A Set-theoretic Setting for Structuring Theories in Proof Development*

Zhaohui Luo and Rod Burstall
Department of Computer Science, University of Edinburgh
The King's Buildings, Edinburgh EH9 3JZ, U.K.

April 1, 1992

Abstract

We present a meta-setting for structured theory development in proof development systems, based on which a theory-structuring language S-CLEAR is defined. A *frame* is a logic endowed with a *lattice* structure and a *renaming* mechanism which capture the basic notions for structured theory development. Besides providing basic theory operations, S-CLEAR supports generic theories. An important feature is that type-checking for the application of a generic theory is decidable. Parameterization also supports structure sharing between theories. Theory bases may be built up using these mechanisms and used for structured development of large proofs. The semantics of S-CLEAR is very simple and logic-independent.

1 Introduction

Interactive proof development systems have been of growing interests in recent years (see [LMR86] for a survey of existing theorem provers). In order for theorem provers to be used in real applications, it is generally believed that a notion of *theory* should be provided and theories in proof development systems should be structurally developed so that theory libraries can be developed systematically and large theorem-proving tasks can be conquered in a structured way.

The notion of theory is often intuitively used to denote a mathematical theory which is similar to that notion in mathematics. Typical examples of mathematical theories would be a theory of natural numbers, a theory of groups, *etc.*. However,

*The work reported here was done in 1988.

how such a notion should be used in a proof development environment seems to be not well investigated yet. Different theory manipulation mechanisms would give people rather different impressions of what a theory might be. Currently, a few proof development systems provide theory structuring facilities. Example systems in which such a notion has been considered include IPE [Rit87], Cambridge LCF [Pau87] and OBJ3 [GW88].

In this paper, we take a simple view that a *theory* in a proof development system basically consists of

- a *signature* consisting of a group of *symbols* representing the basic notions (say, constants and function symbols),
- a *hypothesis-set* consisting of a group of *hypotheses* (say, axioms), and
- a *theorem-set* consisting of the *proved theorems* (possibly together with their *proofs*).

Based on this and influenced by the notion of institution [GB84,90] and the ideas in designing specification language CLEAR [BG80][SB83], we propose a set-theoretic setting called *frames* based on which a theory-structuring language S-CLEAR is defined.

A frame endows a logic with a (meta-level) lattice structure and a renaming mechanism. This provides a set-theoretic setting in which basic relationships between theories can be captured. Specifically, the embedded partial order expresses a notion of *subtheory* and 'extension/restriction' of theories; the least upper bound operator expresses 'putting theories together'; and the greatest lower bound operator 'selecting parts of theories'. *Generic theories* (or *parameterized theories*) can be defined based on the renaming mechanism; it is a very useful tool in abstracting and structuring theories. Type-checking for applications of generic theories is decidable. Correct structure *sharing* between theories can be controlled by parameterization using generic theories; this sharing style is in a similar spirit of that in programming language Pebble [Bur84][BL84][LB88]. It can be seen that the semantics of S-CLEAR is very simple. It should not be difficult to implement the ideas in this paper in a proof development system as the examples show.

1.1 Institutions and frames

Before presenting the work, we first give a brief explanation of the notion of institution [GB84,90] and its similarity with and differences from the notion of frame we are about to describe. This would also be helpful for those familiar with institutions to understand the basic ideas of our current work.

The notion of institution was introduced [GB84] to formalize in general the informal notion of 'logical system' in a model-theoretic way. Informally, an institution consists of

- a category of signatures (which provides vocabularies for sentence constructions and signature changing facilities through signature morphisms), and for each signature,

- a set of sentences over the signature,

- a set of models over the signature, and

- a satisfaction relation between models and sentences (over the same signature),

such that when signatures change by a signature morphism, the satisfaction of sentences by models changes consistently. Based on the notion of institution, one can describe different logical systems in the same framework. This gives many benefits in various applications in theoretical computer science, for example, semantics for structured specification languages. (see [GB90] for a recent account.)

The work in this paper was strongly influenced by the notion of institution among others. The introduction of the notion of frame (see below) is based on a similar idea; it is intended to be used to describe an (arbitrary) logical system and, at the same time, to capture the basic notions for structured theory development. This exactly corresponds to the idea of structured specification as proposed in *e.g.*, Clear [BG80], ACT1 [EFH83], *etc.*. However, instead of taking a categorical and model-theoretic approach, we take a more set-theoretic and proof-oriented approach, which seems to be suitable for studying structured proof development. There are several basic differences between the notion of frame and that of institution and their corresponding notions like theory and parameterization in specifications:

- Signatures of a frame are *ordered* by a partial order (which furthermore gives a lattice structure over signatures) and signature changing is controlled by *renaming mappings* instead of signature morphisms;

- Each frame has a notion of *proof*, which relates hypotheses with sentences and determines a notion of consequence relation. This can be seen as a proof-theoretic counterpart to the notion of satisfaction which relates models with sentences in an institution;

- The basic notions in theory development (for proof development) like subtheories, theory enrichment and putting theories together are given based on the embedded partial orders and lattice operations instead of, as in the framework of institutions, by signature morphisms and categorical notions like pushout.

- As to the notion of *theory*, we do *not* view a theory as consisting of all of the logical consequences of the axioms, but as consisting of the *proved theorems* (with their proofs). This, in our view, is more suitable to describing theories in proof development systems and has a nice consequence that the problem of checking well-typedness of an application of a generic theory becomes decidable.

2 Frames and Theory Structuring

In this section, we define the notion of frame and show how the notion of theory can be described in a frame and, in particular, how the notions in theory structuring can be captured by set-theoretic (lattice-theoretic, more specifically) operations provided by a frame. We shall give two examples of frames: one for many-sorted equational logic (see [MG81] for example), a representative for 'ordinary' logics, and the other for the calculus of constructions [CH88], a typical type theory which can be viewed as a logical system. We believe that we have thus shown the generality of our setting for discussing theory structuring in proof development systems.

2.1 Frames

A frame is a meta-setting which endows a logical system with a lattice structure and a renaming mechanism so that some ideas in theory-structuring in proof development can be captured. In the following definition of frame,

- *Lat* is the class of lattices with 0 and 1 partially ordered by the sublattice relation;[1]

- *Set* is the class of sets partially ordered by set inclusion $\subseteq$; and

- $\stackrel{m}{\to}$ denotes the order-preserving (*i.e.*, monotonic) function space from a poset to a partially ordered class;

Here is the definition followed by some explanations.

[1]For a lattice A, we also use A to denote its supporting set and, use $\sqsubseteq_A$, $\sqcup_A$, $\sqcap_A$, 0_A and 1_A (often with the subscript A omitted) to denote the partial order, the least upper bound, the greatest lower bound, the least element and the greatest element, respectively, over A.

Definition 2.1 (frame) *A* frame F *is a five-tuple*

$$F = (\mathbf{Sig}, \mathbf{Sen}, \mathbf{Hyp}, \mathbf{Prf}, \mathbf{R})$$

consisting of

- $\mathbf{Sig} \in Lat$;
- $\mathbf{Sen} : \mathbf{Sig} \stackrel{m}{\to} Set$;
- $\mathbf{Hyp} : \mathbf{Sig} \stackrel{m}{\to} Lat$;
- $\mathbf{Prf}$*, a set-valued function which gives a set* $\mathbf{Prf}(\sigma, H, e)$ *of proofs for every signature* $\sigma \in \mathbf{Sig}$*, hypothesis* $H \in \mathbf{Hyp}(\sigma)$ *and sentence* $e \in \mathbf{Sen}(\sigma)$*; these sets satisfy the following condition: for* $\sigma, \sigma' \in \mathbf{Sig}$*,* $H \in \mathbf{Hyp}(\sigma)$*,* $H' \in \mathbf{Hyp}(\sigma')$*, and* $e \in \mathbf{Sen}(\sigma)$*,*
  - $\sigma \sqsubseteq \sigma' \wedge H \sqsubseteq H' \Rightarrow \mathbf{Prf}(\sigma, H, e) \subseteq \mathbf{Prf}(\sigma', H', e)$;
- $\mathbf{R}$*, a set of mappings (i.e., single-valued functions) such that, for* $r \in \mathbf{R}$*,*
  - $\forall \sigma \in \mathbf{Sig}.\ r\sigma \in \mathbf{Sig}$,
  - $\forall e \in \mathbf{Sen}(\sigma).\ re \in \mathbf{Sen}(r\sigma)$,
  - $\forall H \in \mathbf{Hyp}(\sigma).\ rH \in \mathbf{Hyp}(r\sigma)$*, and*
  - $\forall p \in \mathbf{Prf}(\sigma, H, e).\ rp \in \mathbf{Prf}(r\sigma, rH, re)$.

We now explain the motivation for the above definition.

Sig is the set of *signatures* which has a lattice structure. A signature is an entity consisting of the basic symbols from which sentences (formulas) of a logic are formed. The lattice structure over the signatures endows **Sig** not only with a subsignature relation $\sqsubseteq$ (a subsignature usually has 'fewer' symbols) but also with an assumption that any two signatures σ and σ' can be put together (combined) to get a new signature $\sigma \sqcup \sigma'$ which contain symbols in both σ and σ'. The 'common part' of two signatures s and s' is $\sigma \sqcap \sigma'$. The existence of a largest signature and a smallest signature is assumed. A typical example of signature structure is the power set of a universal set of symbols partially ordered by set inclusion.

$\mathbf{Sen}(\sigma)$ is the set of *sentences* over a signature σ which are the basic statements (or, propositions) in a logic. A sentence over a subsignature of σ is also a sentence over σ; this is expressed by the monotonicity of **Sen**.

$\mathbf{Hyp}(\sigma)$ is the set of *hypothesis units* over a signature σ. A hypothesis unit can be thought of as consisting of the axioms of a theory.[2] There is a lattice structure

[2] A hypothesis unit may be more general entities, for example, consisting of the axioms and rules of an object logic. In such a case, one may discuss 'structured specification of logics'.

over $\mathbf{Hyp}(\sigma)$ for every signature σ. In a simple and typical case, $\mathbf{Hyp}(\sigma)$ is the power set of $\mathbf{Sen}(\sigma)$ and the lattice structure is then naturally defined by the set operations; and in this case, we may call a hypothesis unit a *hypothesis set*. The lattice structure endows $\mathbf{Hyp}(\sigma)$ with a partial order. $H \sqsubseteq H'$ usually means that H has 'less power' or has 'fewer consequences' than H', and this is reflected in the condition for proofs. Putting two hypothesis units H and H' together, one gets the minimal hypothesis unit $H \sqcup H'$ such that it is stronger than both H and H'. (Dually for $H \sqcap H'$.) For $\sigma \sqsubseteq \sigma'$, $\mathbf{Hyp}(\sigma)$ is a sublattice of $\mathbf{Hyp}(\sigma')$; this is expressed by the monotonicity of $\mathbf{Hyp}$.

Hypothesis units and sentences are related through *proofs*. $\mathbf{Prf}(\sigma, H, e)$ is the set of proofs of sentence e (more precisely, the judgement, say, 'e is true') under the assumption of H. The proofs also induce a notion of consequence relation between hypothesis units and sentences ($H \vdash_\sigma e$ if and only if $\mathbf{Prf}(\sigma, H, e) \neq \emptyset$). A proof of a sentence over a subsignature of σ under the assumption of a weaker hypothesis unit than $H \in \mathbf{Hyp}(\sigma)$ is also a proof of the sentence under the assumption of H; this is expressed as the condition for $\mathbf{Prf}$.

The mappings in $\mathbf{R}$ are called *renamings* (or *renaming operations*), which are syntactic operations which rename basic symbols in the frame under certain conditions of respecting the required relations; those conditions say that, under a renaming $r \in \mathbf{R}$,

- a signature is transformed to a signature;
- a sentence over a signature σ is transformed to a sentence over the resulting signature of σ under r;
- a hypothesis unit over a signature σ is transformed to a hypothesis unit over the resulting signature of σ under r; and
- a proof of a sentence under the assumption of a hypothesis unit is transformed to a proof of the transformed sentence under the transformed hypothesis unit.

When a universal set of symbols is assumed, a typical renaming is a function from the symbol universe to itself which respects the above requirements. Renamings are used to deal with name controlling in theory structuring; particularly for applications of generic theories.

Some properties about frames are useful in considering theory-structuring operations. We give two definitions here which describe some technical conditions to be used later.

Definition 2.2 ($\sqcap$-preserving frames) *A frame is called* $\sqcap$-preserving *if the following hold for all signatures* $\sigma, \sigma' \in \mathbf{Sig}$*:*

$$\mathbf{Sen}(\sigma \sqcap \sigma') = \mathbf{Sen}(\sigma) \cap \mathbf{Sen}(\sigma')$$
$$\mathbf{Hyp}(\sigma \sqcap \sigma') = \mathbf{Hyp}(\sigma) \sqcap \mathbf{Hyp}(\sigma')$$
$$\mathbf{Prf}(\sigma \sqcap \sigma', H \sqcap H', e) = \mathbf{Prf}(\sigma, H, e) \cap \mathbf{Prf}(\sigma', H', e)$$

Definition 2.3 (left-closed frames) *A frame is called* left-closed *if*

$$\forall H \in \mathbf{Hyp}(1_{\mathbf{Sig}}) \forall \sigma \in \mathbf{Sig}.\ H \sqsubseteq 1_{\mathbf{Hyp}(\sigma)} \Rightarrow H \in \mathbf{Hyp}(\sigma)$$

where $1_{\mathbf{Sig}}$ *and* $1_{\mathbf{Hyp}(\sigma)}$ *are the largest signature and largest hypothesis unit over* σ*, respectively.*

2.2 The notion of theory in a frame

Based on an arbitrary frame, we can develop a notion of *theory* in it; that is, a theory consists of a signature, a hypothesis set over the signature and a set of theorems.

Definition 2.4 (theory) *Let* $F = (\mathbf{Sig}, \mathbf{Sen}, \mathbf{Hyp}, \mathbf{Prf}, \mathbf{R})$ *be a frame. A* theory *(in frame* F*) is a triple*

$$(\sigma, H, C)$$

consisting of

- *a signature* $\sigma \in \mathbf{Sig}$*,*
- *a hypothesis unit* $H \in \mathbf{Hyp}(\sigma)$*, and*
- *a set* C *of theorems which are sentence-proof pairs* (e, p) *with* $e \in \mathbf{Sen}(\sigma)$ *and* $p \in \mathbf{Prf}(\sigma, H, e)$*.*

We shall use $\mathbf{TH}_F$ *(or just* $\mathbf{TH}$*) to denote the set of theories (in frame* F*), which is the following set:*

$$\mathbf{TH} =_{\mathrm{df}} \sum \sigma \in \mathbf{Sig} \sum H \in \mathbf{Hyp}(\sigma).\ Pow(\mathbf{Thm}_\sigma(H))$$

where $\mathbf{Thm}_\sigma(H)$ *is the set of* theorems with proofs *(or simply* theorems*):*

$$\mathbf{Thm}_\sigma(H) =_{\mathrm{df}} \sum e \in \mathbf{Sen}(\sigma).\ \mathbf{Prf}(\sigma, H, e)$$

Pow is the power-set operator and $\sum$ *is the set-theoretic dependent sum operator, i.e., for a set* A *and a set-valued function* $B : A \to Set$*,*

$$\sum x \in A.\ B(x) =_{\mathrm{df}} \{\, (a, b) \mid a \in A, b \in B(a) \,\}$$

Note that the theorems in a theory are in general a *subset* of $\mathbf{Thm}_\sigma(H)$; in other words, it is meant to be the set of the *proved* theorems instead of all of the *provable* theorems $\mathbf{Thm}_\sigma(H)$. In practice, one only proves a (finite) set of the theorems in a theory and these proved theorems (together with their proofs) are incorporated as a part of the theory.

One may compare this notion of theory with that considered in algebraic specifications (*e.g.*, [BG80]). The latter considers the *closure* (*i.e.*, all of the consequences) of an axiom set as a theory and a specification denotes such a closure (usually an undecidable infinite set). When parameterized theories (theory procedures called in [BG80]) are considered, it is not decidable whether an application of a parameterized theory to actual parameters is legal. This undecidability seems to be harmless for specification languages. However, as theory-structuring languages for proof development systems, this decidability becomes rather important. When the notion of theory given in the above definition is adopted, the set of the proved theorems is always finite in practice and hence the problem of type-checking is decidable.

Notations We shall use the following notations in the sequel:

- **HYP** denotes the set of all hypothesis units:

$$\mathbf{HYP} = \bigcup_{\sigma \in \mathbf{Sig}} \mathbf{Hyp}(\sigma)$$

- **THM** denotes the set of all of the (provable) theorems with their proofs:

$$\mathbf{THM} = \bigcup_{\sigma \in \mathbf{Sig}} \bigcup_{H \in \mathbf{Hyp}(\sigma)} \mathbf{Thm}_\sigma(H)$$

- Renamings in $\mathbf{R}$ are extended to theorems, theorem-sets and theories in a natural way. Suppose $r \in \mathbf{R}$. Then,

  - for $(e, p) \in \mathbf{THM}$, $r(e, p) =_{\mathrm{df}} (re, rp)$;
  - for $C \subseteq \mathbf{THM}$, $rC =_{\mathrm{df}} \{\, r(e, p) \mid (e, p) \in C \,\}$;
  - for $(\sigma, H, C) \in \mathbf{TH}$, $r(\sigma, H, C) =_{\mathrm{df}} (r\sigma, rH, rC)$.

  Note that, by the conditions that a renaming satisfies, the above transformations send a theorem, a hypothesis unit and a theory to a theorem, a hypothesis unit and a theory, respectively.

- **sign** , **hyps** and **thms** are used as the obvious projection functions which take a theory and result in its signature, hypothesis unit and theorem-set, respectively.

We can define partial orders between theories. One of the natural ones is to require that each component of one theory is less than or equal to the corresponding component of the other, *i.e.*, for $A, B \in \mathbf{TH}$,

$$A \sqsubseteq B \overset{\mathrm{df}}{\Leftrightarrow} \mathbf{sign}\ (A) \sqsubseteq \mathbf{sign}\ (B) \wedge \mathbf{hyps}\ (A) \sqsubseteq \mathbf{hyps}\ (B) \wedge \mathbf{thms}\ (A) \subseteq \mathbf{thms}\ (B)$$

However, one can also define other reasonable orders according to different motivations. (See section 2.3 for an example.) We shall take a partial order over theories as a parameter when giving the semantics of the theory-structuring language in section 3.2.

2.3 EQ: a frame for many-sorted equational logic

In this and the following sections, we give two example frames, one for the many-sorted equational logic and the other for the calculus of constructions.

Many-sorted equational logic (see [MG81] for example) is chosen as a representative of the ordinary logics. We show how it can be endowed with a lattice structure to become a frame so that we can talk about theories as described in the previous section. We believe that this gives a rather general (and typical) way of viewing a logic as a frame.

A frame for many-sorted equational logic can be given as:

$$\mathbf{EQ} = (\mathbf{Sig}_{EQ}, \mathbf{Sen}_{EQ}, \mathbf{Hyp}_{EQ}, \mathbf{Prf}_{EQ}, \mathbf{R}_{EQ})$$

each of whose components is described as follows.

Signatures: A signature of **EQ** consists of a set of *sorts* and a set of function symbols with finite sequence of sorts as their ranks; more precisely,

$$\mathbf{Sig}_{EQ} =_{\mathrm{df}} \sum S \in Pow(Sorts).\ Pow(Opns_S)$$

where

- $Sorts$ is the set of sort symbols, and
- for a set of sorts $S \subseteq Sorts$, $Opns_S$ is the set of function symbols together with their ranks which are in $S^* \times S$, where, S^* is the set of finite sequences of sorts in S. We shall use $Opns$ to denote the set of all function symbols, *i.e.*,

$$Opns =_{\mathrm{df}} \bigcup_{S \subseteq Sorts} Opns_S$$

The lattice operations $\sqsubseteq$, $\sqcup$ and $\sqcap$ for $\mathbf{Sig}_{EQ}$ are the pairwise set operations $\subseteq$, $\cup$ and $\cap$, respectively. Note that the largest and smallest signatures are then $(Sorts, Opns)$ and $(\emptyset, \emptyset)$, respectively.

Sentences: A sentence in **EQ** is an equation between terms of the same sort indexed by a set of (free) variables; more precisely, for $\sigma = (S, O) \in \mathbf{Sig}_{EQ}$,

$$\mathbf{Sen}_{EQ}(\sigma) =_{\text{df}} \sum X \in Pow(V_S) \sum k \in S.\ T_\sigma(X^\circ)_k \times T_\sigma(X^\circ)_k$$

where

- V_S is the set of variables whose sorts are in S,
- X° is the S-indexed set of variables induced by $X \subseteq V_S$ defined as, for $k \in S$,

$$X^\circ_k =_{\text{df}} \{\, x \in X \mid sort(x) = k \,\}$$

  and
- $T_\sigma(X^\circ)$ is (the supporting set of) the term algebra of σ over X°.

Hypothesis units: A hypothesis unit (set) in **EQ** is just a subset of sentences; *i.e.*,

$$\mathbf{Hyp}_{EQ}(\sigma) =_{\text{df}} Pow(\mathbf{Sen}_{EQ}(\sigma))$$

The lattice operations for $\mathbf{Hyp}_{EQ}(\sigma)$ are just the set operations $\subseteq$, $\cup$ and $\cap$.

Proofs: Let $\vdash_\sigma$ be the consequence relation defined by the usual set of rules for many-sorted equational logic (see [MG81]). A proof of a sentence over σ under the assumption H is just a derivation of judgement $H \vdash_\sigma e$, *i.e.*,

$$\mathbf{Prf}_{EQ}(\sigma, H, e) =_{\text{df}} \{\, p \mid p \text{ is a derivation of } H \vdash_\sigma e \,\}$$

Renamings: The renamings for **EQ** are induced by the pairs of functions of the form

$$(f{:}Sorts \to Sorts,\quad g{:}Opns \to Opns)$$

such that g is rank-preserving for f, *i.e.*, for any $op \in Opns$ with rank uk, $g(op)$ has rank $f^\#(uk)$, where $f^\#$ is the free extension of f to $Sorts^*$. More precisely, any such pair (f, g) induces a renaming $r \in \mathbf{R}_{EQ}$ such that, for $\sigma = (S, O) \in \mathbf{Sig}_{EQ}$,

$$r\sigma = (f(S),\ g(O))$$

and the images for sentences, hypothesis sets and proofs are the free extensions of r.

Proposition 2.1 **EQ** *defined above is a frame. Furthermore, it is $\sqcap$-preserving and left-closed.*

The notion of theory as we described in the previous section can be specialized to this frame of equational logic. Examples of theory specifications and structuring will be discussed after we introduce a theory-structuring language.

Remark As we mentioned before, one may define other partial orders between theories for different frames, besides the one given in the last section. Here is another useful order between theories in **EQ**: for $A, B \in \mathbf{TH}_{EQ}$,

$$A \sqsubseteq B \overset{\mathrm{df}}{\Leftrightarrow} \mathbf{sign}\ (A) \sqsubseteq \mathbf{sign}\ (B) \wedge \mathbf{hyps}\ (A) \subseteq \mathbf{hyps}\ (B) \cup \{\ e \mid \exists p.(e,p) \in \mathbf{thms}\ (B)\ \}$$

This says that B is not weaker than A. In particular, we require that each hypothesis for theory A be either a hypothesis for B or a proved theorem in B. The theorems proved in A are not necessarily the assumptions or theorems in B. With this order, one then has a different and more useful parameterization mechanism for theory structuring.

As one may see from the above construction of the frame **EQ**, we can construct frames for ordinary logics which are presented in a proof-theoretic way (by giving inductively a set of formulas and a notion of proof which determines a notion of theoremhood). One can similarly deal with, for example, first-order (or higher-order) logics, modal logics, *etc.*.

We now turn to another kind of presentation of logical systems, that is, via type systems and explain how one may view them as frames.

2.4 CC: a frame for the calculus of constructions

The calculus of constructions [CH88] is a typed functional calculus which provides a nice formalism for constructive proofs in natural deduction style. By Curry-Howard principle of formulas-as-types [CF58][How69], there is an embedded (intuitionistic) higher-order logic in Constructions. Various extensions of the calculus have also been considered (*e.g.*, [Luo89]). We will use the notations and terminology in [Luo89,90], where one can also find how the type system can be viewed as a logic. We first explain intuitively how to endow a type theory like Constructions with a lattice structure to become a frame. (In fact, what is described below is a general way to do so for an arbitrary type theory, as the required properties like weakening are really a good type theory should have.)

A *context* in a type theory consists of a group of assumed constants with specified types and (logical) assumptions (*e.g.*, some axioms). In general, one may consider such a context Γ to consist of two parts: $\Gamma \equiv Sig_\Gamma, Hyp_\Gamma$ where Sig_Γ contains a sequence of constants and Hyp_Γ a sequence of (usually logical) assumptions about the constants. For example, the formalization of the semigroup theory may be expressed as the following context:

$$\Gamma_{SG} \equiv Sig_{SG}, Hyp_{SG}$$

where

$$Sig_{SG} \equiv X{:}Type_0, \circ{:}X \to X \to X$$

$$Hyp_{SG} \equiv ass{:}\Pi x, y, z{:}X.\ (x \circ (y \circ z) = (x \circ y) \circ z)$$

(Thus, ass is an assumed proof of the associativity axiom.)

Remark The separation of signature and axiomatic hypotheses seems to be desirable as well as possible in general. This idea is influenced by the idea of institution [GB84] and is reflected in our definition of frame.

Based on such a view that a context consists of two parts, we can consider how to deal with contexts so that a type theory may be endowed with a lattice structure to consider theory-structuring based on type theories. However, instead of dealing with contexts as sequences, we shall consider how contexts can be viewed as sets.

2.4.1 environments

It is in general easy to impose a lattice structure over sets, but it is less evident for sequences. As in usual presentations of a context in a theory of dependent types contexts are presented as sequences of bindings (*i.e.*, variable-term pairs) in a way that an assumed variable (or constant) may occur free in latter components, we try to consider how to view such contexts as sets in a consistent way without losing the useful information of dependency.[3]

A basic starting point of making a linear context into a set without losing useful information is to notice that the *weakening lemma* always holds for most (if not all) of the type theories; that is,

weakening lemma If $\Gamma \vdash M : A$ is a derivable judgement and Γ' is a valid context which contains every component of Γ, then $\Gamma' \vdash M : A$ is derivable as well.

[3]In general, one may consider more complicated structures of contexts (*e.g.*, as partially ordered trees). For our purpose here, considering them as sets is the simplest choice.

In the following, we introduce a notion of *environment* which are sets corresponding to contexts. We assume that the variables under consideration are all associated with their 'ranks' (*i.e.*, types); so it is assured that name-clashing between variables having different types will not occur.

Definition 2.5 (environments) *Let γ and γ' be finite sets of bindings (i.e., variable-term pairs of the form $x{:}M$) and $\#A$ denote the cardinality of a set A.*

1. *γ is called a* (valid) signature environment *if and only if there exists a permutation*
$$\eta : \{1, ..., \#\gamma\} \to \{1, ..., \#\gamma\}$$
such that, letting $\gamma = \{\, x_i{:}M_i \mid 1 \leq i \leq \#\gamma \,\}$,
$$\eta\gamma =_{\mathrm{df}} x_{\eta(1)}{:}M_{\eta(1)}, ..., x_{\eta(\#\gamma)}{:}M_{\eta(\#\gamma)}$$
is a valid context.

2. *γ' is called a* (valid) hypothesis environment under γ, *where γ is a signature environment by permutation η, if and only if*
   - *for every $x{:}M \in \gamma'$, M is an $\eta\gamma$-type, and*
   - *there exists a permutation*
$$\eta' : \{1, ..., \#\gamma'\} \to \{1, ..., \#\gamma'\}$$
   such that $\eta\gamma, \eta'\gamma'$ is a valid context.

Furthermore, we also take the (infinite) set of all variables with their types as a special signature environment; if γ is a signature environment with permutation η, the (infinite) set of all variables with their types which are $\eta\gamma$-types as a special hypothesis environment under γ.

The following is an important lemma which assures that the operations union and intersection between sets preserve the property of being an environment.

Lemma 2.1 *If γ_1 and γ_2 are signature environments, then so are $\gamma_1 \cup \gamma_2$ and $\gamma_1 \cap \gamma_2$. If γ_1' and γ_2' are hypothesis environments under γ_1 and γ_2, respectively, then $\gamma_1' \cup \gamma_2'$ and $\gamma_1' \cap \gamma_2'$ are hypothesis environments under $\gamma_1 \cup \gamma_2$ and $\gamma_1 \cap \gamma_2$, respectively.*

2.4.2 the frame CC

A frame for Constructions can be defined as:

$$\mathbf{CC} =_{\text{df}} (\mathbf{Sig}_{CC}, \mathbf{Sen}_{CC}, \mathbf{Hyp}_{CC}, \mathbf{Prf}_{CC}, \mathbf{R}_{CC})$$

each of whose components can be described as follows:

Signatures: The signatures of **CC** are just the signature environments.

The lattice operations are the operations $\subseteq$, $\cup$ and $\cap$ for sets. The smallest signature is the empty signature environment and the largest is the special signature environment containing all of the variables with their types. Note that, by lemma 2.1, the union and intersection of any two signatures are signatures as well.

Sentences: Let $\gamma \in \mathbf{Sig}_{CC}$ with η as its permutation. The sentences over γ are the $\eta\gamma$-types.

Hypothesis units: Let $\gamma \in \mathbf{Sig}_{CC}$ with η as its permutation. The hypothesis sets over γ are the hypothesis environments under γ.

The lattice operations are also $\subseteq$, $\cup$ and $\cap$ for sets. The smallest hypothesis set is the empty set and the largest is the special hypothesis environment under γ containing all variables with their types being $\eta\gamma$-types.

Proofs: For $\gamma \in \mathbf{Sig}_{CC}$ with η as its permutation, $\gamma' \in \mathbf{Hyp}_{CC}(\gamma)$ with η' as its permutation, and $A \in \mathbf{Sen}_{CC}(\gamma)$, a proof of A under γ' is a term which inhabits A in context $\eta\gamma,\ \eta'\gamma'$, *i.e.*,

$$\mathbf{Prf}_{CC}(\gamma, \gamma', A) =_{\text{df}} \{\, t \mid \eta\gamma, \eta'\gamma' \vdash t : A \,\}$$

(It is easy to show that $\mathbf{Prf}_{CC}(\gamma, \gamma', A)$ is uniquely defined, independent of η and η'.)

Renamings: The renamings for the frame **CC** are induced by the functions from the variable space Var to Var which are 'type-preserving' in the obvious sense with type dependency being preserved[4].

Proposition 2.2 **CC** *is a frame. Furthermore, it is left-closed and* $\sqcap$*-preserving.*

Proof By induction and using lemma 2.1

[4] For example, let $X{:}Type$ and $x{:}X$, and $rX = X'$ and $rx = x'$. Then, $r \in \mathbf{R}$ only if $X'{:}Type$ and $x'{:}X'$.

3 S-CLEAR — a Theory Structuring Language

In this section, we present a simple theory-structuring language which we shall call S-CLEAR. As its name suggests, S-CLEAR inherits many basic properties of the algebraic specification language CLEAR [BG80]. However, it is based on the notion of frame we introduced in the previous section. The semantics of S-CLEAR is frame-independent and very simple. In particular, applications of generic theories (or parameterized theory procedures) are in general decidable, since our semantics will be based on the notion of theory we considered in the previous section, which reflects the idea that the theorems included in a theory are just those *proved* ones instead of the provable ones. Furthermore, the parameterization mechanism can be used to control correct structure sharing in a style similar to that of Pebble [BL84][Bur84].

3.1 The syntax of S-CLEAR

The main category of expressions of S-CLEAR is that of *theory expressions*, defined by the following BNF-like grammar:

$$\begin{aligned} A ::= \;& x \mid \mathbf{theory}\ (\sigma, H, C) \mid \\ & \mathbf{join}(A, A') \mid \mathbf{meet}(A, A') \mid \\ & \mathbf{enrich}(A, \sigma, H, C) \mid \mathbf{select}(A, \sigma, H, C) \mid \\ & ([x_1{::}A_1, ..., x_n{::}A_n]A)(r, A'_1, ..., A'_n) \\ & rA \end{aligned}$$

where the A's range over theory expressions, x's over *theory variables*, and σ, H, C and r range over *signature expressions*, *hypothesis-unit expressions*, *theorem-set expressions* and *renaming expressions*, respectively; these latter categories of expressions are given by the following grammars:

$$\begin{aligned} \sigma &::= \mathbf{sign}\ \sigma_0 \mid r\sigma \\ H &::= \mathbf{hyps}\ H_0 \mid rH \\ C &::= \mathbf{thms}\ C_0 \mid rC \\ r &::= r_0 \mid r \circ r' \end{aligned}$$

where σ_0, H_0, C_0 and r_0 stand for atomic expressions of signatures, hypothesis-sets, theorem-sets and renamings, respectively, which are frame-dependent and left open here. (We shall see examples of them for **EQ** in latter examples.)

The semantics of the operations given in the above syntax will be formally given in the next section. We first give some intuitive explanations here. The *basic theory*

$$\mathbf{theory}\ (\sigma, H, C)$$

stands for the theory consisting of the signature, hypothesis unit and theorem set denoted by σ, H and C, respectively.

The operation **join** puts two theories together; that is,

$$\mathbf{join}(A, A')$$

denotes the least upper bound of the theories denoted by A and A'. Dually,

$$\mathbf{meet}(A, A')$$

stands for the greatest lower bound — the shared common part of the theories denoted by A and A'.

The operation **enrich**, as the name suggests, enlarges a theory by the given compatible signature, hypothesis unit and theorems; while **select** selects a part of a theory. One can view **select** as an operation complementary to an operation 'delete'; that is, we can delete some part of a theory to form a new theory if what is left after the deletion is indeed a theory.

If r is a renaming expression, the theory expression rA, roughly speaking, denotes the theory obtained by the theory expression got from A by renaming the basic symbols in A by the renaming denoted by r (see below for a precise semantics).

An expression of the form

$$[x_1 :: A_1, ..., x_n :: A_n]A$$

may be viewed as expressing a *generic theory expression* with n parameters; using a syntactic sugar, we may declare a generic theory as follows by assigning it a name G:

Generic G $(x_1 :: A_1, ..., x_n :: A_n)$
body A
end

which can be applied to n theory expressions $A'_1, ..., A'_n$ through a renaming r as

$$G(r, A'_1, ..., A'_n)$$

The semantics of such an application shall be given in the next section.

Remark Note that we do not formally take generic theory expressions as entities in the language but only their applications are considered. We could have introduced **let**-expressions to treat declarations of generic theories formally. Also, we could have introduced syntactic operations such as **join** for signatures, hypothesis units and theorem-sets and allow parameterization over them. For simplicity, these are not dealt with here.

The renaming expressions are supposed to be *syntactic* operations rather than semantic. We give the following *reduction rules* for renamings. *Convertible* theory expressions are considered to be identical, where convertibility is defined as the reflexive, symmetric and transitive closure of the following notion of one-step reduction.

Definition 3.1 (renaming reduction on expressions) *The* renaming reduction *relation on the theory expressions is induced by the following contraction schemata, where r and r' are arbitrary renaming expressions and G is a generic theory expressions as defined above:*

$$rx \quad x$$

$$\begin{array}{rl} r(r'A) & (r \circ r')A \\ r(\mathbf{join}(A, A')) & \mathbf{join}(rA, rA') \\ r(\mathbf{meet}(A, A')) & \mathbf{meet}(rA, rA') \\ r(\mathbf{enrich}(A, \sigma, H, C)) & \mathbf{enrich}(rA, r\sigma, rH, rC) \\ r(\mathbf{select}(A, \sigma, H, C)) & \mathbf{select}(rA, r\sigma, rH, rC) \\ r(G(r', A'_1, ..., A'_n)) & G(r \circ r', rA'_1, ..., rA'_n) \end{array}$$

The renaming reduction on expressions b *of signatures, hypothesis units or theorem sets, is induced by the following contraction:*

$$r(r'b) \quad (r \circ r')b$$

The expressions of the forms in the left hand side above are called redexes *with the corresponding right-hand-side expressions called their* contractums. *An expression is in* normal form *if it contains no redex.*

Proposition 3.1 (normalization) *There is no infinite reduction sequence. And, every expression reduces to a unique normal form under all reducing orders.*

Henceforth, we will write $\mathrm{nf}(A)$ for the normal form of an expression A.

3.2 The semantics of S-CLEAR

The semantics of S-CLEAR will be given subject to an arbitrary frame which is $\sqcap$-preserving and left-closed; let $F = (\mathbf{Sig}, \mathbf{Sen}, \mathbf{TH}, \mathbf{Prf}, \mathbf{R})$ be such a frame.

First, we introduce some (semantic) theory operations corresponding to the syntactic operations **join**, **meet**, **enrich** and **select**. In the following definition, we use $\prod$ to denote the operator for set-theoretic product (or dependent function space), *i.e.*, for set A and a set-valued function $B : A \to Set$,

$$\prod x \in A.\ B(x) =_{\mathrm{df}} \{\, f : A \to \bigcup_{x \in A} B(x) \mid \forall x \in A.\ f(x) \in B(x) \,\}$$

Definition 3.2 (some theory operations)

- *The union operation on theories:*

$$\sqcup : \mathbf{TH} \times \mathbf{TH} \to \mathbf{TH}$$

  is defined as the component-wise 'unions':

$$A \sqcup B =_{\mathrm{df}} (\mathbf{sign}\ (A) \sqcup \mathbf{sign}\ (B), \mathbf{hyps}\ (A) \sqcup \mathbf{hyps}\ (B), \mathbf{thms}\ (A) \cup \mathbf{thms}\ (B))$$

- *The meet operation on theories:*

$$\sqcap : \mathbf{TH} \times \mathbf{TH} \to \mathbf{TH}$$

  is defined as the component-wise 'meets':

$$A \sqcap B =_{\mathrm{df}} (\mathbf{sign}\ (A) \sqcap \mathbf{sign}\ (B), \mathbf{hyps}\ (A) \sqcap \mathbf{hyps}\ (B), \mathbf{thms}\ (A) \cap \mathbf{thms}\ (B))$$

- *The enrichment operation is a function of the following type:*

$$\begin{aligned} enrich \ : \ & \prod A \in \mathbf{TH} \\ & \prod \sigma \in \{\, \sigma' \in \mathbf{Sig} \mid \mathbf{sign}\ (A) \sqsubseteq \sigma' \,\} \\ & \prod H \in \mathbf{Hyp}(\sigma) \\ & \prod C \in Pow(\mathbf{Thm}_{\sigma}(\mathbf{hyps}\ (A) \sqcup H)).\ \ \mathbf{TH} \end{aligned}$$

  defined as

$$enrich(A, \sigma, H, C) =_{\mathrm{df}} (\sigma, \mathbf{hyps}\ (A) \sqcup H, \mathbf{thms}\ (A) \cup C)$$

- *The selection operation is a function of the following type:*

$$\begin{aligned} \mathit{select} \;\; : \;\; & \prod A \in \mathbf{TH} \\ & \prod \sigma \in \{\, \sigma' \in \mathbf{Sig} \mid \sigma' \sqsubseteq \mathbf{sign}\ (A) \,\} \\ & \prod H \in \{\, H \in \mathbf{Hyp}(1_{\mathbf{Sig}}) \mid H \sqsubseteq \mathbf{hyps}\ (A) \,\} \\ & \prod C \in \mathit{Pow}(\mathbf{thms}\ (A)).\;\; \mathbf{TH} \end{aligned}$$

defined as

$$\mathit{select}(A, \sigma, H, C) =_{\mathrm{df}} (\sigma, H \sqcap 1_{\mathbf{Hyp}(\sigma)}, C \cap \mathbf{Thm}_\sigma(H \sqcap 1_{\mathbf{Hyp}(\sigma)}))$$

When these operations are applied to arguments that are not in the required domains, they are undefined.

Remark $\sqcup$ and *enrich* are well-defined in any frame where the requirement that the frame under consideration be $\sqcap$-preserving and left-closed is not needed. *select* is well-defined in any left-closed frame. $\sqcap$ is well-defined in any frame which is left-closed and $\sqcap$-preserving.

Now, we give the semantics of the theory expressions. In the definition, we write Var, $ThExp$, $RenExp$, $SigExp$, $HypExp$ and $ThmExp$ for the sets of theory variables, theory expressions, renaming expressions, signature expressions, hypothesis-unit expressions and theorem-set expressions, respectively.

Definition 3.3 (semantics) *Let ρ be a (theory) valuation, i.e., a function $\rho{:}Var \to$ $\mathbf{TH}$ and assume that the semantics $[\_]$ of the atomic expressions is given, that is, for any atomic expression c of renamings, signatures, hypothesis-sets and theorem-sets, $[c]$ is an element of $\mathbf{R}$, $\mathbf{Sig}$, $\mathbf{HYP}$ and $\mathbf{THM}$, respectively.*

Then, we define the semantics of the theory expressions (under valuation ρ) as a partial function

$$[\![\_]\!]_\rho{:}ThExp \rightharpoonup \mathbf{TH}$$

as follows:

1. *Variable theory expressions: for $x \in Var$,*

$$[\![x]\!]_\rho =_{\mathrm{df}} \rho(x)$$

2. *Basic theory expressions:*

$$[\![\mathbf{theory}\ \sigma, H, C]\!]_\rho =_{\mathrm{df}} \begin{cases} ([\![\sigma]\!], [\![H]\!], [\![C]\!]) & \textit{if } ([\![\sigma]\!], [\![H]\!], [\![C]\!]) \in \mathbf{TH} \\ \uparrow & \textit{otherwise} \end{cases}$$

where 'undefined' is denoted by $\uparrow$.

3. *Join operation:*
$$[\![\mathbf{join}(A,B)]\!]_\rho =_{\rm df} [\![A]\!]_\rho \sqcup [\![B]\!]_\rho$$

4. *Meet operation:*
$$[\![\mathbf{meet}(A,B)]\!]_\rho =_{\rm df} [\![A]\!]_\rho \sqcap [\![B]\!]_\rho$$

5. *Enrichment operation:*
$$[\![\mathbf{enrich}(A,\sigma,H,C)]\!]_\rho =_{\rm df} enrich([\![A]\!]_\rho, [\![\sigma]\!], [\![H]\!], [\![C]\!])$$

6. *Selection operation:*
$$[\![\mathbf{select}(A,\sigma,H,C)]\!]_\rho =_{\rm df} select([\![A]\!]_\rho, [\![\sigma]\!], [\![H]\!], [\![C]\!])$$

7. *Renaming of a theory:*
$$[\![rA]\!]_\rho =_{\rm df} \begin{cases} [\![r]\!]([\![A]\!]_\rho) & \textit{if } A \textit{ is of the form } \mathbf{theory}\ (\sigma,H,C) \\ [\![\mathrm{nf}(rA)]\!]_\rho & \textit{otherwise} \end{cases}$$

8. *Applications of generic theories:*
$$[\![([x_1{::}A_1,...,x_n{::}A_n]A)(r,A_1',...,A_n')]\!]_\rho =_{\rm df} \begin{cases} [\![rA]\!]_{\rho[x_i\mapsto[\![A_i']\!]_\rho]} & \textit{if } [\![rA_i]\!]_\rho \sqsubseteq [\![A_i']\!]_\rho \\ \uparrow & \textit{otherwise} \end{cases}$$
where $\rho[x_i \mapsto [\![A_i']\!]_\rho]$ *is the valuation that is the same as* ρ *except that* x_i *is assigned to* $[\![A_i']\!]_\rho$ *(*$i = 1,...,n$*) and* $\sqsubseteq$ *is the partial order between theories defined for the underlying frame*[5]*.*

In the above, the semantics for expressions of renamings, signatures, hypothesis units and theorem-sets are functions defined as follows:

- *renamings:* $[\![]\!] : RenExp \to \mathbf{R}$ *given by*
$$[\![r_0]\!] =_{\rm df} [r_0]$$
$$[\![r \circ r']\!] =_{\rm df} [\![r]\!] \circ [\![r']\!]$$
where $\circ$ *in the right-hand-side is the composition of maps.*

- *signatures:* $[\![\_]\!] : SigExp \to \mathbf{Sig}$ *given by*
$$[\![\mathbf{sign}\ s_0]\!] =_{\rm df} [\sigma_0]$$
$$[\![r\sigma]\!] =_{\rm df} \begin{cases} [\![r]\!]([\sigma_0]) & \textit{if } \sigma \equiv \mathbf{sign}\ \sigma_0 \\ [\![\mathrm{nf}(r\sigma)]\!] & \textit{otherwise} \end{cases}$$

[5]As we have noted before, we take such a partial order as a parameter of the semantics.

- *hypothesis units:* $[\![\_]\!]$ *:ThExp* $\to$ **TH** *given by*

$$[\![\textbf{hyps}\ H_0]\!] =_{\rm df} [H_0]$$

$$[\![rH]\!] =_{\rm df} \begin{cases} [\![r]\!]([H_0]) & \textit{if}\ \ H \equiv \textbf{hyps}\ H_0 \\ [\![\mathrm{nf}(rH)]\!] & \textit{otherwise} \end{cases}$$

- *theorem sets:* $[\![\_]\!]$ *:ThmExp* $\to$ **THM** *given by*

$$[\![\textbf{thms}\ C_0]\!] =_{\rm df} [C_0]$$

$$[\![rC]\!] =_{\rm df} \begin{cases} [\![r]\!]([C_0]) & \textit{if}\ \ C \equiv \textbf{thms}\ C_0 \\ [\![\mathrm{nf}(rC)]\!] & \textit{otherwise} \end{cases}$$

Some notes on the above definition of semantics follow.

1. Renamings are *syntactic* operations; this is reflected in the semantics of rA (clause 7) which is defined in the following way: we first *normalize* the expression rA and then calculate its denotation. For example, the semantics of theory expression
$$r\ \textbf{join}(A, B)$$
is *not*
$$[\![r]\!]([\![\textbf{join}(A, B)]\!])$$
but
$$[\![\textbf{join}(rA, rB)]\!] = [\![rA]\!] \sqcup [\![rB]\!]$$
This syntactic feature of renamings is important for the mechanism of generic theories, as we see below.

2. The definition of the semantics for generic theory applications (clause 8) is the central part of the above definition. Suppose that G is the generic theory declared as follows:

   Generic G $(x_1{::}A_1, x_2{::}A_2)$
   body B
   end

   and A_1' and A_2' are two theory expressions. Then, G can legally be applied to A_1' and A_2' through a renaming r subject to some *satisfaction requirements*. The application
$$G(r, A_1', A_2')$$

is intended to denote the theory expressed by the body of G (B in this case) after the formal variables x_1 and x_2 are instantiated as the theories denoted by the actual arguments A_1' and A_2'; that is, $G(r, A_1', A_2')$ denotes the theory

$$[\![rB]\!]_{\rho[x_1 \mapsto [\![A_1']\!]_\rho,\ x_2 \mapsto [\![A_2']\!]_\rho]}$$

Renamings play an essential role in such an application. The most basic role of renaming r is to rename the symbols in the body of the generic theory to match the symbols in the theories applied to so that the resulting theory can be formed as intended. Note that it is their syntactic feature that makes renamings play such a role in the intended way. Furthermore, as a renaming may map different symbols to the same one, *structure sharing* can be controlled by parameterization. (See later for more discussions on this.)

The above application of generic theories is only 'legal' under the condition that, under renaming r, the theory denoted by the actual argument A_i' *satisfies* the requirements specified by the formal parameter theory expression A_i, *i.e.*,

- $[\![rA_1]\!] \sqsubseteq [\![A_1']\!]$ and $[\![rA_2]\!] \sqsubseteq [\![A_2']\!]$,

The formal parameters A_i specifies the 'type' of the legal actual arguments so that the resulting theory is well-formed and indeed the intended one.

Examples of applications of generic theories will be given in the following section.

3. The semantics of the other theory-structuring operations are simple and clear enough not to need explanation. We remark that one might consider an auxiliary operation **delete**, a complement to **select**, which removes some symbols, hypotheses and theorems from a given theory to form a new theory. This can not be directly defined in our semantic framework as there is no complementary operator in our lattice structure. However, **select** can be used to mimic **delete** at the meta-level — to delete a part of a theory is just to select the remaining part.

4. In the current language, every expression of signature, hypothesis unit or theorem set has its normal form of the form

$$(r_1 \circ \ldots \ \circ r_n)b_0$$

where b_0 is a basic expression (*e.g.*, **sign** σ_0). The semantics of the above is simply

$$([\![r_1]\!] \circ \ldots \circ [\![r_n]\!])([\![b_0]\!])$$

4 Theory Examples in S-CLEAR

In this section, we give some examples of theory development in the language S-CLEAR described in the previous section. These examples, which are given in the frame **EQ** of the many-sorted equational logic as described in section 2.3, show how the basic structuring operations can be used to construct larger theories by 'putting together' several smaller theories or extract useful parts of a large theory to form theories interested. Examples of generic theories will show the power of parameterization in abstract and structured reasoning.

In the examples, we shall use some syntactic sugars to ease readability, most of which we think are clear enough to self-explain. In particular, we shall use '**with** ... **as** ...' as the syntactic sugar for renamings.

4.1 Basic examples

We can define the theories for semigroups, monoids, groups and rings by stepwise enrichments as follows. The proved theorems for each theory are omitted as '... ...'.

semigroup $\equiv$ **theory**
  sign X, $\circ{:}X^2 \to X$
  hyps $\forall x_1, x_2, x_3 \in X.\ x_1 \circ (x_2 \circ x_3) = (x_1 \circ x_2) \circ x_3$
  thms ... ...

monoid $\equiv$ **enrich** **semigroup**
  sign $id{:}X$
  hyps $\forall x \in X.\ id \circ x = x \wedge x \circ id = x$
  thms ... ...

group $\equiv$ **enrich** **monoid**
  sign $-1{:}X \to X$
  hyps $\forall x \in X.\ x \circ x^{-1} = id$
  thms ... ...

ring $\equiv$ **enrich** **join**(**group with** $+$,0 **as** $\circ, id$
  monoid with $\times$,1 **as** $\circ, id$)
  sign
  hyps $\forall a, b \in X.\ a + b = b + a$

$$\forall a, b, c \in X.\ a \times (b + c) = (a \times b) + (a \times c)$$
$$\forall a, b, c \in X.\ (b + c) \times a = (b \times a) + (c \times a)$$

thms ... ...

If one have first defined the theory of groups directly as follows,

group $\equiv$ **theory**
- **sign** $X,\ \circ{:}X^2 \to X,\ id{:}X,\ {-1}{:}X \to X$
- **hyps** $\forall x_1, x_2, x_3 \in X.\ x_1 \circ (x_2 \circ x_3) = (x_1 \circ x_2) \circ x_3$
  - $\forall x \in X.\ id \circ x = x$
  - $\forall x \in X.\ x \circ id = x$
  - $\forall x \in X.\ x \circ x^{-1} = id$
- **thms** ... ...

then, the theories **semigroup** and **monoid** can be defined by using **select** operation to select the relevant parts of **group**; for instance,

monoid $\equiv$ **select** **group**
- **sign** X, $\circ$, id
- **hyps** **hyps** (**group**)
- **thms** ... ...

Note that, according to the semantics of the operation **select**, only the hypotheses about the function symbols $\circ$ and id are selected as those for **monoid**.

4.2 Some examples of generic theories

Generic theories are normally used to express abstraction in the course of theory development. We give examples to show how they may be used in the following aspects:

1. abstract reasoning,
2. structured theory development, and
3. simulation of 'open/close' operations for theory libraries.

Recall that the semantics of generic theory applications are parameterized by a partial order between theories. For **EQ**, we take the order described in section 2.3, that is,

$A \sqsubseteq B \stackrel{\mathrm{df}}{\Leftrightarrow}$ **sign** $(A) \sqsubseteq$ **sign** $(B) \wedge$ **hyps** $(A) \subseteq$ **hyps** $(B) \cup \{\, e \mid \exists p.(e, p) \in$ **thms** $(B)\,\}$

Example (theorems inheritance) For any theory A, we can define a family of generic theories **Inherit_**A, one for each theory A, which applied to an actual theory A_0 will result in a new theory that inherits all of the theorems in A as its own theorems:

Generic Inherit_A (x::A)
body **enrich** x **by thms** (A)
end

For instance, for the theory **group** defined in the last section, we can define

Generic Inherit_group (x::**group**)
body **enrich** x **by thms** (**group**)
end

If **int** is a theory of integers specified as follows:

int $\equiv$ **theory**
  sign int, +: $\text{int}^2 \rightarrow$ int, -:int$\rightarrow$int, 0: int, ...
  hyps axioms for integers
  thms '(int,+,0,-) is a group', ...

then the application

$$\textbf{Inherit\_group}(\textbf{int})\ \textbf{with}\ \ X, \circ, id, -1\ \textbf{as}\ \ int, +, 0,$$

will result in the theory of integers with all theorems proved in **group** as its theorems as well. This gives a way of *abstract reasoning*; one can first prove theorems about some 'abstract' theories (like **group**) and take advantage of genericity to inherit the proved theorems to various theories (like **int**) which satisfy the hypotheses of the abstract theory. Note that, according to the partial order between theories above, we only require that the hypotheses (but not the theorems) in **group** be contained in **int** as theorems (or hypotheses).

One may also use generic theories to structure theory development (for example, when building up a theory library). The following simple example shows how parameterization can be used to control correct structure sharing when trying to develop larger theories from smaller ones.

Example (sharing by parameterization) We specify a simple generic theory **distr** as follows, where we use '**sign** ...' to denote a theory whose hypothesis set and theorem set are empty:

Generic distr (x :: **sign** X, $*{:}X^2 \to X$, y :: **sign** X, $+{:}X^2 \to X$)
body **enrich join**(x, y) **by hyps** $Distr(X, +, *)$
end

where $Distr(X, +, *)$ expresses the distributive laws about the binary function symbols $+$ and $*$ (*c.f.*, the specification of the theory **ring** before). This generic theory **distr**, when applied successfully, will combine the two theories as actual arguments with an additional hypothesis of the distributive laws. For instance, applying it to a theory of monoids and a theory of a theory of abelian groups will give us a theory of rings.

Although very simple, this example explains an important mechanism generic theories provide for structured theory development — structure sharing by parameterization. Note that the two formal parameters of **distr** has the same sort symbol X. First, this is necessary for the hypothesis $Distr(X, +, *)$ to make sense. More importantly, it guarantees that the two actual arguments of the generic theory must have the **same** sort symbol corresponding to X as well; otherwise, type-checking for the application will fail. This style of structure sharing is similar to and inspired by that in the programming language PEBBLE [BL84][Bur84].

As the final example in this section, we show that the generic theory mechanism can be used to simulate the operations of opening and closing a theory.

Example (open and close by genericity) Every theory A can be *closed* by being made into a generic theory:

Generic close$_A$ ()
body A
end

In order to use such a theory, one must provide a renaming to *open* ('apply') it; Application opens the closed theory and results in a theory with names being renamed by the supplied renaming.

The above example shows that it is possible to build up theory libraries using the mechanism of generic theories. In general, a theory library consists of developed generic theories; in other words, when a library developer puts a theory into the library, he closes the theory (*i.e.*, he makes it generic). When the theories are used, they are opened by application using different renamings and different theories as actual arguments to form new theories. (See Figure 1.) To summarize, the names in a closed (parameterized) theory are just 'placeholders' and, when they are opened, real names are provided by renamings. In practice, this should make

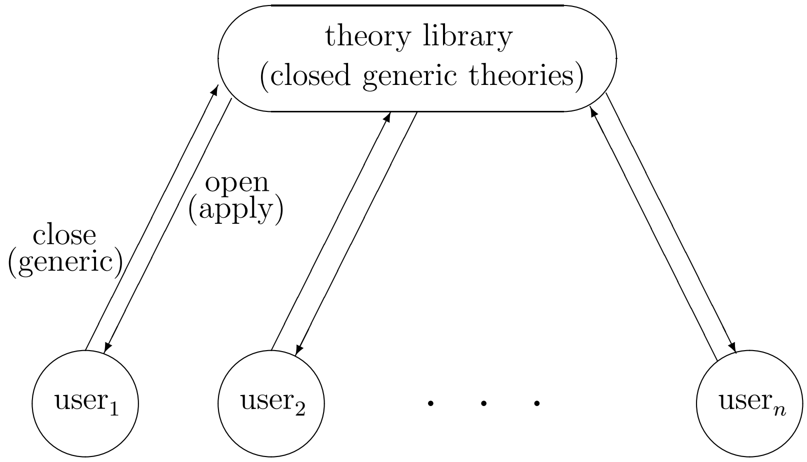

Figure 1: Theory library.

it possible for different people working cooperatively on a large theorem-proving task communicating through a common theory library.

Remark In general, we assume that there is a global name space from which users can choose names for entities to be described. This assumption of global name space might seem to violate a desirable property of 'independence' of theory development. However, our suggestion is that a group of cooperative users (or a user doing a large theorem-proving task) should use a theory library as described above as a tool of communication. We remark that a theory version control mechanism would be very helpful.

5 Discussions and Related Work

Frames are introduced as a set-theoretic meta-setting for theory development in proof development based on the idea that lattice structures can be used to capture the basic notions of structured theory constructions. A simple language S-CLEAR for structured theory development is described; its semantics is given based on arbitrary frames (and hence in a logic-independent way). There are several topics for further research and some related work to be discussed in this section.

The renaming operations considered in this paper are syntactic operations which are used to deal with name-control in proof development. We remark here that the usual naming convention of 'dot notation', which appears in various programming or specification languages, may be introduced as a special case of renamings. In other words, we may *define* the meaning of the dot notation as that of some special renaming operation. Consider the following syntax of renamings with $\mathbf{dot}_N$ as a new renaming expression:

$$r ::= \ r_0 \mid r \circ r' \mid \mathbf{dot}_N$$

where N stands for an arbitrary name. With such a special renaming expression **dot**, the renaming reduction remains the same as before; in particular, for a theory variable x, $\mathbf{dot}_N x$ contracts to x. We can assign the basic semantics of a renaming of the form $\mathbf{dot}_N$ as

$$[\mathbf{dot}_N] : n \mapsto N.n$$

Based on this, naming a theory specified by theory expression A by name N can be done by introducing the following definition (a special zero-argument generic theory):

$$N \ =_{\mathrm{df}} \ \mathbf{dot}_N(A)$$

One may also have a special useful renaming expression of the form

$$\mathbf{dot}_N \ \mathbf{except} \ n_1, ..., n_m$$

which will denote the renaming

$$[N.n_1 \mapsto n_1, ..., N.n_m \mapsto n_m] \circ [\mathbf{dot}_N]$$

Among the related work, the work on institutions [GB84,90] and algebraic specification languages (like Clear [BG80]) has most influenced ours. In [SB83], a structured theory development approach to LCF is discussed, which is later further developed in a more general setting by considering the Edinburgh Logical Framework [HST89] and is related to the idea of using consequence relations to describe a logical system [FS88]. Category theory is used in these approaches while we use set-theoretic notions. Also, the notion of theory they take is that which contains all of the logical consequences, and we do not feel that this is suitable for proof development systems and it causes problems when parameterization of theories is considered, as we have discussed.

Based on a type-theoretic setting, one can also talk about the notion of theory as shown in [Luo91][LPT89], where Σ-types are used to describe a notion of theory and an approach to structured abstract reasoning is developed. It would be interesting to compare the two approaches and look for a way of combining them into a unified proof development environment.

acknowledgements We would like to thank Don Sannella for his helpful remarks on this work.

References

[BG80] R. Burstall and J.Goguen, *'The Semantics of CLEAR, a Specification Language',* LNCS 86.

[BL84] R. Burstall and B.Lampson, *'Pebble, a Kernel Language for Modules and Abstract Data Types',* LNCS 173.

[Bur84] R. Burstall, *'Programming with Modules as Typed Functional Programming',* Proc. Inter. Conf. on Fifth Generation Computer Systems, Tokyo.

[CH88] Th.Coquand and G.Huet, *'The Calculus of Constructions',* Information and Computation, 2/3, vol. 76.

[EFH83] H. Ehrig, W. Fey and H. Hansen, *ACT ONE: an Algebraic Specification Language with Two Levels of Semantics*, Report 83-03, Technical University of Berlin, Fachbereich Informatik, 1983.

[FS88] J. Fiadeiro and A. Sernadas, *'Structuring Theories on Consequences',* in Recent Trends in Data Type Specification (eds., D. Sannella and A. Tarlecki), Springer-Verlag.

[GB84] J. Goguen and R. Burstall, *'Introducing Institutions',* LNCS 164.

[GB90] J. Goguen and R. Burstall, *Institutions: Abstract Model Theory for Specification and Programming,* LFCS Report Series ECS-LFCS-90-106, Dept. of Computer Science, University of Edinburgh.

[GW88] J. Goguen and T. Winkler, *Introducing OBJ3*, SRI report SRI-CSL-88-9, 1988.

[HST89] R. Harper, D. Sannella and A. Tarlecki, *'Structure and Representation in LF',* Proc. of the Fourth Ann. Symp. on Logic in Computer Science, June 1989, Asilomar, California, U.S.A.

[LB88] B.Lampson and R.Burstall, *'Pebble, a Kernel Language for Modules and Abstract Data Types',* Information and Computation, 2/3, vol. 76.

[LMR86] P. Lindsay, R. Moore and B. Ritchie, *Review of Existing Theorem Provers,* IPSE 2.5 report 060/00047/1.3, 1986.

[LPT89] Z. Luo, R. Pollack and P. Taylor. *How to Use LEGO: a preliminary user's manual.* LFCS Technical Notes LFCS-TN-27, Dept. of Computer Science, Edinburgh Univ., 1989.

[Luo89] Zhaohui Luo, *'***ECC***, an Extended Calculus of Constructions',* Proc. of the Fourth Ann. Symp. on Logic in Computer Science, June 1989, Asilomar, California, U.S.A.

[Luo90] Zhaohui Luo, *An Extended Calculus of Constructions,* PhD thesis, University of Edinburgh.

[Luo91] Zhaohui Luo, *'A Higher-order Calculus and Theory Abstraction',* Information and Computation, 90(1), 1991.

[MG81] J. Meseguer and J. Goguen, *'Initiality, Induction and Computability'* in Algebraic Methods in Semantics, (eds.) M. Nivat and J. Reynolds.

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

[Rit87] B. Ritchie, *The Design and Implementation of an Interactive Proof Editor,* Ph.D. Thesis, Dept. of Computer Science, Edinburgh University.

[SB83] D.Sannella and R.Burstall, *'Structured Theories in LCF',* 8th Colloquium on Trees in Algebra and Programming.