be an arbitrary logic with an intuitionistic consequence relation with proofs. An indexed category, denoted by $\mathcal{L}_P : A^{op} \to Cat$, has the term category for LOG (definition 6.2.2) as its base category and, for each $X \in obj(A)$, the fibre $\mathcal{L}_P(X)$ consisting of:

objects finite sequences of proof assumptions $\langle p_1{:}j_1, \ldots, p_n{:}j_n\rangle$;

morphisms finite tuples of proof expressions
$(\Pi_1, \ldots, \Pi_m) : \langle p_1{:}j_1, \ldots, p_n{:}j_n\rangle \to \langle q_1{:}k_1, \ldots, q_m{:}k_m\rangle$;

composition for $(\Pi_1, \ldots, \Pi_m) : \langle p_1{:}j_1, \ldots, p_n{:}j_n\rangle \to \langle q_1{:}k_1, \ldots, q_m{:}k_m\rangle$ and for $(\Sigma_1, \ldots, \Sigma_r) : \langle q_1{:}k_1, \ldots, q_m{:}k_m\rangle \to \langle s_1{:}l_1, \ldots, s_r{:}l_r\rangle$, the composition $(\Sigma_1, \ldots, \Sigma_r) \circ (\Pi_1, \ldots, \Pi_m)$ is
$(\Sigma_1[\overline{\Pi}/\overline{q}], \ldots, \Sigma_r[\overline{\Pi}/\overline{q}]) : \langle p_1{:}j_1, \ldots, p_n{:}j_n\rangle \to \langle s_1{:}l_1, \ldots, s_r{:}l_r\rangle$;

identity $(p_1, \ldots, p_n) : \langle p_1{:}j_1, \ldots, p_n{:}j_n\rangle \to \langle p_1{:}j_1, \ldots, p_n{:}j_n\rangle$.

For each morphism $(t_1, \ldots, t_n) : X \to Y = \langle y_1, \ldots, y_n\rangle$ in A, the functor $\mathcal{L}_P((t_1, \ldots, t_n)^{op} : Y \to X) = (\overline{t})^* : \mathcal{L}_P(Y) \to \mathcal{L}_P(X)$ is given by

$$(\overline{t})^*(\langle p_1{:}j_1, \ldots, p_m{:}j_m\rangle) = \langle p_1 : j_1[\overline{t}/\overline{y}], \ldots, p_m : j_m[\overline{t}/\overline{y}]\rangle;$$
$$(\overline{t})^*((\Pi_1, \ldots, \Pi_r) : \langle p_1{:}j_1, \ldots, p_m{:}j_m\rangle \to \langle q_1{:}k_1, \ldots, q_r{:}k_r\rangle) =$$
$$(\Pi_1[\overline{t}/\overline{y}], \ldots, \Pi_r[\overline{t}/\overline{y}]) : (\overline{t}^*)(\langle p_1{:}j_1, \ldots, p_m{:}j_m\rangle) \to (\overline{t}^*)(\langle q_1{:}k_1, \ldots, q_r{:}k_r\rangle).$$

Proof The proof follows from the substitution results (proposition 3.1.15) and the consequence relation being closed under substitution of logic and proof variables, as specified in section 5.2.2. □

6.5.2 DEFINITION Let LOG be an arbitrary logic with an intuitionistic consequence relation with proofs. The *complete indexed category for LOG* is that defined in proposition 6.5.1.

The inhabitants of the ELF$^+$ judgements are incorporated into the indexed category determined by (ELF$^+$, Σ_{Log}) in a similar fashion.

6.5.3 PROPOSITION Let (ELF$^+$, Σ_{Log}) be the type theory representing an arbitrary logic. We may define an indexed category, denoted by $\mathcal{E}_P : B^{op} \to Cat$, with the sort category for (ELF$^+$, Σ_{Log}) (definition 6.3.2) as its base category and, for each $\Gamma_S \in obj(B)$, the fibre $\mathcal{E}_P(\Gamma_S)$ consisting of

objects precontexts Γ_J, where Γ_J is $\langle p_1{:}J_1, \ldots, p_n{:}J_n\rangle$, such that $J_i \in judge^{\beta\eta}_{\Gamma_S}$ for $i \in \{1, \ldots, n\}$ and $\Gamma_S, p_1{:}J_1, \ldots, p_n{:}J_n$ is a context;

morphisms finite tuples of ELF$^+$ terms $(\Pi_1, \ldots, \Pi_m) : \Gamma_J \to \Delta_J = \langle q_1{:}K_1, \ldots, q_m{:}K_m\rangle$ such that $\Gamma_S, \Gamma_J \vdash_{\Sigma_{LOG}} \Pi_i : K_i$ for $i \in \{1, \ldots, m\}$;

composition for $(\Pi_1, \ldots, \Pi_m) : \Gamma_J \to \Delta_J$ and $(\Sigma_1, \ldots, \Sigma_l) : \Delta_J \to \Theta_J$, the composition is $(\Sigma_1[\overline{\Pi}/\overline{q}], \ldots, \Sigma_l[\overline{\Pi}/\overline{q}]) : \Gamma_J \to \Theta_J$;

identity $(p_1, \ldots, p_n) : \Gamma_J \to \Gamma_J = \langle p_1{:}J_1, \ldots, p_n{:}J_n\rangle$.

For morphism $(t_1, \ldots, t_n) : \Gamma_S \to \Delta_S = \langle y_1{:}A_1, \ldots, y_n{:}A_n\rangle$ in B, the functor $\mathcal{E}_P((t_1, \ldots, t_n)^{op} : \Gamma_S \leftarrow \Delta_S) = (\overline{t})^* : \mathcal{E}_P(\Delta_S) \to \mathcal{E}_P(\Gamma_S)$ is given by

$(\overline{t})^*(\langle p_1{:}J_1, \ldots, p_m{:}J_m\rangle) = \langle p_1 : J_1[\overline{t}/\overline{y}], \ldots, p_m : J_m[\overline{t}/\overline{y}]\rangle$;
$(\overline{t})^* : ((\Pi_1, \ldots, \Pi_l) : \Gamma_J \to \Delta_J) = (\Pi_1[\overline{t}/\overline{y}], \ldots, \Pi_l[\overline{t}/\overline{y}]) : \Gamma_J[\overline{t}/\overline{y}] \to \Delta_J[\overline{t}/\overline{y}]$.

Proof The proof that the $\mathcal{E}_P(\Gamma_S)$ are categories follows the same reasoning as in the proof of proposition 6.3.1. For each morphism $(t_1, \ldots, t_n)^*$, the $(\overline{t})^*$ provides a well-defined functor using the generalised substitution lemma (lemma 2.3.5) and the start lemma (lemma 2.3.3). Finally, $\mathcal{E}_P$ is a functor, again from the generalised substitution lemma. □

6.5.4 DEFINITION Let (ELF$^+$, Σ_{Log}) represent an arbitrary logic. The *complete indexed category for (ELF$^+$, Σ_{Log})* is that defined in proposition 6.5.3.

6.5.5 THEOREM Let $(\eta, \xi, \delta, \chi)$ be a complete encoding of a logic with an intuitionistic consequence relation with proofs represented in ELF^+, and let the complete indexed categories be $\mathcal{L}_P : A^{op} \to Cat$ and $\mathcal{E}_P : B^{op} \to Cat$ respectively. An indexed functor from $\mathcal{L}_P$ to $\mathcal{E}_P$ can be defined, with the base functor $e_{base} : A \to B$, defined in theorem 6.4.1, and natural transformation $ce : \mathcal{L}_P \to \mathcal{E}_P \circ e_{base}$ defined, for each $X \in obj(A)$, by

$$ce_X(\langle p_1{:}j_1, \ldots, p_m{:}j_m\rangle) = \langle \chi_{X;\Delta}(p_1) : \delta_X(j_1), \ldots, \chi_{X;\Delta}(p_m) : \delta_X(j_m)\rangle;$$
$$ce_X((\Pi_1, \ldots, \Pi_l) : \langle p_1{:}j_1, \ldots, p_m{:}j_m\rangle \to \langle q_1{:}k_1, \ldots, q_l{:}k_l\rangle)$$
$$= (\chi_{X;\Delta}(\Pi_1), \ldots, \chi_{X;\Delta}(\Pi_l)) : ce_X(\langle p_1{:}j_1, \ldots, p_m{:}j_m\rangle) \to ce_X(\langle q_1{:}k_1, \ldots, q_l{:}k_l\rangle),$$

where Δ is $\langle p_1{:}j_1, \ldots, p_m{:}j_m\rangle$.

Proof We have a natural transformation $ce : \mathcal{L}_P \to \mathcal{E}_P \circ e_{base}$ by the compositionality of δ_X and $\chi_{X;\Delta}$, for $X \in obj(A)$ and $\Delta \in obj(\mathcal{L}_P(X))$. □

Remark Just as in the encoding case, we believe that, with more analysis of the structure of these indexed categories, the converse may be proved. This analysis is beyond the scope of this thesis.

6.5.6 DEFINITION Let $(\eta, \xi, \delta, \chi)$ be a complete encoding of a logic with an intuitionistic consequence relation with proofs in ELF^+. The *indexed functor determined by* $(\eta, \xi, \delta, \chi)$ is that defined in theorem 6.5.5.

Remark By definition we know that a complete encoding is an encoding with an extra correspondence satisfying certain conditions. This is reflected in the categorical treatment by defining the indexed functors $(id_A, F^{\mathcal{L}}) : \mathcal{L}_P \to \mathcal{L}$ and $(id_B, F^{\mathcal{E}}) : \mathcal{E}_P \to \mathcal{E}$, where $F^{\mathcal{L}}$ and $F^{\mathcal{E}}$ are natural transformations whose components are the obvious forgetful functors losing the information regarding the proof expressions and the inhabitants of judgements respectively, to obtain the equality

$$(e_{base}, e) \circ (id_A, F^{\mathcal{L}}) = (id_B, F^{\mathcal{E}}) \circ (e_{base}, ce);$$

that is, $e_{base} \circ id_A = id_B \circ e_{base}$ and, for each $X \in obj(A)$, $e_X \circ F^{\mathcal{L}}_X = F^{\mathcal{E}}_{e_{base}(X)} \circ ce_X$.

6.5.7 THEOREM Let $(\eta, \xi, \delta, \chi)$ be an encoding of a logic with an intuitionistic consequence relation with proofs in ELF^+, and indexed functor $(e_{base}, e) : \mathcal{L}_P \to \mathcal{E}_P$

be the indexed functor determined by $(\eta, \xi, \delta, \chi)$, where $\mathcal{L}_P : A^{op} \rightarrow Cat$ and $\mathcal{E}_P : B^{op} \rightarrow Cat$. Then $(\eta, \xi, \delta, \chi)$ is natural if and only if (e_{base}, ce) is an indexed isomorphism.

Proof Adapt lemma 6.4.3 for $(f_{base}, cf) : \mathcal{E}_P \rightarrow \mathcal{L}_P$, the inverse indexed functor of $(e_{base}, ce) : \mathcal{L}_P \rightarrow \mathcal{E}_P$. The proof is similar to that of theorem 6.4.4. □

Chapter 7

Some Directions for Future Research

In this thesis, we have introduced and explored the new framework ELF$^+$. With this framework, we are now able to characterise what it means to represent a logic. In particular, we have concentrated on characterising the representation of two kinds of consequence relation given by the proof system of a logic: the usual consequence relation associated with natural deduction systems and the consequence relation with explicit reference to proofs. Throughout the earlier chapters we have identified specific areas for future research where appropriate. In this short concluding chapter, we mention some of the immediate questions arising from our work on ELF$^+$.

Consequence relations from derivations

In section 5.2.2, we define natural representations using the notion of consequence relation with proofs to give some criterion for identifying when the derivations of a logic have been well-represented in ELF$^+$. An alternative approach is to use a consequence relation of sequents, in the style of Avron [Avr89]. We envisage a relation of the form

$$\Gamma_1 \Rightarrow_{X_1} J_1, \ldots, \Gamma_n \Rightarrow_{X_n} J_n \vdash \Gamma \Rightarrow_X J,$$

where $\vdash$ captures the notion of derivation tree; thus the above relation expresses the fact that there is a derivation tree whose assumptions are $\Gamma_1 \Rightarrow_{X_1} J_1$, ..., $\Gamma_n \Rightarrow_{X_n} J_n$ and whose conclusion is $\Gamma \Rightarrow_X J$. Since we are now able to identify the part of the ELF$^+$ entailment relation which corresponds to the underlying

logic, it should be possible to identify those derivation trees in the type theory which correspond to the consequence relation described above. This would yield a better analysis of the representations of derivations in the framework.

All the notions of consequence relation considered in this thesis have been defined with respect to sets of assumptions and indexed by sets of variables, since the restriction to sets is required if the consequence relation is to be represented by the ELF^+ entailment relation. Viewing derivations in natural deduction systems as trees, it makes more sense to consider *multisets* of assumptions (as advocated by Avron [Avr91]): that is, collections of judgements where the ordering is not important but the number of occurrences is. This leads us to the concept of a 'linear' consequence relation defined using multisets of assumptions and indexed by sets of variables; these ideas are compatible with the consequence relations of sequents discussed above. We cannot represent a multiset of assumptions using a context in a standard type theory since we have unrestricted use of declared variables in this context. In order to give a 'true' representation of a linear consequence relation in a type theory, it is necessary to adapt the standard notion of context. Work in progress on a linear ELF [MPP92], transforming ideas from Girard's linear logic [Gir87], is relevant here and should lead to a framework in which more logics can be represented.

Schematic consequence relations

When using logics in practice, one usually works with derivations and the consequence relation at the schematic level. The study of schematic consequence relations arising from proof systems, an area also considered by Aczel [Acz91], is therefore worth exploring. The aim is to capture the notion of a derived rule and provide a methodology for representing rules of standard form in ELF^+. It may be more appropriate, when investigating the schematic nature of logics, to look at an alternative approach to frameworks where the standard variables and schematic variables of the logics are treated separately. This separation of variables is motivated by the observation that, at the schematic level, substitution is a common notion, whereas a common behaviour of variables of the logic is not so obvious: for example, the variables of first-order logic, Hoare logic [Apt81] and the

π-calculus [MPW89] have very different behaviours (see section 3.1). With this approach to frameworks, we would aim, for instance, to capture α-conversion at the logic level and substitution at the schematic level.

Algebraic frameworks

Our work on the algebraic formulation of logics and their representations in ELF^+ provides a link between syntactic and algebraic notions of frameworks. In chapter 6, we present logics and their representing type theories as (strict) indexed categories where the amount of information within these indexed categories depends on the particular consequence relation under investigation. We have shown that, for both types of consequence relation discussed in this thesis, encodings determine indexed functors such that 'correct' encodings give rise to indexed isomorphisms. We conjecture that further analysis of the general structure of these indexed categories, perhaps using multicategories [Lam89], should result in an exact correspondence between encodings and indexed functors, and hence a precise link between the syntactic and algebraic presentations. Our ideas thus form the beginnings of an algebraic framework for representing logics.

Appendix A

The Type Theory ELF

The original presentation of ELF [HHP89] is given in this appendix. In chapter 2, we present an equivalent type theory which is notationally more concise and easier to understand.

A countably infinite set of variables is given and two countably infinite sets of constants, disjoint from each other and from the variables: one for object-level constants, the other for family-level constants. The metavariables x, y, and z range over the variables, c and d range over the object-level constants, and a and b over the family-level constants. The abstract syntax of the terms of ELF is given by the following grammar:

Kinds	K	::=	$\mathsf{Type} \mid \Pi x{:}A.K$
Families	A	::=	$a \mid \Pi x{:}A.B \mid \lambda x{:}A.B \mid AM$
Objects	M	::=	$c \mid x \mid \lambda x{:}A.M \mid MN$

The abstract syntax for signatures and contexts is given by the grammar:

Signatures	Σ	::=	$\langle\rangle \mid \Sigma, a{:}K \mid \Sigma, c{:}A$
Contexts	Γ	::=	$\langle\rangle \mid \Gamma, x{:}A$

The ELF type theory is a formal system for deriving assertions of one of the following forms (the intended meaning is in brackets):

Σ sig	(Σ is a valid signature)
$\vdash_\Sigma \Gamma$	(Γ is a valid context in Σ)
$\Gamma \vdash_\Sigma K$	(K is a kind in Γ and Σ)
$\Gamma \vdash_\Sigma A : K$	(A has kind K in Γ and Σ)
$\Gamma \vdash_\Sigma M : A$	(M has type A in Γ and Σ)

Valid Signatures

(B-EMPTY-SIG) $$\frac{}{\langle\rangle \text{ sig}}$$

(B-KIND-SIG) $$\frac{\Sigma \text{ sig} \quad \vdash_\Sigma K \quad a \notin \text{dom}(\Sigma)}{\Sigma, a{:}K \text{ sig}}$$

(B-TYPE-SIG) $$\frac{\Sigma \text{ sig} \quad \vdash_\Sigma A : \text{Type} \quad c \notin \text{dom}(\Sigma)}{\Sigma, c{:}A \text{ sig}}$$

Valid Contexts

(B-EMPTY-CTX) $$\frac{\Sigma \text{ sig}}{\vdash_\Sigma \langle\rangle}$$

(B-TYPE-CTX) $$\frac{\vdash_\Sigma \Gamma \quad \Gamma \vdash_\Sigma A : \text{Type} \quad x \notin \text{dom}(\Gamma)}{\vdash_\Sigma \Gamma, x{:}A}$$

Valid Kinds

(B-TYPE-KIND) $$\frac{\vdash_\Sigma \Gamma}{\Gamma \vdash_\Sigma \text{Type}}$$

(B-PI-KIND) $$\frac{\Gamma, x{:}A \vdash_\Sigma K}{\Gamma \vdash_\Sigma \Pi x{:}A.K}$$

Table A.1: The LF Type System

We write $\Gamma \vdash_\Sigma \alpha$ for an arbitrary assertion of one of the forms $\Gamma \vdash_\Sigma K$, $\Gamma \vdash_\Sigma A : K$, or $\Gamma \vdash_\Sigma M : A$. The rules for deriving the formation assertions of the ELF type theory are given in Tables A.1 and A.2.

The inference rules of the ELF type theory make use of a definitional equality, consisting of the following three forms of assertion:

$\Gamma \vdash_\Sigma K \equiv K'$ (K and K' are definitionally equal kinds in Γ and Σ)
$\Gamma \vdash_\Sigma A \equiv A'$ (A and A' are definitionally equal families in Γ and Σ)
$\Gamma \vdash_\Sigma M \equiv M'$ (M and M' are definitionally equal objects in Γ and Σ)

The first two of these relations are used directly (rules B-CONV-FAM and B-CONV-OBJ); the third is used to define the others.

The definitional equality relation considered is β-conversion of the entities at all levels. Thus we define the definitional equality relation, $\equiv$, between entities of all three levels to be the symmetric and transitive closure of the *parallel nested*

Valid Families

(B-CONST-FAM) $$\frac{\vdash_\Sigma \Gamma \quad c{:}K \in \Sigma}{\Gamma \vdash_\Sigma c : K}$$

(B-PI-FAM) $$\frac{\Gamma, x{:}A \vdash_\Sigma B : \mathsf{Type}}{\Gamma \vdash_\Sigma \Pi x{:}A.B : \mathsf{Type}}$$

(B-ABS-FAM) $$\frac{\Gamma, x{:}A \vdash_\Sigma B : K}{\Gamma \vdash_\Sigma \lambda x{:}A.B : \Pi x{:}A.K}$$

(B-APP-FAM) $$\frac{\Gamma \vdash_\Sigma A : \Pi x{:}B.K \quad \Gamma \vdash_\Sigma M : B}{\Gamma \vdash_\Sigma AM : [M/x]K}$$

(B-CONV-FAM) $$\frac{\Gamma \vdash_\Sigma A : K \quad \Gamma \vdash_\Sigma K' \quad \Gamma \vdash_\Sigma K \equiv K'}{\Gamma \vdash_\Sigma A : K'}$$

Valid Objects

(B-CONST-OBJ) $$\frac{\vdash_\Sigma \Gamma \quad c{:}A \in \Sigma}{\Gamma \vdash_\Sigma c : A}$$

(B-VAR-OBJ) $$\frac{\vdash_\Sigma \Gamma \quad x{:}A \in \Gamma}{\Gamma \vdash_\Sigma x : A}$$

(B-ABS-OBJ) $$\frac{\Gamma, x{:}A \vdash_\Sigma M : B}{\Gamma \vdash_\Sigma \lambda x{:}A.M : \Pi x{:}A.B}$$

(B-APP-OBJ) $$\frac{\Gamma \vdash_\Sigma M : \Pi x{:}A.B \quad \Gamma \vdash_\Sigma N : A}{\Gamma \vdash_\Sigma MN : [N/x]B}$$

(B-CONV-OBJ) $$\frac{\Gamma \vdash_\Sigma M : A \quad \Gamma \vdash_\Sigma A' : \mathsf{Type} \quad \Gamma \vdash_\Sigma A \equiv A'}{\Gamma \vdash_\Sigma M : A'}$$

Table A.2: The LF Type System (continued)

(R-REFL) $$\frac{}{M \to M}$$

(R-BETA-OBJ) $$\frac{M \to M' \quad N \to N'}{(\lambda x{:}A.M)N \to [N'/x]M'}$$

(R-BETA-FAM) $$\frac{B \to B' \quad N \to N'}{(\lambda x{:}A.B)N \to [N'/x]B'}$$

(R-APP-OBJ) $$\frac{M \to M' \quad N \to N'}{MN \to M'N'}$$

(R-APP-FAM) $$\frac{A \to A' \quad M \to M'}{AM \to A'M'}$$

(R-ABS-OBJ) $$\frac{A \to A' \quad M \to M'}{\lambda x{:}A.M \to \lambda x{:}A'.M'}$$

(R-ABS-FAM) $$\frac{A \to A' \quad B \to B'}{\lambda x{:}A.B \to \lambda x{:}A'.B'}$$

(R-PI-FAM) $$\frac{A \to A' \quad B \to B'}{\Pi x{:}A.B \to \Pi x{:}A'.B'}$$

(R-PI-KIND) $$\frac{A \to A' \quad K \to K'}{\Pi x{:}A.K \to \Pi x{:}A'.K'}$$

Table A.3: Parallel Reduction

reduction relation, $\rightarrow$, defined by the rules of Table A.3. The transitive closure of parallel reduction is denoted by $\rightarrow^*$.

Bibliography

[Acz80] P. Aczel. Frege Structures and the Notions of Proposition, Truth and Set, *The Kleene Symposium*, ed.s Barwise, Keisler and Kunen, North-Holland, pp 31-59.

[Acz90] P. Aczel. Replacement Systems and the Axiomatisation of Situation Theory, *Situation Theory and its Applications*, CSL1 Lecture Notes No. 22, Stanford University.

[Acz91] P. Aczel. *The Notion of a Logic*, draft.

[ACM90] P. Aczel, D.P. Carlisle and N. Mendler. Two frameworks of theories and their implementation in Isabelle, in [HP91], pp 3-40.

[Amb91] S. Ambler. *First-order Linear Logic in Semi-monoidal Closed Categories*, Ph.D. thesis, Edinburgh University.

[And71] P.B. Andrews. Resolution in Type Theory. *Journal of Symbolic Logic*, Vol. 36, pp 414-432.

[Apt81] K.R. Apt. Ten Years of Hoare's Logic: A Survey—*A.C.M. Transactions on Programming Languages and Systems*, Part 1, Vol. 3, No. 4, pp 431-483.

[ACN90] L. Augustsson, T. Coquand and B. Nordström. A short description of Another Logical Framework, *Informal Proceedings of the First Workshop on Logical Frameworks*, Antibes, ed.s G. Huet and G. Plotkin.

[Avr86] A. Avron. *Internalising S_4 and S_5 in the LF*, preprint.

[Avr89] A. Avron. *Hypersequents and logical consequences*, Technical report 140/89, Institute of Computer Sciences, University of Tel Aviv.

[Avr91] A. Avron. Simple Consequence Relations, *Information and Computation*, Part 1, Vol. 91, pp 105–139.

[AHM89] A. Avron, F.A. Honsell and I.A.Mason. *Using Typed Lambda Calculus to Implement Formal Systems on a Machine*, LFCS report series ECS-LFCS-87-31, Edinburgh University.

[AHMP87] A. Avron, F.A. Honsell, I.A.Mason and R. Pollack. *Using typed lambda calculus to implement formal systems on a machine*, LFCS report series ECS-LFCS-87-31, Edinburgh University.

[Bar84] H. Barendregt. *The lambda calculus, its syntax and semantics*, 2nd revised ed., North-Holland, Amsterdam .

[Bar90] H. Barendregt. Lambda Calculi with Types, to appear in: *Handbook of Logic in Computer Science*, OUP.

[Bar91] H. Barendregt et al. *Lambda Calculus*, Summer school, University of Nijmegen, The Netherlands.

[BW90] M. Barr and C. Wells, *Category Theory for Computing Science*, Prentice Hall.

[BF85] J. Barwise and S. Feferman. *Model-theoretic Logics*, Perspectives in Mathematical Logic, Spinger-Verlag, New York.

[Ben85] J. Bénabou. Fibred Categories and the Foundations of Naive Category Theory, *Journal of Symbolic Logic*, Vol. 50, pp 10-37.

[Ber90] S. Beradi. *Type dependence and constructive mathematics*, Ph.D. thesis, Mathematical Institute, Torino, Italy.

[dB70] N. G. de Bruijn. The mathematical language AUTOMATH, its usage and some of its extensions, in *Symposium on Automatic Demonstration*, Lecture Notes in Mathematics 125, Springer, pp 29 -61.

[dBr72] N.G. de Bruijn. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, *Indag. Math*, Vol. 34.

[dB80] N. G. de Bruijn. A survey of the AUTOMATH project, in [HS88], pp 580-606.

[BG90] R. Burstall and J. Goguen, *Institutions: Abstract Model Theory for Specification and Programming*, LFCS report series ECS-LFCS-90-106, Edinburgh University.

[Car78] J. Cartmell. *Generalised Algebraic Theories and Contextual Categories*, PhD thesis, Oxford University.

[Che80] B.F. Chellas. *Modal Logic: an introduction*, CUP.

[Chu40] A. Church. A formulation of the simple theory of types, *Journal of Symbolic Logic*, pp 56-68.

[Con86] R.L. Constable et al. *Implementing Mathematics with the NuPrl Proof Development System*, Prentice-Hall.

[CH90] R. Constable and D. Howe. NuPRL as a general logic, *Logic and Computation*, ed. P. Odifreddi, Acedemic Press.

[Coq85] T. Coquand. *Une Theorie des Constructions*, Ph.D. thesis, Paris 7.

[Coq91] T. Coquand. *An Algorithm for testing conversion in Type Theory*, Inria and the University of Göteborg/Chalmers, Sweden.

[CF58] H.B. Curry and R. Feys. *Combinatory Logic*, Vol. 1, Studies in Logic and the Foundations of Mathematics, North Holland.

[Cut86] N. J. Cutland. *Computability: An Introduction to recursive function theory*, CUP.

[Dow91] G. Dowek. *Démonstration Automatique dans le Calcul des Constructions*, These de Doctorat, Paris 7.

[Dun84] J.M. Dunn. Revelant Logic and Entailment, *Handbook of Philosophical Logic*, eds. D. GAbbay and F. Guenthner, pp 117–224.

[Ell90] C. Elliot. *Extensions and Applications of Higher-order unification*, Ph.D. thesis, School of Computer Science, Carnegie Mellon University.

[Fef89] S. Feferman. Finitary inductively presented logics, *Logic Colloquium 1988*, pp 191–220, Amsterdam, North Holland.

[Fel89] A. Felty. *Specifying and Implementing Theorem Provers in a Higher-order Logic Programming Language*, Ph.D. Thesis, Department of Computer and Information Science, University of Pennsylvania.

[Gab81] D. Gabbay. *Semantical Investigations in Heyting's Intuitionistic Logic*, Reidel: Dordrecht, Holland: Boston.

[Gar92] P.A. Gardner. Unpublished manuscript.

[Gen69] G. Gentzen. Investigations into Logical Deduction, *Collected works of Gentzen*, ed. M.E. Szabo, North-Holland, Amsterdam.

[Geu90] J.H. Geuvers. *Type systems for Higher-order Logic.* Draft. Faculty of Mathematics and Computer Science, University of Nijmegen.

[Geu91] J.H. Geuvers. *The Church–Rosser property for $\beta\eta$-reduction in typed lambda calculi*, draft, University of Nijmegen.

[GN91] J.H. Geuvers and M.J. Nederhof. A Modular Proof of Strong Normalisation for the Calculus of Constructions, *Journal of Functional Programming*, Part 2, Vol. 1, pp 61-70.

[Gir72] J.Y. Girard. *Interprétation foctionelle et élimination des coupures dans l'arithmétique d'ordre supérieure*, Ph.D. thesis, Paris 7.

[Gir87] J.Y. Girard. Linear Logic, *Theoretical Computer Science*, Vol. 50, pp 1-102.

[Ham80] A.G. Hamilton. *Logic for Mathematicians*, CUP.

[Har90] R. Harper. *Systems of Polymorphic Type Assignment in LF*, Technical Report CMU-CS-90-144, School of Computer Science, Carnegie Mellon University.

[HHP89] R. Harper, F. Honsell and G. Plotkin. A Framework for Defining Logics, *J. Assoc. Comp. Mach.*

[HST89] R. Harper, D. Sannella and A. Tarlecki. Structure and Representation in LF, *Proceedings of the 4th IEEE Symposium on Logic in Computer Science*, Asilomar, California.

[HS88] J.R. Hindley and J.P. Seldin. *Introduction to Combinators and the λ-calculus*, CUP, 2nd ed.

[How80] W.A.Howard. The formulae-as-types notion of construction, in [HS88], pp 479-490.

[Hue75] G.A. Huet. A Unification Algorithm for Typed λ-calculus, *Theoretical Computer Science*, Vol. 1, pp 27-57.

[HP91] G. Huet and G. Plotkin, editors. *Logical Frameworks*, CUP.

[Jac91] B.P.F. Jacobs. *Categorical Type theory*, Ph.D. thesis, Nijmegen University.

[Jut77] L. S. Jutting. *Checking Landau's Grundlagen in the AUTOMATH System*, PH.D. thesis, Eindhoven University, Netherlands.

[Law70] F.W. Lawvere. Equality in Hyperdoctrines and Comprehension Schema as an Adjoint Functor, *Proceedings of Symposia in Pure Mathematics*, Vol. XVIII: Applications of Categorical Algebra, American Mathematical Society.

[Lam89] J. Lambek. Multicategories Revisited, *Boulder Proceedings*, AMS Vol. 92, pp 217–234.

[Luo90] Z. Luo. *Extended Calculus of Constructions*, Ph.D. thesis.

[Mac88] S. MacLane. *Categories for the Working Mathematician*, Springer-Verlag.

[Mar80] P. Martin-Löf. *Intuitionistic Type Theory*, Padova notes.

[Mar85] P. Martin-Löf. *On the meanings of the logical constants and the justifications of the logical laws*, Technical report 2, Scula di Specializzazione in Logica Matematica, Dipartimento di Mathematica, Università di Siena.

[Men64] E. Mendelson. *Introduction to Mathematical Logic*, Princeton, Van Nostrad.

[MA88] P.F. Mendler and P. Aczel. *The notion of a framework and a framework for LTC.*, Proc. Third Annual Symposium on Logic in Computer Science, Edinburgh.

[Mes89] J. Meseguer. General Logics, *Proceedings of the Logic Colloquium '87*, ed. H.D. Ebbinghaus et al., North Holland.

[MN86] D. Miller and G. Nadathur. *Higher-order logic programming*, Proc. 3rd International Logic Programming Conference, London.

[MPP92] D. Miller, G.D. Plotkin and D. Pym. *Linear ELF*, to appear.

[MPW89] R. Milner, J.G. Parrow and D.J. Walker, *A Calculus of Mobile Mobile Processes, Parts I and II*, LFCS report series ECS-LFCS-89-85 and -86, Edinburgh, to appear in *Journal of Information and Computation.*

[Mit91] J.C. Mitchell. Type Systems for programming languages, *Handbook of Theoretical Computer Science*, ed. Jan van Leewen, Vol B: Formal Models and Semantics.

[Nip91] T. Nipkov. Order-sorted polymorphism in Isabelle, *Proceeding of the Second Annual Workshop on Logical Frameworks*, Edinburgh.

[NPS90] B. Nordström, K. Petersson and J. Smith. *Programming in Martin-Löf's Type Theory*, OUP.

[PS78] R. Paré and D. Schumacher. Abstract Families and the Adjoint Functor Theorems, *Indexed Categories and their Applications*, ed.s P.T. Johnstone and R. Paré.

[Pau87] L. Paulson. *The foundations of a generic theorem prover*, Technical Report 130, Computer Laboratory, University of Cambridge.

[Pfe91] F. Pfenning, Logic Programming in the LF logical framework, in [HP91], pp 149-183.

[Pit89] A. M. Pitts. *Categorical Semantics of Dependent Types*, notes on a talk given at SRI Menlo Park and at the Logic Colloquium in Berlin.

[Plo74] G. D. Plotkin. Call-by-name, Call-by-value and the λ-calculus, *Theoretical Computer Science*, North-Holland, Vol. 1, pp 125-159.

[Pol9-] R. Pollack. Ph.D thesis, Edinburgh.

[Pra65] D. Prawitz. *Natural Deduction: A Proof-Theoretical study*. Almquist and Wiksell, Stockholm.

[Pym90] D.J. Pym. *Proofs, Search and Computation in General Logic*, Ph.D. Thesis, Edinburgh University.

[PW91] D.J. Pym and L.A. Wallen. *Proof search in the $\lambda\Pi$-calculus*, in [HP91].

[Pym91] D.J. Pym. Talk at the Second Annual Workshop on Logical Frameworks Edinburgh.

[Sal89] A. Salvesen. *The Church-Rosser Property for LF with $\beta\eta$-reduction*, talk at the First Annual Workshop on Logical Frameworks, Antibes.

[Sal91] A. Salvesen. *The Church-Rosser Property for Pure Type Systems with $\beta\eta$-reduction*, draft.

[Schm83] D. Schmidt. *A Programming Notation for Tactical Reasoning*, Report CSR-141-83, Edinburgh University.

[Sch77] K. Schütte. *Proof Theory*, Springer-Verlag.

[See83] R.A.G. Seely. *Hyperdoctrines, Natural Deduction and the Beck Condition*, Zeitschrift für Mathematisch Logik ud Grundlagen der Mathematic, Vol. 29.

[Sim93] A. Simpson. Ph.D. Thesis, Edinburgh.

[Smu61] R. M. Smullyan.*Theory of Formal Systems*, Princeton University Press, New Jersey.

[Sto88] A. Stoughton, Substitution Revisited, *Theoretical Computer Science*, Vol. 59.

[Str89] T. Streicher. *Correctness and Completeness of a Categorical Semantics of the Calculus of Constructions*, Ph.D. thesis, Passau University.

[Tak75] G. Takeuti. *Proof Theory*, North-Holland.

[Tar56] A. Tarski. *Logic, Semantics and Metamathematics*, OUP.

[Ter89] J. Terlouw. *Een nadere bewijstheoretishe analyse van GSTT's*, Internal Report, Faculty of Mathematics and Computer Science, University of Nijmegen, Holland.