# Structured theory presentations and logic representations*

Robert Harper† Donald Sannella‡ Andrzej Tarlecki§

Draft: 10th December 1992

**Abstract**

The purpose of a logical framework such as LF is to provide a language for defining logical systems suitable for use in a logic-independent proof development environment. All inferential activity in an object logic (in particular, proof search) is to be conducted in the logical framework via the representation of that logic in the framework. An important tool for controlling search in an object logic, the need for which is motivated by the difficulty of reasoning about large and complex systems, is the use of structured theory presentations. In this paper a rudimentary language of structured theory presentations is presented, and the use of this structure in proof search for an arbitrary object logic is explored. The behaviour of structured theory presentations under representation in a logical framework is studied, focusing on the problem of "lifting" presentations from the object logic to the metalogic of the framework. The topic of imposing structure on logic presentations, so that logical systems may themselves be defined in a modular fashion, is also briefly considered.

## 1 Introduction

In logic, the traditional way to present a theory is by giving a set of axioms. This is sufficient for dealing with the simplest examples like groups or monoids. However, in Computer Science applications, such a presentation of a theory describing the behaviour of a complex real-life software system would involve a huge list of axioms, and the scale of such presentations makes them effectively useless. A commonly-accepted way to cope with this problem is to impose structure on theory presentations [BG77] and to build complex theories by combining smaller components. One advantage of such "modular" or "structured" theory presentations is that they provide a basis for guiding proof search in large theories. This was first considered in [SB83] in the context of Edinburgh LCF [GMW79]. An LCF theory is presented by declaring base types, constants, and function symbols (*i.e.*, by giving an LCF signature), and by giving a set of axioms over the language induced by these declarations. The fundamental idea in [SB83] is to exploit the invariance of consequence under changes of signature described by "signature morphisms." The language of structured presentations considered there (and in this paper) uses signature morphisms to mediate the combination of theories and to provide a form of "information hiding." The primitives of the presentation language are sufficient for the definability of a variety of interesting constructions such as instantiation of parametric presentations. All this

*This is an expanded and revised version of the paper "Structure and Representation in LF" which appeared in *Proc. 4th IEEE Symp. on Logic in Computer Science*, Asilomar, California, June 1989, pp. 226–237.

†School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, USA.

‡Laboratory for Foundations of Computer Science, Department of Computer Science, Edinburgh University, Edinburgh, Scotland.

§Institute of Informatics, Warsaw University, and Institute of Computer Science, Polish Academy of Sciences, Warsaw, Poland.

can be generalized to the context of logical systems other than LCF; the main purpose of this paper is to make this explicit and to consider the ramifications of these ideas in the context of a "universal metalogic" such as LF.

The Logical Framework (LF) [HHP93] is a meta-language for defining formal systems. It is a three-level typed $\lambda$-calculus with $\Pi$-types, closely related to the AUTOMATH type theories [dB80, vD80]. A formal system is specified by giving an LF signature, a finite list of constant declarations that specifies the syntax, judgement forms, and inference rules of the system. All of the syntactic apparatus of the formal system, including proofs, are represented as LF terms. The LF type system is sufficiently expressive to capture the uniformities of a large class of logical systems of interest to Computer Science, including notions of schematic rules and proofs, derived rules of inference, and higher-order judgement forms expressing consequence and generality.

According to the methodology of [HHP93, AHMP87], a necessary condition for the correctness of an encoding of an object logic $\mathcal{L}$ in LF is that the consequence relation $\vdash^{\mathcal{L}}$ of $\mathcal{L}$ be fully and faithfully embedded in the consequence relation $\vdash^{\mathcal{LF}}$ of LF by an encoding of the syntax of $\mathcal{L}$ as LF terms. (The consequence relation of LF is given by considering type inhabitation assertions, as in NuPRL [Con86].) By focusing on the embedding of consequence relations, LF may be viewed as a universal metalogic in which all inferential activity is to be conducted: object logics "exist" (for the purposes of implementation) only insofar as they are encodable in LF.

One important form of inferential activity in a logical system $\mathcal{L}$ is proof search: given a set of axioms or assumptions $\Phi$ and a conjecture $\phi$, determine whether or not $\Phi \vdash^{\mathcal{L}} \phi$. In keeping with the view of LF as a universal metalogic, proof in $\mathcal{L}$ is to be reduced to proof in LF via the encoding of $\mathcal{L}$ in LF. Numerous interesting questions arise in the process of carrying out this program. Pym [Pym90] considers a variety of issues related to proof search, in particular the definition of a unification algorithm and methods for conducting proof search in the context of an arbitrary LF signature. Elliott [Ell89] has also developed a unification algorithm for LF, and Pfenning [Pfe89], [Pfe91] bases a logic programming language on it.

In this paper, we consider proof search in structured theory presentations. We focus on "lifting" structured presentations from the level of the object logic to the level of the metalogic, in particular, on the conditions under which proofs in the metalogic for lifted structured presentations soundly represent proofs for structured presentations in the object logic.

Another important aspect of LF is that it opens up the possibility of using several logical systems at once. For example, one may view the encoding of **S4** modal logic given in [AHMP87] as a combination of the truth and validity consequence relations of **S4**. In this paper we suggest some basic machinery of a language of structured logic presentations that allows for "putting together logics," just as structured theory presentations provide the machinery for "putting together theories." This machinery may be used to formalize examples such as adding a connective to a logic, or the parameterization of Hoare logic by the logic of assertions.

This paper is organized as follows. In Section 2 we introduce a general definition of a logical system as a family of consequence relations indexed by signatures that satisfies a certain uniformity condition with respect to change of signature. This resembles the formalization of a logical system as an *institution* from [GB84a]; the crucial difference is that institutions present a model-theoretic view of logical systems while the formulation in this paper is centered directly around the notion of a consequence relation. (See also [FS87].) The sorts of consequence relations that we consider are motivated by the strictures of encoding in LF, and thus are limited to one-sided consequence relations that are closed under weakening, permutation, contraction, and cut, and which satisfy compactness. In Section 3 we consider structured presentations in an arbitrary logical system. Structured presentations denote theories (sets of sentences closed under consequence), and the structure of the presentations induces a natural proof search procedure guided by this structure, which we discuss in Section 4. Generalizing the methodology of [HHP93], in Section 5 we introduce the notion of a *representation* of one logical

system in another, taking account of variability in signatures, and then consider the problem of "lifting" a structured presentation along a representation of one logical system in another. Structured presentations may not be simply translated via the representation and used in the target logic. Instead, we define a notion of proof that is conditioned by the representation, and give restrictions under which we may achieve the goal of working entirely within the metalogic. In Section 6 we introduce the metalogic of interest, LF, as a logical system, and define the notion of a logic presentation. A logic presentation is essentially an LF signature (with an indication of which terms encode the judgements of the object logic), together with a representation of the object logic in the logical system given by the presentation. In Section 7 we return to the problem of proof in structured theory presentations in the specific setting of logics encoded in LF. In Section 8 we explore the colimit construction as a tool for building logics in a structured way; these ideas are more tentative than those in the rest of the paper. Finally, in Sections 9 and 10 we discuss related work and suggest directions for future research.

# 2 Consequence Relations and Logical Systems

Our treatment of logical systems centers on consequence relations (see [Avr91] for a survey). We take a consequence relation to be a binary relation between finite subsets and elements of a set of "sentences" satisfying three conditions to be given below. We use $\phi$ and $\psi$ to range over sentences, $\Phi$ to range over arbitrary sets of sentences, and $\Delta$ to range over finite sets of sentences. We write $\Delta, \Delta'$ for union, and $\phi, \Delta$ for $\{\phi\}, \Delta$. If $s : \Phi_1 \to \Phi_2$ is a function, then the extension of $s$ to subsets of $\Phi_1$ is denoted by $s$ as well. Function application will often be denoted by concatenation, *e.g.*, $s\phi$ stands for $s(\phi)$.

**Definition 2.1** *A* consequence relation (CR) *is a pair* $(S, \vdash)$ *where* $S$ *is a set of* sentences *and* $\vdash \subseteq Fin(S) \times S$ *is a binary relation such that*

1. *(Reflexivity)* $\phi \vdash \phi$.
2. *(Transitivity) If* $\Delta \vdash \phi$ *and* $\phi, \Delta' \vdash \psi$, *then* $\Delta, \Delta' \vdash \psi$.
3. *(Weakening) If* $\Delta \vdash \psi$, *then* $\phi, \Delta \vdash \psi$.

The choice of conditions on consequence relations is motivated by our intention to consider encodings of logical systems in LF (in a sense to be made precise below.) By considering only finite sets of sentences, we implicitly restrict attention to compact consequence relations. Although the technical development does not depend in any way on this choice, only compact consequence relations are amenable to machine implementation.

The following apparently more general properties are easily seen to hold of any consequence relation:

**Proposition 2.2**

1. *If* $\phi \in \Delta$, *then* $\Delta \vdash \phi$.
2. *If* $\Delta \vdash \phi$ *and* $\Delta, \phi, \Delta' \vdash \psi$, *then* $\Delta, \Delta' \vdash \psi$.
3. *If* $\Delta \vdash \phi$, *then* $\Delta, \Delta' \vdash \phi$.

**Definition 2.3** *Let* $(S, \vdash)$ *be a consequence relation and let* $S' \subseteq S$. *The* restriction *of* $(S, \vdash)$ *to* $S'$, *written* $(S, \vdash) \restriction S'$, *is the consequence relation* $(S', \vdash \cap (Fin(S') \times S'))$.

**Proposition 2.4** *If $(S, \vdash)$ is a consequence relation and $S' \subseteq S$, then $(S, \vdash) \restriction S'$ is indeed a consequence relation.*

**Definition 2.5** *A consequence relation $(S, \vdash)$ induces a closure operation on sets of sentences $\Phi \subseteq S$ defined by*

$$Cl_{\vdash}(\Phi) = \{\, \phi \mid \Delta \vdash \phi \text{ for some finite set } \Delta \subseteq \Phi \,\}.$$

We usually write $\overline{\Phi}$ for $Cl_{\vdash}(\Phi)$ when $\vdash$ is clear from context.

**Proposition 2.6** *The function $Cl_{\vdash} : \mathcal{P}ow(S) \to \mathcal{P}ow(S)$ is indeed a closure operation:*

1. *If $\Phi_1 \subseteq \Phi_2$, then $Cl_{\vdash}(\Phi_1) \subseteq Cl_{\vdash}(\Phi_2)$;*
2. *$\Phi \subseteq Cl_{\vdash}(\Phi)$;*
3. *$Cl_{\vdash}(\Phi) = Cl_{\vdash}(Cl_{\vdash}(\Phi))$.*

**Definition 2.7** *A set $\Phi$ is sentences is* closed under $\vdash$ *iff $Cl_{\vdash}(\Phi) = \Phi$. A* theory (wrt $\vdash$) *is a set of sentences closed under $\vdash$.*

**Definition 2.8** *A* morphism of consequence relations (CR morphism) *$s : (S_1, \vdash_1) \to (S_2, \vdash_2)$ is a function $s : S_1 \to S_2$ (the* translation of sentences*) such that if $\Delta \vdash_1 \phi$, then $s\Delta \vdash_2 s\phi$. The CR morphism $s$ is an* inclusion *if it is an inclusion as a function in the category of sets, and is* conservative *if $\Delta \vdash_1 \phi$ whenever $s\Delta \vdash_2 s\phi$.* **CR** *is the category whose objects are consequence relations and whose morphisms are CR morphisms, with identities and composition inherited from the category of sets.*

**Proposition 2.9** *If $s : (S_1, \vdash_1) \to (S_2, \vdash_2)$ is a CR morphism and $\Phi_1 \subseteq S_1$, then*

$$s(\overline{\Phi_1}) \subseteq \overline{s(\Phi_1)}.$$

The containment is, in general, proper since the image of a theory under a CR morphism need not be a theory. However, it follows from the above proposition (by Proposition 2.6) that $\overline{s(\overline{\Phi_1})} = \overline{s(\Phi_1)}$.

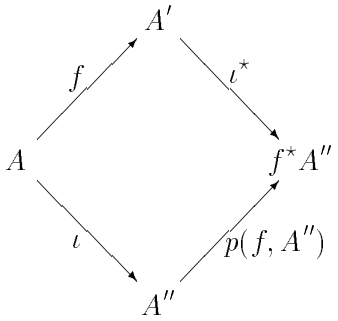
Techniques for structuring theory presentations are based on the idea of keeping explicit track of the language of a theory. Building large theories from smaller ones generally involves expansion of this language and/or change of the type/constant/function symbols used. Sometimes it is appropriate to "hide" some of the symbols in the language, restricting the vocabulary in order to abstract away from details of secondary interest. Consequently, a logical system is not viewed as being defined over an arbitrary but fixed language, but is instead considered to be a family of consequence relations indexed by a collection of *signatures* which determine the set of sentences considered. Variation in signature (for example, renaming constants or replacing constants by terms over another signature) gives rise to a natural translation of sentences over the signatures involved. Moreover, it is important that consequence be preserved under this translation. This partly captures the idea that the consequence relations in the family are defined uniformly with respect to their signatures, and leads to the following definition:

**Definition 2.10** *A* logical system, *or* logic, *is a functor $\mathcal{L} : \mathbf{Sig}^{\mathcal{L}} \to \mathbf{CR}$.*[1]

[1] Of course this definition captures only some aspects of what is usually meant by the informal notion of "logical system."

The category $\mathbf{Sig}^{\mathcal{L}}$ is called the *category of signatures* of $\mathcal{L}$, with objects denoted by $\Sigma$ and morphisms by $\sigma : \Sigma_1 \to \Sigma_2$. A signature morphism $\sigma : \Sigma_1 \to \Sigma_2$ is to be thought of as specifying a "relative interpretation" of the language defined by $\Sigma_1$ into the language defined by $\Sigma_2$. Writing $\mathcal{L}(\Sigma) = (|\mathcal{L}|_\Sigma, \vdash^{\mathcal{L}}_\Sigma)$, the definition of logical system implies that if $\sigma : \Sigma_1 \to \Sigma_2$ and $\Delta \vdash^{\mathcal{L}}_{\Sigma_1} \phi$, then $\mathcal{L}(\sigma)(\Delta) \vdash^{\mathcal{L}}_{\Sigma_2} \mathcal{L}(\sigma)(\phi)$. The function $\mathcal{L}(\sigma)$ underlying the CR morphism is called the *translation function* induced by $\sigma$. To simplify notation, we write $\sigma(\phi)$ for $\mathcal{L}(\sigma)(\phi)$ and $\sigma(\Delta)$ for $\mathcal{L}(\sigma)(\Delta)$ when no confusion is likely.

A logical system $\mathcal{L}$ *has inclusions*[2] iff the objects of $\mathbf{Sig}^{\mathcal{L}}$ are pre-ordered by a distinguished subcategory of morphisms, which will be referred to as *inclusions*, and $\mathcal{L}$ maps signature inclusions to inclusions of consequence relations. Inclusions are designated by $\iota : \Sigma_1 \hookrightarrow \Sigma_2$. In the particular cases that we study, signature morphisms are functions of some kind; we will normally assume without explicit mention that the signature inclusions are inclusions in the usual sense. The requirement that $\mathcal{L}$ preserve inclusions means that if $\iota : \Sigma_1 \hookrightarrow \Sigma_2$ and $\Delta \vdash^{\mathcal{L}}_{\Sigma_1} \phi$, then $\Delta \vdash^{\mathcal{L}}_{\Sigma_2} \phi$. If $\mathcal{C}$ is a category with a distinguished pre-order subcategory of inclusions, then we say that $\mathcal{C}$ *has pushouts along inclusions* iff whenever $f : A \to A'$ and $\iota : A \hookrightarrow A''$ are morphisms of $\mathcal{C}$, the pushout of $f$ and $\iota$ exists, and, moreover, the morphism opposite the inclusion in the pushout diagram may be chosen to be an inclusion:




We require a canonical choice of $p(f, A'')$ (and $f^\star A''$) which is functorial in $f$, *i.e.*, $p(f; f', A'') = p(f, A''); p(f', f^\star A'')$ (dually to contextual categories, *cf.* [Car86]). This will be needed for Prop. 8.3 only.

As an example, we define the logical system associated with many-sorted equational logic.

**Definition 2.11** *Let* $\mathbf{Sig}^{\mathcal{EQ}}$ *be the category of many-sorted algebraic signatures having:*

**Objects:** *Pairs* $\Sigma = (S, \Omega)$ *consisting of a set* $S$ *of type symbols and a family of sets* $\Omega = \langle \Omega_{w,s} \rangle_{w \in S^\star, s \in S}$ *of function symbols.*

**Morphisms:** $\sigma : (S, \Omega) \to (S', \Omega')$ *consists of a function* $t : S \to S'$ *together with a family of functions* $\langle f_{w,s} : \Omega_{w,s} \to \Omega'_{t^\star(w), t(s)} \rangle_{w \in S^\star, s \in S}$. *The composition of morphisms is the composition of their corresponding components as functions. Inclusions are pairs consisting of an inclusion and a family of inclusions.*

Let $\Sigma = (S, \Omega)$ be an algebraic signature. Define the set $Eq(\Sigma)$ of $\Sigma$-equations to be the set of triples $(X, t_1, t_2)$, where $X$ is a finite sequence of mutually distinct variables decorated with elements of $S$ and $t_1, t_2$ are $\Sigma$-terms of the same sort with variables from $X$. The equation $(X, t_1, t_2)$ will be written $\forall X.\ t_1 = t_2$, or $t_1 = t_2$ if $X$ is the empty sequence. Equations with no variables will be called *ground* equations. The consequence relation $(Eq(\Sigma), \vdash^{\mathcal{EQ}}_\Sigma)$ is defined in the standard model-theoretic way via a notion of satisfaction of a $\Sigma$-equation by a $\Sigma$-algebra, or equivalently by appropriate rules of equational deduction (reflexivity, symmetry, *etc.* [GM81]).

[2] This is a much weaker concept than that of an inclusion system as introduced in [DGS92]. The requirements stated here are sufficient for our purposes.

**Definition 2.12** *The functor* $\mathcal{EQ} : \mathbf{Sig}^{\mathcal{EQ}} \to \mathbf{CR}$ *is defined by*

$$\begin{array}{rcl} \mathcal{EQ}(\Sigma) & = & (Eq(\Sigma), \vdash^{\mathcal{EQ}}_{\Sigma}) \\ \mathcal{EQ}(\sigma : \Sigma \to \Sigma') & = & \textit{the usual extension of } \sigma \textit{ to a function } Eq(\Sigma) \to Eq(\Sigma') \end{array}$$

*$\mathcal{GEQ}(\Sigma)$ is the restriction of $\mathcal{EQ}(\Sigma)$ to ground $\Sigma$-equations, and $\mathcal{GEQ}(\sigma)$ is the corresponding restriction of $\mathcal{EQ}(\sigma)$.*

**Proposition 2.13**

1. *$\mathcal{EQ}$ and $\mathcal{GEQ}$ are logical systems with inclusions.*
2. *$\mathbf{Sig}^{\mathcal{EQ}}$ has pushouts along inclusions (in fact, is co-complete).*

**Proof** *$\mathcal{EQ}(\sigma : \Sigma \to \Sigma')$ is a CR morphism because of the Satisfaction Lemma [BG80]. Similarly for $\mathcal{GEQ}$. Pushouts in $\mathbf{Sig}^{\mathcal{EQ}}$ are defined as in [GB84b].*

In a similar manner we can present first-order logic with equality. The logical system $\mathcal{FOEQ}$ has the same signatures as $\mathcal{EQ}$ (we take equality to be the only predicate) and for any many-sorted algebraic signature $\Sigma$, $\mathcal{FOEQ}(\Sigma)$ is the set of closed first-order logical formulae with equalities as atomic formulae, with the consequence relation induced by the usual inference rules (or equivalently by the usual satisfaction relation). For any signature morphism $\sigma : \Sigma \to \Sigma'$, $\mathcal{FOEQ}(\sigma)$ translates closed $\Sigma$-formulae to closed $\Sigma'$-formulae in the obvious way. Then $\mathcal{FOEQ}$ is a logical system with inclusions.

Note that CR morphisms induced by signature morphisms in a logical system need not be conservative. Non-conservativity arises in $\mathcal{EQ}$ and $\mathcal{FOEQ}$ (due to the infamous empty carrier phenomenon — see [GM81]) but not in $\mathcal{GEQ}$.

# 3 Theory Presentations

Let $\mathcal{L}$ be an arbitrary logical system. As formulated above, $\mathcal{L}$ comprises a family of consequence relations satisfying some additional conditions. Thus, concepts introduced for consequence relations lift to $\mathcal{L}$. Of particular importance is the concept of theory:

**Definition 3.1** *An* $\mathcal{L}$-theory *with signature $\Sigma$ is a set $T \subseteq |\mathcal{L}|_\Sigma$ of sentences closed under $\vdash^{\mathcal{L}}_{\Sigma}$.*

Notice that in any given logical system, theories are classified by their signatures. Thus, for example, the equational theory of monoids and the equational theory of Abelian groups have different signatures although both are $\mathcal{EQ}$-theories.

As mentioned in the introduction, the complexity of real-life software systems means that theories describing their behavior must be built in a modular or structured fashion. Only the simplest theories are presented in the traditional way by giving (a signature $\Sigma$ and) a set $\Phi \subseteq |\mathcal{L}|_\Sigma$ of axioms, denoting the theory $\overline{\Phi}$. We define below a rudimentary language of structured theory presentations for building more complex theories by combining and enriching such simple ones. The presentation language that we choose is adapted from [SB83] for use in an arbitrary logical system.

**Definition 3.2** *A* structured theory presentation in $\mathcal{L}$ ($\mathcal{L}$-presentation) *is an expression in the language generated by the following grammar:*

$$\begin{array}{rcl} P & ::= & (\Sigma, \Phi) \\ & | & P_1 \cup P_2 \\ & | & \textsf{translate } P \textsf{ along } \sigma \\ & | & \textsf{derive } P \textsf{ via } \sigma \end{array}$$

*(Here $\Sigma$ is a $\mathbf{Sig}^{\mathcal{L}}$-signature, $\sigma$ is a $\mathbf{Sig}^{\mathcal{L}}$-morphism and $\Phi$ is a set of $\mathcal{L}$-sentences.) Structured presentations of the form $(\Sigma, \Phi)$ are called* basic *presentations. A structured presentation is* finite *if all the basic presentations it contains involve only finite sets of sentences.*

In the above grammar we do not specify how signatures, signature morphisms, or sets of sentences are presented. For logics with finite signatures, it is unproblematic to define a presentation language for signatures and signature morphisms (*e.g.*, [Wir86]). In practice infinite presentations are given using some form of schematization. For the sake of simplicity we do not make this explicit here.

**Definition 3.3** *The* signature $\mathbf{Sg}^{\mathcal{L}}(P)$ *of an $\mathcal{L}$-presentation $P$ is defined by induction on the structure of $P$ as follows:* $\mathbf{Sg}^{\mathcal{L}}(P) = \Sigma$ *iff*

- $P = (\Sigma, \Phi)$, *or*
- $P = P_1 \cup P_2$ *and* $\mathbf{Sg}^{\mathcal{L}}(P_1) = \mathbf{Sg}^{\mathcal{L}}(P_2) = \Sigma$, *or*
- $P =$ translate $P_1$ along $\sigma$, $\sigma : \Sigma_1 \to \Sigma$, *and* $\mathbf{Sg}^{\mathcal{L}}(P_1) = \Sigma_1$, *or*
- $P =$ derive $P_1$ via $\sigma$, $\sigma : \Sigma \to \Sigma_1$, *and* $\mathbf{Sg}^{\mathcal{L}}(P_1) = \Sigma_1$.

$P$ *is* well-formed *iff* $\mathbf{Sg}^{\mathcal{L}}(P)$ *is defined.*

**Definition 3.4** *Let $P$ be a well-formed $\mathcal{L}$-presentation. The* theory *determined by $P$ is defined as follows:*

$$\begin{aligned}
\mathbf{Th}^{\mathcal{L}}((\Sigma, \Phi)) &= \overline{\Phi} \\
\mathbf{Th}^{\mathcal{L}}(P_1 \cup P_2) &= \overline{\mathbf{Th}^{\mathcal{L}}(P_1) \cup \mathbf{Th}^{\mathcal{L}}(P_2)} \\
\mathbf{Th}^{\mathcal{L}}(\text{translate } P_1 \text{ along } \sigma) &= \overline{\mathcal{L}(\sigma)(\mathbf{Th}^{\mathcal{L}}(P_1))} \\
\mathbf{Th}^{\mathcal{L}}(\text{derive } P_1 \text{ via } \sigma) &= \mathcal{L}(\sigma)^{-1}(\mathbf{Th}^{\mathcal{L}}(P_1))
\end{aligned}$$

**Proposition 3.5** *For any well-formed $\mathcal{L}$-presentation $P$, $\mathbf{Th}^{\mathcal{L}}(P)$ is an $\mathcal{L}$-theory with signature $\mathbf{Sg}^{\mathcal{L}}(P)$.*

**Proof** *The non-trivial case is that of* derive*, where one has to notice that the co-image of a theory under a signature morphism is a theory.*

The language of structured presentations allows large theories to be built in a flexible and well-structured fashion. Union is used to combine separate theories over the same signature. Theories over different signatures may be combined using union together with translate. If a signature in $\mathbf{Sig}^{\mathcal{L}}$ is thought of as a vocabulary of type, constant and function symbols, where a morphism is a renaming of the symbols in one signature to those in another, then the translate operation is useful for applying such a renaming to a theory while the derive operation is used to "abstract" from a theory by hiding some symbols (for example, auxiliary function symbols needed to finitely axiomatize some other function) and perhaps renaming the rest. The operations used are almost the same as those in [SB83] (union is inessentially different). The theory-building operations of the specification language CLEAR [BG80] may be defined in terms of these primitives.

A few concrete examples should help to clarify the motivation behind structured theory presentations. See [BG81], [SB83] and Section 4 for further examples. We will consider structured theory presentations over the logic $\mathcal{EQ}$ defined in Section 2.

**Example 3.6** Let

$$\begin{array}{rcll}
\Sigma Group & = & \text{type} & G \\
 & & \text{constant} & \epsilon : G \\
 & & \text{functions} & \circ : G \times G \to G \\
 & & & inv : G \to G
\end{array}$$

and

$$\begin{aligned} EqGroup \quad = \quad & \{\forall g{:}G.\ \epsilon \circ g = g, \\ & \forall g{:}G.\ g \circ \epsilon = g, \\ & \forall g, g', g''{:}G.\ g \circ (g' \circ g'') = (g \circ g') \circ g'', \\ & \forall g{:}G.\ g \circ inv(g) = \epsilon, \\ & \forall g{:}G.\ inv(g) \circ g = \epsilon\} \end{aligned}$$

(using $\circ$ as an infix function). We use a self-explanatory notation to introduce types and functions (and constants, which are 0-ary functions). Then $Group = (\Sigma Group, EqGroup)$ is an $\mathcal{EQ}$-presentation. Let

$$\begin{aligned} \Sigma Abelian \quad = \quad & \textsf{type} \quad T \\ & \textsf{function} \quad op : T \times T \to T \end{aligned}$$

and

$$EqAbelian \quad = \quad \{\forall t, t'{:}T.\ op(t, t') = op(t', t)\}$$

Then $Abelian = (\Sigma Abelian, EqAbelian)$ is an $\mathcal{EQ}$-presentation, and so is $AbelianGroup = Group \cup$ (**translate** $Abelian$ **along** $\sigma_{AG}$) where $\sigma_{AG} : \Sigma Abelian \to \Sigma Group$ is defined by $\sigma_{AG}(T) = G$ and $\sigma_{AG}(op) = \circ$. This shows how union may be used to combine separate theories, and how **translate** is used for adjusting signatures (in this case, to make union applicable). Let

$$\begin{aligned} \Sigma Monoid \quad = \quad & \textsf{type} \quad M \\ & \textsf{constant} \quad \epsilon : M \\ & \textsf{function} \quad \circ : M \times M \to M \end{aligned}$$

Then $Monoid =$ **derive** $Group$ **via** $\sigma_{MG}$ and $AbelianMonoid =$ **derive** $AbelianGroup$ **via** $\sigma_{MG}$ are $\mathcal{EQ}$-presentations, where $\sigma_{MG} : \Sigma Monoid \to \Sigma Group$ is defined by $\sigma_{MG}(M) = G$, $\sigma_{MG}(\circ) = \circ$ and $\sigma_{MG}(\epsilon) = \epsilon$. This shows how **derive** may be used to hide functions, in this case $inv$. Note that this only hides $inv$; its existence is not totally forgotten. It happens to be the case here that this does not have any consequences which are expressible as $\Sigma Monoid$-equations. If this example were formulated in first-order logic with equality, then the resulting theory would include sentences like $\forall x{:}M.\ \exists y{:}M.\ x \circ y = \epsilon$.

Another example of the use of **derive** would be defining a function *sort* by specifying auxiliary boolean-valued functions *permutation* and *ordered*, with the equations

$$\begin{aligned} & \forall l{:}list.\ ordered(sort(l)) = true \\ & \forall l{:}list.\ permutation(l, sort(l)) = true \end{aligned}$$

to define *sort*, and then using **derive** to hide *permutation* and *ordered*. In general, it is possible to give finite $\mathcal{EQ}$-presentations using **derive** for theories which have no finite $\mathcal{EQ}$-presentations without **derive** [TWW82]. This is in contrast with the other theory-building operations; it is easy to see that any finite $\mathcal{L}$-presentation built using only basic presentations, union and **translate** has the same theory as a finite basic presentation.

In general, we will say that two well-formed $\mathcal{L}$-presentations with the same signature are *equivalent* if they determine the same theory. A more general situation is when we want to compare $\mathcal{L}$-presentations with different signatures. This comparison is mediated by a morphism between the two signatures involved.

**Definition 3.7** *An* $\mathcal{L}$-presentation morphism $\sigma : P \to P'$ *is a* $\mathbf{Sig}^{\mathcal{L}}$*-morphism* $\sigma : \mathbf{Sg}^{\mathcal{L}}(P) \to \mathbf{Sg}^{\mathcal{L}}(P')$ *such that*

$$\sigma(\mathbf{Th}^{\mathcal{L}}(P)) \subseteq \mathbf{Th}^{\mathcal{L}}(P')$$

$\mathbf{ThPres}^{\mathcal{L}}$ *is the category of* $\mathcal{L}$*-presentations and morphisms between them, with identities, composition, and inclusions inherited from* $\mathbf{Sig}^{\mathcal{L}}$.

Pushouts in the category of presentations may be used to define CLEAR-style instantiation of parameterized theories [BG80], [Ehr 82]. A presentation $P$ is "parametric" in a presentation $R$ if there is a $\mathbf{ThPres}^{\mathcal{L}}$-inclusion $\iota : R \hookrightarrow P$. The idea is that $R$ is a "requirement" specification for the theory $P$ which may be regarded as taking any theory "matching" $R$ as a parameter. The parametric presentation $P$ may be instantiated by any presentation $A$ provided that there is a "fitting morphism" $\sigma : R \to A$ specifying how $A$ is to be regarded as satisfying the requirements of $R$. The instance of $P$ by $A$ via $\sigma$, written $P(A[\sigma])$, is obtained by taking the pushout of $\sigma$ and $\iota$ in $\mathbf{ThPres}^{\mathcal{L}}$. The conditions under which this works are expressed by the following proposition:

**Proposition 3.8** *If* $\mathbf{Sig}^{\mathcal{L}}$ *has pushouts along inclusions, then so does* $\mathbf{ThPres}^{\mathcal{L}}$. *More specifically, suppose that* $\iota : R \hookrightarrow P$ *and* $\sigma : R \to A$ *are* $\mathbf{ThPres}^{\mathcal{L}}$ *morphisms, and let* $\Sigma = \mathbf{Sg}^{\mathcal{L}}(P)$. *The pushout of* $\iota$ *and* $\sigma$ *is given by the object*

$$\sigma^\star P = (\mathsf{translate}\ P\ \mathsf{along}\ p(\sigma, \Sigma)) \cup (\mathsf{translate}\ A\ \mathsf{along}\ \iota^\star)$$

*and morphisms* $p(\sigma, P) = p(\sigma, \Sigma) : P \to \sigma^\star P$ *and* $\iota^\star : A \to \sigma^\star P$ *given by the pushout construction in* $\mathbf{Sig}^{\mathcal{L}}$.

Thus the instance $P(A[\sigma])$ of $P$ by $A$ is explicitly definable in the language of structured presentations. It is important for the sequel that complex presentations involving parameterization, *etc.*, may be reduced to structured theory presentations involving only the primitives given above.

**Example 3.9** Suppose *Monoid* is as defined in Example 3.6. Let

$$\begin{array}{lcll} \Sigma Alphabet & = & \mathsf{type} & A \\ \Sigma InjAlph & = & \mathsf{types} & A, M \\ & & \mathsf{function} & in : A \to M \end{array}$$

Let $Alphabet = (\Sigma Alphabet, \emptyset)$, $InjAlph = (\Sigma InjAlph, \emptyset)$ and

$$MonoidWithAlphabet = (\mathsf{translate}\ Monoid\ \mathsf{along}\ \sigma_{mon}) \cup (\mathsf{translate}\ InjAlph\ \mathsf{along}\ \sigma_{inj})$$

where $\sigma_{mon} : \Sigma Monoid \hookrightarrow \Sigma Monoid \cup \Sigma InjAlph$ and $\sigma_{inj} : \Sigma InjAlph \hookrightarrow \Sigma Monoid \cup \Sigma InjAlph$ are the inclusions into the union of the two signatures. There is an obvious $\mathbf{ThPres}^{\mathcal{EQ}}$-inclusion $\iota : Alphabet \hookrightarrow MonoidWithAlphabet$; thus the $\mathcal{EQ}$-presentation *MonoidWithAlphabet* is parametric in *Alphabet*.

In the rest of the example we construct a simple instance of *MonoidWithAlphabet*. Let

$$\begin{array}{lcll} \Sigma Nat & = & \mathsf{type} & N \\ & & \mathsf{constant} & 0 : N \\ & & \mathsf{functions} & succ : N \to N \\ & & & + : N \times N \to N \\ & & & \times : N \times N \to N \end{array}$$

and

$$\begin{array}{lcl} EqNat & = & \{\forall n{:}N.\ n + 0 = n, \\ & & \ \forall n, n'{:}N.\ n + succ(n') = succ(n + n'), \\ & & \ \forall n{:}N.\ n \times 0 = 0, \\ & & \ \forall n, n'{:}N.\ n \times succ(n') = (n \times n') + n\} \end{array}$$

(using $+$ and $\times$ as infix). Let $Nat = (\Sigma Nat, EqNat)$ and let $\sigma : Alphabet \to Nat$ be the presentation morphism defined by $\sigma(A) = N$.

Then $MonoidWithAlphabet(Nat[\sigma])$ is the pushout of $\iota$, $\sigma$. This is (a rough approximation of[3]) the theory of sequences of natural numbers with types $M$ and $N$, constants $\epsilon$ and 0 and functions $\circ$, $in$, $succ$, $+$ and $\times$. If type names such as $sequence$ and $nat$ and/or different constant and function names are required, these may be obtained using translate.

**Example 3.10** Repeat example 3.9, but add the following to *Alphabet*:

$$\begin{array}{ll} \textsf{constant} & neutral : A \\ \textsf{function} & op : A \times A \rightarrow A \\ \textsf{axioms} & \forall a{:}A.\ op(neutral, a) = a, \\ & \forall a{:}A.\ op(a, neutral) = a, \\ & \forall a, a', a''{:}A.\ op(a, op(a', a'')) = op(op(a, a'), a'') \end{array}$$

Add the above to *InjAlph* (and so to *MonoidWithAlphabet*) along with:

$$\begin{array}{ll} \textsf{function} & accum : M \rightarrow A \\ \textsf{axioms} & accum(\epsilon) = neutral \\ & \forall a{:}A.\ accum(in(a)) = a \\ & \forall m, m'{:}M.\ accum(m \circ m') = op(accum(m), accum(m')) \end{array}$$

We still have the inclusion $\iota : Alphabet \hookrightarrow MonoidWithAlphabet$, and thus *MonoidWithAlphabet* is still parametric in *Alphabet*. To construct an instance of it, we now have to ensure that the parameter presentation matches *Alphabet*, which requires that it satisfies the new requirements in *Alphabet*.

Add the following axioms to *EqNat*:

$$\begin{array}{l} \forall n{:}N.\ 0 + n = n \\ \forall n, n', n''{:}N.\ n + (n' + n'') = (n + n') + n'' \end{array}$$

(these may be proved from the axioms already in *EqNat* if an appropriate induction principle is available). Then let $\sigma : Alphabet \rightarrow Nat$ be defined by $\sigma(A) = N$, $\sigma(neutral) = 0$ and $\sigma(op) = +$ (this is a presentation morphism).

Then $MonoidWithAlphabet(Nat[\sigma])$ is the pushout of $\iota$, $\sigma$. This is the theory of sequences of natural numbers with a summation function ($accum$). In this example we have included the monoid axioms in the requirement theory *Alphabet* since these characterize the intended actual parameters. The axioms for $accum$ may force elements of the type matching $A$ to be identified if the function matching $op$ is not associative or the constant matching $neutral$ is not an identity for this function. Requiring the fitting morphism to be a presentation morphism protects against such undesirable instantiations.

A variety of other constructions are definable in $\mathbf{ThPres}^{\mathcal{L}}$. For example, $\mathbf{ThPres}^{\mathcal{L}}$ has coproducts whenever $\mathbf{Sig}^{\mathcal{L}}$ does, and the theory of the coproduct is the disjoint union of the theories of the components. Colimits of more complex diagrams in $\mathbf{ThPres}^{\mathcal{L}}$ may be used to express sharing; such colimits exist if they exist in $\mathbf{Sig}^{\mathcal{L}}$. In particular, diagrams in $\mathbf{Sig}^{\mathcal{L}}$ consisting only of inclusions arise in a natural way from the hierarchical construction of theories by extension. In many interesting cases all such diagrams have colimits, and we may therefore use colimits as the basis for a CLEAR-like or ML-like syntax for managing sharing [ST85, ST86].

[3]It is well-known that the first-order equational theory of the standard model of the natural numbers is not r.e. and hence cannot be effectively presented in $\mathcal{EQ}$. By enriching the logical system (*e.g.*, by introducing induction schemes or by adding non-first-order notions such as the "data constraints" of [BG80, MS85]), better approximations to the theory may be given.

A parametric presentation amounts to a function which maps a presentation and a fitting morphism to another presentation, with the result determined by the pushout construction. Another obvious mechanism for defining such functions is $\lambda$-abstraction, and it would have been possible to use this here instead of pushout-style parameterization as above. In fact, one advantage of this alternative is that higher-order parameterization may be achieved without additional complications; see [SW83], [Wir86] and [SST92].

# 4 Proof Search in Structured Presentations

For a basic $\mathcal{L}$-presentation $(\Sigma, \Phi)$, the consequence relation $\vdash^{\mathcal{L}}_{\Sigma}$ can be used directly to deduce consequences of $\Phi$, that is, sentences in $\mathbf{Th}^{\mathcal{L}}((\Sigma, \Phi))$. More generally, for an arbitrary presentation $P$, we would like to deduce sentences in $\mathbf{Th}^{\mathcal{L}}(P)$ from consequences of the component presentation(s) of $P$. To capture this, we will introduce judgements of the form $P \Vdash^{\mathcal{L}} \phi$, where $\phi$ is a $\mathbf{Sg}^{\mathcal{L}}(P)$-sentence, and show how they can be proved.

**Definition 4.1** *We define a family of relations $P \Vdash^{\mathcal{L}}_{\Sigma} \phi$ between well-formed $\mathcal{L}$-presentations $P$ with signature $\Sigma$ and sentences $\phi \in |\mathcal{L}|_{\Sigma}$ by induction on the structure of $P$ as follows:*

1. *$(\Sigma, \Phi) \Vdash^{\mathcal{L}}_{\Sigma} \phi$ iff there exists $\Delta \subseteq \Phi$ such that $\Delta \vdash^{\mathcal{L}}_{\Sigma} \phi$.*
2. *$P_1 \cup P_2 \Vdash^{\mathcal{L}}_{\Sigma} \phi$ iff there exists $\Delta_1 \subseteq |\mathcal{L}|_{\Sigma}$ and $\Delta_2 \subseteq |\mathcal{L}|_{\Sigma}$ such that $P_1 \Vdash^{\mathcal{L}}_{\Sigma} \Delta_1$, $P_2 \Vdash^{\mathcal{L}}_{\Sigma} \Delta_2$, and $\Delta_1, \Delta_2 \vdash^{\mathcal{L}}_{\Sigma} \phi$.*
3. *translate $P_1$ along $\sigma \Vdash^{\mathcal{L}}_{\Sigma} \phi$ (where $\sigma : \Sigma_1 \to \Sigma$) iff there exists $\Delta_1 \subseteq |\mathcal{L}|_{\Sigma_1}$ such that $P_1 \Vdash^{\mathcal{L}}_{\Sigma_1} \Delta_1$ and $\mathcal{L}(\sigma)(\Delta_1) \vdash^{\mathcal{L}}_{\Sigma} \phi$.*
4. *derive $P_1$ via $\sigma \Vdash^{\mathcal{L}}_{\Sigma} \phi$ (where $\sigma : \Sigma \to \Sigma_1$) iff $P_1 \Vdash^{\mathcal{L}}_{\Sigma_1} \mathcal{L}(\sigma)(\phi)$.*

*Here, $P \Vdash^{\mathcal{L}}_{\Sigma} \Delta$ stands for $P \Vdash^{\mathcal{L}}_{\Sigma} \phi$ for all $\phi \in \Delta$.*

**Proposition 4.2** *The relation $P \Vdash^{\mathcal{L}}_{\mathbf{Sg}^{\mathcal{L}}(P)} \phi$, where $P$ is a well-formed $\mathcal{L}$-presentation and $\phi$ is a $\mathbf{Sg}^{\mathcal{L}}(P)$-sentence, holds iff $\phi \in \mathbf{Th}^{\mathcal{L}}(P)$.*
**Proof** *By structural induction on $P$, directly from Definition 4.1 and 3.4.*

It should be obvious that Definition 4.1 embodies a proof system for entailment $P \Vdash^{\mathcal{L}}_{\Sigma} \phi$ with rules like:

$$\frac{P_1 \Vdash^{\mathcal{L}}_{\Sigma} \Delta_1 \qquad P_2 \Vdash^{\mathcal{L}}_{\Sigma} \Delta_2 \qquad \Delta_1, \Delta_2 \vdash^{\mathcal{L}}_{\Sigma} \phi}{P_1 \cup P_2 \Vdash^{\mathcal{L}}_{\Sigma} \phi}$$

We can use this proof system as the basis for a proof procedure based on structured presentations.

To illustrate how such a proof procedure may take advantage of the structure of a presentation, consider a logical system $\mathcal{L}$ with inclusions. Let $P_1$ be an $\mathcal{L}$-presentation with signature $\Sigma_1$, and let $\iota : \Sigma_1 \hookrightarrow \Sigma$ be an inclusion. If $\phi \in |\mathcal{L}|_{\Sigma_1}$, then a useful heuristic for testing translate $P_1$ along $\iota \Vdash^{\mathcal{L}}_{\Sigma} \phi$ is to take $\Delta_1$ in the above proposition to be $\{\phi\}$, and to test $P_1 \Vdash^{\mathcal{L}}_{\Sigma_1} \phi$. According to Definition 4.1(3) this is sufficient (but not necessary, in general), for since $\mathcal{L}$ preserves inclusions, $\iota\Delta_1 = \{\phi\}$, and hence the requirement $\iota\Delta_1 \vdash^{\mathcal{L}}_{\Sigma} \phi$ is trivial. A generalization of this heuristic is embodied in the following rule:

$$\frac{P_1 \Vdash^{\mathcal{L}}_{\Sigma_1} \phi}{\text{translate } P_1 \text{ along } \sigma \Vdash^{\mathcal{L}}_{\Sigma} \mathcal{L}(\sigma)(\phi)} \qquad \text{(Translate)}$$

Analogous rules arise from the other parts of Definition 4.1:

$$\frac{\phi \in \Phi}{(\Sigma, \Phi) \Vdash^{\mathcal{L}}_{\Sigma} \phi} \qquad \text{(Basic)}$$

$$\frac{P_1 \Vdash^{\mathcal{L}}_{\Sigma} \phi}{P_1 \cup P_2 \Vdash^{\mathcal{L}}_{\Sigma} \phi} \qquad \text{(Union-left)}$$

$$\frac{P_2 \Vdash^{\mathcal{L}}_{\Sigma} \phi}{P_1 \cup P_2 \Vdash^{\mathcal{L}}_{\Sigma} \phi} \qquad \text{(Union-right)}$$

$$\frac{P_1 \Vdash^{\mathcal{L}}_{\Sigma_1} \mathcal{L}(\sigma)(\phi)}{\mathsf{derive}\ P_1\ \mathsf{via}\ \sigma \Vdash^{\mathcal{L}}_{\Sigma} \phi} \qquad \text{(Derive)}$$

Proof in the context of a structured theory presentation is fundamentally different from proof in an ordinary (unstructured) theory presentation. Both kinds of presentations contain axioms which form the basic constituents of proofs. In the case of an ordinary theory presentation, we have a single set of axioms, and use of an axiom in a proof involves the application of the rule:

$$\frac{\phi \in \Delta}{\Delta \vdash^{\mathcal{L}}_{\Sigma} \phi} \qquad \text{(Axiom)}$$

In contrast, the axioms of a structured theory presentation tend to be scattered throughout the structure. An axiom must be extracted from the basic presentation in which it resides when it is needed in a proof, using rules such as Translate. Proof in a structured theory can thus involve frequent changes of context, where proof fragments in the context of "small" theories correspond to the proofs of lemmas which are then brought to bear on the main proof via translation to the context of an appropriate "larger" theory. An analogy may perhaps be drawn with the use and discharge of assumptions in natural deduction proofs, where different parts of the proof of a theorem take place in the context of different sets of assumptions.

Given the goal of finding a proof for a theorem $\phi$ in a structured theory presentation $P$, where the proof may potentially involve axioms from a number of different subpresentations of $P$, there are two basic strategies which may be applied. Probably the most obvious of these involves reduction to the familiar case of proof in an unstructured presentation, using a technique referred to in [SB83] as *dredging*. One proceeds by extracting ("dredging up") all of the possibly relevant axioms from subpresentations and translating these to theorems in $P$ using inference rules such as Translate. These theorems may then be used in the proof of $\phi$. This is the strategy which is (implicitly) used in systems with facilities for building new theories by combining and extending existing theories, such as Edinburgh LCF [GMW79] (also Cambridge LCF [Pau87] and Isabelle [Pau92]), in which the new theory automatically contains all the axioms and theorems of its component theories. An alternative, called *diving* in [SB83], is to translate $\phi$ to the context of an appropriate subpresentation $P'$ of $P$, using rules such as Translate "backwards". If a proof for the translation of $\phi$ can be found in $P'$, then applying the same translation rules in the forward direction gives a proof of $\phi$ in $P$.

There are at least two problems with dredging. First, if $P$ is large then dredging up *all* the axioms in $P$ yields a large and unstructured set of axioms, many of which will (probably) make no contribution to the proof of $\phi$. Second, dredging axioms from a structured presentation of the form $\mathsf{derive}\ P'\ \mathsf{via}\ \sigma$ tends to lead to loss of information. For example, consider the following structured theory presentation based on the basic presentation *Nat* (over $\mathcal{EQ}$) from Example 3.9:

$$Nat^{\times} = \mathsf{derive}\ Nat\ \mathsf{via}\ \iota : \Sigma Nat \setminus \{+\} \hookrightarrow \Sigma Nat$$

The only axiom of $Nat$ which can be directly translated to the context of $Nat^\times$ is $\forall n{:}N.\ n \times 0 = 0$, since the remaining axioms of $Nat$ involve the function $+$ which is not available in $\mathbf{Sg}^{\mathcal{EQ}}(Nat^\times)$. Theorems of $Nat^\times$ such as $succ(0) \times succ(0) = succ(0)$ are expressible in $\mathbf{Sg}^{\mathcal{EQ}}(Nat^\times)$, but there are infinitely many such consequences (and in $\mathcal{EQ}$ there is no finite way to present them all).

Likewise, diving by itself is not an appropriate strategy. For example, a proof of $\phi$ in a structured theory presentation of the form $P_1 \cup P_2$ may involve the use of axioms from both $P_1$ and $P_2$. A successful attempt to prove $\phi$ will involve either dredging up axioms from both $P_1$ and $P_2$, or the formation of lemmas $\phi_1$ and $\phi_2$ such that $P_1 \Vdash^{\mathcal{L}} \phi_1$, $P_2 \Vdash^{\mathcal{L}} \phi_2$ (establishing these may involve further diving) and $\phi_1, \phi_2 \vdash^{\mathcal{L}} \phi$, or some combination of these. Similar problems can arise in structured presentations of the form translate $P'$ along $\sigma$.

The most promising strategy for proving $\phi$ in $P$ involves a mixture of diving and dredging. First, diving is used to focus on the smallest subpresentation $P'$ of $P$ containing all the information relevant to $\phi$. This is the most appropriate context in which to attempt the proof; if $P$ is a large structured presentation, such as the specification of a compiler, then in many cases $P'$ will be very much smaller than $P$. Dredging may be used to extract axioms from $P'$, and the proof is then carried out using these axioms by employing the following rule:

$$\frac{P \Vdash^{\mathcal{L}}_{\Sigma} \phi_1 \quad \cdots \quad P \Vdash^{\mathcal{L}}_{\Sigma} \phi_n \qquad \phi_1, \ldots, \phi_n \vdash^{\mathcal{L}}_{\Sigma} \phi}{P \Vdash^{\mathcal{L}}_{\Sigma} \phi} \qquad \text{(CUT)}$$

Alternatively, a small set of lemmas may be formulated from which (the translated version of) $\phi$ can be proved (again using CUT to lift this proof to the level of structured presentations); each of these lemmas may then be proved in $P'$ separately, perhaps by means of further diving and dredging. Lemma formation may be unavoidable if $P'$ involves union or translate with nested subpresentations involving derive.

The first step of this strategy is to find a subpresentation $P'$ of $P$ containing information relevant to the goal $\phi$. The search for an appropriate $P'$ is helped by the fact that translate forms an effective barrier to diving "too deeply": given a presentation translate $P_1$ along $\sigma : \Sigma_1 \to \Sigma$ and a $\Sigma$-sentence $\phi$, diving down to the level of $P_1$ will be impossible if $\phi$ is not expressible in the vocabulary which $\Sigma_1$ provides, i.e. if there is no $\phi_1 \in |\mathcal{L}|_{\Sigma_1}$ such that $\mathcal{L}(\sigma)(\phi_1) = \phi$. A common pattern is (translate $P_1$ along $\sigma_1$) $\cup$ (translate $P_2$ along $\sigma_2$), where the two uses of translate "guard" the branches of the union against inappropriate diving. Because translate has this effect, in a well-structured theory presentation it is often possible to find an approximately correct subpresentation $P'$ by simple depth-first search. See [SB83] for concrete details of the above strategies in the context of the Edinburgh LCF system.

**Example 4.3** The following structured theory presentation over $\mathcal{EQ}$ specifies symbol tables for an Algol-like programming language with nested blocks. See [GHM78] and [BG77] for variations on this well-known example. For convenience we make use of a presentation-structuring operation enrich which is defined in terms of union and translate as follows:

$$\text{enrich } P \text{ by types } T \text{ constants } C \text{ functions } F \text{ axioms } A = (\text{translate } P \text{ along } \iota) \cup (\Sigma, A)$$

where $\Sigma = \mathbf{Sg}^{\mathcal{EQ}}(P) \cup (T, C \cup F)$ and $\iota : \mathbf{Sg}^{\mathcal{EQ}}(P) \to \Sigma$ is the inclusion.

$Item$ = type $item$

$Stack$ = enrich $Item$ by
- type $stack$
- constant $nilstack : stack$
- functions $push : item \times stack \to stack$
  $pop : stack \to stack$
  $top : stack \to item$
- axioms $\forall x{:}item, s{:}stack.\ pop(push(x, s)) = s$
  $\forall x{:}item, s{:}stack.\ top(push(x, s)) = x$

$Boolean$ = type $bool$
- constants $true : bool$
  $false : bool$
- functions $not : bool \to bool$
- axioms $not(true) = false$
  $not(false) = true$

$Index$ = enrich $Boolean$ by
- type $index$
- function $eq : index \times index \to bool$

$Cond$ = enrich $Boolean$ by
- type $value$
- function $cond : bool \times value \times value \to value$
- axioms $\forall v, v'{:}value.\ cond(true, v, v') = v$
  $\forall v, v'{:}value.\ cond(false, v, v') = v'$

$Array$ = enrich (translate $Index$ along $\iota$) $\cup$ (translate $Cond$ along $\iota'$) by
- type $array$
- constant $nilarray : array$
- functions $put : index \times value \times array \to array$
  $get : index \times array \to value$
  $present : index \times array \to bool$
- axioms $\forall i{:}index, v{:}value, a{:}array.\ get(i, put(i, v, a)) = v$
  $\forall i, j{:}index, v{:}value, a{:}array.\ not(eq(i, j)) \Rightarrow get(i, put(j, v, a)) = get(i, a)$
  $\forall i{:}index.\ present(i, nilarray) = false$
  $\forall i{:}index, v{:}value, a{:}array.\ present(i, put(i, v, a)) = true$
  $\forall i, j{:}index, v{:}value, a{:}array.\ not(eq(i, j)) \Rightarrow present(i, put(j, v, a)) = present(i, a)$

Here, $\iota$ and $\iota'$ are the inclusions of $\mathbf{Sg}^{\mathcal{EQ}}(Index)$ and $\mathbf{Sg}^{\mathcal{EQ}}(Cond)$ respectively into $\mathbf{Sg}^{\mathcal{EQ}}(Index) \cup \mathbf{Sg}^{\mathcal{EQ}}(Cond)$, and $b \Rightarrow t = t'$ is an abbreviation for $cond(b, t, t') = t'$.

There is a $\mathbf{ThPres}^{\mathcal{EQ}}$ inclusion $Item \hookrightarrow Stack$, and a $\mathbf{ThPres}^{\mathcal{EQ}}$ morphism $\sigma : Item \to Array$ given by $\sigma(item) = array$. Thus *Stack* is parametric in *Item* and *Stack* can be applied to *Array*.

$$ArrayStack = Stack(Array[\sigma])$$

$$\begin{array}{lll} SymTab1 & = & \textsf{enrich } ArrayStack \textsf{ by} \\ & & \textsf{functions} \quad add : index \times value \times stack \to stack \\ & & \qquad\qquad retrieve : index \times stack \to value \\ & & \qquad\qquad enterblock : stack \to stack \\ & & \qquad\qquad leaveblock : stack \to stack \\ & & \textsf{axioms} \quad \forall i{:}index, v{:}value, s{:}stack.\ add(i,v,s) = push(put(i,v,top(s)), pop(s)) \\ & & \qquad\qquad \forall i{:}index, a{:}array, s{:}stack.\ present(i,a) \Rightarrow retrieve(i, push(a,s)) = get(i,a) \\ & & \qquad\qquad \forall i{:}index, a{:}array, s{:}stack. \\ & & \qquad\qquad\qquad not(present(i,a)) \Rightarrow retrieve(i, push(a,s)) = retrieve(i,s) \\ & & \qquad\qquad \forall s{:}stack.\ enterblock(s) = push(nilarray, s) \\ & & \qquad\qquad \forall s{:}stack.\ leaveblock(s) = pop(s) \end{array}$$

$$SymTab = \textsf{derive } SymTab1 \textsf{ via } \sigma'$$

where $\sigma' : \Sigma \to \mathbf{Sg}^{\mathcal{EQ}}(SymTab1)$ is defined by $\sigma'(symtab) = stack$, $\sigma'(niltable) = nilstack$, and $\sigma'(x) = x$ for all other symbols $x$ in $\Sigma$, where:

$$\begin{array}{lll} \Sigma & = & \textsf{types} \quad symtab, index, value, bool \\ & & \textsf{constants} \quad true : bool \\ & & \qquad\qquad false : bool \\ & & \qquad\qquad niltable : symtab \\ & & \textsf{functions} \quad eq : index \times index \to bool \\ & & \qquad\qquad add : index \times value \times symtab \to symtab \\ & & \qquad\qquad retrieve : index \times symtab \to value \\ & & \qquad\qquad enterblock : symtab \to symtab \\ & & \qquad\qquad leaveblock : symtab \to symtab \end{array}$$

Expanding all the uses of enrich and the single use of application, we obtain a structured theory presentation containing eight basic presentations, seven uses of union, nine uses of translate and one use of derive. All of the signature morphisms involved are inclusions except for $\sigma'$ (used in the final derive step) and one which arises from the application of *Stack* to *Array*.

*SymTab* is expressible as a basic presentation so it is possible in principle to dredge up all the information it contains. In practice this would be difficult: most of the axioms in the subpresentations of *SymTab* are not directly expressible in the signature $\mathbf{Sg}^{\mathcal{EQ}}(SymTab)$, so dredging would lose information unless appropriate theorems which *are* expressible in that signature are first formulated and proved in these subpresentations.

Now, suppose that we wish to prove that

$$SymTab \Vdash^{\mathcal{EQ}}_{\mathbf{Sg}^{\mathcal{EQ}}(SymTab)} \forall i{:}index, s{:}symtab.\ retrieve(i, leaveblock(enterblock(s))) = retrieve(i,s)$$

According to DERIVE, it is sufficient to prove that

$$SymTab1 \Vdash^{\mathcal{EQ}}_{\mathbf{Sg}^{\mathcal{EQ}}(SymTab1)} \forall i{:}index, s{:}stack.\ retrieve(i, leaveblock(enterblock(s))) = retrieve(i,s)$$

(This reduction can be viewed as diving to the level of *SymTab1*.) One way to proceed is to prove the following lemma, from which the desired result follows by substitutivity:

$$SymTab1 \Vdash^{\mathcal{EQ}}_{\mathbf{Sg}^{\mathcal{EQ}}(SymTab1)} \forall s{:}stack.\ leaveblock(enterblock(s)) = s$$

The last two axioms in the enrichment used to build *SymTab1* can be dredged up to the level of *SymTab1* (by UNION-RIGHT) and used to reduce the goal to:

$$SymTab1 \Vdash^{\mathcal{EQ}}_{\mathbf{Sg}^{\mathcal{EQ}}(SymTab1)} \forall s{:}stack.\ pop(push(nilarray, s)) = s$$

We can dive down to the level of *ArrayStack* with this goal (using UNION-LEFT followed by TRANSLATE) to obtain the goal:

$$ArrayStack \Vdash^{\mathcal{EQ}}_{\mathbf{Sg}^{\mathcal{EQ}}(ArrayStack)} \forall s{:}stack.\ pop(push(nilarray, s)) = s$$

It is not possible to dive down to the level of *Stack* with this goal: the use of translate (along $p(\sigma, \mathbf{Sg}^{\mathcal{EQ}}(Stack))$, implicit in the application of *Stack* to *Array*) acts as a barrier, since there is no constant corresponding to *nilarray* in *Stack*. But it is sufficient to prove:

$$ArrayStack \Vdash^{\mathcal{EQ}}_{\mathbf{Sg}^{\mathcal{EQ}}(ArrayStack)} \forall x{:}array, s{:}stack.\ pop(push(x, s)) = s$$

and it *is* possible to dive down to the level of *Stack* with this goal (using UNION-LEFT, TRANSLATE, UNION-RIGHT), where this is found to be an axiom in the enrichment used to build *Stack*.

The above procedure represents a successful LCF-style top-down goal-directed search for a proof of the original theorem; such a proof may now be obtained by applying the corresponding inference rules, proceeding bottom-up.

# 5 Logic Representations

The next issue to address is the sense in which one logical system can be represented or encoded in terms of another logical system. The essence of such a representation is a mapping from the sentences of the first system to those of the second, in such a way that consequence is accurately preserved.

**Definition 5.1** *A* morphism of logics $\gamma : \mathcal{L} \to \mathcal{L}'$ *is a pair* $(\gamma^{Sig}, \gamma^{CR})$ *where* $\gamma^{Sig} : \mathbf{Sig}^{\mathcal{L}} \to \mathbf{Sig}^{\mathcal{L}'}$ *is a functor and* $\gamma^{CR} : \mathcal{L} \dot{\to} \gamma^{Sig}; \mathcal{L}' : \mathbf{Sig}^{\mathcal{L}} \to \mathbf{CR}$ *is a natural transformation. The identity is the pair consisting of the identity functor on* $\mathbf{Sig}^{\mathcal{L}}$ *and the identity natural transformation on* $\mathcal{L}$*. Composition is defined by*[4]

$$(\gamma_1^{Sig}, \gamma_1^{CR}); (\gamma_2^{Sig}, \gamma_2^{CR}) = (\gamma_1^{Sig}; \gamma_2^{Sig},\ \gamma_1^{CR}; (\gamma_1^{Sig}; \gamma_2^{CR})).$$

**Log** *is the category of logics and logic morphisms.*

A morphism of logics is to be thought of as an "encoding" of one logical system in another in such a way that consequence is preserved. Let $\gamma : \mathcal{L} \to \mathcal{L}'$ be a morphism of logics. The requirement that $\gamma^{CR}$ be a natural transformation may be expressed by the equation

$$\gamma^{CR}(\sigma(\phi)) = \gamma^{Sig}(\sigma)(\gamma^{CR}(\phi)).$$

In words: it doesn't matter whether we encode the translation $\sigma(\phi)$ of $\phi$, or translate the encoding $\gamma^{CR}(\phi)$ of $\phi$ along the encoding $\gamma^{Sig}(\sigma)$ of $\sigma$. To simplify notation, we write $\gamma(\Sigma)$ for $\gamma^{Sig}(\Sigma)$, and $\gamma(\phi)$ for $\gamma^{CR}_{\Sigma}(\phi)$ (for appropriate choice of $\Sigma$.)

**Proposition 5.2** *If* $\gamma : \mathcal{L} \to \mathcal{L}'$ *is a logic morphism,* $\Sigma$ *is an* $\mathcal{L}$*-signature, and* $\Phi \subseteq |\mathcal{L}|_{\Sigma}$*, then*

$$\gamma(\overline{\Phi}) \subseteq \overline{\gamma(\Phi)}.$$

[4]We use ";" to denote not only composition in a category (*e.g.*, the usual composition of functions and functors) written in diagrammatic order, but also both vertical composition of natural transformations and the composition of a natural transformation with a functor so that $(\gamma_1^{CR}; (\gamma_1^{Sig}; \gamma_2^{CR}))_{\Sigma} = (\gamma_1^{CR})_{\Sigma}; (\gamma_2^{CR})_{\gamma_1^{Sig}(\Sigma)}$.

Similarly as in Proposition 2.9, the containment is in general proper since the image of a theory under a logic morphism need not be a theory. However, it follows from the above proposition that $\overline{\gamma(\overline{\Phi})} = \overline{\gamma(\Phi)}$.

Further requirements on a logic morphism $\gamma : \mathcal{L} \to \mathcal{L}'$ must be imposed to ensure that consequence in $\mathcal{L}'$ for translated sentences is sound with respect to consequence in $\mathcal{L}$.

**Definition 5.3** *A logic morphism $\gamma : \mathcal{L} \to \mathcal{L}'$ is a* representation *iff $\gamma^{Sig}$ is an embedding and each $\gamma^{CR}_\Sigma$ is conservative. A representation is* surjective *iff each $\gamma^{CR}_\Sigma$ is surjective as a function on the underlying sets. A logic $\mathcal{L}$ is* representable *in a logic $\mathcal{L}'$ iff there is a representation $\rho : \mathcal{L} \to \mathcal{L}'$.*

Let $\rho$ range over representations. It is easy to see that identities are representations and that the composition of two representations is again a representation. Thus representations form a subcategory of **Log**. The requirement that $\rho^{Sig}$ be an embedding implies that the category of signatures of the source logic is faithfully encoded in the target logic, and the requirement of conservativity implies that each consequence relation of the source is fully and faithfully encoded in the target. Thus if $\rho : \mathcal{L} \to \mathcal{L}'$ is a logic representation then $\Phi \vdash^{\mathcal{L}}_{\Sigma} \phi$ iff $\rho\Phi \vdash^{\mathcal{L}'}_{\rho\Sigma} \rho\phi$. Note that surjectivity of $\rho$ does not entail that $\rho^{Sig}$ be full, only that $\rho^{CR}$ be onto.

**Example 5.4** For any algebraic signature $\Sigma$, there is an inclusion $\iota_\Sigma$ of the set $|\mathcal{EQ}(\Sigma)|$ of $\Sigma$-equations into the set $|\mathcal{FOEQ}(\Sigma)|$ of first-order $\Sigma$-sentences. These inclusions preserve consequence: if a set of $\Sigma$-equations $\Delta$ entails a $\Sigma$-equation $\phi$ in $\mathcal{EQ}$ then $\iota_\Sigma(\Delta)$ entails $\iota_\Sigma(\phi)$ in $\mathcal{FOEQ}$, and so $\iota_\Sigma : \mathcal{EQ}(\Sigma) \to \mathcal{FOEQ}(\Sigma)$ is a CR morphism. The opposite implication holds as well, *i.e.* $\iota_\Sigma$ is conservative. Moreover, $\iota$ is compatible with the translation of $\Sigma$-equations under signature morphisms $\sigma : \Sigma \to \Sigma'$. Thus we have a logic representation $\rho : \mathcal{EQ} \to \mathcal{FOEQ}$ where $\rho^{Sig}$ is the identity functor and for each signature $\Sigma$, $\rho^{CR}_\Sigma = \iota_\Sigma$ as defined above. Notice that $\rho$ is (obviously) not surjective.

Proposition 5.2 of course holds for representations as well, but they also satisfy a stronger property:

**Proposition 5.5** *If $\rho : \mathcal{L} \to \mathcal{L}'$ is a representation, $\Sigma$ is an $\mathcal{L}$-signature, and $\Phi \subseteq |\mathcal{L}|_\Sigma$, then*

$$\overline{\Phi} = \rho^{-1}(\overline{\rho(\Phi)})$$

*where $\rho^{-1}(\Psi)$ is the co-image of $\Psi$ under $\rho$.*

If $\rho$ is a representation of $\mathcal{L}$ in $\mathcal{L}'$, then we may use $\mathcal{L}'$ as an "inference engine" for $\mathcal{L}$. We would like to consider how this interacts with the ideas in the previous section concerning proofs in structured theory presentations in $\mathcal{L}$. The first step is to use the proof system introduced in Definition 4.1, replacing all uses of $\Delta \vdash^{\mathcal{L}}_{\Sigma} \phi$ by $\rho(\Delta) \vdash^{\mathcal{L}'}_{\rho(\Sigma)} \rho(\phi)$. Then, the proof methodology discussed in the previous section need not make any use of $\mathcal{L}$ for elementary inference. The proof process is still, however, driven by an $\mathcal{L}$-presentation $P$, and so involves the sentences, signatures and translations induced by signature morphisms of $\mathcal{L}$. But if our goal is to reduce all inferential activity in $\mathcal{L}$ to inferential activity in $\mathcal{L}'$, then we would like to "lift" $P$ to an $\mathcal{L}'$-presentation, and perform structured proof in $\mathcal{L}'$ guided by the lifted presentation. To make this precise, we first define a natural lifting of presentations.

**Definition 5.6** *Suppose that $\rho : \mathcal{L} \to \mathcal{L}'$ is a representation, and let $P$ be an $\mathcal{L}$-presentation with signature $\Sigma$. The* representation of $P$ in $\mathcal{L}'$ wrt $\rho$ *is given by the following function defined by induction on the structure of $P$:*

$$\begin{aligned}
\tilde{\rho}((\Sigma, \Phi)) &= (\rho^{Sig}(\Sigma), \rho^{CR}_\Sigma(\Phi)) \\
\tilde{\rho}(P_1 \cup P_2) &= \tilde{\rho}(P_1) \cup \tilde{\rho}(P_2) \\
\tilde{\rho}(\mathsf{translate}\ P_1\ \mathsf{along}\ \sigma) &= \mathsf{translate}\ \tilde{\rho}(P_1)\ \mathsf{along}\ \rho^{Sig}(\sigma) \\
\tilde{\rho}(\mathsf{derive}\ P_1\ \mathsf{via}\ \sigma) &= \mathsf{derive}\ \tilde{\rho}(P_1)\ \mathsf{via}\ \rho^{Sig}(\sigma)
\end{aligned}$$

**Proposition 5.7** *If $P$ is an $\mathcal{L}$-presentation, then $\tilde{\rho}(P)$ is an $\mathcal{L}'$-presentation with $\mathbf{Sg}^{\mathcal{L}'}(\tilde{\rho}(P)) = \rho^{Sig}(\mathbf{Sg}^{\mathcal{L}}(P))$. Moreover, $\rho(\mathbf{Th}^{\mathcal{L}}(P)) \subseteq \mathbf{Th}^{\mathcal{L}'}(\tilde{\rho}(P))$.*
**Proof** *By straightforward induction on the structure of $P$.*

The above discussion suggests that in order to test $P \Vdash^{\mathcal{L}}_{\Sigma} \phi$, one should encode $P$ and $\phi$ in $\mathcal{L}'$, and test $\tilde{\rho}(P) \Vdash^{\mathcal{L}'}_{\rho(\Sigma)} \rho(\phi)$. We would hope that this strategy is sound and complete, that is, that the following conjecture holds.

**Conjecture 5.8** *Suppose that $\rho : \mathcal{L} \to \mathcal{L}'$ is a representation, and let $P$ be an $\mathcal{L}$-presentation with signature $\Sigma$ and $\phi \in |\mathcal{L}|_{\Sigma}$. Then $P \Vdash^{\mathcal{L}}_{\Sigma} \phi$ iff $\tilde{\rho}(P) \Vdash^{\mathcal{L}'}_{\rho(\Sigma)} \rho(\phi)$.*

Joining Proposition 5.7 with Proposition 4.2, we immediately obtain the implication from left to right (i.e. completeness). Unfortunately, the converse implication (i.e. soundness) fails, as the following counterexample illustrates.

**Counterexample 5.9** Consider the following presentations in $\mathcal{EQ}$. Let $\Sigma_0$ be the signature

| | |
|---|---|
| types | $s, s'$ |
| constants | $a : s$ |
| | $b, c : s'$ |

Let $\Sigma$ be the same signature with $a$ removed, and let $\iota : \Sigma \hookrightarrow \Sigma_0$ be the corresponding signature inclusion. Let $P_0 = (\Sigma_0, \emptyset)$, $P_1 = \mathsf{derive}\ P_0\ \mathsf{via}\ \iota$ and $P = P_1 \cup (\Sigma, \{\forall x{:}s.\ b = c\})$.

Then, $\mathbf{Th}^{\mathcal{EQ}}(P)$ does not include $b = c$ since, in the context of the signature $\Sigma$, $\forall x{:}s.\ b = c$ does not entail $b = c$.

Recall that we have a logic representation $\rho : \mathcal{EQ} \to \mathcal{FOEQ}$. Then, $\mathbf{Th}^{\mathcal{FOEQ}}(\tilde{\rho}(P_0))$ contains the sentence $\exists x{:}s.\ true$. Consequently, $\exists x{:}s.\ true \in \mathbf{Th}^{\mathcal{FOEQ}}(\tilde{\rho}(P_1))$, and $b = c \in \mathbf{Th}^{\mathcal{FOEQ}}(\tilde{\rho}(P))$, since $\exists x{:}s.\ true, \forall x{:}s.\ b = c \vdash^{\mathcal{FOEQ}}_{\Sigma} b = c$.

In terms of entailment relations, we have that $\tilde{\rho}(P) \Vdash^{\mathcal{FOEQ}}_{\Sigma} b = c$ even though $P \Vdash^{\mathcal{EQ}}_{\Sigma} b = c$ does not hold.

A simpler but even more contrived counterexample may be found in [HST89a] (Counterexample 4.6).

The source of this failure of equivalence is a discrepancy between $\mathbf{Th}^{\mathcal{L}}(P)$ and $\mathbf{Th}^{\mathcal{L}'}(\tilde{\rho}(P))$. The use of the derive operation may cause the following crucial property to be lost:

$$\mathbf{Th}^{\mathcal{L}'}(\tilde{\rho}(P)) \subseteq \overline{\rho(\mathbf{Th}^{\mathcal{L}}(P))}.$$

(The reverse inclusion follows directly from Proposition 5.7.) In the counterexample, the sentence $\exists x{:}s.\ true$ is a witness to the failure of this containment, but is not, by itself, sufficient to refute the conjecture, for it lies outside of the image of $\rho$. The union operation used in the counterexample to construct $P$ "exploits" this discrepancy to produce a sentence in the "lifted" theory that lies within the image of $\rho$, and hence refutes the conjecture. The translate operation may be used instead of union to exploit the discrepancy created by derive in a similar manner. But neither union nor translate are able to create such a discrepancy. In fact, if an $\mathcal{L}$-presentation $P$ does not involve derive, then $\mathbf{Th}^{\mathcal{L}'}(\tilde{\rho}(P)) = \overline{\rho(\mathbf{Th}^{\mathcal{L}}(P))}$.

**Proposition 5.10** *Consider an arbitrary representation $\rho : \mathcal{L} \to \mathcal{L}'$.*

1. *Suppose $(\Sigma, \Phi)$ is a well-formed basic $\mathcal{L}$-presentation. Then*

$$\mathbf{Th}^{\mathcal{L}'}(\tilde{\rho}((\Sigma, \Phi))) = \overline{\rho(\mathbf{Th}^{\mathcal{L}}((\Sigma, \Phi)))}.$$

2. *Suppose $P, P'$ are well-formed $\mathcal{L}$-presentations with the same signature satisfying* $\mathbf{Th}^{\mathcal{L}'}(\tilde{\rho}(P)) = \overline{\rho(\mathbf{Th}^{\mathcal{L}}(P))}$ *and* $\mathbf{Th}^{\mathcal{L}'}(\tilde{\rho}(P')) = \overline{\rho(\mathbf{Th}^{\mathcal{L}}(P'))}$. *Then*

$$\mathbf{Th}^{\mathcal{L}'}(\tilde{\rho}(P \cup P')) = \overline{\rho(\mathbf{Th}^{\mathcal{L}}(P \cup P'))}.$$

3. *Suppose $P$ is a well-formed $\mathcal{L}$-presentation satisfying* $\mathbf{Th}^{\mathcal{L}'}(\tilde{\rho}(P)) = \overline{\rho(\mathbf{Th}^{\mathcal{L}}(P))}$ *and* $\sigma : \mathbf{Sg}^{\mathcal{L}}(P) \to \Sigma$. *Then*

$$\mathbf{Th}^{\mathcal{L}'}(\tilde{\rho}(\mathsf{translate}\ P\ \mathsf{along}\ \sigma)) = \overline{\rho(\mathbf{Th}^{\mathcal{L}}(\mathsf{translate}\ P\ \mathsf{along}\ \sigma))}.$$

4. *Suppose $P$ is a well-formed $\mathcal{L}$-presentation satisfying* $\mathbf{Th}^{\mathcal{L}'}(\tilde{\rho}(P)) = \overline{\rho(\mathbf{Th}^{\mathcal{L}}(P))}$ *and* $\sigma : \Sigma \to \mathbf{Sg}^{\mathcal{L}}(P)$. *Then*

$$\mathbf{Th}^{\mathcal{L}'}(\tilde{\rho}(\mathsf{derive}\ P\ \mathsf{via}\ \sigma)) \supseteq \overline{\rho(\mathbf{Th}^{\mathcal{L}}(\mathsf{derive}\ P\ \mathsf{via}\ \sigma))}.$$

*The inclusion may be proper. However, we have:*

$$\rho^{-1}(\mathbf{Th}^{\mathcal{L}'}(\tilde{\rho}(\mathsf{derive}\ P\ \mathsf{via}\ \sigma))) = \rho^{-1}\left(\overline{\rho(\mathbf{Th}^{\mathcal{L}}(\mathsf{derive}\ P\ \mathsf{via}\ \sigma))}\right) \quad (= \mathbf{Th}^{\mathcal{L}}(\mathsf{derive}\ P\ \mathsf{via}\ \sigma)).$$

Since the most obvious approach fails as explained above, is there a sense in which we can lift structured presentations from $\mathcal{L}$ to $\mathcal{L}'$? The answer is given by considering an alternative definition of the theory of an $\mathcal{L}$-presentation that is conditioned by the representation $\rho$.

**Definition 5.11** *Let $\rho : \mathcal{L} \to \mathcal{L}'$ be a representation. For any well-formed $\mathcal{L}$-presentation with signature $\Sigma$, the $\mathcal{L}'$-*theory of $P$ wrt $\rho$, *written*[5] $\mathbf{Th}^{\rho}(P)$, *is defined as follows:*

$$\begin{aligned}
\mathbf{Th}^{\rho}((\Sigma, \Phi)) &= \overline{\rho(\Phi)} \\
\mathbf{Th}^{\rho}(P_1 \cup P_2) &= \overline{\mathbf{Th}^{\rho}(P_1) \cup \mathbf{Th}^{\rho}(P_2)} \\
\mathbf{Th}^{\rho}(\mathsf{translate}\ P_1\ \mathsf{along}\ \sigma) &= \overline{\rho(\sigma)(\mathbf{Th}^{\rho}(P_1))} \\
\mathbf{Th}^{\rho}(\mathsf{derive}\ P_1\ \mathsf{via}\ \sigma) &= \overline{\rho(|\mathcal{L}|_{\Sigma}) \cap \rho(\sigma)^{-1}(\mathbf{Th}^{\rho}(P_1))}
\end{aligned}$$

Note that we are defining the $\mathcal{L}'$-theory of an $\mathcal{L}$-presentation, conditioned by the representation $\rho$ of $\mathcal{L}$ in $\mathcal{L}'$. This is because in the case of derive, the restriction to the range of $\rho$ makes reference to the $\mathcal{L}$-signature $\Sigma$ of the $\mathcal{L}$-presentation $P$. Although this restriction ensures that only $\mathcal{L}$-sentence images are taken from $P$, the closure of the result under $\vdash^{\mathcal{L}'}$ admits non- $\mathcal{L}$-sentence images into the result. In effect, in the case of derive, only $\mathcal{L}$-sentence images are admitted as intermediate lemmas, whereas arbitrary $\mathcal{L}'$-sentences are admitted as consequences of these lemmas. This will be reflected in the proof search procedure associated with this definition.

**Theorem 5.12** *If $P$ is a well-formed $\mathcal{L}$-presentation, then $\mathbf{Th}^{\rho}(P)$ is an $\mathcal{L}'$-theory with signature $\rho(\mathbf{Sg}^{\mathcal{L}}(P))$ such that $\mathbf{Th}^{\rho}(P) = \overline{\rho(\mathbf{Th}^{\mathcal{L}}(P))}$.*

**Proof** *By a straightforward induction on the structure of $P$. The only interesting case is that of* derive*; the others follow from Proposition 5.10. Suppose that $P_1$ is a well-formed $\mathcal{L}$-presentation and $\sigma : \Sigma \to \mathbf{Sg}^{\mathcal{L}}(P_1)$. Using the inductive hypothesis $\mathbf{Th}^{\rho}(P_1) = \overline{\rho(\mathbf{Th}^{\mathcal{L}}(P_1))}$, we have to prove $\overline{\rho(|\mathcal{L}|_{\Sigma}) \cap \rho(\sigma)^{-1}(\mathbf{Th}^{\rho}(P_1))} = \overline{\rho(\sigma^{-1}(\mathbf{Th}^{\mathcal{L}}(P_1)))}$.*

$\subseteq$: *Take $\phi' \in \rho(|\mathcal{L}|_{\Sigma}) \cap \rho(\sigma)^{-1}(\mathbf{Th}^{\rho}(P_1))$. Then $\phi' = \rho(\phi)$, for some $\phi \in |\mathcal{L}|_{\Sigma}$, and $\rho(\sigma)(\phi') \in \overline{\mathbf{Th}^{\rho}(P_1)}$. By naturality of $\rho$ and by the inductive assumption, we have $\rho(\sigma(\phi)) \in \overline{\rho(\mathbf{Th}^{\mathcal{L}}(P_1))}$, that is for some $\Delta \subseteq \mathbf{Th}^{\mathcal{L}}(P_1)$, $\rho(\Delta) \vdash^{\mathcal{L}'}_{\rho(\Sigma)} \rho(\sigma(\phi))$. Since representations are conservative, this implies that $\Delta \vdash^{\mathcal{L}}_{\Sigma} \sigma(\phi)$ and so $\sigma(\phi) \in \mathbf{Th}^{\mathcal{L}}(P_1)$. Thus, $\phi' = \rho(\phi) \in \rho(\sigma^{-1}(\mathbf{Th}^{\mathcal{L}}(P_1)))$.*

[5] Since $\mathcal{L}$ and $\mathcal{L}'$ are implicit in $\rho$, this notation carries all the data involved.

$\supseteq$: *Take* $\phi' \in \rho(\sigma^{-1}(\mathbf{Th}^{\rho}(P_1)))$. *Then* $\phi' = \rho(\phi)$, *for some* $\phi \in \sigma^{-1}(\mathbf{Th}^{\mathcal{L}}(P_1))$, *that is* $\sigma(\phi) \in \mathbf{Th}^{\mathcal{L}}(P_1)$. *By the inductive assumption,* $\rho(\sigma(\phi)) \in \mathbf{Th}^{\rho}(P_1)$ *and since by naturality of* $\rho$, $\rho(\sigma)(\rho(\phi)) = \rho(\sigma(\phi))$, *we conclude that* $\phi' = \rho(\phi) \in (\rho(\sigma))^{-1}(\mathbf{Th}^{\rho}(P_1))$, *which completes the proof in this case since clearly* $\phi' \in \rho(|\mathcal{L}|_{\Sigma})$.

As in the previous section, Definition 5.11 induces a corresponding entailment relation, between $\mathcal{L}$-presentations $P$ and $\mathcal{L}'$-sentences $\phi'$ with signature $\rho(\mathbf{Sg}^{\mathcal{L}}(P))$.

**Definition 5.13** *We define a family of relations* $P \Vdash^{\rho}_{\Sigma} \phi$ *between well-formed* $\mathcal{L}$*-presentations* $P$ *with signature* $\Sigma$ *and sentences* $\phi' \in |\mathcal{L}'|_{\rho(\Sigma)}$ *by induction on the structure of* $P$ *as follows:*

1. $(\Sigma, \Phi) \Vdash^{\rho}_{\Sigma} \phi'$ *iff there exists* $\Delta \subseteq \Phi$ *such that* $\rho(\Delta) \vdash^{\mathcal{L}'}_{\rho(\Sigma)} \phi'$.
2. $P_1 \cup P_2 \Vdash^{\rho}_{\Sigma} \phi'$ *iff there exists* $\Delta'_1, \Delta'_2 \subseteq |\mathcal{L}'|_{\rho(\Sigma)}$ *such that* $P_1 \Vdash^{\rho}_{\Sigma} \Delta'_1$, $P_2 \Vdash^{\rho}_{\Sigma} \Delta'_2$, *and* $\Delta'_1, \Delta'_2 \vdash^{\mathcal{L}'}_{\rho(\Sigma)} \phi'$.
3. translate $P_1$ along $\sigma \Vdash^{\rho}_{\Sigma} \phi'$ *(where* $\sigma : \Sigma_1 \to \Sigma$*) iff there exists* $\Delta'_1 \subseteq |\mathcal{L}'|_{\rho(\Sigma)}$ *such that* $P_1 \Vdash^{\rho}_{\Sigma_1} \Delta'_1$ *and* $\rho(\sigma)(\Delta'_1) \vdash^{\mathcal{L}'}_{\rho(\Sigma)} \phi'$.
4. derive $P_1$ via $\sigma \Vdash^{\rho}_{\Sigma} \phi'$ *(where* $\sigma : \Sigma \to \Sigma_1$*) iff there exists* $\Delta' \subseteq \rho(|\mathcal{L}|_{\Sigma})$ *such that* $P_1 \Vdash^{\rho}_{\Sigma_1} \rho(\sigma)(\Delta')$ *and* $\Delta' \vdash^{\mathcal{L}'}_{\rho(\Sigma)} \phi'$.

*Here,* $P \Vdash^{\rho}_{\Sigma} \Delta'$ *stands for* $P \Vdash^{\rho}_{\Sigma} \phi'$ *for all* $\phi' \in \Delta'$.

**Proposition 5.14** *The relation* $P \Vdash^{\rho}_{\Sigma} \phi'$, *where* $P$ *is a well-formed* $\mathcal{L}$*-presentation with signature* $\Sigma$ *and* $\phi' \in |\mathcal{L}'|_{\rho(\Sigma)}$, *holds iff* $\phi' \in \mathbf{Th}^{\rho}(P)$.
**Proof** *By structural induction on* $P$, *directly from Definition 5.13 and 5.11.*

Theorem 5.12 may be restated in terms of the entailment relations we have introduced.

**Corollary 5.15** *For any* $\mathcal{L}$*-presentation* $P$ *with signature* $\Sigma$ *and* $\phi \in |\mathcal{L}|_{\Sigma}$, $P \Vdash^{\mathcal{L}}_{\Sigma} \phi$ *iff* $P \Vdash^{\rho}_{\Sigma} \rho(\phi)$.

Definition 5.13 provides the basis for a proof procedure for $\mathcal{L}'$ sentences relative to an $\mathcal{L}$-presentation. As we remarked above, we would like to achieve a complete reduction to $\mathcal{L}'$ by working with the representation $\tilde{\rho}(P)$ of $P$. The conditions under which we can achieve this may be derived by comparing the proof system determined by Definition 5.13 for $P \Vdash^{\rho}_{\Sigma} \rho(\phi)$ with that determined by Definition 4.1 for the case of $\tilde{\rho}(P) \Vdash^{\mathcal{L}'}_{\rho(\Sigma)} \rho(\phi)$.

First, if we restrict attention to $\mathcal{L}'$-sentences $\phi'$ in the image of $\rho$ (*i.e.*, such that there exists an $\mathcal{L}$-sentence $\phi$ with $\phi' = \rho(\phi)$), then case (4) of Definition 5.13 may be simplified to

$$\text{derive } P_1 \text{ via } \sigma \Vdash^{\rho}_{\Sigma} \phi' \text{ iff } P_1 \Vdash^{\rho}_{\Sigma_1} \rho(\sigma)(\phi'),$$

since $\rho(\sigma)(\phi') = \rho(\sigma)(\rho(\phi)) = \rho(\sigma(\phi))$ (the last step by naturality), and so we can take $\Delta' = \{ \rho(\phi) \}$, for which the condition $\Delta' \vdash^{\mathcal{L}'}_{\rho(\Sigma)} \rho(\phi)$ is trivial. Thus if $\rho$ were surjective, then the proof procedure given by Definition 5.13 would be essentially identical to the ordinary proof procedure, except that it is guided by an $\mathcal{L}$-presentation:

**Corollary 5.16** *Let* $\rho : \mathcal{L} \to \mathcal{L}'$ *be a surjective representation. Then for any well-formed* $\mathcal{L}$*-presentation* $P$ *with signature* $\Sigma$ *and* $\mathcal{L}$*-sentence* $\phi \in |\mathcal{L}|_{\Sigma}$, $P \Vdash^{\mathcal{L}}_{\Sigma} \phi$ *iff* $\tilde{\rho}(P) \Vdash^{\mathcal{L}'}_{\rho(\Sigma)} \rho(\phi)$.

Note, however, that requiring surjectivity is a rather strong restriction. As we shall see below, in practical situations it is necessary to admit the use of $\mathcal{L}'$-sentences lying outside of the range of $\rho$ as intermediate lemmas in the process of proving sentences lying within the range of $\rho$. It is therefore important to admit arbitrary $\mathcal{L}'$ sentences as goals of the "lifted" proof procedure.

Second, although the proof procedure induced by Definition 5.13 is guided by an $\mathcal{L}$-presentation $P$, it does not make direct use of any of the components of $P$, but rather only of their representations in $\mathcal{L}'$. For example, in the case of translate, the procedure applies $\rho(\sigma)$, not $\sigma$ (*i.e.*, $\mathcal{L}'(\rho(\sigma))$, not $\mathcal{L}(\sigma)$). In a sense the proof procedure forms $\tilde{\rho}(P)$ "on the fly," taking the representations of each component of $P$ in order to carry out the proof. The essential difference between an $\mathcal{L}'$ proof guided by $\tilde{\rho}(P)$ and the above $P$-guided proof procedure lies in the restriction on $\Delta'$ in the case of derive. To enforce this restriction, the proof procedure must be able to decide, given $\rho$ and $\phi' \in |\mathcal{L}'|_{\rho(\Sigma)}$, whether $\phi' = \rho(\phi)$ for some $\phi \in |\mathcal{L}|_\Sigma$. Such a test requires only the signature $\Sigma$ of derive $P$ via $\sigma$ and the representation $\rho$. But since $\rho$ is a representation, the component $\rho^{Sig}$ is an embedding, and hence $\Sigma$ is determined by $\rho(\Sigma)$. Therefore no $\mathcal{L}$-entities are needed; it is enough to have the image $\rho(\Sigma)$ of $\Sigma$. We may therefore use $\tilde{\rho}(P)$ to guide the proof, provided that $\rho$ is a representation and we can test membership in the range of $\rho$. To make this explicit, let us introduce yet another entailment relation.

**Definition 5.17** *Let $\mathcal{L}$ be a logical system, and let $\mathcal{S}$ be a family of sets $\mathcal{S}_\Sigma \subseteq |\mathcal{L}|_\Sigma$ for $\Sigma \in \mathbf{Sig}^{\mathcal{L}}$. We define a family of relations $P \Vdash_\Sigma^{\mathcal{L},\mathcal{S}} \phi$ between well-formed $\mathcal{L}$-presentations $P$ with signature $\Sigma$ and sentences $\phi \in |\mathcal{L}|_\Sigma$ by induction on the structure of $P$ as follows:*

1. $(\Sigma, \Phi) \Vdash_\Sigma^{\mathcal{L},\mathcal{S}} \phi$ *iff there exists* $\Delta \subseteq \Phi$ *such that* $\Delta \vdash_\Sigma^{\mathcal{L}} \phi$.
2. $P_1 \cup P_2 \Vdash_\Sigma^{\mathcal{L},\mathcal{S}} \phi$ *iff there exists* $\Delta_1 \subseteq |\mathcal{L}|_\Sigma$ *and* $\Delta_2 \subseteq |\mathcal{L}|_\Sigma$ *such that* $P_1 \Vdash_\Sigma^{\mathcal{L},\mathcal{S}} \Delta_1$, $P_2 \Vdash_\Sigma^{\mathcal{L},\mathcal{S}} \Delta_2$, *and* $\Delta_1, \Delta_2 \vdash_\Sigma^{\mathcal{L}} \phi$.
3. translate $P_1$ along $\sigma \Vdash_\Sigma^{\mathcal{L},\mathcal{S}} \phi$ *(where $\sigma : \Sigma_1 \to \Sigma$) iff there exists* $\Delta_1 \subseteq |\mathcal{L}|_{\Sigma_1}$ *such that* $P_1 \Vdash_{\Sigma_1}^{\mathcal{L},\mathcal{S}} \Delta_1$ *and* $\mathcal{L}(\sigma)(\Delta_1) \vdash_\Sigma^{\mathcal{L}} \phi$.
4. derive $P_1$ via $\sigma \Vdash_\Sigma^{\mathcal{L},\mathcal{S}} \phi$ *(where $\sigma : \Sigma \to \Sigma_1$) iff there exists* $\Delta \subseteq \mathcal{S}_\Sigma$ *such that* $P_1 \Vdash_{\Sigma_1}^{\mathcal{L},\mathcal{S}} \sigma(\Delta)$ *and* $\Delta \vdash^{\mathcal{L}} \phi$.

*Here, $P \Vdash_\Sigma^{\mathcal{L},\mathcal{S}} \Delta$ stands for $P \Vdash_\Sigma^{\mathcal{L},\mathcal{S}} \phi$ for all $\phi \in \Delta$.*

**Corollary 5.18** *Let $\rho : \mathcal{L} \to \mathcal{L}'$ be a representation and let $\mathcal{S}'$ be a family of sets of $\mathcal{L}'$-sentences indexed by $\mathbf{Sig}^{\mathcal{L}'}$ such that $\mathcal{S}'_{\rho(\Sigma)} = \rho(|\mathcal{L}|_\Sigma)$ for $\Sigma \in \mathbf{Sig}^{\mathcal{L}}$. Then for any well-formed $\mathcal{L}$-presentation $P$ with signature $\Sigma$ and $\mathcal{L}$-sentence $\phi \in |\mathcal{L}|_\Sigma$, $P \Vdash_\Sigma^{\mathcal{L}} \phi$ iff $\tilde{\rho}(P) \Vdash_{\rho(\Sigma)}^{\mathcal{L}',\mathcal{S}'} \rho(\phi)$.*
**Proof** *By Corollary 5.16 and Definition 5.17.*

**Example 5.19** Recall Counterexample 5.9. In the context of the definitions given there, we do not have that $\tilde{\rho}(P) \Vdash_\Sigma^{\mathcal{FOEQ},\mathcal{S}'} b = c$ where $\mathcal{S}'_\Sigma = \rho(\mathcal{EQ}_\Sigma)$. The reason for this is that the proof of $\tilde{\rho}(P) \Vdash_\Sigma^{\mathcal{FOEQ}} b = c$ cannot be repeated here: the necessary mediating formula $\exists x{:}s.\ true$ is filtered out since it is not a representation of an equation ($\exists x{:}s.\ true \notin \mathcal{S}'_\Sigma$). This illustrates the difference between $\Vdash_\Sigma^{\mathcal{FOEQ}}$ (used in Counterexample 5.9) and $\Vdash_\Sigma^{\mathcal{FOEQ},\mathcal{S}'}$.

To assess the practical implications of the requirement to keep track of the image of $\rho$ and to check (in the case of derive) whether a sentence is in this image, we turn in the next section to the representation of logics in LF.

# 6 Logical Systems and LF

We refer to [HHP93] for a complete definition of LF. A complete understanding of the detailed technicalities below requires a reasonable acquaintance with the intricacies of the LF type theory. However, we hope that the general ideas are intelligible even without such background.

In order to discuss representations of logical systems in LF, we first define the logical system associated with the LF type theory. The basic form of assertion in this logic is that a closed type is inhabited. The restriction to closed types is a simplification that suffices for the purposes of this paper, but would have to be relaxed in practice. (See Section 10 and [HST89b] for further discussion.)

An LF signature consists of a sequence of declarations of constants and types (and type families). The former are written as $c{:}A$ where $A$ is the type of $c$, and the latter as $c{:}\mathsf{Type}$ (or for type families indexed by elements of types $A_1, \ldots, A_n$, $c : \Pi x_1{:}A_1. \ldots . \Pi x_n{:}A_n.\mathsf{Type}$). See [HHP93] for examples.

**Definition 6.1** *An* LF signature morphism $\sigma : \Sigma_1 \to \Sigma_2$ *is a function* $\sigma$ *mapping constants to closed terms such that if* $c{:}A$ *(*$c{:}K$*) occurs in* $\Sigma_1$*, then* $\vdash_{\Sigma_2} \sigma(c) : \sigma^\sharp A$ *(*$\vdash_{\Sigma_2} \sigma(c) : \sigma^\sharp K$*). (The function* $\sigma^\sharp$ *is the natural extension of* $\sigma$ *to LF terms.) The identity morphism on* $\Sigma$ *is the identity map, and composition is defined by* $\sigma_1;\sigma_2 = \sigma_1;\sigma_2^\sharp$*. Inclusions are the inclusion functions on the underlying sets of constants.* $\mathbf{Sig}^{\mathcal{LF}}$ *is the category of LF signatures and LF signature morphisms.*

Note that if $\iota$ is an inclusion between LF signatures, then $\iota^\sharp$ is an inclusion, and hence is usually omitted.

The following proposition expresses the stability of the assertions of the LF type theory under change of signature, which is the crucial fact used to justify the well-formedness of the definitions given throughout the rest of this section; detailed proofs are omitted.

**Proposition 6.2** *If* $\sigma : \Sigma_1 \to \Sigma_2$*, and* $\vdash_{\Sigma_1} \alpha$*, then* $\vdash_{\Sigma_2} \sigma^\sharp\alpha$ *for each assertion* $\alpha$ *of the LF type system.*

**Definition 6.3** *Let* $\Sigma$ *be an LF signature.* $\mathcal{LF}(\Sigma)$ *is the consequence relation* $(\mathrm{Types}_\Sigma, \vdash_\Sigma^{\mathcal{LF}})$ *where* $\mathrm{Types}_\Sigma = \{\, A \mid \vdash_\Sigma A : \mathsf{Type} \,\}$ *and*

$$A_1, \ldots, A_n \vdash_\Sigma^{\mathcal{LF}} A \quad \textit{iff} \quad x_1{:}A_1, \ldots, x_n{:}A_n \vdash_\Sigma M : A$$

*for some* $M$ *and any pairwise distinct variables* $x_1, \ldots, x_n$*. That is,* $\mathcal{LF}(\Sigma)$ *is the set of closed LF types over the signature* $\Sigma$*, with the consequence relation induced by type inhabitation.*

This consequence relation has a straightforward Gentzen-style axiomatization similar to that used in NuPRL [Con86] that may be used as the basis for interactive proof search.

This construction extends to a functor in a straightforward way.

**Definition 6.4** *The functor* $\mathcal{LF} : \mathbf{Sig}^{\mathcal{LF}} \to \mathbf{CR}$ *is defined by taking:*

- $\mathcal{LF}(\Sigma)$*, for* $\Sigma \in \mathbf{Sig}^{\mathcal{LF}}$*, to be the consequence relation of Definition 6.3.*
- $\mathcal{LF}(\sigma)$*, for* $\sigma : \Sigma_1 \to \Sigma_2$*, to be* $\sigma^\sharp \restriction \mathrm{Types}_{\Sigma_1}$*, the restriction of* $\sigma^\sharp$ *to closed* $\Sigma_1$*-types.*

**Proposition 6.5** $\mathcal{LF}$ *is a logical system with inclusions.*

For the purposes of encoding a logical system $\mathcal{L}$, we are interested in "specializations" of $\mathcal{LF}$ obtained by fixing a "base" signature $\Sigma_{\mathcal{L}}$ specifying the syntax, assertions, and rules of $\mathcal{L}$ [HHP93], [AHMP87]. The signatures of $\mathcal{L}$ are then represented as extensions to $\Sigma_{\mathcal{L}}$, and signature morphisms are represented as LF signature morphisms on these extensions leaving $\Sigma_{\mathcal{L}}$ fixed. Inferential activity for $\mathcal{L}$ is then reduced to inferential activity in the specialization of $\mathcal{LF}$ to $\Sigma_{\mathcal{L}}$. To make this precise, some additional machinery is needed.

**Definition 6.6** *Let $\Sigma$ be an LF signature. The* category of extensions *of $\Sigma$, written $\mathbf{Sig}_\Sigma^{\mathcal{LF}}$, is the full subcategory of the slice category $\Sigma \downarrow \mathbf{Sig}^{\mathcal{LF}}$ determined by the inclusions $\iota : \Sigma \hookrightarrow \Sigma'$. More explicitly, $\mathbf{Sig}_\Sigma^{\mathcal{LF}}$ has as objects pairs consisting of a signature $\Sigma' \in |\mathbf{Sig}^{\mathcal{LF}}|$ together with an inclusion $\iota : \Sigma \hookrightarrow \Sigma'$. In the following we will simply write $\iota : \Sigma \hookrightarrow \Sigma'$ for objects of $\mathbf{Sig}_\Sigma^{\mathcal{LF}}$. A morphism from $\iota_1 : \Sigma \hookrightarrow \Sigma_1$ to $\iota_2 : \Sigma \hookrightarrow \Sigma_2$ in $\mathbf{Sig}_\Sigma^{\mathcal{LF}}$ is a signature morphism $\sigma : \Sigma_1 \to \Sigma_2$ in $\mathbf{Sig}^{\mathcal{LF}}$ such that $\iota_1; \sigma = \iota_2$. The identities and composition are inherited from $\mathbf{Sig}^{\mathcal{LF}}$.*

Every LF signature induces a logical system based on that signature as follows:

**Definition 6.7** *Let $\Sigma$ be an LF signature. The functor $\mathcal{LF}_\Sigma : \mathbf{Sig}_\Sigma^{\mathcal{LF}} \to \mathbf{CR}$ is defined on objects by*

$$\mathcal{LF}_\Sigma(\iota : \Sigma \hookrightarrow \Sigma') = \mathcal{LF}(\Sigma')$$

*and on morphisms $\sigma : \Sigma' \to \Sigma''$ (in the category of extensions of $\Sigma$) by*

$$\mathcal{LF}_\Sigma(\sigma) = \mathcal{LF}(\sigma).$$

**Proposition 6.8** *$\mathcal{LF}_\Sigma$ is a logical system with inclusions.*

An encoding of a logical system $\mathcal{L}$ in LF comprises not only an LF signature $\Sigma_\mathcal{L}$, but also an "internal type family" distinguishing the basic judgements of $\mathcal{L}$ in the encoding. For example, in the encoding of first-order logic given in [HHP93], the constant *true* of kind $o \to \mathsf{Type}$ represents the basic judgement form of first-order logic. The significance of *true* for the encoding becomes apparent in the statement of the adequacy theorem: terms of type $true(\widehat{\phi})$ in a context with variables $x_i$ of type $true(\widehat{\phi}_i)$ represent proofs of $\phi$ from the $\phi_i$'s (where $\widehat{\phi}$ is the syntactic coding of $\phi$ in LF). This methodology is formalized in our setting as follows.

**Definition 6.9** *An* internal type family of $\Sigma$ *is a term $F$ such that $\vdash_\Sigma F : K$ for some kind $K$. (Note that if $\vdash_\Sigma K$, then $K$ has normal form $\Pi x_1{:}A_1. \ldots . \Pi x_k{:}A_k.\mathsf{Type}$ for some $x_1, \ldots, x_k$ and $A_1, \ldots, A_k$.) The* range *of an internal type family $F$ of $\Sigma$ is defined to be the set*

$$\mathrm{Rng}_\Sigma(F) = \{\, \mathrm{lnf}(F\, M_1\, \ldots\, M_k) \mid \vdash_\Sigma F\, M_1\, \ldots\, M_k : \mathsf{Type} \,\},$$

*where $\mathrm{lnf}(M)$ is the long $\beta\eta$-normal form of $M$.*

If $\mathcal{J}$ is a set of internal type families of $\Sigma$, then

$$\mathrm{Rng}_\Sigma(\mathcal{J}) = \bigcup_{F \in \mathcal{J}} \mathrm{Rng}_\Sigma(F).$$

**Definition 6.10** *A* logic presentation *is a pair $(\Sigma, \mathcal{J})$ where $\Sigma$ is an LF signature and $\mathcal{J}$ is a finite set of internal type families of $\Sigma$.*

**Definition 6.11** *Let $(\Sigma, \mathcal{J})$ be a logic presentation. The* logical system presented by $(\Sigma, \mathcal{J})$, *$\mathcal{P}(\Sigma, \mathcal{J})$, is the restriction of $\mathcal{LF}_\Sigma$ to the range of $\mathcal{J}$. Specifically, $\mathcal{P}(\Sigma, \mathcal{J}) : \mathbf{Sig}_\Sigma^{\mathcal{LF}} \to \mathbf{CR}$ is defined on objects by*

$$\mathcal{P}(\Sigma, \mathcal{J})(\iota : \Sigma \hookrightarrow \Sigma') = \mathcal{LF}(\Sigma') \restriction \mathrm{Rng}_{\Sigma'}(\mathcal{J})$$

*and on morphisms $\sigma : \Sigma' \to \Sigma''$ in the slice category by*

$$\mathcal{P}(\Sigma, \mathcal{J})(\sigma) = \mathcal{LF}(\sigma) \restriction \mathrm{Rng}_{\Sigma'}(\mathcal{J})$$

*(notice that since $\iota : \Sigma \hookrightarrow \Sigma'$ is an inclusion, $\mathcal{J}$ is also an internal type family of $\Sigma'$).*

**Proposition 6.12** *$\mathcal{P}(\Sigma, \mathcal{J}) : \mathbf{Sig}_{\Sigma}^{\mathcal{LF}} \to \mathbf{CR}$ is indeed a logical system and has inclusions.*

**Definition 6.13** *A logical system is* uniformly encodable *(in LF) iff there exists a logic presentation $(\Sigma_{\mathcal{L}}, \mathcal{J}_{\mathcal{L}})$ and a surjective representation $\rho_{\mathcal{L}} : \mathcal{L} \to \mathcal{P}(\Sigma_{\mathcal{L}}, \mathcal{J}_{\mathcal{L}})$. The triple $(\Sigma_{\mathcal{L}}, \mathcal{J}_{\mathcal{L}}, \rho_{\mathcal{L}})$ is called a* uniform encoding *of $\mathcal{L}$.*

The word "uniform" reflects the fact that we require a "natural" encoding of the entire family of consequence relations of $\mathcal{L}$ in LF, rather than a signature-by-signature encoding as is suggested by the account in [HHP93]. The requirement of surjectivity ensures that $\mathcal{J}$ accurately describes the images of $\mathcal{L}$-sentences in LF. For example, in the encoding of first-order logic in [HHP93], all closed long normal forms of the shape $true(M)$ represent first-order sentences.

As an example of a uniform encoding, we consider the logical system $\mathcal{EQ}$ defined in Section 2. The encoding of $\mathcal{EQ}$ will be given in Example 6.16 but for the benefit of readers unfamiliar with LF we will work our way up to this gradually, beginning with the single-sorted case. Thus single-sorted algebraic signatures are families of the form $\langle \Omega_n \rangle_{n \geq 0}$ where the type of individuals is left implicit and for $n \geq 0$, $\Omega_n$ is the set of $n$-ary function symbols. Call this logical system $\mathcal{EQ}_1$. We begin with the even simpler case of single-sorted ground equational logic, $\mathcal{GEQ}_1$.

**Example 6.14** Let $\Sigma\mathcal{GEQ}_1$ be the LF signature

$$\begin{array}{lcl}
\iota & : & \mathsf{Type} \\
eq & : & \iota \to \iota \to \mathsf{Type} \\
refl & : & \Pi x{:}\iota.\ eq\, x\, x \\
sym & : & \Pi x{:}\iota.\ \Pi y{:}\iota.\ eq\, x\, y \to eq\, y\, x \\
trans & : & \Pi x{:}\iota.\ \Pi y{:}\iota.\ \Pi z{:}\iota.\ eq\, x\, y \to eq\, y\, z \to eq\, x\, z \\
cong & : & \Pi f{:}\iota \to \iota.\ \Pi x{:}\iota.\ \Pi y{:}\iota.\ eq\, x\, y \to eq(f\, x)(f\, y)
\end{array}$$

and let $\mathcal{J}\mathcal{GEQ}_1 = \{eq\}$. A uniform encoding of $\mathcal{GEQ}_1$ is the triple $(\Sigma\mathcal{GEQ}_1, \mathcal{J}\mathcal{GEQ}_1, \rho)$ where $\rho : \mathcal{GEQ}_1 \to \mathcal{P}(\Sigma\mathcal{GEQ}_1, \mathcal{J}\mathcal{GEQ}_1)$ is a surjective representation defined as follows:

- For each single-sorted algebraic signature $\Omega = \langle \Omega_n \rangle_{n \geq 0}$, $\rho^{Sig}(\Omega)$ is the extension of $\Sigma\mathcal{GEQ}_1$ by the constant $f : \underbrace{\iota \to \cdots \to \iota \to}_{n \text{ times}} \iota$ for each $f \in \Omega_n$. We assume that $\Omega_n$ does not contain $eq$, $refl$, *etc.* for each $n \geq 0$. Then $\rho^{Sig}$ extends to a functor $\rho^{Sig} : \mathbf{Sig}^{\mathcal{GEQ}_1} \to \mathbf{Sig}^{\mathcal{LF}}_{\Sigma\mathcal{GEQ}_1}$ in the obvious way.

- There is an obvious bijection between the set of ground $\Omega$-terms and the set of closed LF terms of type $\iota$ in $\rho^{Sig}(\Omega)$ and hence between ground $\Omega$-equations and the set of closed LF types of the form $eq\, t\, t'$ in $\rho^{Sig}(\Omega)$. This determines a surjective function $\rho_\Omega : |\mathcal{GEQ}_1|_\Omega \to \mathrm{Rng}_{\rho^{Sig}(\Omega)}(eq)$ which is natural in $\Omega$ and which is a conservative CR morphism $\rho_\Omega : \mathcal{GEQ}_1(\Omega) \to \mathcal{LF}(\rho^{Sig}(\Omega)) \restriction \mathrm{Rng}_{\rho^{Sig}(\Omega)}(eq)$.

Let $\Omega$ be an algebraic signature with $\{a, a', b, b'\} \subseteq \Omega_0$ and $\{f\} \subseteq \Omega_2$. The following is derivable in LF:

$$\begin{array}{l}
ax1{:}eq(a, a'), ax2{:}eq(b, b') \quad \vdash_{\rho^{Sig}(\Omega)} \\
\quad trans\,(f\, a\, b)\,(f\, a\, b')\,(f\, a'\, b') \\
\qquad (cong\,(f\, a)\, b\, b'\, ax2) \\
\qquad (cong\,(\lambda x{:}\iota.\ f\, x\, b')\, a\, a'\, ax1) \quad : eq\,(f\, a\, b)\,(f\, a'\, b')
\end{array}$$

This represents a proof of the congruence property for the two-argument function $f$:

$$a = a', b = b' \vdash_{\Omega}^{\mathcal{GEQ}_1} f(a, b) = f(a', b').$$

The generalization of this example to single-sorted non-ground equational logic is not entirely straightforward. Since sentences of a logic are represented by closed types, it is necessary to introduce an explicit quantifier to bind the free variables and to simulate the implicit universal quantification of sentences in $\mathcal{EQ}$. In order to conform with the type theory of LF, we add an intermediate type $o$ of formulae, where the universal quantifier takes a term of type $\iota \to o$ and produces a term of type $o$ [HHP93].

**Example 6.15** Let $\Sigma\mathcal{GEQ}_1^o$ be the LF signature

$$\begin{array}{lcl}
\iota & : & \mathsf{Type} \\
o & : & \mathsf{Type} \\
true & : & o \to \mathsf{Type} \\
eq & : & \iota \to \iota \to o \\
refl & : & \Pi x{:}\iota.\ true(eq\, x\, x) \\
sym & : & \Pi x{:}\iota.\ \Pi y{:}\iota.\ true(eq\, x\, y) \to true(eq\, y\, x) \\
trans & : & \Pi x{:}\iota.\ \Pi y{:}\iota.\ \Pi z{:}\iota.\ true(eq\, x\, y) \to true(eq\, y\, z) \to true(eq\, x\, z) \\
cong & : & \Pi f{:}\iota \to \iota.\ \Pi \phi{:}\iota \to o.\ \Pi x{:}\iota.\ \Pi y{:}\iota. \\
 & & \qquad true(eq\, x\, y) \to true(\phi(f\, x)) \to true(\phi(f\, y))
\end{array}$$

and let $\mathcal{JGEQ}_1^o = \{\, true \,\}$.

We have generalized the earlier congruence rule to allow predicates other than $eq$ to be added without additional congruence rules. This is not necessary for the examples below in which we deal with logics having equality as the only atomic predicate. It may be shown that the rule *cong* above is equivalent (in the presence of *refl*) to the following two rules:

$$\begin{array}{lcl}
cong_f & : & \Pi f{:}\iota \to \iota.\ \Pi x{:}\iota.\ \Pi y{:}\iota.\ true(eq\, x\, y) \to true(eq(f\, x)(f\, y)) \\
cong_\phi & : & \Pi \phi{:}\iota \to o.\ \Pi x{:}\iota.\ \Pi y{:}\iota.\ true(eq\, x\, y) \to true(\phi\, x) \to true(\phi\, y)
\end{array}$$

The new presentation of single-sorted ground equational logic allows universal quantification to be introduced in a natural way. Let $\Sigma\mathcal{EQ}_1$ be the signature

$$\begin{array}{lcl}
\Sigma\mathcal{GEQ}_1^o, & & \\
\forall & : & (\iota \to o) \to o \\
\forall \mathrm{I} & : & \Pi \phi{:}\iota \to o.\ (\Pi x{:}\iota.\ true(\phi\, x)) \to true(\forall\, \phi) \\
\forall \mathrm{E} & : & \Pi \phi{:}\iota \to o.\ \Pi x{:}\iota.\ true(\forall\, \phi) \to true(\phi\, x)
\end{array}$$

and let $\mathcal{JEQ}_1 = \{true\}$. We can define a surjective representation $\rho : \mathcal{EQ}_1 \to \mathcal{P}(\Sigma\mathcal{EQ}_1, \mathcal{JEQ}_1)$ similarly as in Example 6.14. This yields a uniform encoding $(\Sigma\mathcal{EQ}_1, \mathcal{JEQ}_1, \rho)$ of $\mathcal{EQ}_1$.

Let $\Omega$ be an algebraic signature with $\{a, b\} \subseteq \Omega_0$ and $\{f\} \subseteq \Omega_1$. The following is derivable in LF:

$$\begin{array}{l}
ax1{:}true(\forall(\lambda x{:}\iota.\ eq(f\, x)a)), ax2{:}true(\forall(\lambda x{:}\iota.\ eq(f\, x)b)) \quad \vdash_{\rho^{Sig}(\Omega)} \\
\qquad (\lambda y{:}\iota.\ trans\, a\, (f\, y)\, b \\
\qquad\qquad (sym\, (f\, y)\, a\, (\forall \mathrm{E}\, (\lambda x{:}\iota.\ eq(f\, x)a)\, y\, ax1)) \\
\qquad\qquad (\forall \mathrm{E}\, (\lambda x{:}\iota.\ eq(f\, x)b)\, y\, ax2) \\
\qquad\qquad\qquad\qquad )a \quad : true(eq\, a\, b)
\end{array}$$

This represents a proof that

$$\forall x.\ f(x) = a, \forall x.\ f(x) = b \vdash_\Omega^{\mathcal{EQ}_1} a = b.$$

A careful analysis of examples like this one shows how the quantifier elimination and introduction rules together with LF's substitution mechanism simulate the substitution rule of equational logic, taking correct account of the possibility that the domain of quantification might be empty [GM81].

The final step is to add mechanisms to encode the possibility of having more than one sort of individuals to the above encoding of single-sorted equational logic. We add a type of sort names and we attach to each sort name the type of its values. Then, both the syntactic operations ($eq$ and $\forall$) and the inference rules must be supplied with a sort name as an additional parameter.

**Example 6.16** Let $\Sigma\mathcal{EQ}$ be the LF signature

$$
\begin{array}{lcl}
\mathit{sorts} & : & \mathsf{Type} \\
\mathit{obj} & : & \mathit{sorts} \to \mathsf{Type} \\
o & : & \mathsf{Type} \\
\mathit{true} & : & o \to \mathsf{Type} \\
\mathit{eq} & : & \Pi s{:}\mathit{sorts}.\ \mathit{obj}\, s \to \mathit{obj}\, s \to o \\
\mathit{refl} & : & \Pi s{:}\mathit{sorts}.\ \Pi x{:}\mathit{obj}\, s.\ \mathit{true}(\mathit{eq}\, s\, x\, x) \\
\mathit{sym} & : & \Pi s{:}\mathit{sorts}.\ \Pi x{:}\mathit{obj}\, s.\ \Pi y{:}\mathit{obj}\, s.\ \mathit{true}(\mathit{eq}\, s\, x\, y) \to \mathit{true}(\mathit{eq}\, s\, y\, x) \\
\mathit{trans} & : & \Pi s{:}\mathit{sorts}.\ \Pi x{:}\mathit{obj}\, s.\ \Pi y{:}\mathit{obj}\, s.\ \Pi z{:}\mathit{obj}\, s. \\
 & & \qquad \mathit{true}(\mathit{eq}\, s\, x\, y) \to \mathit{true}(\mathit{eq}\, s\, y\, z) \to \mathit{true}(\mathit{eq}\, s\, x\, z) \\
\mathit{cong} & : & \Pi s{:}\mathit{sorts}.\ \Pi s'{:}\mathit{sorts}.\ \Pi f{:}\mathit{obj}\, s \to \mathit{obj}\, s'.\ \Pi \phi{:}\mathit{obj}\, s' \to o.\ \Pi x{:}\mathit{obj}\, s.\ \Pi y{:}\mathit{obj}\, s. \\
 & & \qquad \mathit{true}(\mathit{eq}\, s\, x\, y) \to \mathit{true}(\phi(f\, x)) \to \mathit{true}(\phi(f\, y)) \\
\forall & : & \Pi s{:}\mathit{sorts}.\ (\mathit{obj}\, s \to o) \to o \\
\forall \mathrm{I} & : & \Pi s{:}\mathit{sorts}.\ \Pi \phi{:}\mathit{obj}\, s \to o.\ (\Pi x{:}\mathit{obj}\, s.\ \mathit{true}(\phi\, x)) \to \mathit{true}(\forall\, s\, \phi) \\
\forall \mathrm{E} & : & \Pi s{:}\mathit{sorts}.\ \Pi \phi{:}\mathit{obj}\, s \to o.\ \Pi x{:}\mathit{obj}\, s.\ \mathit{true}(\forall\, s\, \phi) \to \mathit{true}(\phi\, x)
\end{array}
$$

and let $\mathcal{JEQ} = \{\, \mathit{true} \,\}$.

A uniform encoding of $\mathcal{EQ}$ is the triple $(\Sigma\mathcal{EQ}, \mathcal{JEQ}, \rho)$ where $\rho : \mathcal{EQ} \to \mathcal{P}(\Sigma\mathcal{EQ}, \mathcal{JEQ})$ is a surjective representation defined as follows:

- For each many-sorted algebraic signature $\Sigma = (S, \Omega)$, $\rho^{Sig}(\Sigma)$ is the extension of $\Sigma\mathcal{EQ}$ by the constant $s : \mathit{sorts}$ for each $s \in S$ and the constant $f : \mathit{obj}(s_1) \to \cdots \to \mathit{obj}(s_n) \to \mathit{obj}(s)$ for each $f : s_1 \times \cdots \times s_n \to s$ in $\Sigma$. We assume that $eq$, $refl$, *etc.* do not occur in $\Sigma$. Then $\rho^{Sig}$ extends to a functor $\rho^{Sig} : \mathbf{Sig}^{\mathcal{EQ}} \to \mathbf{Sig}^{\mathcal{LF}}_{\Sigma\mathcal{EQ}}$ in the obvious way.

- The surjective mapping $\rho_\Sigma : |\mathcal{EQ}|_\Sigma \to \mathrm{Rng}_{\rho^{Sig}(\Sigma)}(\mathit{true})$ is determined as in the examples above, e.g.

  $$\rho_\Sigma(\forall x{:}s, y{:}s'.\ f(x,y) = b) = \forall\, s\, (\lambda x{:}\mathit{obj}\, s.\ \forall\, s'\, (\lambda y{:}\mathit{obj}\, s'.\ \mathit{true}\, (\mathit{eq}\, s\, (f\, x\, y)\, b)))$$

  for a signature $\Sigma$ with sorts $s, s'$, constant $b : s$ and a binary operation $f : s \times s' \to s$. This function $\rho_\Sigma$ is natural in $\Sigma$ and is a conservative CR morphism $\rho_\Sigma : \mathcal{EQ}(\Sigma) \to \mathcal{LF}(\rho^{Sig}(\Sigma)) \restriction \mathrm{Rng}_{\rho^{Sig}(\Sigma)}(\mathit{true})$.

Let $\Sigma = (S, \Omega)$ be an algebraic signature with sorts $s$ and $s'$, constants $a{:}s$ and $b{:}s$, and an operation $f : s' \to s$. Then $\rho^{Sig}(\Sigma)$ is the extension of $\Sigma\mathcal{EQ}$ as described above. The following is derivable in LF:

$$
\begin{array}{l}
\mathit{ax1}{:}\mathit{true}(\forall\, s'\, (\lambda x{:}\mathit{obj}\, s'.\ \mathit{eq}\, s\, (f\, x)\, a)), \mathit{ax2}{:}\mathit{true}(\forall\, s'\, (\lambda x{:}\mathit{obj}\, s'.\ \mathit{eq}\, s\, (f\, x)\, b)) \quad \vdash_{\rho^{Sig}(\Sigma)} \\
\qquad \forall \mathrm{I}\, s'\, (\lambda x{:}\mathit{obj}\, s'.\ \mathit{eq}\, s\, a\, b) \\
\qquad\quad (\lambda y{:}\mathit{obj}\, s'.\ \mathit{trans}\, s\, a\, (f\, y)\, b \\
\qquad\qquad\qquad (\mathit{sym}\, s\, (f\, y)\, a\, (\forall \mathrm{E}\, s'\, (\lambda x{:}\mathit{obj}\, s'.\ \mathit{eq}\, s\, (f\, x)\, a)\, y\, \mathit{ax1})) \\
\qquad\qquad\qquad (\forall \mathrm{E}\, s'\, (\lambda x{:}\mathit{obj}\, s'.\ \mathit{eq}\, s\, (f\, x)\, b)\, y\, \mathit{ax2}) \\
\qquad\qquad\qquad\qquad ) \quad : \mathit{true}(\forall\, s'\, (\lambda x{:}\mathit{obj}\, s'.\ \mathit{eq}\, s\, a\, b))
\end{array}
$$

This represents a proof that

$$\forall x : s'.\ f(x) = a, \forall x : s'.\ f(x) = b \vdash^{\mathcal{EQ}}_{\Sigma} \forall x : s'.\ a = b.$$

Notice that in $\mathcal{EQ}$ we *cannot* derive

$$\forall x : s'.\ f(x) = a, \forall x : s'.\ f(x) = b \vdash^{\mathcal{EQ}}_{\Sigma} a = b.$$

and correspondingly this cannot be derived in the LF encoding either: in the LF signature $\rho^{Sig}(\Sigma)$ and the context $ax1{:}true(\forall\, s'\,(\lambda x{:}obj\, s'.\ eq\, s\,(f\, x)\, a))$, $ax2{:}true(\forall\, s'\,(\lambda x{:}obj\, s'.\ eq\, s\,(f\, x)\, b))$ there is no term of type $true(\, eq\, s\, a\, b)$.

More complex examples may be built using the various representations of logical systems in LF presented in [AHMP87].

# 7 Proof Search under Uniform Encodings

Let us now return to the problem of proof in structured theory presentations. Given a uniform encoding of a logical system $\mathcal{L}$, we intend to exploit the proof mechanisms of LF to conduct proofs in structured $\mathcal{L}$-presentations.

Since the representation part of a uniform encoding of a logic $\mathcal{L}$ is required to be surjective, it might be thought that we may use the naïve lifting of $\mathcal{L}$-presentations to LF, relying on Corollary 5.16. But this is not the case, for in practice we work not in $\mathcal{P}(\Sigma_{\mathcal{L}}, \mathcal{J}_{\mathcal{L}})$, but in $\mathcal{LF}(\Sigma_{\mathcal{L}})$, which is to say that we cannot restrict attention to sentences in the range of $\mathcal{J}_{\mathcal{L}}$ only. For example, in the encoding of **S4** [AHMP87], sentences are represented by terms of the form $true(M)$. But to prove, say, $true(\Box(M))$, we must, in certain cases, prove $valid(M)$. But this type lies outside of the image of $\rho$ (and cannot be soundly included in it).

Now since $\mathcal{P}(\Sigma, \mathcal{J})$ is defined to be the restriction of $\mathcal{LF}_{\Sigma}$ to the range of $\mathcal{J}$, there is an obvious "inclusion" of $\mathcal{P}(\Sigma, \mathcal{J})$ into $\mathcal{LF}$ which is typically not surjective. However, the set of sentences considered is explicitly determined by $\mathcal{J}$.

Consider a uniform encoding $(\Sigma_{\mathcal{L}}, \mathcal{J}_{\mathcal{L}}, \rho_{\mathcal{L}})$ of $\mathcal{L}$. Let $\mathcal{R}$ be the family of LF types given by $\mathcal{R}_{\Sigma'} = \mathrm{Rng}_{\Sigma'}(\mathcal{J}_{\mathcal{L}})$ for $\Sigma'$ extending $\Sigma_{\mathcal{L}}$. By Corollary 5.18, for any well-formed $\mathcal{L}$-presentation $P$ with signature $\Sigma$ and $\mathcal{L}$-sentence $\phi \in |\mathcal{L}|_{\Sigma}$, $P \Vdash^{\mathcal{L}}_{\Sigma} \phi$ iff $\tilde{\rho}_{\mathcal{L}}(P) \Vdash^{\mathcal{LF},\mathcal{R}}_{\rho_{\mathcal{L}}(\Sigma)} \rho_{\mathcal{L}}(\phi)$. It is important to realize that the right-hand side of this equivalence refers only to LF entities, and the corresponding proof search as determined by Definition 5.17 can be carried out entirely within LF.

An essential part of this proof activity is to test whether a type $A$ of $\mathcal{LF}(\Sigma')$ is in the image of $\rho_{\mathcal{L}}$, where $\Sigma' = \rho_{\mathcal{L}}(\Omega)$ for some $\Omega \in \mathbf{Sig}^{\mathcal{L}}$, that is whether or not $A \in \mathrm{Rng}_{\Sigma'}(\mathcal{J}_{\mathcal{L}})$. This amounts to matching in the LF type theory: $A \in \mathrm{Rng}_{\Sigma'}(\mathcal{J}_{\mathcal{L}})$ iff there exists $J \in \mathcal{J}$ and $M_1, \ldots, M_k$ (where $k$ is determined by $J$) such that $A$ is convertible to $J(M_1, \ldots, M_k)$. This test may be implemented using the unification algorithms developed by Pym [Pym90] or Elliot [Ell89]. In practice, $\mathcal{J}$ is often a single constant, in which case this test is trivial; it is an open problem whether the matching problem is, in general, decidable.

**Example 7.1** Recall the presentations given in Counterexample 5.9: $\Sigma_0$ is the signature with sorts $s$ and $s'$ and constants $a{:}s$ and $b, c{:}s'$, $\Sigma$ is the same signature with $a$ removed, $\iota : \Sigma \hookrightarrow \Sigma_0$ is the inclusion, $P_0 = (\Sigma_0, \emptyset)$, $P_1 = \mathsf{derive}\ P_0\ \mathsf{via}\ \iota$ and $P = P_1 \cup (\Sigma, \{\forall x{:}s.\ b = c\})$. Consider the uniform encoding $(\Sigma\mathcal{EQ}, \mathcal{J}\mathcal{EQ}, \rho)$ of $\mathcal{EQ}$ in LF given in Example 6.16. We will conduct inference in $P$ via this encoding.

The LF signature $\rho(\Sigma_0)$ is the extension of $\Sigma\mathcal{EQ}$ by constants $s, s'{:}sorts$, $a{:}obj\, s$ and $b, c{:}obj\, s'$. The following is derivable in LF:

$$t{:}\mathsf{Type} \vdash_{\rho(\Sigma_0)} (\lambda f{:}obj\, s \to t.\ f\, a) : (obj\, s \to t) \to t$$

Thus, in the inhabitation logic of LF, we have in particular

$$\vdash^{\mathcal{LF}}_{\rho(\Sigma_0)} (obj\ s \to true\,(eq\ s'\ b\ c)) \to true\,(eq\ s'\ b\ c)$$

The following is also derivable:

$$\begin{array}{l} ax1{:}true\,(\forall s\,(\lambda x{:}s.\ \ eq\ s'\ b\ c)), ax2{:}(obj\ s \to true\,(eq\ s'\ b\ c)) \to true\,(eq\ s'\ b\ c) \quad \vdash_{\rho(\Sigma)} \\ \qquad\qquad ax2\,(\lambda x{:}obj\ s.\ \forall \mathrm{E}\ s\,(\lambda x{:}obj\ s.\ \ eq\ s'\ b\ c)\ x\ ax1) : true\,(eq\ s'\ b\ c) \end{array}$$

Consequently, in the inhabitation logic of LF, we have

$$true\,(\forall s\,(\lambda x{:}s.\ \ eq\ s'\ b\ c)), (obj\ s \to true\,(eq\ s'\ b\ c)) \to true\,(eq\ s'\ b\ c) \vdash^{\mathcal{LF}}_{\rho(\Sigma)} true\,(eq\ s'\ b\ c)$$

The above entailments in the inhabitation logic of LF justify the following (cf. Definition 4.1):

$$\begin{array}{rll} \tilde{\rho}(P_0) & \Vdash^{\mathcal{LF}}_{\rho(\Sigma_0)} & (obj\ s \to true\,(eq\ s'\ b\ c)) \to true\,(eq\ s'\ b\ c) \\ \tilde{\rho}(P_1) & \Vdash^{\mathcal{LF}}_{\rho(\Sigma)} & (obj\ s \to true\,(eq\ s'\ b\ c)) \to true\,(eq\ s'\ b\ c) \\ \tilde{\rho}(P) & \Vdash^{\mathcal{LF}}_{\rho(\Sigma)} & true\,(eq\ s'\ b\ c) \end{array}$$

Note that we have just proved that $\tilde{\rho}(P) \Vdash^{\mathcal{LF}}_{\rho(\Sigma)} \rho(b = c)$ even though $P \Vdash^{\mathcal{EQ}}_{\Sigma} b = c$ does not hold. In essence, what is happening here is similar to what was illustrated in Counterexample 5.9, except that a higher-order type is used in place of an existential formula. As in Example 5.19, this shows the need for keeping track of the image of the encoding. The crucial mediating type, $(obj\ s \to true\,(eq\ s'\ b\ c)) \to true\,(eq\ s'\ b\ c)$ which is inhabited in $\rho(\Sigma_0)$, does not encode a sentence of $\mathcal{EQ}$ (is not in the range of *true*) and so will be filtered out in the modified proof procedure determined by Definition 5.17: we do *not* have $\tilde{\rho}(P_1) \Vdash^{\mathcal{LF},\mathcal{R}}_{\rho(\Sigma)} (obj\ s \to true\,(eq\ s'\ b\ c)) \to true\,(eq\ s'\ b\ c)$ where $\mathcal{R}_{\Sigma'} = \mathrm{Rng}_{\Sigma'}(\mathcal{JEQ})$ for $\Sigma'$ extending $\Sigma\mathcal{EQ}$.

# 8 Putting Together Logics

In this section we consider the adaptation of the idea of presenting theories in a structured way to logic presentations. As a first step in this direction we investigate the use of pushouts to give an account of parameterization and instantiation of logic presentations. We have in mind such examples as: the parameterization of Peano arithmetic by the underlying predicate calculus, with instantiations like classical Peano arithmetic and Heyting arithmetic; the parameterization of Hoare logic by the logic of assertions; the parameterization of the calculus of synchronization trees by the synchronization algebra [Win81].

**Proposition 8.1** $\mathbf{Sig}^{\mathcal{LF}}$ *has pushouts along inclusions.*
**Proof** *If* $\sigma : \Sigma \to \Sigma'$ *and* $\iota : \Sigma \hookrightarrow \Sigma''$, *then the pushout is given by*

$$\sigma^{\star}\Sigma'' = \Sigma', p(\sigma, \Sigma'')^{\sharp}\Sigma'''$$

*(where* $\Sigma'' = \Sigma, \Sigma'''$ *— since* $\iota$ *is a signature inclusion, it is always possible to present* $\Sigma''$ *in this way) and*

$$p(\sigma, \Sigma'')(c) = \begin{cases} \sigma(c) & \text{if } c \in \mathrm{dom}(\Sigma) \\ c & \text{otherwise} \end{cases}$$

*(This assumes that* $\Sigma'''$ *is disjoint from* $\Sigma'$*; otherwise,* $p(\sigma, \Sigma'')$ *would have to rename symbols appropriately.)*

Note that $\mathbf{Sig}^{\mathcal{LF}}$ is not finitely co-complete.

**Definition 8.2** *A* logic presentation morphism $\sigma : (\Sigma, \mathcal{J}) \to (\Sigma', \mathcal{J}')$ *is a signature morphism* $\sigma : \Sigma \to \Sigma'$ *in* $\mathbf{Sig}^{\mathcal{LF}}$ *such that for every* $F \in \mathcal{J}$ *with*

$$\vdash_{\Sigma} F : \Pi x_1{:}A_1 . \ldots . x_k{:}A_k . \mathsf{Type},$$

*there exists* $F' \in \mathcal{J}'$ *such that*

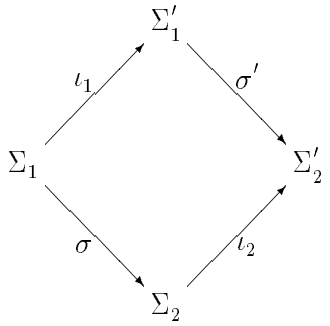
$$\sigma^{\sharp} F =_{\beta\eta} \lambda x_1{:}\sigma^{\sharp} A_1 . \ldots . x_k{:}\sigma^{\sharp} A_k . F'(M_1, \ldots, M_n)$$

*for some* $M_1$, ..., $M_n$. *Identity and composition are inherited from* $\mathbf{Sig}^{\mathcal{LF}}$. **LogPres** *is the category of logic presentations and logic presentation morphisms.*

Note that $x_i \notin \mathrm{FV}(F')$ $(1 \leq i \leq k)$ since $F'$ is closed.

**Proposition 8.3** *The assignment* $(\Sigma, \mathcal{J}) \mapsto \mathcal{P}(\Sigma, \mathcal{J})$ *extends to a functor* $\mathcal{P} : \mathbf{LogPres} \to \mathbf{Log}$.
**Sketch of construction** *Consider a presentation morphism* $\sigma : (\Sigma_1, \mathcal{J}_1) \to (\Sigma_2, \mathcal{J}_2)$. *The logic morphism* $\mathcal{P}(\sigma) : \mathcal{P}(\Sigma_1, \mathcal{J}_1) \to \mathcal{P}(\Sigma_2, \mathcal{J}_2)$ *may be defined as follows:*

- $\mathcal{P}(\sigma)^{Sig} : \mathbf{Sig}^{\mathcal{LF}}_{\Sigma_1} \to \mathbf{Sig}^{\mathcal{LF}}_{\Sigma_2}$ *is defined on objects using the pushout construction:* $\mathcal{P}(\sigma)^{Sig}(\iota_1 : \Sigma_1 \hookrightarrow \Sigma'_1) = (\iota_2 : \Sigma_2 \hookrightarrow \Sigma'_2)$ *where*




  *is a pushout in* $\mathbf{Sig}^{\mathcal{LF}}$.

  *This extends to morphisms using the co-universal property of pushouts.*

- *For any* $\iota_1 : \Sigma_1 \hookrightarrow \Sigma'_1$, $\sigma' : \Sigma'_1 \to \Sigma'_2$ *in the construction above induces the translation* $(\sigma')^{\sharp} : \mathrm{Rng}_{\Sigma'_1}(\mathcal{J}_1) \to \mathrm{Rng}_{\Sigma'_2}(\mathcal{J}_2)$. *(This uses the fact that* $\sigma$ *is a logic presentation morphism.) This is a CR morphism* $(\sigma')^{\sharp} : \mathcal{P}(\Sigma_1, \mathcal{J}_1)(\iota_1 : \Sigma_1 \hookrightarrow \Sigma'_1) \to \mathcal{P}(\Sigma_2, \mathcal{J}_2)(\mathcal{P}(\sigma)^{Sig}(\iota_1) : \Sigma_2 \hookrightarrow \Sigma'_2)$ *which is natural in* $\iota_1$. *This defines* $\mathcal{P}(\sigma)^{CR} : \mathcal{P}(\Sigma_1, \mathcal{J}_1) \dashrightarrow \mathcal{P}(\sigma)^{Sig}; \mathcal{P}(\Sigma_2, \mathcal{J}_2)$.

We propose to use colimits in the category of logic presentations to build logics in the same way as colimits were used in Section 3 to build theories. Although the category of logic presentations is not finitely co-complete, it may be shown that a diagram in **LogPres** has a colimit iff its projection to $\mathbf{Sig}^{\mathcal{LF}}$ has a colimit. The most pertinent case is that of pushouts along inclusions:

**Definition 8.4** *A logic presentation morphism* $\iota : (\Sigma, \mathcal{J}) \hookrightarrow (\Sigma', \mathcal{J}')$ *is an* inclusion *if* $\iota : \Sigma \hookrightarrow \Sigma'$ *is an inclusion and* $\mathcal{J} \subseteq \mathcal{J}'$.

**Proposition 8.5 LogPres** *has pushouts along inclusions.*
**Proof** *The pushout of* $\sigma : (\Sigma, \mathcal{J}) \to (\Sigma_1, \mathcal{J}_1)$ *and* $\iota : (\Sigma, \mathcal{J}) \hookrightarrow (\Sigma_2, \mathcal{J}_2)$ *is given by the object* $\sigma^\star(\Sigma_2, \mathcal{J}_2) = (\sigma^\star\Sigma_2, \sigma^\star\mathcal{J}_2)$*, where* $\sigma^\star\Sigma_2$ *is the pushout object in* $\mathbf{Sig}^{\mathcal{LF}}$ *and*

$$\sigma^\star\mathcal{J}_2 = p(\sigma, \Sigma_2)^\sharp(\mathcal{J}_2) \cup \mathcal{J}_1,$$

*and the morphism* $p(\sigma, (\Sigma_2, \mathcal{J}_2)) = p(\sigma, \Sigma_2)$ *is given by the pushout construction in* $\mathbf{Sig}^{\mathcal{LF}}$*.*

A **LogPres** inclusion can be seen as a parameterized logic presentation where the pushout of this morphism with a "fitting" morphism amounts to instantiation, by analogy with parameterized structured theory presentations.

**Example 8.6**

$$\begin{array}{lcll}
\Sigma\mathcal{BASE}^o & = & o & : \mathsf{Type} \\
 & & \mathit{true} & : o \to \mathsf{Type} \\
\mathcal{J}\mathcal{BASE}^o & = & \{\mathit{true}\} & \\
 & & & \\
\Sigma\mathcal{PROP} & = & \Sigma\mathcal{BASE}^o, & \\
 & & \neg & : o \to o \\
 & & \wedge & : o \to o \to o \\
 & & \vee & : o \to o \to o \\
 & & \supset & : o \to o \to o \\
 & & & \vdots \\
 & & \supset\mathrm{I} & : \Pi\phi{:}o.\ \Pi\psi{:}o.\ (\mathit{true}(\phi) \to \mathit{true}(\psi)) \to \mathit{true}(\supset\phi\,\psi) \\
 & & & \vdots \\
\mathcal{JPROP} & = & \{\mathit{true}\} &
\end{array}$$

$\mathcal{BASE}^o = (\Sigma\mathcal{BASE}^o, \mathcal{J}\mathcal{BASE}^o)$ presents a trivial logic containing only the type of formulae ($o$) and the judgement form *true*. $\mathcal{PROP} = (\Sigma\mathcal{PROP}, \mathcal{JPROP})$ presents propositional logic; only one of the standard inference rules is given above. There is an obvious inclusion $\iota : \mathcal{BASE}^o \hookrightarrow \mathcal{PROP}$, which may be seen as propositional logic parameterized by the type of atomic propositions.

Instantiation of this parameterized logic presentation to the presentation of single-sorted ground equational logic $(\Sigma\mathcal{GEQ}^o_1, \mathcal{JGEQ}^o_1)$ (see Example 6.15), via the inclusion of $\mathcal{BASE}^o$, yields a presentation $\mathcal{PROP}(\mathcal{GEQ}^o_1)$ of a propositional logic where atomic formulae are ground equations.

**Example 8.7**

$$\begin{array}{lcll}
\Sigma\mathcal{BASE}^{\iota,o} & = & \iota & : \mathsf{Type} \\
 & & o & : \mathsf{Type} \\
 & & \mathit{true} & : o \to \mathsf{Type} \\
\mathcal{J}\mathcal{BASE}^{\iota,o} & = & \{\mathit{true}\} & \\
 & & & \\
\Sigma\mathcal{UNIV} & = & \Sigma\mathcal{BASE}^{\iota,o}, & \\
 & & \forall & : (\iota \to o) \to o \\
 & & \forall\mathrm{I} & : \Pi\phi{:}\iota \to o.\ (\Pi x{:}\iota.\ \mathit{true}(\phi\,x)) \to \mathit{true}(\forall\,\phi) \\
 & & \forall\mathrm{E} & : \Pi\phi{:}\iota \to o.\ \Pi x{:}\iota.\ \mathit{true}(\forall\,\phi) \to \mathit{true}(\phi\,x) \\
\mathcal{JUNIV} & = & \{\mathit{true}\} &
\end{array}$$

$\mathcal{BASE}^{\iota,o} = (\Sigma\mathcal{BASE}^{\iota,o}, \mathcal{J}\mathcal{BASE}^{\iota,o})$ presents a logic containing only the type of individuals ($\iota$) and formulae ($o$) and the judgement form *true*. The logic presentation $\mathcal{UNIV} = (\Sigma\mathcal{UNIV}, \mathcal{JUNIV})$

presents a logic of universal quantification. There is an obvious inclusion $\iota : \mathcal{BASE}^{\iota,o} \hookrightarrow \mathcal{UNIV}$, which may be seen as a pure logic of universal quantification parameterized by the types of individuals and formulae.

The result of instantiating this parameterized logic presentation to $(\Sigma\mathcal{GEQ}_1^o, \mathcal{JGEQ}_1^o)$ via the inclusion of $\mathcal{BASE}^{\iota,o}$ is the logic presentation $(\Sigma\mathcal{EQ}_1, \mathcal{JEQ}_1)$ from Example 6.15. The result of instantiating it to $\mathcal{PROP}(\mathcal{GEQ}_1^o)$ yields a presentation $\mathcal{UNIV}(\mathcal{PROP}(\mathcal{GEQ}_1^o))$ of a version of first-order logic with an equality predicate. Existential quantifiers are absent although they are expressible since the logic includes universal quantification and negation. It would be easy to add them explicitly by extending $\mathcal{UNIV}$ or by forming a parameterized presentation $\mathcal{EXIST}$ of the logic of existential quantification and applying this to $\mathcal{UNIV}(\mathcal{PROP}(\mathcal{GEQ}_1^o))$.

In the above examples we used pushouts in the category of logic presentations as a mechanism for instantiating parameterized logic presentations. The functor $\mathcal{P}$ allows us to view this as a combination of the corresponding logical systems. A more straightforward method would be to combine logical systems directly, using pushouts in **Log**.[6] In general, the result would be different ($\mathcal{P}$ is not finitely co-continuous). The reason for the difference is that in **Log** the internal structure of sentences is not visible and so the combination is done in a coarse, superficial way. For example, consider two extensions of $\mathcal{EQ}_1$, one which adds negation (giving equations, negations of equations, negations of negations of equations, *etc.*) and another which adds conjunction (giving equations, conjunctions of equations, conjunctions of conjunctions of equations, *etc.*). The pushout of these in **Log** is a logical system with the union of the two sets of sentences but not including (for instance) conjunctions of negations of equations. This is in contrast to the result of taking the pushout of the obvious presentations of these logics in **LogPres**, in which the fine detail of the structure of sentences is visible. The resulting presentation has negation and conjunction built in as operations on the type $o$ of formulae and hence the logical system it presents contains sentences with arbitrarily deep interleaving of conjunction and negation, as expected. The same phenomenon may be illustrated using Examples 8.6 and 8.7. As we have mentioned, $\mathcal{UNIV}(\mathcal{PROP}(\mathcal{GEQ}_1^o))$ is a presentation of a version of first-order logic. Performing the analogous construction at the level of logics, a much smaller set of sentences would be obtained; for example, the existential quantifer would not be expressible. Summing up, this suggests that the proper way to combine logics is at the level of logic presentations rather than at the level of the logics themselves.

The same problems of sharing mentioned in Section 3 with reference to building large theories arise when building complex logics (such as seem to be appropriate for reasoning about Standard ML programs [ST91]). More complicated colimits are again applicable here, and as before the relevant diagrams arise in a natural way from the way that logics are combined using the notation of a language such as CLEAR [BG80]. A difference is that some diagrams in **LogPres** have no colimit, so it is useful to consider a subcategory (with inclusions) of **LogPres** in which all colimits exist. By proceeding in this way we obtain a CLEAR-like or ML-like language for defining logics in a structured way.

# 9 Related Work

Our notion of a logical system is inspired by Goguen and Burstall's work on institutions [GB84a] and by Fiadeiro and Sernadas's $\pi$-institutions [FS87]. Roughly speaking, institutions are a model-theoretic view of logical systems based on signature-indexed families of satisfaction relations that are well-behaved under variation in signature. $\pi$-institutions are a theory-based view of logical systems

[6] Although for foundational reasons, **Log** is not finitely co-complete, it may be shown that it has pushouts involving logical systems with *small* categories of signatures [TBG92]. This is a reasonable assumption since it holds for example if all the "names" in signatures come from an infinite but fixed vocabulary.

based on closure operations on sets of sentences, and are equivalent to our logical systems. We prefer to take consequence as basic both as a matter of taste and because this framework admits generalizations that are not available in $\pi$-institutions (*e.g.*, multi-conclusioned CR's, CR's based on multisets or sequences, rather than sets, and CR's that are not closed under weakening.)

Institutions were first used to parameterize the semantics of CLEAR [BG80] by the logical system used to write specifications, an idea which has been pursued for other specification languages [ST86], [ST88a] and in connection with the foundations of formal program development [BV85, ST88b], since then. The ideas in Sections 3 and 8 concerning building theories and logics in a structured fashion have their roots in CLEAR and are related to Goguen's earlier work on general systems theory [Gog71].

[ST88a] considers a language of structured *specifications* which is similar to but richer than the language of structured presentations introduced in Section 3. As discussed in [ST92], there is an essential difference between the view of structured presentations purely as theory presentations, which we take here, and the view of them as specifications as in [ST88a]. The main role of specifications is to describe the class of their admissible realizations (models), and hence the primary semantics of the specification language in [ST88a] is given in terms of model classes. One way to construe the work in Sections 3 and 4 is as providing a sound proof-theoretic counterpart to this model-theoretic semantics. The proof search procedure presented in Section 4 is not complete for this semantics, at least in the case of logical systems such as $\mathcal{EQ}$ and $\mathcal{FOEQ}$. But its advantage is that it strictly follows the structure of the presentation written by the user of the formalism, with benefits such as those sketched in Section 4. Completeness seems to be the price we have to pay for this: the complete proof systems given in [Far92] and [Wir91] require the structure of the presentation to be altered in the course of proof.

In [GB84a] a notion of *institution morphism* is presented, and used to investigate (among other things) the question of when a theorem prover for one logic can be used to prove theorems on theories from another. A morphism of logics in our sense corresponds roughly to a *sound* institution morphism in the framework of [GB84a]. However, since the two kinds of morphisms are motivated by different concerns (an institution morphism indicates how one institution can be viewed as having been built over the other, while a logic morphism indicates how one logic can be encoded in the other), this comparison is not very accurate. Further work on providing a notion of morphism between institutions which adequately captures preservation of a proof-theoretic entailment relation associated with the model-theoretic satisfaction relation of an institution is presented in [Mes89] and [AC92].

In [Tar86] it is shown that the category of institutions has limits and the idea of using limits to combine institutions is briefly discussed. Translating this to the present setting, these limits are related to colimits in the category **Log** of logical systems. In [GB86] the concepts of *charter* and *parchment* are presented; these are progressively more primitive in that charters are used as tools for constructing institutions while parchments are in turn used to construct charters. A parchment for a logical system seems to correspond very roughly with an encoding of that logical system in LF, except that dependent types are not available in parchments. All such comparisons can only be vague since institutions, charters and parchments are fundamentally model-theoretic notions while our logical systems (and $\pi$-institutions) are proof theoretic.

Like this paper, [Gar92] attempts a careful explication of the concept of logic representation in LF which was not made fully formal in [HHP93]. The main idea of the version of LF studied there is to refine the type theory in such a way that it is possible to extract the logic defined by a signature given only the signature. This is accomplished by distinguishing judgements from other types in the representing type theory, rather than using the "extra-logical" methods (the type family component of uniform encodings) that we have considered here. Our notion of uniform encoding corresponds roughly to the notion of *adequate encoding* in [Gar92], although variation of signatures is not taken into account there.

Drawing on some of the ideas considered in this paper and on [SW92], [HP92] proposes a modules

system for Elf, a logic programming language based on LF [Pfe89], [Pfe91]. This system provides structuring operations on LF signatures with which one may give structured presentations of logical systems and theories within a given logical system. An analogue of our presentation morphisms is provided via the notion of a *realizor*, which is essentially an interpretation of one signature in another given by a sequence of terms of the LF $\lambda$-calculus. The structuring operations considered in [HP92] do not, by design, include an analogue of our `derive` operation. Proof search is provided by the `solve` primitive which not only attempts to determine if a given type is inhabited (*i.e.*, whether the judgement it encodes is provable), but also computes an inhabiting term. The search procedure is conditioned by the `using` primitive with which the relevant portions of a structured logic presentation are marshalled for use by the solver. The absence of `derive` ensures that the problems with the behavior of the structured search procedure under representation (discussed in Section 5) are avoided.

# 10 Directions for Future Research

The definition of logical system, and especially the definition of uniform encoding, reflects the intention that sentences be "closed." The definition of logical system and uniform encoding could be generalized to admit "open" sentences, but it is important to realize that there are (at least) two different ways to construe consequence in this situation [Avr91]. Under the "truth" interpretation, free variables behave essentially as constants, and hence could be handled within our notion of logical system (the situation is more complicated in free logics such as PX [HN88]). Under the "validity" interpretation, free variables are implicitly universally quantified at each formula. Hilbert-type presentations of first-order logic usually take the validity interpretation, whereas natural deduction presentations take the truth interpretation. Some ideas on how the notion of logical system may be extended to accommodate free variables are in [HST89b].

The definition of basic theory presentation admits the possibility of an infinite set of axioms. In practice such sets are presented schematically since theories of interest are recursively presentable. The notion of logical system can be extended to treat axiom schemes explicitly, and the definition of uniform encoding can be correspondingly generalized to encode schemes using $\Pi$-types. This extension becomes important in the case of certain truth-type logical systems lacking a universal quantifier, for there it is not possible to think of an axiom scheme as standing for all of its instances. It would be interesting to work out a treatment of schematization for both truth-type and validity-type logical systems.

The emphasis in this paper has been on provability, rather than on finding proofs. This is reflected in our decision to view logical systems as consequence relations, and in the concomitant definition of search in structured presentations. It would be interesting to develop a general notion of logical system that includes an explicit representation of proofs. With this in mind, we have considered a categorical generalization of the notion of consequence relation whereby proofs become morphisms in a *consequence category* satisfying some weak closure properties (as in linear categories [GL87]). It seems difficult, however, to develop the notions of structured presentation and structured search in such a way that a witness to the fact that a sentence is a consequence of a structured presentation may be extracted. The difficulty seems to lie in the fact that structured presentations rely on working simultaneously with a family of consequence relations, rather than just one. When generalized to admit proofs, this means that we must consider a hybrid notion of proof that spans a family of consequence categories.

The CLEAR-style parameterization methods outlined in Section 3 require that a signature morphism be a presentation morphism. This is, in general, an infinitary proof obligation, and so cannot be considered as an instance of proof within the (encoding of) the logical system at hand. However, in many commonly-arising situations (in particular, in typical applications of parameterization), it must

be shown that $\sigma : P \to P'$ is a presentation morphism where $P = (\Sigma, \Delta)$ is a finite basic presentation. This reduces to showing that $P' \Vdash \sigma\Delta$, and hence is an instance of structured search, as explored in [HP92] for presentations without the derive operation. But for general $P$ the proof obligations are not "internalizable" in this way. Finding a fully satisfactory answer to this question is the subject of ongoing research; see [Wir91], [Far92] for proposed solutions and relevant discussion.

It is useful to consider a notion of uniform encoding that is not based on treating LF as a logical system. The idea is to regard basic theory presentations as contexts (more or less as now), and to "internalize" the presentation-structuring operations in an extension of the LF type theory. In particular, the derive operation seems closely related to existential types [MP85]. Part of this program, for the fragment of the language without derive, is carried out in [HP92].

A related idea is to view the presentation-structuring operations as "internal" logical operations, and to explore the analogy with (higher-order) categorical logic. In this way we hope to obtain a better proof theory for both deriving consequences of structured presentations and deriving entailments between such presentations [HT92]. This would provide a simple way to represent proofs in structured presentations (since these would just be proofs in this richer logic) and to prove that a signature morphism is a presentation morphism (since this would reduce to an entailment between structured presentations). As mentioned earlier, such a proof system would necessarily involve altering the structure of presentations in the course of proof; this would be captured by rules allowing commutation of translate with derive (corresponding to the Beck condition) and of union with derive (Frobenius reciprocity), much in the style of the proof systems in [Wir91] and [Far92].

Finally, the language of structured presentations may be generalized to admit translation and inverse image along logic morphisms. This would allow for the combination of theories from several different logical systems, giving rise to an "inter-logic" search space similar to the "intra-logic" search space given by structured theory presentations. It would be interesting to develop these ideas further, and to consider their application to formal program development where there is some indication that such hybrid logics and inter-logic search will be of some use [ST88b].

**Acknowledgements:** Thanks to Rod Burstall (from DS) for earlier collaboration on structured theories. This research has been partially supported by the U.K. Science and Engineering Research Council, the ESPRIT-funded COMPASS basic research working group, and Edinburgh University (RH, DS, AT), Carnegie-Mellon University (RH), the Polish Academy of Sciences and Linköping University (AT).

# References

[AC92] E. Astesiano and M. Cerioli. Relationships between logical frameworks. In M. Bidoit and C. Choppy, editors, *Recent Trends in Data Type Specification*, pages 126–143, Springer-Verlag, 1992.

[AHMP87] A. Avron, F. Honsell, I. Mason and R. Pollack. Using typed lambda calculus to implement formal systems on a machine. Technical Report ECS–LFCS–87–31, Laboratory for Foundations of Computer Science, Edinburgh University, June 1987. To appear, *Journal of Automated Reasoning.*

[Avr91] A. Avron. Simple consequence relations. *Information and Computation*, 91(1):105–139, 1991.

[BG77] R. Burstall and J. Goguen. Putting theories together to make specifications. *Proc. 5th Intl. Joint Conf. on Artificial Intelligence*, Cambridge, Massachusetts, pages 1045–1058, 1977.

[BG80] R. Burstall and J. Goguen. The semantics of CLEAR, a specification language. In *Proceedings of Advanced Course on Abstract Software Specifications*, pages 292–332, Springer-Verlag, Copenhagen, 1980.

[BG81] R. Burstall and J. Goguen. An informal introduction to specifications using CLEAR. In R. S. Boyer and J S. Moore, editors, *The Correctness Problem in Computer Science*, pages 185–213, Academic Press, New York, 1981.

[BV85] C. Beierle and A. Voss. Implementation specifications. In H.-J. Kreowski, editor, *Recent Trends in Data Type Specification*, pages 39–53, Springer-Verlag, 1985.

[Car86] J. Cartmell. Generalised algebraic theories and contextual categories. *Annals of Pure and Applied Logic*, 32:209–243, 1986.

[Con86] R. L. Constable, *et. al. Implementing Mathematics with the NuPRL Proof Development System*. Prentice–Hall, Englewood Cliffs, NJ, 1986.

[dB80] N. G. de Bruijn. A survey of the project AUTOMATH. In J. P. Seldin and J. R. Hindley, editors, *To H. B. Curry: Essays in Combinatory Logic, Lambda Calculus, and Formalism*, pages 589–606, Academic Press, 1980.

[DGS92] R. Diaconescu, J. Goguen, and P. Stefaneas. Logical support for modularisation. In G. Plotkin and G. Huet, editors, *Proceedings of Workshop on Types and Logical Frameworks*, Cambridge Univ. Press, to appear, 1992.

[Ehr 82] H.-D. Ehrich. On the theory of specification, implementation and parameterization of abstract data types. *Journal of the Association for Computing Machinery* 29, 206–227, 1982.

[Ell89] C. Elliott. Higher-order unification with dependent function types. In *Proceedings of Rewriting Techniques and Applications*, Chapel Hill, NC, April 1989.

[Far92] Jordi Farrés-Casals. *Verification in ASL and Related Specification Languages*. PhD thesis, Report CST–92–92, Edinburgh University, 1992.

[FS87] J. Fiadeiro and A. Sernadas. Structuring theories on consequence. In D. Sannella and A. Tarlecki, editors, *Proc. of the Fifth Workshop on Specification of Abstract Data Types*, Gullane, September 1987.

[Gar92] P. Gardner. *Representing Logics in Type Theory*. PhD thesis, Report CST–93–92, Edinburgh University, 1992.

[GB84a] J. Goguen and R. Burstall. Introducing institutions. In E. Clarke and D. Kozen, editors, *Logics of Programs*, pages 221–256, Springer-Verlag, 1984.

[GB84b] J. Goguen and R. Burstall. Some fundamental algebraic tools for the semantics of computation. Part I: Comma categories, colimits, structures and theories. *Theoretical Computer Science*, 31:175–209, 1984.

[GB86] J. Goguen and R. Burstall. A study in the foundations of programming methodology: specifications, institutions, charters and parchments. In *Proc. Workshop on Category Theory and Computer Programming*, pages 313–333, Springer-Verlag, Guildford, 1986.

[GHM78] J. Guttag, E. Horowitz and D. Musser. Abstract data types and software validation. *Communications of the ACM*, 21(12):1048–1064, 1978.

[GL87] J.-Y. Girard and Y. Lafont. Linear logic and lazy computation. In H. Ehrig, R. Kowalski, G. Levi, and U. Montanari, editors, *TAPSOFT '87: Colloquium on Functional and Logic Programming and Specifications*, pages 52–66, Pisa, Italy, March 1987.

[GM81] J.A. Goguen and J. Meseguer. Completeness of many-sorted equational logic. *SIGPLAN Notices*, 16:24–32, 1981.

[GMW79] M. Gordon, R. Milner and C. Wadsworth. *Edinburgh LCF*. Springer-Verlag, 1979.

[Gog71] J.A. Goguen. Mathematical representation of hierarchically organized systems. In E. Attinger, editor, *Global Systems Dynamics*, pages 112–128, S. Karger, 1971.

[HHP93] R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. *Journal of the Association for Computing Machinery*, 40(1):143–184, 1993. Preliminary version in *Proceedings of the Symposium on Logic in Computer Science*, pages 194–204, Ithaca, New York, June 1987.

[HST89a] R. Harper, D. Sannella, and A. Tarlecki. Structure and representation in LF. Technical Report ECS–LFCS–89–75, Laboratory for Foundations of Computer Science, Edinburgh University, 1989. Abridged version in *Proceedings of the Symposium on Logic in Computer Science*, pages 226–237, Asilomar, California, June 1989.

[HST89b] R. Harper, D. Sannella, and A. Tarlecki. Logic representation in LF. *Proc. 3rd Summer Conference on Category Theory and Computer Science*, pages 250–272, Springer-Verlag, Manchester, 1989.

[HN88] S. Hayashi and H. Nakano. *PX: A Computational Logic. Foundations of Computing Series*, The MIT Press, Cambridge, Mass., 1988.

[HP92] R. Harper and F. Pfenning. A module system for a programming language based on the LF logical framework. Report CMU–CS–92–191, Carnegie Mellon University, 1992.

[HT92] R. Harper and A. Tarlecki. Logics, modularity and representation in a logical framework. Working draft, 1992.

[Mes89] J. Meseguer. General logics. *Proc. Logic Colloquium '87*, Granada. North Holland (1989).

[MP85] J. C. Mitchell and G. Plotkin. Abstract types have existential type. In *Proceedings of the 12th ACM Symposium on the Principles of Programming Languages*, pages 37–51, 1985.

[MS85] D. MacQueen and D. Sannella. Completeness of proof systems for equational specifications. *IEEE Transactions on Software Engineering*, SE-11:454–461, 1985.

[Pau87] L. Paulson. *Logic and Computation: Interactive Proof with Cambridge LCF*. Cambridge University Press, 1987.

[Pau92] L. Paulson. The Isabelle reference manual. Computer Laboratory, Cambridge University, 1992.

[Pfe89] F. Pfenning. Elf: a language for logic definition and verified metaprogramming. In *Proceedings of the Symposium on Logic in Computer Science*, pages 313–322, Asilomar, 1989.

[Pfe91] F. Pfenning. Logic programming in the LF logical framework. In G. Plotkin and G. Huet, editors, *Logical Frameworks*, Cambridge Univ. Press, 1991.

[Pym90] D. Pym. *Proofs, search and computation in general logic*. PhD thesis, Report CST–69–90, Edinburgh University, 1990.

[SB83] D. T. Sannella and R. M. Burstall. Structured theories in LCF. In *Proc. of the 8th Colloquium on Algebra and Trees in Programming*, pages 377–391, L'Aquila, Italy, 1983.

[SST92] D. Sannella, S. Sokołowski and A. Tarlecki. Toward formal development of programs from algebraic specifications: parameterisation revisited. *Acta Informatica*, to appear, 1992.

[ST85] D. Sannella and A. Tarlecki. Program specification and development in Standard ML. In *Proceedings of the 12th ACM Symposium on the Principles of Programming Languages*, pages 67-77, 1985.

[ST86] D. Sannella and A. Tarlecki. Extended ML: an institution-independent framework for formal program development. In *Proc. Workshop on Category Theory and Computer Programming*, pages 364–389, Springer-Verlag, Guildford, 1986.

[ST88a] D. Sannella and A. Tarlecki. Specifications in an arbitrary institution. *Information and Computation*, 76:165–210, 1988.

[ST88b] D. Sannella and A. Tarlecki. Toward formal development of programs from algebraic specifications: implementations revisited. *Acta Informatica*, 25:233–281, 1988.

[ST91] D. Sannella and A. Tarlecki. Extended ML: past, present and future. In *Proc. 7th Intl. Workshop on Specification of Abstract Data Types*, pages 297–322, Springer-Verlag, Wusterhausen, 1991.

[ST92] D. Sannella and A. Tarlecki. Toward formal development of programs from algebraic specifications: model-theoretic foundations. In *Proc. Intl. Colloq. on Automata, Languages and Programming*, pages 656–671, Springer-Verlag, Vienna, 1992.

[SW83] D. Sannella and M. Wirsing. A kernel language for algebraic specification and implementation. In *Proc. 1983 Intl. Conf. on Foundations of Computation Theory*, pages 413–427, Springer-Verlag, Borgholm, Sweden, 1983.

[SW92] D. Sannella and L. Wallen. A calculus for the construction of modular Prolog programs. *Journal of Logic Programming*, 12:147–177, 1992. Preliminary version in *Proc. 4th IEEE Symp. on Logic Programming*, San Francisco, 1987.

[Tar86] A. Tarlecki. Bits and pieces of the theory of institutions. In *Proc. Workshop on Category Theory and Computer Programming*, pages 334–363, Springer-Verlag, Guildford, 1986.

[TBG92] A. Tarlecki, R. Burstall, and J. Goguen. Some fundamental algebraic tools for the semantics of computation, Part III: Indexed categories. *Theoretical Computer Science*, 91:239–264, 1992.

[TWW82] J.W. Thatcher, E.G. Wagner, and J.B. Wright. Data type specification, parameterization, and the power of specification techniques. *ACM Transactions on Programming Languages and Systems*, 4:711–732, 1982.

[vD80] D. T. van Daalen. *The Language Theory of AUTOMATH.* PhD thesis, Technical University of Eindhoven, Eindhoven, Netherlands, 1980.

[Win81] G. Winskel. Synchronization trees. *Theoretical Computer Science*, 34:33–82, 1981.

[Wir91] M. Wirsing. Proofs in structured specifications. Report MIP–9008, Fakultät für Mathematik und Informatik, Universität Passau, 1991.

[Wir86] M. Wirsing. Structured algebraic specifications: a kernel language. *Theoretical Computer Science*, 42:123–249, 1986.