A Meta–theory for Structured Presentations in the COC

Sherri Shulman
B.S., Shimer College, 1973
M.S. Illinois Institute of Technology, 1979

A dissertation submitted to the faculty of the
Oregon Graduate Institute of Science and Technology
in partial fulfillment of the
requirements for the degree
Doctor of Philosophy
in
Computer Science and Engineering

February 1997

The dissertation "A Meta–theory for Structured Presentations in the COC" by Sherri Shulman has been examined and approved by the following Examination Committee:

James Hook
Associate Professor
Thesis Research Adviser

David Maier
Professor

Francoise Bellegarde
Research Associate Professor

Thaddeus Kowalski
Director, AT&T Intelligent Voice

Richard Kieburtz
Professor

Acknowledgements

I am very grateful to my department for their patience in allowing me to complete this thesis. Jim Hook waded through several versions and helped me to refine my thoughts and presentation significantly. Ted Kowalski not only read and commented on my thesis, but made a cross country round trip in one day to attend my defense. Francoise Bellegarde read my thesis, commented, and encouraged me to persevere. Dave Maier provided extensive remarks on presentation and organization at some cost (time and mental energy) to himself. I was and am appreciative of all the comments made and have learned a great deal during my time at OGI.

There are others that I should acknowledge but this space is inadequate to represent their contributions.

Contents

List of Figures

Abstract

A Meta–theory for Structured Presentations in the COC

Sherri Shulman
Oregon Graduate Institute of Science and Technology, 1996

Supervising Professor: James Hook

This thesis presents the specification and design for a meta–theory of constructive presentations in the Calculus of Constructions. It describes a system that supports constructing and sharing theories and theory fragments (called theory presentations) in a structured way, the capability to discharge dependencies without rechecking type status unless the type status has been destroyed, the capability to encode a consistent set of presentations, the ability to discharge a presentation reference with consistent extensions of a presentation, and to translate a given structure into a target object language of the presentation.

This thesis differs from other work in the area by focusing on the structure and relationship of theory presentations. The structure of a theory presentation is the genealogy: how the theory presentation was constructed from other theory presentations. This structure is isolated from the content by limiting manipulable theory presentations to those that have been "named". The set of names theory presentations then becomes a persistent base of artifacts available for construction.

Chapter 1

Introduction: Structural Organization of Constructive Logics through Theory Presentations

A mathematical discourse presents the organization of mathematical theories and objects. The most common forms of discourse include textbooks, journal articles, and lectures. People organize what they consider to be the relevant knowledge for various purposes: for collecting related information, for teaching a particular subject, for collecting and referring to connected or related arguments, for presenting a new result in a required context or setting. Arguments in one discourse might refer to arguments or results in another by referring to the name of the argument or result as it is known in its defining discourse.

This scaffolding that surrounds the presentation of a theory or collection of theories is usually not explicit. We usually have no need to refer to the organization of a book into its chapters and sections. The purpose of the text is to convey the contents in whatever form may be available. The text is then read by an individual, such as a logician or student. The methods used by the reader to understand or analyze an argument may be unrelated to the methods used to structure the presentation of the argument.

In the domain of automatic construction and analysis of arguments, we generally focus on the tools that people use to understand an argument rather than the structure of the books that people access for their knowledge. In this thesis we focus on this structure: How can this structure be used effectively for both construction and analysis? What are the advantages of promoting structure to a first class object for both construction and

analysis?

As logicians study language, they introduce terms that cover the concepts of the language. In this thesis, we introduce concepts that are not covered by existing terminology. In changing the domain of discussion, we modify existing terms to reflect a new usage, because our goal has changed. We specialize and adapt the classical definitions to the context of the logical framework. These new meanings and usage are defined in this chapter.

There are three aspects to this organization of knowledge: the description of some concept or argument or theory, the prior developments on which this description relies, and the description of the structure of the prior developments (usually implicit by its presence in the book). Theories may be described in many different languages, using different presentation styles and discourses. A logical framework is a formal system that may be used to codify such discourses of mathematical arguments. We use the term logical framework in the same sense as is used by Harper, Honsell, and Plotkin [23]: a language for defining formal systems. It must be sufficiently powerful to encode the statements in the prior development and the current theory. The Calculus of Constructions [15] is such a language, as is LF [23]. We present another approach to constructing a logical framework in which the structure is defined and manipulated separately from the statements of the theory.

An example of a mathematical discourse is the propositional logic defined using truth and falsity. This presentation would depend on the prior development of a Boolean theory giving meaning to *TRUE* and *FALSE*. The definitions of the new connectives of propositional logic are accomplished by defining what the resultant true or false value will be for all combinations of connectives. The primitive objects of the propositional logic are the propositional symbols. The constructors for producing new propositions are the logical connectives $(\vee, \wedge, \rightarrow, \neg)$, and the properties of these connectives are defined in terms of the resulting truth values from the Boolean theory.

The above description of the propositional logic uses an implicit abstraction of a discourse: a theory may be described by providing the primitives of the theory, the connectors of the theory, and the properties of the connectors. This abstraction defines what information needs to be provided, but not how it will be provided. The differences in discourses

may be seen in differences in how the information is described.

Alternatively, classical propositional logic may also be defined using natural deduction methods: the set of proposition and property symbols remains the same, as does the set of logical connectives. However, the discourse of natural deduction defines under what conditions a formula may be introduced or eliminated, using axiom schema. The primitives of natural deduction as a discourse are the axiom schema, which are composed of premises and conclusions. Defining propositional logic in this discourse is done by giving the meaning of each logical connective formula by showing what conclusions can be drawn. Rather than discuss if a formula of propositions and connectives is true or false, the schema rules encode when a formula is derivable.

These two presentations encode the same mathematical system in different discourses: they differ in how the meaning of the system is presented. The discourse may be said to define the meta-theory for the presentation. The propositional logic defined using truth and falsehood can make statements about truth. The propositional logic defined using axiom schema and natural deduction makes statements about derivability and derivations.

For any given discourse, one may introduce new theories. While mathematically a theory designates a deductively closed set of sentences, in this thesis a theory refers to a collection of statements true or provable in a context. When a new theory has been defined the discourse may be thought of as extended by that theory. So the discourse of natural deduction extended by the theory of propositional logic*Proplogic* can make use of the methods of natural deduction as well as the statements of *Proplogic*. Other theories may subsequently be defined in this discourse either using the theory *Proplogic* (i.e. using the primitives of *Proplogic*) or independent of *Proplogic*. However in the discourses described above, the concept of a theory is external. There is no way to designate that a theory being elaborated requires the theory *Proplogic*. Our interest here is the discourse of theories: how to specify what theories are required, what theories may be used, and how these theories may be related to one another.

The goal of this research is the organization of bodies of knowledge that are internally structured as collections of theories. We will call the language used to organize these theories *MF* (*Meta-Framework*). In the current definition, the theories themselves are

defined in the Calculus of Constructions. The use of the Calculus of Constructions means that the discourse used inside a theory is the Calculus of Constructions extended by any theories required for further development. The theories required in constructing or using a new theory are specified as the required environment. Manipulation and specification of the environment is done with *MF*.

The new primitive objects introduced in *MF* are:

- The theory as a linguistic unit. The semantics of a theory are similar to that of encapsulation. It not only identifies a linguistic part of the language but also identifies a theory as a collection of features.

- The environment of a theory, where the environment is the (structured) collection of theories required.

Theories are constructed in relation to other theories, starting with the empty or null theory. New theories and their relationships are constructed and maintained by theory combiners. A theory combiner is a function that maps theories and environments to new theories and environments. Theories may be derived from others through extension, unions, specializations, and generalizations. A theory constructed in this way has a relationship to the theory or theories it was based on.

The theory relationships supported are extension and specialization. A theory that extends another is consistent with that theory. In this case, consistency is used to denote that one theory entails the other.

As an example, in constructing some theory A with a declared environment of the theory *Proplogic*, one may assume the primitives of *Proplogic*, without knowing entirely what is included in *Proplogic*, and one may not refer to the constituents of *Proplogic* without having the entire theory present. If two theories B and A are combined and their environments are not independent, the resulting environment of the combined theories must be determined. If they both use *Proplogic*, or a theory consistent with *Proplogic*, the required environment would be some theory C which satisfies the environment requirements of both B and A: a theory consistent with *Proplogic*.

In the following, A and B refer to arbitrary theories, constructed using arbitrary theory

combiners. A and B have environments E^A and E^B respectively. E^A, A is the theory A with its environment E^A. $TH(E^A, A)$ is a function that maps a set of sentences to its deductive closure. As a shorthand, $TH(A)$ denotes $TH(E^A, A)$. The environment closure of a theory is the list of all the theories that a theory requires. A^* or $(E^A)^*$ denotes the environment closure of the theory A or its environment E^A.

The goals of such a system encompass constructing new theories in a manner that preserves the theory structure and allows the following judgements to be made:

- Theory A is consistent with B, capturing the notion that A may be used wherever B is needed, $(TH(B) \subseteq TH(A))$
- Theory B requires theory A, or more generally, theory B requires a theory consistent with A. $(A \in E^B$ or $A \in (E^B)^*)$.

These judgements are then used to answer questions about theories, such as:

- What is the minimal environment of a theory, A? (The least E', s.t. $E' \subseteq E$ and $TH(E', A) = TH(E, A)$.)
- Can theory A be used in place of B? $(TH(B) \subseteq TH(A)$.)
- If two or more theories are combined, does the combination of their environments have a unique and minimal solution? (Is there a minimal E s.t. $TH(E, A \cdot B) = Th((E^A, E^B), A \cdot B)$?, where $\cdot$ is a theory combiner.)

The Calculus of Constructions, like the Automath system, provides a logic into which simple mathematical discourse may be embedded. In this thesis we expand this discourse into richer notions of organization and theory combinations. The soundness of the resulting system relies on the expansion into the base discourse: the theory organization is invisible to the Calculus of Constructions. This organization serves to mediate between how people organize knowledge and the requirements of the underlying formal system.

1.1 The Calculus of Constructions as a language of discourse

The *MF* language describes relationships among theories, relying on a base language that is used to describe the theories themselves. The base language provides a default discourse in which to describe theories. There are many potential candidates for such a base language.

Because the goal of this system is to facilitate the description of theories, we would like the base language to be as expressive as possible. Some of the general attributes that are desirable in the base language are:

- The logic must be capable of describing what statements and derivations are legal in a theory.
- The logic should be capable of expressing the specification of a theory or problem.
- The logic should be capable of expressing proofs of correctness in the logic.
- The logic should be capable of stating and proving properties about the theory.

The key characteristic to a language's ability to generally describe theories using arbitrary discursive methods is the first item. Barendregt in *Introduction to Generalised Type Systems* [4] characterizes type systems by the dependencies of the terms and the types in the systems. In his model, languages that allow types to be dependent on terms are capable of describing what is legal or provable in a theory. Languages with this facility are often called logical frameworks because they are able to describe other logics or theories.

The Calculus of Constructions has all the above attributes. It is capable of defining theories whose role is to act as a discourse for the further development of other theories. And, it is sufficiently expressive to define the discourse of theories within the logic itself. This thesis proposes that the discourse of theories is more productively supported outside the logical framework or mathematical discourse. We will discuss this issue more fully later in this chapter.

The Calculus of Constructions has the desired expressiveness as well as some particular characteristics: the explicit disjunction between assumptions and the values used to

discharge the assumptions, and the pragmatic consideration that an implementation exists for the language. The Calculus of Constructions has an internal notion of context, encoding what assumptions are made; higher-order types, allowing complex propositions and properties to be expressed in the logic; a mechanism for associating constructive proofs with types, enabling types to have executable members; and the ability to discharge context elements with objects of the appropriate type, known as the cut rule, allowing types and proofs to be used in other types and proofs. The existence of an implementation allows further explorations to be conducted in a lab-like environment. Other logics could be used as a base language supporting a discourse of theory management.

A weaker logic would result not only in a weaker discourse for building new theories, but would also lack the scaffolding for building the theory structure. Without a notion of context internal to the logic, a context would be part of the theory structure itself. This lack of context in turn would necessitate support for discharge of assumptions in the theory structure rather than in the logic. A less expressive language of types would limit what theories could be built. Perhaps only one discourse would be definable, altering the fit between a discourse and a target theory. A different notion of proof would alter what properties the resultant theory would have: constructive, executable, intensional, extensional, etc. A lack of the ability to define what is legal in the logic itself would require a discourse to be described by augmenting the language with externally defined constants, as in Automath.

While the Calculus of Constructions has many advantages, it has weaknesses as well when considered as a vehicle for organizing knowledge. These weaknesses fall into two areas: those that arise when the calculus is considered as a logical framework for developing theories and those that arise when considering the calculus as a vehicle for implementing a discourse of theories. Looking at the calculus as a logical framework, its discourse forces a style that is particularly cumbersome for people to use and manage. Its strong requirements for proof, while providing very nice formal properties such as termination and consistency, also necessitates a great deal of detail to express, discharge, and formalize even quite small theories. The cut rule, while providing a powerful and concise mechanism to connect assumptions with objects that discharge them, is not capable of maintaining

that connection. Cut, combined with contexts, provides a flexible way to abstract out assumptions and then discharge them independently. But in theory management, the connection between the assumptions and their dischargees needs to be maintained as well. Lastly, there is no internal notion of completeness criteria, either for assumptions or for theories. Because a theory may be only partially described, we use completeness to refer to how much of the theory is present or elaborated. As it stands now, a theory defined in the Calculus of Constructions may be defined with a context that identifies what particular constructions are required by the theory. Those constants are presumed to be defined elsewhere, in their own theories, that assure their correctness. However, there is no requirement that the entire theory be present when the assumed constant is discharged. By allowing contexts to be expressed with minimal assumptions the Calculus of Constructions supports the definition of new types and theories, even if those assumptions themselves require the existence of other constructions or constants.

Looking at the calculus as a vehicle for implementing a discourse of theories, there are comparable weaknesses. As an impredicative system (a system expecting the definition of assumptions in the system itself) the calculus does not have the primitives associated with theory construction, management, use, and reuse. These primitives would have to be developed within the discourse of the Calculus of Constructions. Because of the constructive nature (both an advantage and a disadvantage, in some ways), these new primitives would require proofs within the calculus. Because the same logic would be used both for the discourse of theories as primitives and for the development of theories, the discursive nature is confusing. This conflation of discourse theory with the object theory leads to a lack of clarity in what exactly is being described: At what level are we discussing a theory? As an object to be manipulated or as a set of logical statements that constitute an argument? How are the connections maintained between contexts and objects? Between theories and their component statements? This dichotomy parallels the two activities mentioned above. The nature of theory construction, management, and retrieval pertains to the way people organize knowledge. The nature of analyzing a theory pertains to the way an argument is analyzed by logicians: the pieces need to be collected and presented in a consistent form to the logician. At that point in time, the

structure of the theories that were used in building the argument is irrelevant. The dual nature of construction vs analysis suggests that embedding a discourse of theories into the Calculus of Constructions will not provide the level of clarity that we wish. When a body of knowledge is presented in a book, we have an expectation that not only will the subject matter be correct, but that it will provide us with a usable form for learning new material and using this material at a later time. We expect the same from a formalized notion of theory structure.

Using a logical framework such as the Calculus of Constructions to describe the theories and a specialized language to describe theory relationships provides several benefits. There is no level ambiguity: statements concerning how theories relate to one another are confined to the *MF*. The language of theory relationships and queries does not have the same requirements as the language for defining theories. It need not be as expressive or as general as the language of theories. By limiting the generality, the theory structure is more evident and can be manipulated more directly.

1.2 An Example of a Discourse of Theories

Because the domain of a discourse of theories is the structure of theory dependencies, rather than the specific details within the theory, in the following theory examples the internal details will be sketches, while the structure will be fully elaborated.

Using a style based on that of the Calculus of Constructions (COC), we will develop a presentation of a theory of the propositional logic, using a natural deduction style discourse. *Prop* refers to the assertion of an arbitrary proposition. *Thm* refers to a theorem. A proposition has a name, *P_name* and a value *P_val*, with the interpretation being that *P_val* is a valid proposition and it may be referred to by *P_name*.

Prop P_name P_val

A theorem has a name *T_name*, value *T_val*, and a type *T_type*. It is interpreted as an assertion that *T_name* is associated with *T_val*, with the type *T_type*.

Thm T_name T_type T_val

Both a proposition and a theorem may be assumed:

$$P\_name : Prop$$

asserts that P_name is an arbitrary proposition.

$$T\_name : T\_type$$

asserts that T_name is a construction (a proof) of type T_type.

The constructions (values or proofs) are constructed using abstraction ($[\cdot\cdot\cdot]$) and function constructors ($\rightarrow$):

$$[a : A][B : Prop](A \rightarrow B) \rightarrow B$$

describes a construction that is abstracted on an object a of type A and a proposition B. The remainder of the construction is interpreted as a function that given a construction of the type $A \rightarrow B$ will produce a construction of the type B.

A set of propositions and theorems may be preceded by a context of assumed propositions and types.

Using this sketch of the Calculus of Constructions, the theory of propositional logic might be described as follows:

$$\begin{array}{lll} Context & \equiv & [A : Prop \\ & & B : Prop] \end{array}$$

$$\begin{array}{llll} Prop & And & & [C : Prop](A \rightarrow B \rightarrow C) \rightarrow C \\ Prop & Or & & [C : Prop](A \rightarrow C) \rightarrow (B \rightarrow C) \rightarrow C \\ Prop & Impl & & [C : Prop](A \rightarrow B) \rightarrow (B \rightarrow C) \rightarrow C \\ Context & \equiv & [a : A & \\ & & b : B] & \end{array}$$

$$\begin{array}{llll} Thm & \wedge & And < A, B > & [C : Prop][h : A \rightarrow B \rightarrow C](h\ a\ b) \\ Thm & \vee_l & Or < A, B > & [C : Prop][h1 : A \rightarrow C][h2 : B \rightarrow C](h1\ a) \\ Thm & \vee_r & Or < A, B > & [C : Prop][h1 : A \rightarrow C][h2 : B \rightarrow C](h2\ b) \\ Thm & \Rightarrow & Impl < A, B > & [C : Prop][h1 : A \rightarrow B][h2 : B \rightarrow C](h2\ (h1\ a)) \end{array}$$

We will call this theory *Proplogic*. This theory might be specialized to Booleans by specifying that the propositions be Booleans rather than arbitrary propositions. A less general propositional logic theory might be defined using the discourse of $True$ and $False$ by defining suitable Boolean constants, defining what constitutes legal terms in the propositional logic, and defining the semantics of each term by giving their Boolean values.

Similarly we might define a theory of natural numbers as:

$Prop$	Nat		$[A : Prop](A \to A) \to A \to A$
$Theorem$	$zero$	Nat	$[A : Prop][f : A \to A][x : A]x$
$Theorem$	$succ$	$Nat \to Nat$	$[n : Nat][A : Prop][f : A \to A][x : A](f\ (n\ A\ f\ x))$

We will call this theory *Naturalnum*.

And for a theory of booleans:

$Prop$	$Bool$		$[A : Prop]A \to A \to A$
Thm	$True$	$Bool$	$[A : Prop][v1 : A][v2 : A]v1$
Thm	$False$	$Bool$	$[A : Prop][v1 : A][v2 : A]v2$

which we will call *Booleans*.

Now, in this theory environment, we have access to the following discourses in which to write new theories: the unextended calculus, called *COC*, the calculus extended by *Proplogic* (*COC + Proplogic*), the calculus extended by *Naturalnum* (*COC + Naturalnum*), *COC + Booleans*, *COC + Proplogic + Naturalnum*, and so on. (The + is used to indicate extension.)

Suppose that we wish to develop a theory of finite state machines, calling it *FSM*. What theory or discourse do we expect prior to this new theory development?

To express the states in the machine we can use the theory of nats, *Naturalnum*. To express the functionality of each state we could use *Proplogic* with an interpretation of booleans for the logical connectives. To specialize the theory *Proplogic* to booleans, we want the value expressed to be in terms of *Bool* rather than an arbitrary $C : Prop$. Let this specialized theory be *Proplogic* · *Booleans*. Then the new theory *FSM* will have an environment of *Proplogic* · *Booleans*, *Naturalnum*. We will not give the details of *FSM* here since they would quickly grow too cumbersome for this introduction.

The meaning of *Proplogic* · *Booleans* in an environment is intuitively that *Proplogic* is specialized to *Booleans*, which is sufficient detail for this discussion. The specialization of *Proplogic* to *Booleans* requires that the specialized theory has the theory *Booleans* as well as the theory *Proplogic* in its environment. The current theory structure of this system can be represented as a set of relationships among the theories, where $\leq$ shows an inclusion relation (if $A < B$ then all the statements of A are in B), and $\prec$ shows a specialization (if $A \prec B$ then the statements of B are statements of A specialized by discharging assumptions with witnesses) and S_0 represents the theory that is included by all subsequent presentations, in this case S_0 is the empty theory.

$$
\begin{array}{lcl}
Proplogic & \geq & S_0 \\
Naturalnum & \geq & S_0 \\
Booleans & \geq & S_0 \\
Proplogic \cdot Booleans & \geq & Booleans \\
Proplogic \cdot Booleans & \prec & Proplogic \\
FSM & \geq & Proplogic \cdot Booleans \\
FSM & \geq & Naturalnum
\end{array}
$$

Now, we can further refine *Proplogic* to include properties that we wish to express or prove, such as statements of soundness, completeness, and consistency. [1] Perhaps we will add the Peano axioms to *Naturalnum* as well. Call these new theories $Proplogic^+$ and $Naturalnum^+$. We may need a theory of equality to augment these theories in this way: call it *Equality.* The new theory structure then satisfies the following relationships, as well as those above:

$$
\begin{array}{lcl}
Proplogic^+ & \geq & Equality \\
Proplogic^+ & \geq & Proplogic \\
Naturalnum^+ & \geq & Equality \\
Naturalnum^+ & \geq & Naturalnum
\end{array}
$$

[1] While the properties of soundness, completeness, and consistency are part of the COC, a more specialized statement may be constructed that refers specifically to the statements in the propositional logic. We could augment the theory with the set *WFF* for the set of well-formed formulas. The theory may be expanded to specify what formulas will be in this set and further what properties this set will have. The statements that are made about the propositional logic are not restricted to the propositional logic formulas themselves.

In this example, there are already environmental interactions that we would like to have access to, for purposes of simplification, conciseness, explanatory reasons, or completeness.

In the interests of completeness, we may desire the most fully elaborated description of a theory. After a theory has been expanded, as in the above example, that expanded theory can be used to discharge any references to the unexpanded theory. So *Proplogic*$^+$ may be used wherever *Proplogic* was expected, since it is an extension of *Proplogic*: it adds information, but takes nothing away. In a subsequent reference to *FSM*, *Proplogic*$^+$ might therefore be used to discharge the environmental dependency on *Proplogic*.

Another question that we would like to answer is what is the minimal environment (the smallest extension of the environment) compatible with the environment specifications. If an extension to *FSM* is considered that requires *Equality* and $Naturalnum^+$, the environment $Naturalnum^+$ is a solution because $Naturalnum^+$ already requires *Equality*.

1.3 Another Example

Another view of the role of theory presentations may be seen in the description of a theory of monoids.

In this description, the theory of monoids is elaborated in two aspects: first a notion of an algebraic system is proposed, then the monoid is defined by successively adding to theories for groupoids and semigroups.

Arity is a variant of the natural numbers, over types rather than propositions. The notation $\{\cdots\}$, from the Lego implementation [32] of the Calculus of Constructions, denotes a product (Π) type. The form:

$$Prop\langle name\rangle[\langle context - elements\rangle]\langle propositionexpression\rangle$$

denotes that $\langle name\rangle$ when applied to elements of the appropriate type, as specified in $\langle context - elements\rangle$ will be a *Prop* with value $\langle propositionexpression\rangle$. The form:

$$Defn\langle name\rangle[\langle context - elements\rangle] = \langle construction\rangle$$

denotes that $\langle name\rangle$, when applied to elements of the appropriate type as specified in the $\langle context - elements\rangle$, will produce a construction as shown in $\langle construction\rangle$. The type

of ⟨*name*⟩ is the type of ⟨*construction*⟩.

ThArity

Prop	*Arity*		$\{A : Type\}(A \to A) \to A \to A$
Thm	*Zero*	*Arity*	$[A : Type][s : A \to A][z : A]z$
Thm	*Succ*	$Arity \to Arity$	$[n : Arity][A : Type][s : A \to A][z : A](s\ (n\ A\ s\ z))$

An algebraic system is an abstraction defining what information will be defined: a set of carriers, A, and a set ω of functions over A. Because this abstraction is being defined independently, it uses its own notion of application rather than the cut rule, and it defines a type to collect all of the functions over A, regardless of arity. List is first defined to collect the arguments for this new application.

ThList

Prop	*list*	$[A : Prop]$	$\{A : Prop\}\{B : Prop\}$
	$(A \to B \to B) \to B \to B$		
Thm	*Nil*	*list A*	$[B : Prop][Cons : A \to B \to B][nil : B]nil$
Thm	*Cons*	$A \to list\ A \to list\ A$	$[a : A][l : list\ A]$
	$[B : Prop][cons : A \to B \to B][nil : B](cons\ a(l\ B\ cons\ nil))$		

ThAlgsys

$Prop \quad arityZero \quad [A : Prop]$

$ZeroProp([A : Prop]A)A$

$Prop \quad arityN \quad [A : Prop][n : Arity]$

$(n\ Prop\ ([B : Prop](arityZero\ A) \rightarrow B)(arityZeroA))$

$Prop \quad Omega$

$\{n : Arity\}arityN\ A\ n$

$Prop \quad Ap \quad [n : Arity]$

$\{f : arityN\ A\ n\}\{l : list\ A\}(arityNAn)$

$Thm \quad map0 \quad (Ap\ Zero)$

$[f : arityNA\ Zero][l : list\ A]f$

$Thm \quad map1 \quad (Ap\ (Succ\ Zero))$

$[f : arityNA\ (Succ\ Zero)][l : list\ A](l(arityNA\ (Succ\ Zero))$

$([a : A][B : arityNA\ (Succ\ Zero)]([b : A](B\ a)))f)$

$Thm \quad map2 \quad (Ap\ (Succ\ (Succ\ Zero)))$

$[f : arityNA\ (Succ\ (Succ\ Zero))][l : list\ A](l(arityNA\ (Succ\ (Succ\ Zero)))$

$([a : A][B : arityNA\ (Succ\ (Succ\ Zero))]([b : A](B\ a)))f)$

$Defn \quad Map0 \quad [f : arityNA\ Zero][l : listA\ ] \quad = \quad (map0\ f\ l)$

$Defn \quad Map1 \quad [f : arityNA\ (Succ\ Zero)][l : list\ A] \quad =$

$(map1\ f\ l)(dummy\ A\ a)$

$Defn \quad Map2 \quad [f : arityNA\ (Succ\ (Succ\ Zero))][l : list\ A] \quad =$

$(map2\ f\ l)(dummy\ Aa\ )(dummy\ A\ a)$

The *map* constructions use the arguments list l as an iterator to apply the function f. The representation type for the list is the type of f. This representation type is used to collect the cumulative results of the application. Because of uniformity constraints in the Calculus of Constructions, the result of *map* is a function of the same arity as that of the original function. In general, for $f : arityN\ A\ n$, we have the definition that $mapn\ f\ l : arityN\ A\ n$. The *Map* definitions then apply this function to dummy

arguments to retrieve the result.[2]

Equality and pair theories are used to express the postulates of the monoid theory:

ThEquality

Prop	*Eq*	$[A : Prop][x, y : A]$	$\{P : A \to Prop\}(Px) \to (Py)$
Thm	*refl*	$\{A : Prop\}\{x : A\}(Eq\ A\ x\ x)$	
		$[A : Prop][x : A][P : A \to Prop][prf : (Px)]prf$	

ThPair

Prop	*And*	$[A, B : Prop]$	$\{C : Prop\}(A \to B \to C) \to C$
Thm	*pair*	$\{A, B : Prop\}A \to B \to (And\ A\ B)$	
		$[A, B : Prop][a : A][b : B][C : Prop][p : A \to B \to C](p\ a\ b)$	

To define a groupoid we need only define the binary function *cdot*, using an environment of ThArity, ThList, and ThAlgsys:

ThGroupoid	$\langle ThArity, ThList, ThAlgsys\rangle$	
Prop	*cdot*	$arityN\ A\ (Succ(Succ\ Zero)))$

A semigroup adds the associative law to the groupoid. It therefore needs an equality theory as well:

ThSemigroup		$\langle ThGroupoid, ThEquality\rangle$	
Prop	*Assoc*	$[cdot : arityNAn]$	$\{x, y, z : A\}(Eq\ A\ (Map2\ cdot$
		$(Cons\ (Map2\ cdot(Cons\ x\ (Cons\ y\ Nil)))(Cons\ z\ Nil)))$	
		$(Map2\ cdot(Cons\ x\ (Cons\ (Map2\ cdot(Cons\ y\ (Cons\ z\ Nil)))Nil))))$	

The monoid is defined by adding the nullary function *Identity* to ω and the identity law

[2] Also, to be correct, map should map over the list's reversal.

describing its behavior to the postulate:

ThMonoid $\langle ThSemigroup, ThPair\rangle$

Prop *Ident* *arityNA Zero*

Prop *LeftIdentityLaw*

$\{x : A\}(Eq\ A(Map2\ cdot(Cons(Map0\ Ident\ Nil)(Cons\ x\ Nil)))x)$

Prop *RightIdentityLaw*

$\{x : A\}(Eq\ A(Map2\ cdot(Cons\ x(Cons(Map0\ Ident\ Nil)Nil)))x)$

Prop *Identity* $(And\ LeftIdentityLaw\ RightIdentityLaw)$

Some of the details are left out because they involve proofs on specific values of functions and would be provided for specific semigroups or monoids. For instance, *cdot* must be associative but we can't demonstrate associativity until *cdot* is instantiated with a particular binary function. At that time, we can provide proofs that a particular algebraic system is a groupoid, semigroup, or monoid by providing the constructions for the following propositions:

(1) *Prop* *Groupoid* $[A : Prop][cdot : arityNA(Succ(SuccZero))]$
trivially true: the type of cdot
is the definition of a binary operation

(2) *Prop* *Semigroup* $[A : Prop][cdot : arityNA\ (Succ\ (Succ\ Zero))]$
$And((Groupoid\ A\ cdot), (Assoc\ A\ cdot))$

(3) *Prop* *Monoid* $[A : Prop][cdot : arityN\ A\ (Succ\ (Succ\ Zero))]$
$[ident : arityNAZero]And((Semigroup\ A\ cdot), (Identity\ ident))$

The resulting structure would look like:

$$
\begin{array}{lcl}
ThMonoid & \geq & ThPair \\
ThMonoid & \geq & ThSemigroup \\
ThSemigroup & \geq & ThEquality \\
ThSemigroup & \geq & ThGroupoid \\
ThSemigroup & \geq & ThAlgsys \\
ThGroupoid & \geq & ThAlgsys \\
ThAlgsys & \geq & ThArity \\
ThAlgsys & \geq & ThList
\end{array}
$$

This example points out some inadequacies still remaining in using this theory discourse. As mentioned earlier, we often use an implicit abstraction that guides us in describing theories. In this last example we first thought of the abstraction of an algebraic system that would later be instantiated with the necessary components for each theory of interest (monoid, for example). The algebraic system is defined to be a set Ω and a carrier set. Ω could have been defined as a set: $(arityNA\ n) \rightarrow Prop$, using a theory of sets. Then, functions could have been inserted in Ω. This detail was not included: it adds significantly to the amount of detail that needs to be carried along. Moreover, it is hard to distinguish the relationship of a theory of P that uses algebraic system theory to structure its information from the relationship of P to any theories that P may extend or specialize. Additionally, some of the theories that we use (such as equality) are not used in an essential way: we would not classify a semigroup as an extension of equality, although we use equality to specify the postulates in a semigroup. This observation points out the different roles a theory may play: abstraction and use. These roles are not adequately supported as yet. We will discuss these issues in Chapter 6.

Both the examples above may be described in the native Calculus of Constructions without using the theory discourse. A common method of doing this kind of development is to use the context as the means of both describing the necessary pieces and of later discharging those pieces with actual constructions. The description of the monoid might

then look like:

$$
\begin{aligned}
Context \equiv\ & \\
& [A\,B : Prop]; \\
& [Arity : Prop]; \\
& [Succ : Arity- > Arity]; \\
& [Zero : Arity]; \\
& [List : Prop- > Prop]; \\
& [Cons : A- > (ListA)- > (ListA)]; \\
& [Nil : (ListA)]; \\
& [arityZero : Prop- > Prop]; \\
& [arityN : Prop- > Arity- > Prop]; \\
& [Ap : Arity- > Prop]; \\
& [n : Arity]; \\
& [map : (arityNA\ n)- > (ListA)- > (Ap\ n)]; \\
& [And : Prop- > Prop- > Prop]; \\
& [pair : AndAB]; \\
& [Assoc : (arityNAn)- > Prop]; \\
\\
\cdots\ & \text{Propositions and theorems for the monoid} \\
& \text{such as those for the identity function}
\end{aligned}
$$

This context captures the constants and their types needed by the monoid. However the identification of where these constants are defined, what theory produced them, and what other statements must hold true has been lost. Two constants with the same types, but belonging to different theories with different behaviors might be confused. It would be impossible to recognize redundancy, as in multiple uses of *Cons*, for instance, in different theories.

1.4 Thesis Organization

The goal of this research is to provide a framework for managing the elaboration of theories. We propose organizing the development, use, and re-use of constructions by organizing them into theory presentations. A theory in the Calculus of Constructions is a set of types and constructions and all logical propositions that describe the required elements and their behavior. A theory may not be finitely representable and it may be only partially defined at any point. We call the finite or partial representation of a theory a theory presentation. A theory presentation comprises the assumptions or proofs that it depends on and the propositions that directly characterize the theory as elaborated so far. There are many similarities with this approach and *CLEAR* [11]. We will discuss the relation of theory presentations to *CLEAR* in Chapter 5.

Using theory presentations as the basic object of analysis, we focus on how the theory presentations are constructed from more basic theory presentations, where a more basic theory presentation may be a required discourse or a partial description of the theory under construction. How the theory presentation is constructed can be viewed as its genealogy. Using this genealogical structure, it is possible to determine when a dependency of a theory presentation on another theory presentation is discharged and if a theory presentation is valid in another context of theory dependencies. Ultimately, in a sufficiently rich base of theory presentations, it may be possible to construct complex theory presentations that are correct by construction using the genealogical structure, the theory presentation relationships, and, only secondarily (through the underlying logical framework), the type information.

The remainder of this thesis is organized as follows: Chapter 2 gives the background of relevant developments in type theory development, proof theory, and logic, specification languages and knowledge organization languages. Chapter 3 describes some examples of theory presentations, environments, and constructions. Chapter 4 describes the formal system of the discourse of theories. Chapter 5 is a comparison of this system with others of similar or overlapping goals. Chapter 6 concludes with suggestions for future work.

Chapter 2

History

This thesis draws upon several areas. In part, the ideas stem from the recent interest in higher-order constructive logics in the context of computer science. In part, the suggestions in this thesis have grown out of interests in methodological, taxonomical, and specification activities when applied to organizing complex bodies of knowledge. Lastly, in reviewing the philosophical history of the field associated with language, type, and proof development, I conceived the idea that perhaps the key to controlling the complexity was to introduce a new entity altogether: structure as a first class object rather than as an artifact of some other organization.

To display these different threads, I will focus on four relevant research areas: the database and AI worlds for taxonomic properties; specification languages; the role of higher-order logics in theory construction, analysis, and use; and philosophic speculations on what a logician is analyzing: the structure versus the content of an argument. Our goal is to support the view that the structure of an argument itself is an object of logical analysis.

2.1 Taxonomic Properties

The organization of large bodies of knowledge, with complex relationships and interdependencies among its components, has been the target of both artificial intelligence and database research. The particular results that contributed to the ideas in this thesis are the work of Aït-Kaci [1] and its formalization in order-sorted equational logic by Smolka and Aït-Kaci [47].

Aït-Kaci introduced the notion of inheritance hierarchies built up from feature types. Feature types are ordered by subtyping and the elements of a feature type are records. The features of a feature type prescribe the fields of its record elements. Unification over feature type terms determines if two feature type terms have an intersection and constructs a feature term that denotes this intersection. Feature terms denote sets and subsets of sorts rather than elements of sorts, acting as descriptions of what the elements must conform to. As more information is accumulated as to what the elements are, or how to discriminate one set of elements from another, that information is added to the feature type as an additional feature or as a refinement of an existing feature.

While there are differences in the kinds of taxonomic data covered by Aït-Kaci and later by Smolka and Aït-Kaci, there are relevant similarities as well. The original terms introduced by Aït-Kaci ($\psi-$terms) were conceived as organizing and defining inheritance hierarchies that represented the subset structure of information. The hierarchy could then be incorporated into a logic query language such as LOGIN [1]. The information in these kinds of hierarchies is typically set descriptions, such as the description of a set of students by describing the characteristics of what is a student; the description of the set of part-time students similarly (a subset of students); the description of the set of students with a particular major, etc.

Their later development incorporated the $\psi-$term approach with order-sorted equational logic. Order-sorted equational logic can be used to specify inheritance hierarchies as well, but the inheritance hierarchy is a constructor hierarchy of elements, rather than set descriptions. The hierarchy is organized around how a member of the set is constructed, yielding an initial algebra approach to specifications. However, the elements being described are constructor elements, called constructor types. These constructor types can be used to define data types by describing the required constructors. The information in these kinds of hierarchies is typically data-type descriptions, such as what constructors are necessary for the natural numbers, for the negative numbers, and ultimately the full set of negative and positive numbers (integers).

The resulting formalization allows both feature types and constructor types to coexist in one inheritance hierarchy. The characteristics of the information in this hierarchy share

some important attributes with a hierarchy of theory presentations. A theory presentation hierarchy needs to capture several aspects:

- What components a theory requires. Like the constructor theory above, where the *integers* require *nat* and *negint*, theory presentations are generally constructed in a context of other theory presentations.

- Theory presentation developments represent a genealogy of sorts. The theory presentation is a description of what is known about the theory up to this point in its elaboration. At each iteration of development, the description of what the theory must look like is further refined, thus limiting which actual theories will satisfy the requirements.

The hierarchy of theory presentations is a modification of the inheritance hierarchy in order-sorted equational logic. Part of the information organized in this hierarchy is the structural information: what theory presentations are required to provide a valid context for this theory presentation development. This structure corresponds to the constructor theory aspect. Part of the information required for a theory presentation is what features belong to the theory. Using the terminology of features and feature terms, we can look at the examples sketched in Chapter 1. The theory of booleans presented there could be said to have three features: a notion of what a *Bool* is, what *True* is, and what *False* is. The theory presentation asserts that a theory of booleans is a theory that includes elements for these three features. If this theory is extended in $Booleans^+$, we can encode the extension by asserting the relationship ($Booleans^+ \geq Booleans$), which in the context of inheritance hierarchies asserts that $Booleans^+$ inherits all the features of *Booleans*. Then we can add the new features, such as boolean operations, $\vee$ and $\wedge$. *Booleans* is the structural requirement for $Booleans^+$. The new features further refine the information known about the theories that conform to this description.

As described, the theory presentations have been concerned with structure and collectivity. Structure refers to where a theory presentation is in a hierarchy of theory presentations. Collectivity refers to what statements are collected in the theory presentation. These two aspects together form a description of the theory. Feature terms viewed as a

description are similar to a signature with no interpretation. In the feature term view, the theory presentation *Booleans* describes the set of all theory presentations that have the features *Bool*, *True*, and *False*. But the members of this set have not yet been defined. So, *Bool* has no interpretation yet. In the inheritance hierarchy formalization, the values are constructed by constructors, assumed to be minimal elements in the hierarchy. Using this approach, *Bool* might be interpreted as a *Prop*: i.e. the feature *Bool* is defined as an arbitrary proposition. Or if more information is known, *Bool* might be a particular construction that has the desired attributes.

Because of constraints of the underlying logic used in this thesis, sorts and feature terms have a rigid structure. The theory presentations describe a set, but the set is ordered. A theory presentation describes a set: for instance *Booleans* describes the set of all extensions of *Booleans*. But if there is more than one way to describe the proposition *Bool*, they must be differentiated in some way. Perhaps they assume different theory presentations in the context in which they are defined. Or their definitions are such that only some of the definitions for *True* or *False* will still be valid. This difference entails an entirely new theory presentation to introduce new values: *Booleans'*. *Booleans'*, while containing the same features as *Booleans*, has different values for these features. So even if we believe that both *Booleans'* and *Booleans* both describe the same theory (semantically), in the hierarchy they describe different sets. If the fact of their connection is a desired piece of information, then they need to be described in that way: i.e. that *Bool* is an arbitrary *Prop* rather than a specific proposition with a value.

So there is additional structural information required of the sets being described in a theory presentation hierarchy: we cannot connect to arbitrary descriptions and we can only conclude theory presentations are the same if they denote the same structure. Additionally, because the intent is to describe a growing, evolving system, combined with the structural meaning, some semantics have been changed. In the original formalization of Aït-Kaci, the subtype relationship is set inclusion. So, the set of *student* is constructed by constructing all the members of *part-time-student* and *full-time-student*. The sub-sort relationship is equated with subsets: *part-time-student* $\cup$ *full-time-student* $\subseteq$ *student*. For theory presentations, however, we are more concerned with manipulating the descriptions

rather than managing the collection of elements. Continuing the student example, from a theory presentation perspective, the theory *student* could come first with the theory of *part-time-student* and the theory of *full-time-student* defined as extensions to *student.* The meaning would then be the features of a part time student are distinct from those of a full time student, and a student defines those features that are common to both. The theory presentation view is that *part-time-student* $\geq$ *student* and *full-time-student* $\geq$ *student*, where the *part-time-student* and *full-time-student* descriptions are extensions of *student.* While the set of *student* contains both the elements in the set of *part-time-student* and those in *full-time-student*, the opposite is true for the descriptions: the description *student* is contained in both *full-time-student* and *part-time-student.* In the order-sorted-equational view, only minimal sorts have constructors: i.e. to construct a student, you either construct a *part-time-student* or a *full-time-student*, assuming these are minimal feature terms or constructor terms. However, in the theory presentation hierarchy, it makes sense to construct an element at any point: the theory of *student* is well-formed in the underlying logic, and has associated values. The additional information in the *part-time-student* theory is not present in the theory of students, so no conclusions based on this information will be available. This contrasts to the feature term approach, where a student must be either one or the other or there is another category which is minimal: an as yet undefined student. [1]

2.2 Specification Languages

The works in specification languages that are most pertinent here are those used to describe data-types, such as CLEAR ([22]), OBJ ([11]), KL-ONE ([6]), and successors of these languages. The role of specification languages or knowledge description languages is to encode the relationships that we want to maintain, as described in the section above. These relationships become part of the descriptive component of the formal system. This

[1] In this discussion of the feature logic, we use $\leq$ to mean extension. So in the expression $A \leq B$, B is an extension of A, meaning that additional information is added. In the feature logic defined by Aït-Kaci, the meaning of $\leq$ was reversed. In the expression $A \leq B$, A is an extension (or refinement) of B. This led to the definition of a minimal sort (in LOGIN) with respect to $\leq$. In this thesis we conform to the notion of subsort as extension, so the use of the term minimal can be misleading.

descriptive component has two aspects: one to construct or describe the formal system (the relationships) and the other to inspect the formal system.

Such a formal system describes a hierarchy. The language used to construct the hierarchy may also be used as a query to inspect the hierarchy for a solution, sometimes in conjunction with unification. It's difficult to completely separate the language from the persistent (or semi-persistent) artifacts that they describe and introduce.

The work reported by Gert Smolka [46] describes a feature logic that generalizes and unifies formalisms developed for knowledge representation, such as Aït-Kaci's $\psi-$terms and linguistic structures of Rounds, Kasper, and Dorre [17], [27]. This work specifies a set description logic and identifies which interpretations are admissible. An interpretation assigns a set of elements to each sort.

This work is more extensive than needed for theory presentations: it provides descriptive terms that would construct theory presentation terms with no meaning. The main thrust of Smolka's work is to unify different descriptive mechanisms (of which this work presents one) and to show the properties that these logics have.

MF is a logic of theory presentations that forms a subset of the descriptive features presented by Smolka. In particular, his feature term logic allows partial descriptions of feature terms in a wide variety of ways:

- By referring to a sort specifically, such as *part-time-student.*
- Referring to a feature in a feature sort: $f : s$, equivalent to referring to the set of all feature terms that have the feature f of sort s.
- Referring to two paths through the feature terms: $p \downarrow q$ is equivalent to referring to the set of all feature terms whose values for these two paths agree. An example might be the subset of *student* described by *part-time-student.major*$\downarrow$*full-time-student.major*, which would describe the subset of students with the same major as each other.
- Referring to two paths through the features where the path values disagree. Again, an example might be the subset of students that don't work might be those who are

retired, independently wealthy, or too young for a work permit.

- Referring to the intersection of two feature term descriptions, capturing the set for whom both descriptions hold true.
- Referring to the union of two feature term descriptions, capturing the set for whom one or the other description holds true.
- Referring to the complement: all feature terms that don't match the feature term description.

While this generality makes sense where the goal is to be as descriptive as possible as to the kind of knowledge represented, the goal of the research described in this thesis is to describe knowledge arranged in a particular way. The structure of theory presentations has certain requirements that are reflected in the organization of the inheritance hierarchy and thus need to be maintained in the logic that constructs and queries this hierarchy. Some of the descriptions are irrelevant in the domain of theory presentations. Because the central item of interest is how the theories are structured, and not the particular values, it is less meaningful to refer to agreement and disagreement of values. For a similar reason, it is not as meaningful in this environment to speak of the theory that is either an s or a t, unless there is some connection between the theories s and t. If that is the case, that connection should be made explicit, in another theory u. *MF* supports theory presentation extension, intersection, and union.

2.3 Higher-Order Logics and Philosophic Issues

As mentioned in the introduction, to some degree the logic chosen as the base of a system such as this is not central to this research. However, the existence of higher-order logics and their fruitfulness as languages for specifying theories, properties, and other aspects of computable artifacts has motivated much research in the area of logical frameworks. The focus of this section is to trace the relevant contributions of higher-order logics and the associated philosophic issues that contributed both to their development and use.

This overview of the developmental history of higher order logic systems covers:

1. Type theory developments, such as Russell's notion of types.
2. Philosophical origins. Brief overview of Brouwer's (and Heyting's) questions concerning classical reasoning.
3. The definition of the λ-calculus, in particular the typed λ-calculus, and its role as a computational model (Church, Rosser, Kleene).
4. Gentzen's contribution of natural deduction with its 'natural' map onto logical reasoning, allowing the idea of a proof to be treated as a syntactic object.[2]
5. The Curry-Howard isomorphism: i.e. the correspondence between logical formulas and types.
6. The correspondence between proofs and terms of functional types.
7. The role of impredicativity and higher order types.

2.3.1 Development of the notion of types

The theory of types originated with Russell [40] with the goal of resolving the contradictions of assuming a set of all sets (*predicates not predicable of themselves*). This contradiction arises in an attempt to formalize mathematics by reducing mathematics to logic. In this school, logic is prior to mathematics and provides the justification and meaning of the mathematical statements.

The theory of types gave meaning to a statement, by restricting statements to those having appropriate typings, yielding *true* or *false* values. Statements that could not be typed in this sense were meaningless.

[2]Note, that this is one area that I elaborate and extend. For me a proof is not limited to a proof of the proposition or conjecture, but includes the proof in a particular context (environment). So, while Gentzen's natural deduction specifies how a particular construction is formed, it does not specify in what way the underlying logic has been extended, or in what way the context has been defined, instantiated, or discharged.

2.3.2 Intuitionism

The intuitionistic (constructive) view of logic due to Brouwer [7] arose in a similar context: that of trying to understand the logical underpinnings of mathematics. Brouwer, however, questioned the logical constants from the standpoint of questioning when a statement can be judged *true* or what is a valid proof of a proposition [25] [20] [51]. From that stance he limits the logical constants and valid inference rules to those that embody his notion of truth as proof. The basic premise is to deny the validity of the excluded middle as a means of establishing truth (proof) since, while classically true, the proof itself would not necessarily give any information as to what exactly was being proven. So one can only conclude $A \vee B$ if there is a proof of one or the other of the propositions. Similarly for $A \vee \neg A$. And for the proof of a proposition involving the existential quantifier, one must provide the object to which the proposition refers.

Having articulated this new notion of proof or truth, it was natural to attempt to use this new logic to formalize mathematics, in the same spirit as Russell did with classical logic. This new notion of proof had the particular advantage that *when* an intuitionistic proof of a mathematical proposition exists, the proof itself yields an object that is the realization of the proposition. For Brouwer, in a sense, the mathematical object is more fundamental than the logic that describes it. So a proof of a logical statement describing this *real* object must produce this object if we are to believe it exists. The object justifies the meaning of the logical statement.

There have been substantial attempts to formalize the constructive portion (i.e., the real portion) of mathematics (Bishop [5], Heyting [25], and Kreisel [28]). While there is a certain philosophical distinction in determining which of mathematics or logic is prior (which provides the meaning for the other), to some extent it is moot. In both approaches there is an attempt to elaborate a portion of mathematics formally in a logic. For the intuitionists, the logic is philosophically justified by this more stringent notion of proof and then used to see what portion of mathematics might be said to be real. For the classical logician, the logic is justified *logically* as it were, and then shown that it is sufficient to describe existing mathematics. For both the goal is to describe mathematics whatever the

semantics ascribed to this description might be.

2.3.3 The λ–calculus

The λ–calculus [12] and combinatory logic [16] were developed as devices to study rules [2], [3]a, where the rules are represented as functions.

Looking at the functional definition as embodied in the λ–terms led to many results from the perspective of what is computable via these functional definitions – what terminates and what can be represented. Again, as in the logics above, the system of λ-terms was intended as a foundation for mathematics [2]. This goal was not entirely realized, though many more limited goals were. In the context of the λ–calculus, many properties of functions have been examined, such as the correspondence of λ–terms with functions (in particular the partial recursive functions), the normal form properties of the typed λ-terms, termination properties of these terms, computational models, etc.

2.3.4 Proof Theory

Proof theory is the study of formal arguments; in particular, it characterizes how we know an argument to be valid [38], [39]. Proof theory has been a topic for both classical and constructive logic. In particular, for constructive logic, proof theory is of interest because of the relation of a proof to a construction. Informally, constructive logic implies that one has proved a proposition if one has provided a construction demonstrating that proposition. So the relation of a proof to a construction is very close. Proof theory studies the process by which a construction is found and how we know that the construction demonstrates the proposition. So proof theory consists in part in analyzing the (constructive) logical connectors to determine their intent, elaborating (explicitly) the connectives and what constitutes a *proof* for each connective. Gentzen's formal system [19] analyzed the deductive operations of constructive logic into its component parts and transformed the study of proof theory from an intensional analysis to an analysis of extensional objects: the proof is a derivation from initial premises to the desired conclusion. Gentzen's system formalized the notion of a proof, changing it from a process to a syntactic, manipulable object, exposing regularities in the structure of proofs. In particular, syntactic proofs

were susceptible to manipulation, a normal form, and had termination properties similar to those for the λ–calculus.

2.3.5 Formulas as Types and the Curry-Howard Isomorphism

In the previous section, we discussed how proofs came to be viewed as syntactic objects. Alternatively, in Heyting's semantics [26], proofs were given a meaning as abstract constructions. For instance, the proof of a sentence $A \wedge B$ is a pair (p, q) consisting of a proof p of A and a proof q of B. A proof of $A \Rightarrow B$ is a function f which maps each proof p of A to a proof $f(p)$ of B. This semantics corresponds closely to the intuitionistic logic of Brouwer.

Given this view of semantics and the syntactic view of proofs, we can see a correspondence between a proof (a syntactic object) and the abstract construction that is its denotation. To complete this correspondence abstract constructions can be formalized as a system of typed terms describing these semantic objects. The Curry-Howard isomorphism describes this connection between the syntactic proofs and the semantic objects that they denote. Once the semantic objects were formalized, formulas (proofs or deductions) could be identified with the typed terms of the abstract constructions. The typed terms have an operational side not present in the proof. This operational aspect is closely tied to typed terms in computer science. The proofs have a logical aspect not present in the typed (functional) terms. By establishing the isomorphism between the two, we are able to move between the two worlds as needed.

2.3.6 Higher-Order Types and Impredicativity

Russell's initial description of a type theory was naturally higher-order. There are types of objects and types of types (sets of sets), and so on. The utility of higher-order types is partially dependent on what one wants to say and how one wants to say it. Automath [8] is first-order; de Bruijn justifies this restriction to first-order by referring to additional axioms (extra-logical notions) that extend the power of the language. Under the proviso that one is willing to assume certain propositions as proven, a first-order language can be extended to encode quantification over appropriate sets.

However, things being as they are, once a concept exists it is hard to restrain oneself from utilizing it and seeing where it leads. In the Hilbert tradition, the more things that can be expressed, formalized, and proven in a particular formalization, the closer we are to some (uniform) ideal of definition. The formalization of mathematics in a first-order language extended with appropriate axioms leads directly to the question as to what additional power (if any) or expressivity (if any) is to be gained by removing the axioms and going to a more fully higher-order language.

Because of the desire to formalize mathematics, and because of the incompleteness theorems of Goedel [48], higher-order types were explored to attain the expressivity necessary to define mathematics in a uniform context in a way that avoided the incompleteness problems. Hierarchies of types allowed the consistency and meta-logical properties of each level to be expressed at a higher level. The properties of constructive languages, with higher-order types, were of particular interest: what these languages can express, their decidability properties, and their consistency properties.

Associated with higher-order types is the consideration on how the system is formed: is it predicative or impredicative? Predicativity refers to those systems that are predicated on predefined assumptions. In contrast, impredicative languages and systems expect the definition of assumptions in the language itself, reducing the a priori information that must be assumed. To some extent, each language has a predicative and an impredicative component, where the predicative component is what primitives are predefined (assumed) in the language. Languages that are generally referred to as impredicative, such as the calculus of constructions, have very few primitive definitions: it supports the concept of the proposition, but not any particular propositions. Using this notion of a proposition, one can define (for instance) a proposition that captures the properties of integers, lists, stacks, and so on. Systems such as Martin Löf's predicative system [33] provide definitions and behaviors for primitive constants, such as the recursive rules for different constructors. These rules are then used in building new types and propositions.

In the drive to formalize mathematics foundationally, the ultimate goal is for a system that is entirely formalized without any extra-logical components. In such a system, we do not need access to beliefs or terms constructed or formed outside the bounds of the

system. Impredicative definitions often seem circular as they don't refer to anything outside themselves. However, there are always extra-logical constants: ones that we believe to be intuitively true or valid in some external sense (see Russell's comments on Peano [40]).

As mentioned above, first-order languages may be extended with axioms allowing higher-order concepts to be expressed; this extension is obviously predicative. Their properties and our understanding of the meaning of the statements produced in such a system are contingent on our understanding or believing the predicative base. If we go to an impredicative system, first-order languages are no longer sufficient. So the conjunction of higher-order types with impredicativity provides the base for a pure, uniform, formalization of constructive mathematics and computer science [14, 15, 29, 30].

2.3.7 The Use of Higher-Order Logic in *MF*

Taking these developments together has led to systems, both predicative and impredicative, exploring what portions of mathematics and computer science are definable.

Martin-Löf concluded at one point with the comment [33]:

> ... formalization taken together with the ensuing proof-theoretical analysis effectuates the computerization of abstract intuitionistic mathematics that above all Bishop has asked for. What is doubtful at present is not whether computerization is possible, because we already know that, but rather whether these proof-theoretical computation procedures are at all useful in practice. So far, they do not seem to have found a single significant application.

Many of the questions confronting researchers today are formal: how to do a particular thing in the context of constructive definitions of programming and specification activities. But, in a sense, the flavor is foundational. While the overall goal of intuitionistic philosophy was to expose the foundations of mathematics, it has met a more limited and fruitful domain in the field of computer science. As Martin-Löf [34] believed, if we view a program as the design of a method for computing a value, then the construction of a program (function) is identical to the construction of a proof of a mathematical object (value).

The research has been driven from two directions. Looking at the logic itself, some research has aimed at giving a computational meaning to new logical connectives. The logic may be expanded either in its connectives, its domain of quantification, or both. Similarly, some research has started with existing computational forms: where a computational form exists, find a suitable meaning expressed in the logic. Both of these perspectives use the type system as an intermediary between the logic and the computational form [4]. Using the type system as an initial point is often confused with the logic, because of the pervasive viewpoint of the proposition as types.

An orthogonal research area has been to investigate the meta-theory associated with the connection of a computable artifact, its type, and its meaning in the logic. When we interpret a $\lambda-$term as the type $A \rightarrow B$, which in a system of total functions is a type equivalent to the proposition $\forall A.B$, we make claims that are outside the logic. If we want to investigate this claim, we need to look at the connection between the artifacts, rather than at the artifacts. Systems such as Harper, Honsell, and Plotkin's LF [23] encode this position in their judgement-as-types paradigm. The kinds of judgement that a system makes can be treated as types. This view of judgements is reminiscent of the role that proof theory took on as it became a manipulable artifact. Until this view took root, proof was external to formalization. With a change in viewpoint, the connection among certain kinds of proofs, $\lambda-$terms, and typing could be investigated.

The problem being investigated in this thesis is a generalization of what a proof is and what kinds of judgements can be made. As logics have become more powerful, more expressive, and capable of expressing and manipulating larger portions of mathematics and computer science the unit of interest is not solely an individual proof or judgement. There is now a need to look at sequences of proofs, where the sequence represents some portion (whole or part) of mathematics or computer science. Where an individual proof requires a context of other proofs, a sequence of related proofs may require a prior sequence. I call the sequence of proofs a theory or theory presentation. A theory presentation is only well formed if the assumed theory presentations are available, much as a proof is only well-formed if the assumed proofs are available.

Because of the similarity of theory dependencies to proof dependencies, some systems

have reflected some of the meta-theory of a sequence of proofs into the logic itself. Systems such as LF [23] define the primitives that allow more complex and interrelated structures to be constructed. Designs such as the work on structure and representation in *LF*'s [24] define higher level structures at a significantly more general level, facilitating a formalism that can be used to describe a system that encodes a meta-theory. Rather than having a framework that allows a discourse or meta-theory to be defined, this supports a framework for the direct definition of a meta-theory that may subsequently be used to describe specific logical arguments. The meta-theory is embedded in the formal system as well as the object logic.

To explore the role of theories and how they relate to one another, I propose that the necessary formalism should primarily focus on the structure and relationships of theories. This exploration can be done most effectively by isolating the structure rather than embedding it in another formal system. Girard suggests that proof theory is the logic of logic [20]. Research in proof theory focuses on the proofs independent of the content of what is being proved. Similarly, I suggest that the formalization of theories should be constructed independent of the content of the theories. Higher-order logics open up the potential to describe these structures because of their expressivity; this expressivity allows them to both express more complex artifacts necessitating a greater need for a unifying formalism as control and suggests the construction of a solution. However, intermingling the structure with the contents has the effect of confusing the issue as to what exactly is being referred to. Moreover, the needs of theory presentation and structure may not be the same as the needs of the theory contents itself.

The process of generalizing from a proof to a sequence of proofs, and from a proof context to a theory environment introduces a new object of discourse (the structure) and abstracts away irrelevant detail (the particular proofs). Isolating the treatment of theories from the treatment of their contents focuses on the different requirements, judgements, and roles of the two. The research described in this dissertation focuses on the required judgements and role of theories, relying on a suitably expressive underlying logic to express the contents of the theories. The correctness of the theories produced is assured by the constraints of the underlying logic. The correctness of the theory connections is assured

by the formalization of the meta-theory, which has been shown to preserve the correctness of the underlying logical argument.

2.4 Goals of This Research

As mentioned in the previous section, this thesis examines a new metaphor for describing logical arguments.

When the focus of a logical argument is on how the set of proofs is organized, the structural choices are made explicit to the user. Rather than starting from the notion of a proof, the metaphor is inverted and the argument starts from the top: what a collection of proofs is intended to represent or denote, independent of the nature of a particular proof. The question that I look at is not how an individual construction (proof) is formed or other properties of a construction per se. There is substantial work on supporting proof construction at present, in several variants of constructive type systems, with varying properties.

Dummett [18] states that what we know is entirely determined by use.

> The meaning of a mathematical statement determines and is exhaustively determined by its use. The meaning of such a statement cannot be, or contain as an ingredient, anything which is not manifest in the use made of it, lying solely in the mind of the individual who apprehends that meaning.

So we can legitimately ask to what use we want to put these systems. My goal is to answer questions such as:

1. What theories do I require to make a particular theory well-formed?

2. When can I judge that a theory reference (dependency) has been discharged?

MF uses the structure of theories as well as the properities of the underlying logical framework to judge when a theory presentation is well-formed and when theory references have been discharged.

Some of these questions rely on a notion of persistence and reusability. The goal is not merely to serialize a set of propositions in some order specified by the *book* as in

Automath, but to encode this order and dependencies such that the order becomes an elementary object in itself. This tangible representation of the order should allow the exposure of regularities in structure much like the regularities exposed via Gentzen's and Prawitz's tangible representation of proof theory.

In that context I suggest that the elementary objects be theory presentations and theory environments. The role of theory environments in this sense breaks into two aspects:

1. Definition: the environment defines implicit abstraction.

2. Use: the environment provides any assumptions required for use.

The role of theories is to collect the pieces of information of the theory, in whatever form it is presented, and to maintain that unity.

I am interested in constructing presentations in a particular structured way that will preserve relationships among theories such that those relationships can be used to discharge dependencies or assumptions. This structure would also accommodate theory changes, both those that maintain relationships among presentations and those that don't, and a *framework* to accommodate state changes in dependencies such as versions.

This approach was suggested by the structural treatment of theories in *Clear* [10, 11]. A similar structural approach is also being used by Harper, Sannella and Tarlecki [24] to describe a treatment and meta-theory for theories They are working on structuring theories in *LF*, which itself is a means to uniformly express the meta–theory of an object logic via encoding. The approach of *MF* is distinguished from theirs in the following characteristics:

- *MF* establishes an ordering on the objects induced by the structure of the theory (how it was built).

- *MF* allows limited higher-order quantification over theories.

- *MF* uses the order of theories as constraints to identify when a theory dependency is discharged.

- *MF* uses a database for the persistent portion of the design process. This persistent database also allows assumptions to be discharged with pre-existing theory presentations. These theory presentations can be viewed as expanding the predicative base of the discourse for a particular argument.

- *MF* is based on a particular logic (the theory of constructions)[14]. However, the initial goal of the research presented here is to explore the particulars of directly manipulating theories through their structure. The general mechanism of encoding and treating the object level indirectly has an associated cognitive cost of sometimes making it difficult to understand the distinction between the objects and their meta–theory. By abstracting out the structure we hope to better understand it's role in theory development.

I use a formalization based on the inheritance hierarchies of Aït-Kaci and Smolka, and an associated logic based on Smolka's generalization of inheritance hierarchies. Together these allow the user to describe, manipulate, and use the theory presentation that comprise a particular discourse. They encode the structure of the logical argument, and allow the argument to be recreated with the required theories.

This research presents a promising solution to organizing this particular kind of knowledge (the discourse). I see this framework as a first step towards a full system to manage, construct, and organize theory presentations such that the presentations can be (re-)used, shared, contrasted, and validated in the relevant theory environment. Rather than view the process of constructing theories as isolated from the process of making specifications or statements about computational entities, the goal is to elaborate and manipulate explicitly the process via the structure it imposes.

Chapter 3

Examples of Presentation Construction and Manipulation

MF defines a language to manage sequences of statements in a logical framework. For the purposes of the definition of *MF* we chose the Calculus of Constructions as implemented in *LEGO* [31] [32], although any logical framework with certain properties will do (see Chapter 1).

The goal of this chapter is to give an understanding of the components of *MF* by introducing some terminology that will be used throughout the description of *MF*, presenting an example of an *MF* style derivation, and showing how an *MF* derivation is translated into statements in the underlying logical framework. The full definition of *MF* is given in Chapter 4. The purpose of the *MF* language is to provide structure to unstructured collections of *LF* statements. Without going into the full details of the *MF* language we let the term "*MF* statement" refer to any legal term in *MF* that denotes an *LF* statement [1]. *MF* is defined in three levels, corresponding to the abstractions introduced (see Figure 3.1). Figure 3.1 shows the construction of two theories (A and D) from theories B anc C. The Level 3 operators (**combine** and **extend**) correspond to Level 2 operators ($\cup$ and $\odot$). *LF* statements are given identifiers (c_i) and values (x_i), and are accessed in *MF* through feature identifiers (ι_i). The lowest level (Level 1) maps identifiers to *LF* statements resulting in a flat context, supporting only β-reduction. The next level (Level 2) introduces the concept of a sequence of statements, supporting sequence combination using theory

[1] I am using the term *MF* statement to cover any of: identifier symbols for *LF* encodings (c_i), type symbols for the c_i ($\hat{c_i}$), and feature symbols (ι_i), that refer to c_i. All of these *MF* expressions are interpreted as *LF* statements, although they occur at different levels in the *MF* system.

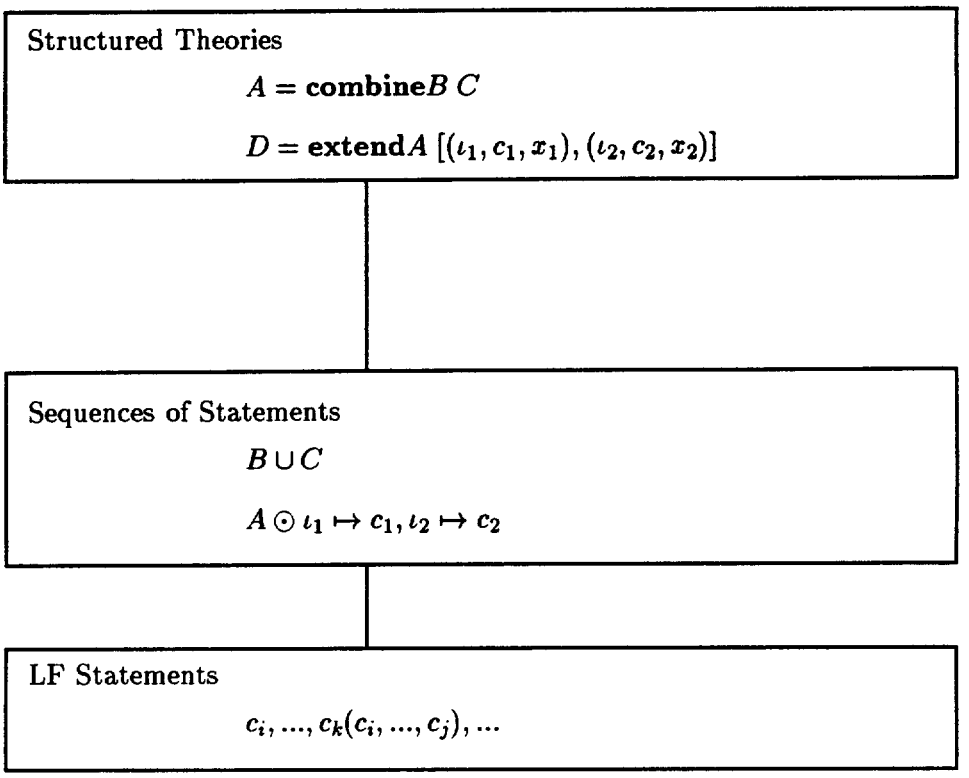

Figure 3.1: High Level View of *MF*

presentation constructors (structured theory operators). Each statement in a sequence is identified by an *MF* identifier, called a feature. The top level (Level 3) introduces the concept of a named theory presentation, allowing sequences of statements to be named and used in the construction of new theory presentations and supporting queries of the persistent structure theory presentations. Each level is interpreted by the structures of the next lower level, so that ultimately *MF* theory presentations are interpreted by the underlying logical framework.

An *MF* statement is a reference to an encoding of a legal statement in the underlying logical framework. In the *LEGO* implementation of the Calculus of Constructions, the legal statements include propositions, types, theorems, and definitions. In an *LF*, a sequence of legal statements may be used to describe theories, statements about theories, specifications, properties of specifications, etc. These statements become part of the *LF*

context, and the *LF* may be regarded as having been extended with these statements, providing a new environment for further definition. Statements introduced in this extended *LF* may refer to any statement in the context. However, the *LF* context is global and flat: all statements are introduced into a global context, accessible to all, without regard to dependencies.

Contrary to this flat environment, when a sequence of *LF* statements is introduced, the intent is that some statements belong together, and when taken together describe a meaningful theory. In *MF* we can name this sequence so that when we reference the name we will access the entire sequence. We call this named sequence of statements a *theory presentation.* For example, the statements introduced in a theory presentation (A) may refer to other *LF* statements ($c_0, ..., c_j$). In this sense, A is only meaningful in the *LF* when the *LF* has been extended with the necessary *LF* statements $c_0, ..., c_j$, assuming that the statements $c_0, ..., c_j$ themselves are meaningful in the *LF*.

However, it is not really sufficient to extend the *LF* solely with the required statements. If the intent of a theory presentation is to collect the statements necessary to describe a theory, we should not separate a statement from the other statements occurring in a theory presentation. To express this requirement, *MF* defines a structural environment for a theory presentation. The *structural environment* required by a theory presentation A is the collection of theory presentations in which all the *LF* statements required by A are introduced. The structural environment of a theory presentation A denotes the portion of the flat, global *LF* context that is required by the statements defined in A as well as those that are dictated by the theory presentation in which each c_i is introduced. The structural environment is an *MF* structure that is mapped to the flat *LF* context, as required.

In *MF* a theory presentation is defined by identifying the introduced *LF* sequence of statements and the structural environment required by these statements. Newly introduced *LF* statements can make use of any statements present in the *LF* context determined by the structural environment and are presumed to be correct assuming the statements in the context are correct (recall that ultimately the correctness is checked by the underlying *LF*). The *MF* relationship of a theory presentation to another is only captured at the level of theory presentations, rather than at the level of *MF* statements or *LF* statements.

To demonstrate this: an *LF* statement may reference another *LF* statement directly by referring to its *LF* identifier. If we refer to the *LF* identifier of the statement c_0 by $id(c_0)$, statement c_1 may reference statement c_0 by referring to $id(c_0)$ in its *LF* definition. Now, c_0 may be introduced into *MF* in theory presentation A, and c_1 in theory presentation B. In that case, we have the dependency that B requires A. We do not, however, have any *MF* dependency on the individual statement as identified by the *LF* identifier or an *MF* statement that identifies an individual statement. This form of dependency relation means that the correctness of the dependency relation in the *LF* is dependent on a correct identification of which theory is required.

In the logical framework there is no explicit distinction between the context for a sequence of statements and the statements themselves. A statement is introduced in a particular context (the statements and assumptions known prior to the introduction of the statement). After the statement has been introduced, the context has been extended. *MF* provides this missing structure and connection between the introduced statements of a presentation and its context by constructing a persistent hierarchy (partial order) of theory presentations relating all presentations to their environments. The partial order is defined as $A < B$ if A is in the environment of B, meaning that B is an extension of A. We use the terms *structural environment* and environment interchangeably, reflecting the structured nature of *MF* environments in contrast to the flat contexts of *LF*. The *presentation base* refers to the persistent hierarchy. The *structural environment* of a theory presentation A is the downward closure of the presentation A in the *presentation base*, and determines the required theory presentations for A.

Once a theory presentation is defined and its relationship to other theory presentations recorded in the presentation base, it is given a meaning based on the interpretation it is given in the underlying *LF*. The meaning given to an *LF* encoding of an *LF* statement is the meaning given by the *LF* to the translation of the statement. The meaning given to a sequence of encodings of *LF* statements $(c_0, \cdots, c_j)$ is the meaning given to each statement in order. The meaning of a theory presentation in the empty environment is simply the meaning given to the sequence of statements that compose the theory presentation. The meaning given to a theory presentation in a non-empty environment is the meaning given

to the environment (preserving the dependencies specified in the partial order, such that if $A < B$ then the meaning of A must be determined prior to the meaning of B, since the statements in B may make use of any statement in A), followed by the meaning given to B. Ultimately, the meaning of a structured *MF* theory presentation is the meaning given to the flat sequence of statements that the theory presentation denotes in the *LF*, as generated by the (informal) rules above. We can manipulate the theory presentations in the presentation base so long as the meaning given to the theory presentations is not altered, allowing us to combine environments in a variety of meaning-preserving transformations.

3.1 The Form of the Underlying Logical Framework Statements

In order to support some of the theory constructors of *MF*, we assume that the statements in the logical framework have a specific form. While a logical framework may include a variety of statement types, they all can be translated into a definition, having the following form:

$$(LF\,identifier,\ [Context,\ Construction,\ Type])$$

This pair captures all statements that may be made. All statements must have an identifier identifying the context of assumptions under which this definition was made, the construction (or proof object) the identifier denotes, and its type. Not all the components are required. For instance, the *LEGO* declaration

$$[A : Prop]$$

asserting that we assume the existence of a *Prop* referred to by the identifier A, would have the pair $(A, [[], [], Prop])$.

A sequence of *LF* statements that are to be encapsulated into a theory may have a hypothetical context. This context holds assumptions in the form of declarations. This context is intended to be discharged at some later time. We assume that sequences of statements having a context of this type have been translated such that the hypothetical

context is defined in the context of each statement, as in the following example.

$[A : Prop]$ The hypothetical component.
$[B : A- > A]$
$[C : A]$

$P_1 = M_1 : T_1$ A proof object.
$P_2 = M_2 : T_2$ A proof object.
...
$P_n = M_n : T_n$ A proof object.

The above context and proof objects can be translated into the following proof objects, equivalent to the proof objects above.

$$P_1[A : Prop][B : A- > A][C : A] = M_1 : T_1$$
$$P_2[A : Prop][B : A- > A][C : A] = M_2 : T_2$$
$$\ldots$$
$$P_n[A : Prop][B : A- > A][C : A] = M_n : T_n$$

A consequence of this assumption is that we can't really insure linguistically that a sequence of assumptions all be discharged from one theory, although *MF* will support the application of a theory in the form above to another theory. I.e., there are no existing constraints that would require that A, B, and C above are all discharged from the same theory. A discussion on this limitation and planned extensions are discussed in Chapter 6.

3.2 *MF* Constructions and the Structural Environment

To construct a theory presentation we need to identify the required structural environment and any introduced *LF* statements. We define the structural environment of a theory presentation according to the way that the theory presentation was constructed. The *LF* statements are recorded using features.

To distinguish the *MF* statement from the *LF* statement, to allow them to be accessed independently, and to support β-reduction in *MF*, each *MF* statement is represented as a

feature of a theory presentation. For example, if we introduce the *MF* theory presentation A, intended to represent the sequence of *LF* statements $c_0, ..., c_j$, each c_i is represented as a feature of the theory presentation A having a feature identifier ι_i. In the presentation base, *LF* statements are referred to by their feature identifier[2].

In *MF* the four methods of theory construction (and an abbreviated syntax for each) are:

- Combine: combine 2 (or more) theory presentations into one theory presentation.

  $$C \equiv \textbf{combine } A\ B$$

  Combining two theories results in a new theory presentation that is the result of combining the meanings of the two theories. The simplest notion of the combination of two theories is simply concatenation. The new theory presentation C will refer to this combination.

- Generalize: two (or more) theory presentations can be generalized, constructing a new theory presentation that is the intersection of both. If a theory presentation is viewed as a flat sequence of statements including the statements in its context, this intersection is the sequence of statements that are common to both theory presentations. If a theory presentation is viewed as a structural environment and a sequence of newly introduced statements, the generalization is the intersection of the two structural environments: i.e. the common structural environment to the two theory presentations.

  $$F \equiv \textbf{generalize } D\ E$$

  If D is a combination of A and B, and if E is a combination of B and C, then the common structural environment of D and E is the theory presentation B. The new theory presentation F will refer to this generalization.

[2]Because features ultimately map to *LF* statements with *LF* identifiers, we can think of the feature identifier as the *LF* identifier. And, if two arbitrary theories are to be allowed to be combined in arbitrary ways, this 1-1 correspondence is appropriate: both the feature identifier space and the *LF* identifier space require uniqueness. However, to leave open the possibilities of re-using the same construction in different theories, outside the theory presentation structure, and also to leave open the possibilities of extending *MF* to allow for multiple uses of features, the two name spaces are distinct.

- Extension: an existing theory presentation can be extended by introducing new statements.

$$B \equiv \textbf{extend } A\ [(\iota_i, c_i, x_i), \cdots, (\iota_n, c_n, x_n)]$$

Features ι_i identify in MF the indicated LF statement c_i, with the value of x_i. The value x_i is assumed to have the LF encoded form [*Context*, *Construction*, *Type*] described in the previous section.

Extension corresponds to adding more statements to a theory presentation. The new theory presentation B is said to have features $\iota_i, \cdots, \iota_n$, each denoting an LF identifier $c_i, \cdots, c_n$, having values $x_i, \cdots, x_n$, respectively.

- Instantiation: an existing theory presentation can be instantiated by applying the statements of one to selected statements of another. The indicated statements of A (designated by the list $[\iota_{A_i}]$) are applied to the corresponding statements of B (designated by the list of $[\iota_{B_i}]$). In the newly constructed theory presentation, the new statements have feature identifiers specified by the list $[\iota_{C_i}]$, and each has a new LF identifier $c_1, \cdots, c_m$ respectively.

$$C \equiv \textbf{instantiate } A\ B[\iota_{A_1}, \cdots, \iota_{A_m}]\ [\iota_{B_1}, \cdots, \iota_{B_n}]\ [\iota_{C_1}, \cdots, \iota_{C_m}]\ [c_1, \cdots, c_m]$$

To give an intuition for instantiate, we will have to use a slight abuse of notation. If we assume that ι_{A_i} refers to the i'th feature defined in the theory presentation A, and similarly for B, and $\iota_{A_j}(\iota_{B_i})$ is a term that will be given an interpretation in the LF through β-application, then the the above construction can be thought of as defining a new extension to the combination of A and B, where the extension has features defined by the triples:

$$(\iota_{C_1}, c_1, \iota_{A_1}(\iota_{B_1}, \cdots, \iota_{B_n}))$$
$$(\iota_{C_2}, c_2, \iota_{A_2}(\iota_{B_1}, \cdots, \iota_{B_n}))$$
$$\cdots$$
$$(\iota_{C_m}, c_m, \iota_{A_m}(\iota_{B_1}, \cdots, \iota_{B_n}))$$

When given a meaning in the LF this will result in the application of the LF statements denoted by the indicated features of A: $\iota_{A_1}, \cdots, \iota_{A_m}$ to the indicated features

of B: $\iota_{B_1}, \cdots, \iota_{B_n}$. If the indicated statements are so constructed that this application is type correct, meaning that the statements denoted by $\iota_{A_1}, \cdots, \iota_{A_m}$ are abstracted on assumptions of the types denoted by the statements $\iota_{B_1}, \cdots, \iota_{B_n}$, this will result in the discharge of the specified assumptions.

The theory presentation B is in the structural environment of A ($B < A$) if

1. $A \equiv$ **combine** $\cdots B \cdots$ or $A \equiv$ **combine** $\cdots C \cdots$ and $B < C$.

   Or:

2. $A \equiv$ **generalize** $C\ D$ and $B < C$ and $B < D$

   Or:

3. $A \equiv$ **extend** $B \cdots$ or $A \equiv$ **extend** $C \cdots$ and $B < C$

   Or:

4. $A \equiv$ **instantiate** $B\ C \cdots$ or $A \equiv$ **instantiate** $C\ B$ or $A \equiv$ **instantiate** $D\ E$ and $B < D$ or $B < E$

3.3 The Presentation Base

A *presentation base* is a persistent structure composed of identifiers bound to sequences of features in a structured hierarchy that determines the structural environment for each theory presentation.

Presentation commands allow a new presentation to be added to the *presentation base* by specifying the introduced statements, the feature identifiers each statement will be known by in *MF*, and the required structural environment using *presentation constructors* (**combine**, **generalize**, **extend**, **instantiate**). *Presentation constructors* identify the direct ancestors (and direct descendents) of a theory presentation in the partial order, and hence a location in the partial order.

The presentation base determines all the theory presentations that have been constructed (named). It defines a closed world of existing building blocks, and how they are all related. Because the structural environment of a theory presentation identifies how

the LF context will be extended, we can think of the theory presentations as the building blocks for the context. Specifying which building blocks are required determines where a new theory presentation will reside in the partial order. The hierarchy of these building blocks gives a map showing which blocks are needed to (re)build any piece. For instance, the construction that combines two sequences into one sequence can be used to construct a new theory presentation by the presentation command:

$$A \equiv \textbf{combine}\ B\ C$$

Now A is a new theory presentation identifier with no introduced statements, with a structural environment of B and C. In the partial order, the new theory is related to its constituents by $B < A, C < A$ and A is a direct descendent of both B and C. To give a meaning to A in LF, we would give a meaning to B and then to C. If B had been constructed by the presentation command (S_0 is the empty theory presentation):

$$B \equiv \textbf{extend}\ S_0[(\iota_1, c_1, x_1), (\iota_2, c_2, x_2)]$$

and C by:

$$C \equiv \textbf{extend}\ S_0[(\iota_3, c_3, x_3), (\iota_4, c_4, x_4)]$$

the presentation base would look like the one shown in Figure 3.2.

A presentation command modifies a presentation base as shown in Figure 3.2, adding new relationships. Presentation constructors may be used independent of a presentation command. When used independently, a *presentation constructor* expression describes how a presentation is constructed and the meaning associated with such an expression is the theory presentation whose structure matches the indicated construction. For instance, the presentation constructor expression **combine** B C above describes the presentation A: this is how A was constructed, by combining the meanings for B and C. The theory presentation A can be referenced either by the presentation identifier (A) or by an expression that matches its construction: **combine** B C. Because the meaning of A is a consistent extension of the meanings B and C, the theory presentation A can be used anywhere B or C can be used. So, it we wanted to know all theory presentations that were consistent with (for example) C, A would be one of those theory presentations.

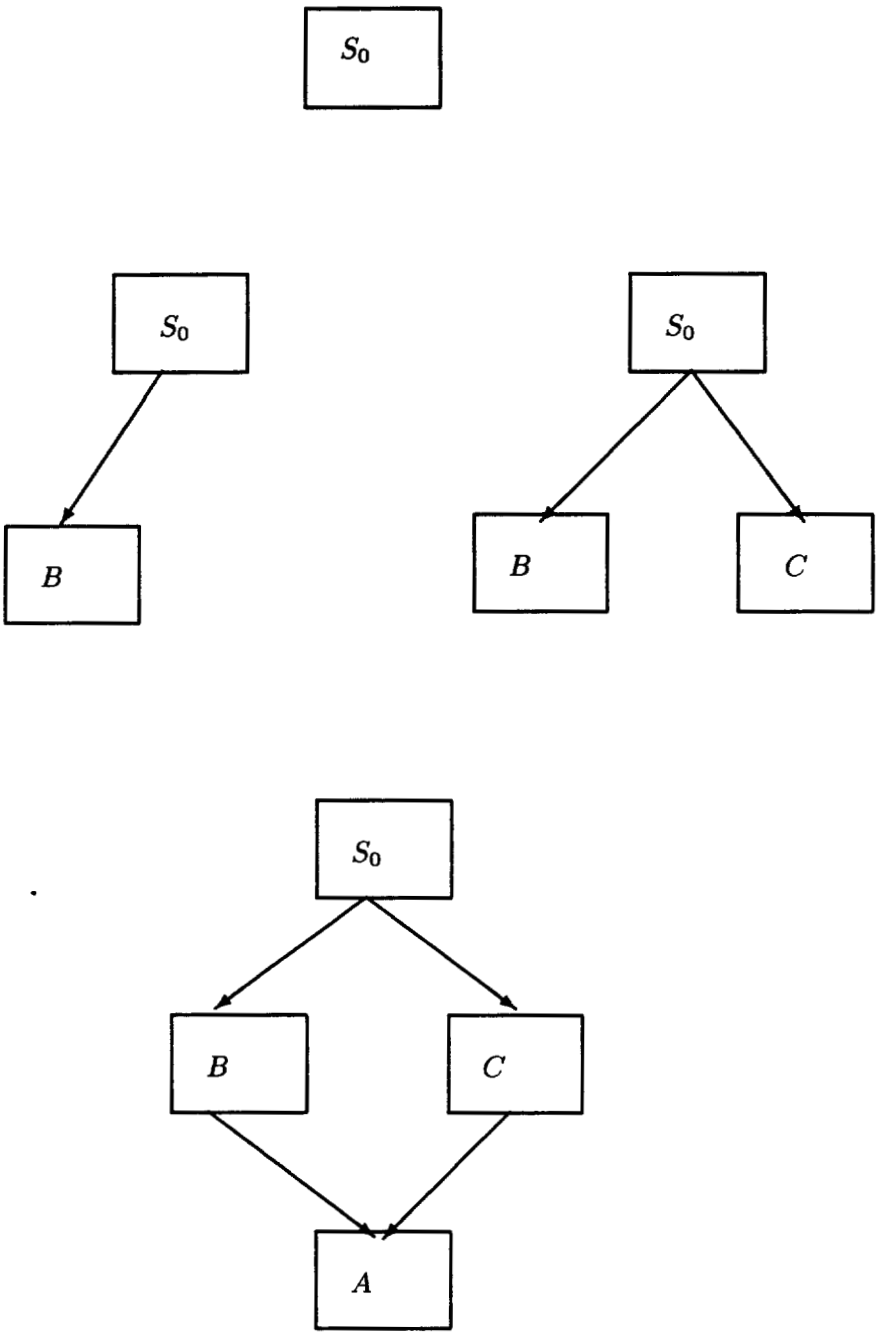

Figure 3.2: Presentation Base for $A \equiv$ **combine** $B\ C$

The presentation base can be accessed in several ways:

1. It (or parts of it) can be used as a structural environment for introducing new sequences and constructing new presentations bound to presentation identifiers.

2. It can be used to "reconstruct" a sequence of *LF* statements for any identifier for interpretation and type checking in the *LF*.

3. Any identifier in the presentation base represents not only the sequence of *LF* statements that compose it but also the extensions to that sequence. These are all the consistent extensions for any *MF* theory presentation. A presentation expression denotes all theory presentations whose construction is consistent with the construction specified in the presentation expression. These can be viewed as all theories that have a common context and at least the indicated introduced statements. So that, in the fragment above, B denotes B and **combine** B C, since **combine** B C is a consistent extension of B.

3.4 An *MF* Derivation of a Presentation Base

We will use the *MF* calculus to construct the examples from Chapter 1, to reconstruct the statements, and to query the resulting presentation base.

3.4.1 *Propositional Logic* Example

We assume that we have an initial presentation base with only the empty theory (S_0) and an environment mapping identifiers c_i to LF encodings [3], such as

c_0	The *And* proposition
c_1	The *Or* proposition
c_2	The *Impl* proposition
c_3	The $\wedge$ theorem
c_4	The $\vee_l$ theorem
c_5	The $\vee_r$ theorem
etc.	

For instance, the *And* proposition would be encoded as:

$$(And, c_0, [[A : Prop, B : Prop, C : Prop], (A \rightarrow B \rightarrow C) \rightarrow C, Prop])$$

This specifies that *And* is the feature identifier, c_0 is the LF identifier, $[A : Prop, B : Prop, C : Prop]$ is the context required for this constructions, $(A \rightarrow B \rightarrow C) \rightarrow C$ is the construction, and *Prop* is its type. Because we are not concerned with the actual LF values, we will refer to the LF encodings for an identifier c_i as v_i

We can construct the theory *Proplogic* by

$$Proplogic \equiv \textbf{extend } S_0 \ [(And, c_0, v_0), (Or, c_1, v_1), \cdots, (Arrow, c_6, v_6)]$$

This states that the theory *Proplogic* is an extension of the empty theory, with the features *And*, *Or*, etc. The features *And*, *Or*, etc. are bound to the LF identifiers which in turn are bound to the LF statements for the associated LF proposition. For example, *And* is bound to the LF identifier c_0, which is bound to the value $[[A : Prop, B : Prop, C : Prop], (A \rightarrow B \rightarrow C) \rightarrow C, Prop]$. MF is not concerned with the particulars of an LF statement, only with recording the relationship that statement has to the theory presentations.

[3] Each c_i represents a specific statement in the logical framework, and is used both as the MF identifier and as the LF identifier for the associated statement.

To continue, we could introduce *Naturalnum* by:

$$Naturalnum \equiv \textbf{extend}\ S_0\ [(Nat, c_7, v_7), (zero, c_8, v_8), (succ, c_9, v_9)]$$

And the theory *Booleans*:

$$Booleans \equiv \textbf{extend}\ S_0\ [(Bool, c_{10}, v_{10}), (True, c_{11}, v_{11}), (False, c_{12}, v_{12})]$$

If we assume that the values for the statements $c_0, c_1, \cdots, c_6$ are organized with a context as specified in Section 3.1 they can be applied in the *LF*. We can then instantiate the theory *Proplogic* to the *Booleans* by:

$$\begin{aligned} PropBools \equiv\ & \textbf{instantiate}\, Proplogic\ Booleans[And, Or, Impl, \wedge, \vee_l, \vee_r, Arrow][Bool] \\ & [AndBool, OrBool, ImplBool, \wedge Bool, \vee_l Bool, \vee_r Bool, \Rightarrow Bool] \\ & [band, bor, bimpl, b\wedge, b\vee_l, b\vee_r, b\Rightarrow] \end{aligned}$$

This will apply the indicated features of *Proplogic* ($[And, Or, Impl, \wedge, \vee_l, \vee_r, Arrow]$) to the feature *Bool* in *Booleans* resulting in a new theory presentation with features *AndBool*, *OrBool*, *ImplBool*, $\wedge Bool$, $\vee_l Bool$, $\vee_r Bool$, and $\Rightarrow Bool$ whose statements are specialized to the *Booleans* through the witness *Bool*. This instantiation is only meaningful if the statements of *Proplogic* are abstracted on an *LF* statement that has the same *LF* type as the statement associated with *Bool* in the theory presentation *Booleans*. Because *MF* does not know about the internal structure of any *LF* statement, instantiation has an ordering requirement: substitution is accomplished by *LF* application rather than direct substitution. This means that in the application above, the statement associated with the feature *Bool* will be substituted for the first abstraction of the features $And, Or, Impl, \wedge, \vee_l, \vee_r$. The example in Chapter 1 was not organized in this way, although it can be. (We present a more detailed example of instantiation later in this chapter.)

3.4.2 The *Group* Example

Next we present the complete example from Chapter 1:

$$
\begin{aligned}
\mathit{ThArity} &\equiv \mathbf{extend}S_0[(\mathit{Arity}, c_{15}, v_{15}), (\mathit{ZeroAr}, c_{16}, v_{16}), (\mathit{SuccAr}, c_{17}, v_{17})] \\
\mathit{ThList} &\equiv \mathbf{extend}S_0[(\mathit{list}, c_{18}, v_{18}), (\mathit{Nil}, c_{19}, v_{19}), (\mathit{Cons}, c_{20}, v_{20})] \\
\mathit{ThAlgSys} &\equiv \mathbf{extend}S_0[(\mathit{arityZero}, c_{21}, v_{21}), (\mathit{arityN}, c_{22}, v_{22}), (\mathit{Omega}, c_{23}, v_{23}), \cdots] \\
\mathit{ThEquality} &\equiv \mathbf{extend}S_0[(\mathit{Eq}, c_{32}, v_{32}), (\mathit{refl}, c_{33}, v_{33})] \\
\mathit{ThPair} &\equiv \mathbf{extend}S_0[(\mathit{And}, c_{34}, v_{34}), (\mathit{Pair}, c_{35}, v_{35})] \\
\mathit{ThGr0} &\equiv \mathbf{combine}\mathit{ThArity}\ \ \mathit{ThList}\ \ \mathit{ThAlgSys} \\
\mathit{ThGroupoid} &\equiv \mathbf{extend}\,\mathit{ThGr0}[(\mathit{cdot}, c_{36}, v_{36})] \\
\mathit{ThSGr0} &\equiv \mathbf{combine}\mathit{ThGroupoid}\ \ \mathit{ThEquality} \\
\mathit{ThSemigroup} &\equiv \mathbf{extend}\,\mathit{ThSGr0}[(\mathit{Assoc}, c_{37}, v_{37})] \\
\mathit{ThMonoid0} &\equiv \mathbf{combine}\mathit{ThSemigroup}\ \ \mathit{ThPair} \\
\mathit{ThMonoid} &\equiv \mathbf{extend}\,\mathit{ThMonoid0}[(\mathit{Ident}, c_{38}, v_{38}), (\mathit{LeftId}, c_{39}, v_{39}), (\mathit{RightId}, c_{40}, v_{40}), \\
&\qquad (\mathit{Identity}, c_{41}, v_{41})]
\end{aligned}
$$

The above *MF* derivation gives the *Presentation Base* shown in Figure 3.3.

We can now extend this. In Chapter 1 we defined 3 propositions that condense what we need to provide to demonstrate that we have a *Groupoid* (proposition (1)), a *Semigroup* (proposition (2)), and a *Monoid* (proposition (3)). If we want to construct a particular groupoid, called *Groupoid1*, we would have to provide a binary operator, show that it is binary (of the type *cdot*). Assuming that c_{42} is an encoding of an *LF* statement demonstrating a proof object for the proposition *cdot*, and that c_{43} is one demonstrating it is binary (of the type *Groupoid*), the following will accomplish this:

$$\mathit{Groupoid1} \equiv \mathbf{extend}\,\mathit{ThGroupoid}[(\mathit{aCdot}, c_{42}, v_{42})(\mathit{GroupoidCdot}, c_{43}, v_{43})]$$

We may then wish to extend this to a semigroup by showing that it is associative (assuming the c_{44} is a proof object of the type *Assoc*, as defined in the complete context for this

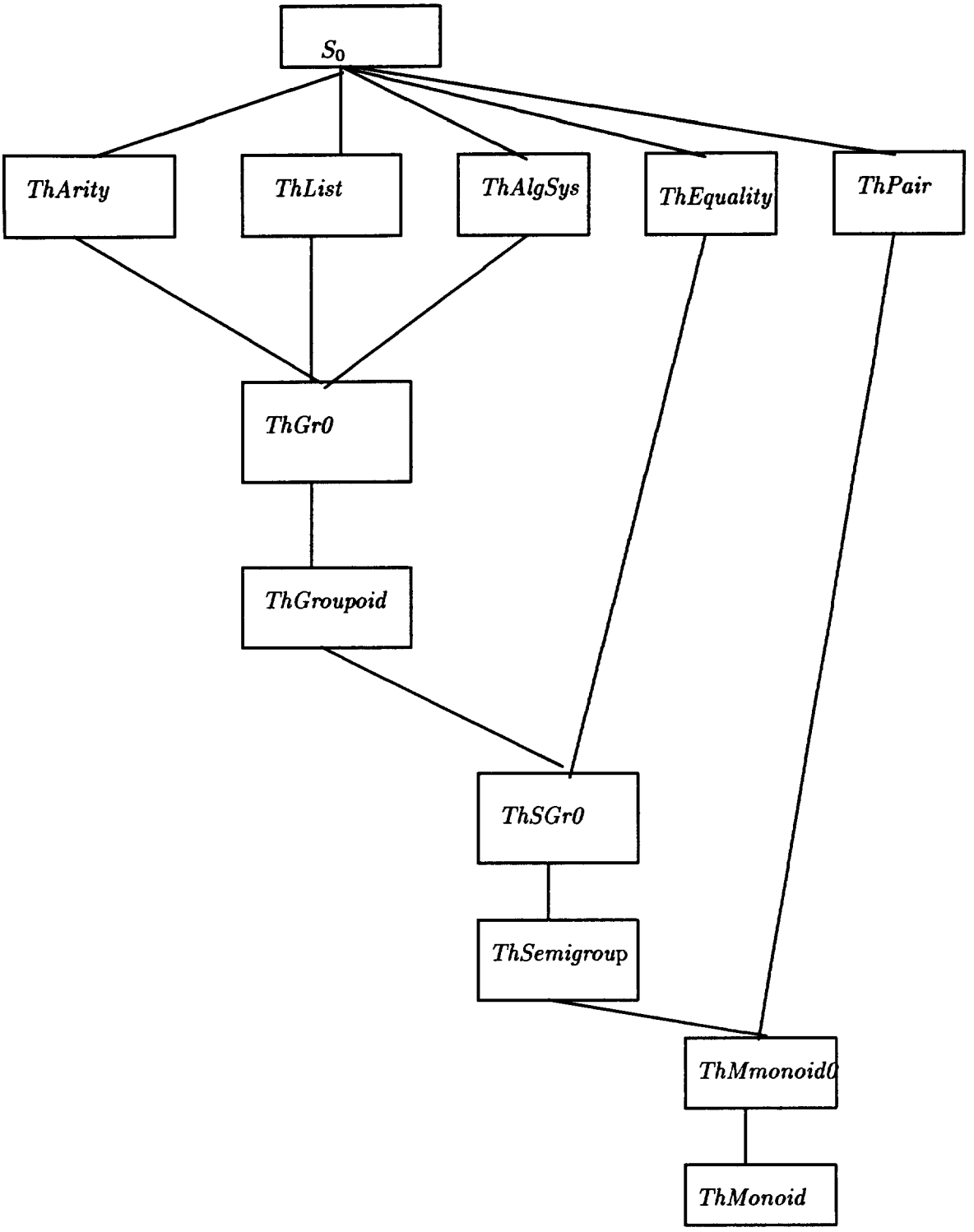

Figure 3.3: Structure of the Presentation Base

example in Chapter 1): [4]

$$\begin{aligned} \mathit{IntermedGrp} &\equiv \textbf{combine}\mathit{Groupoid1}\ \mathit{ThSemigroup} \\ \mathit{AssocGrp1} &\equiv \textbf{extend}\mathit{IntermedGrp}[(\mathit{PrfAssoc}, c_{44}, v_{44})] \end{aligned}$$

However, while *AssocGrp1* is now a groupoid with an associative binary operator and so is a semigroup, it has no relationship to any theory that makes statements about semigroups in the Presentation Base. (See Figure 3.4.)

Instantiation is the means by which *LF* statements introduced in a theory presentation can be applied to other statements intended to discharge specific dependencies. Let the theory presentation *SemiGrpStmts* be a theory composed of such statements. Proposition (2) (which is not included in any theory) states that something is a semigroup if it has a substructure that is a *Groupoid* and the operator is associative. To assure that all statements are indeed about a semigroup, each statement will be contextualized with the necessary assumptions:

C_{S_1} $[A : \mathit{Prop}][b : \mathit{cdot}][g : (\mathit{Groupoid}\ A\ \mathit{cdot})][p : (\mathit{Assoc}\ A\ \mathit{cdot})]$ Statement Body.

C_{S_2} $[A : \mathit{Prop}][b : \mathit{cdot}][g : (\mathit{Groupoid}\ A\ \mathit{cdot})][p : (\mathit{Assoc}\ A\ \mathit{cdot})]$ Statement Body.

...

$$\mathit{SemiGrpStmts} \equiv \textbf{extend}\mathit{ThSemigroup}[(\iota_{s_1}, c_{s_1}, v_{s_1},), (\iota_{s_2}, c_{s_2}, v_{s_2}), \cdots]$$

Assuming that *GroupoidCdot* is of the type (*Groupoid A cdot*), we can instantiate *SemiGrpStmts* by:

$$\mathit{SGrp} \equiv \textbf{instantiate}\ \mathit{SemiGrpStmts}\ \mathit{Groupoid1}\ [\iota_{s_1}, \iota_{s_2}, \cdots]\ [A\ \mathit{aCdot}\ \mathit{GroupoidCdot}]\ [N\iota_{s_1}, N\iota_{s_2}, \cdots]\ \cdots$$

SGrp is now instantiated with all the statements from the groupoid to discharge its dependency on a group.

$$\mathit{ASGrp} \equiv \textbf{instantiate}\ \mathit{SGrp}\ \mathit{AssocGrp1} \quad [N_{\iota_{S_1}},, N_{\iota_{S_2}}, \cdots]\ [\mathit{PrfAssoc}] \cdots$$

[4] We have an intermediate step that serves to name the combination, as a consequence of the decision to only allow named constructions. This can be fairly cumbersome but can be obviated by a macro constructor that performs the two steps automatically. Or, more generally, *MF* could allow local naming in some contexts. This issue will be discussed more in extensions and conclusions.

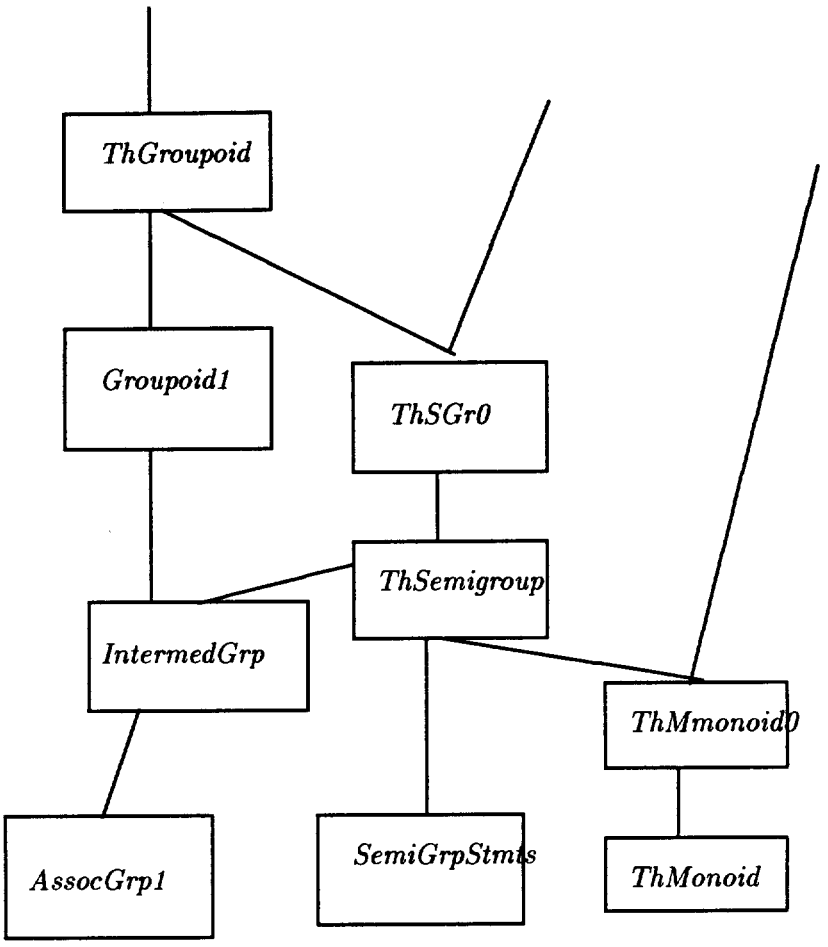

Figure 3.4: *AssocGrp1* in the Presentation Base

ASGrp now has all the semigroup dependencies discharged. This approach will allow *SemiGrpStmts* to be instantiated with any number of specific semigroups. The resulting Presentation Base is shown in Figure 3.5.

We glossed over a few details in the discussion above. Where did the context element $[A : Prop]$ come from? How is it discharged? Is there a relationship between the A in *SemiGrpStmts*, *SGrp*, and *ASGrp*? In general, theories are dependent on some set of assumptions, which may or may not be discharged. If we discharge a dependency $[A : Prop]$ with another $[B : Prop]$, it is still dependent on the assumption that there is an arbitrary proposition that we will call B. More than one theory may be dependent on an arbitrary B. How can we express that theories are dependent an an arbitrary B, but that the B must be the same for all? We address this restriction next.

3.4.3 Continuation of *instantiate* example

In Chapter 1 we gave a general context that specifies the assumptions needed without committing to any proof object. There may be more than one representation of a theory of list, or of arity, for example. A context such as the one in Chapter 1 only commits to the type. If we have a common context of expectations, we can express a conditional theory, where we expect to discharge some or all of the conditions later.

For instance, we might express the dependencies of a theory presentation (i.e. the statements of a theory presentation) by a context to be discharged. We can define LF encodings for c_{50}, c_{51}, c_{52} formulated with a common context expressing this dependency:

$$\begin{aligned} c_{50} &= [A : Prop][L : Prop \to Prop][C : A \to (L\ A) \to (L\ A)][N : (L\ A)] \cdots \\ c_{51} &= [A : Prop][L : Prop \to Prop][C : A \to (L\ A) \to (L\ A)][N : (L\ A)] \cdots \\ c_{52} &= [A : Prop][L : Prop \to Prop][C : A \to (L\ A) \to (L\ A)][N : (L\ A)] \cdots \end{aligned}$$

Now we can construct a theory with these statements without any dependencies on the theories that might be used to discharge these assumptions in each context:

$$S \equiv \mathbf{extend} S_0[(Xone, c_{50}, v_{50}), (Xtwo, c_{51}, v_{51}), (Xthree, c_{52}, v_{52})]$$

The $[A : Prop]$ in the context above is an assumption that must be discharged, along with the expectation that a witness for L, C, N will be provided. We will alter the definition

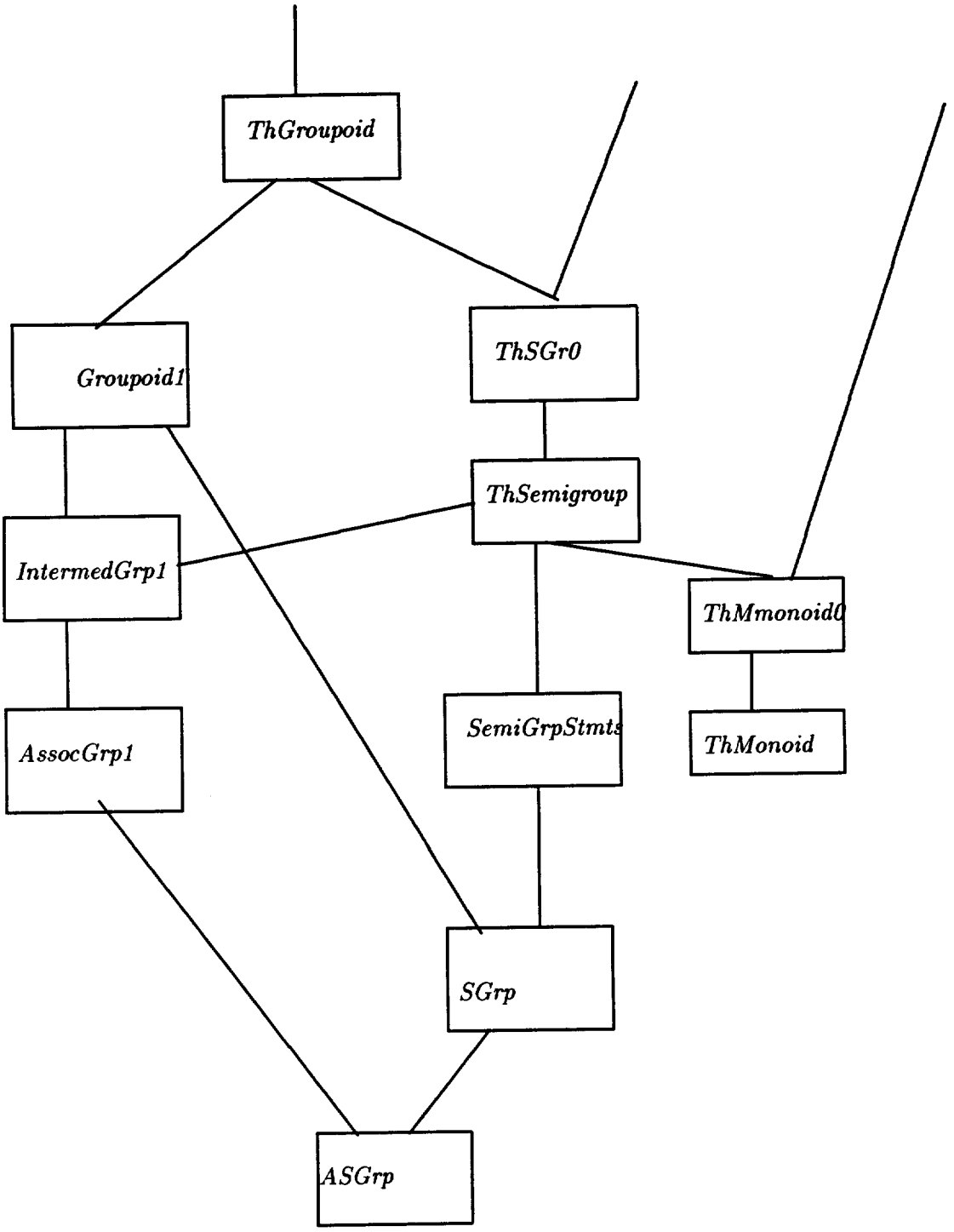

Figure 3.5: Instantiated Semigroup

of *ThList* to provide a consistent theory for use in instantiation:

$$\begin{aligned} c_{18} &= list \quad [A : Prop] \quad [B : Prop](A \to B \to B) \to B \to B \\ c_{19} &= Nil \quad [A : Prop] \quad [B : Prop][C : A \to B \to B][N : B] \quad N \\ c_{20} &= Cons \quad [A : Prop] \quad [a : A][l : list\ A][B : Prop][C : A \to B \to B][N : B] \\ &\qquad (C\ a\ (l\ B\ C\ N)) \end{aligned}$$

This version of the theory *ThList* has all its statements dependent on A such that all can be applied meaningfully. So if we applied them all to the proposition *Nat*, we would be referring to a list of Nats.

How do we discharge the dependencies of the theory presentation S with the witnesses of *ThList* while preserving the overall dependency on an arbitrary A? We continue the example by providing a theory presentation whose sole purpose is to capture the role of the common hypothetical dependency, allowing us to provide a common dependency on an arbitrary A.

$$c_{53} = A \quad ([], [], Prop)$$

$$AHyp \equiv \textbf{extend}\ S_0\ [(FeatureA, c_{53}, v_{53})]$$

Now we can discharge the dependency in *ThList* by:

$$ThListDisch \equiv \textbf{instantiate}\ ThList\ AHyp\ [list, Nil, Cons]\ [FeatureA]\ [Alist, ACons, ANil]\ [al, ac, an]$$

Similarly, we can discharge the dependency in S:

$$SDisch \equiv \textbf{instantiate}\ S\ AHyp\ [Xone, Xtwo, Xthree]\ [FeatureA]\ [aXone, aXtwo, aXthree]\ [aX_0, aX_1, aX_2]$$

These two theories only differ from their parent theory in that they are no longer dependent on a witness for A. We can now instantiate *SDisch* with witness from *ThList-Disch*:

$$T \equiv \textbf{instantiate}\ SDisch\ ThListDisch\ [aXone, aXtwo, aXthree]\ [Alist, ACons, ANil]\ [nXone, nXtwo, nXthree]\ [aLX_0, aLX_1, aLX_2]$$

The new presentation base structure is shown in Figure 3.6.

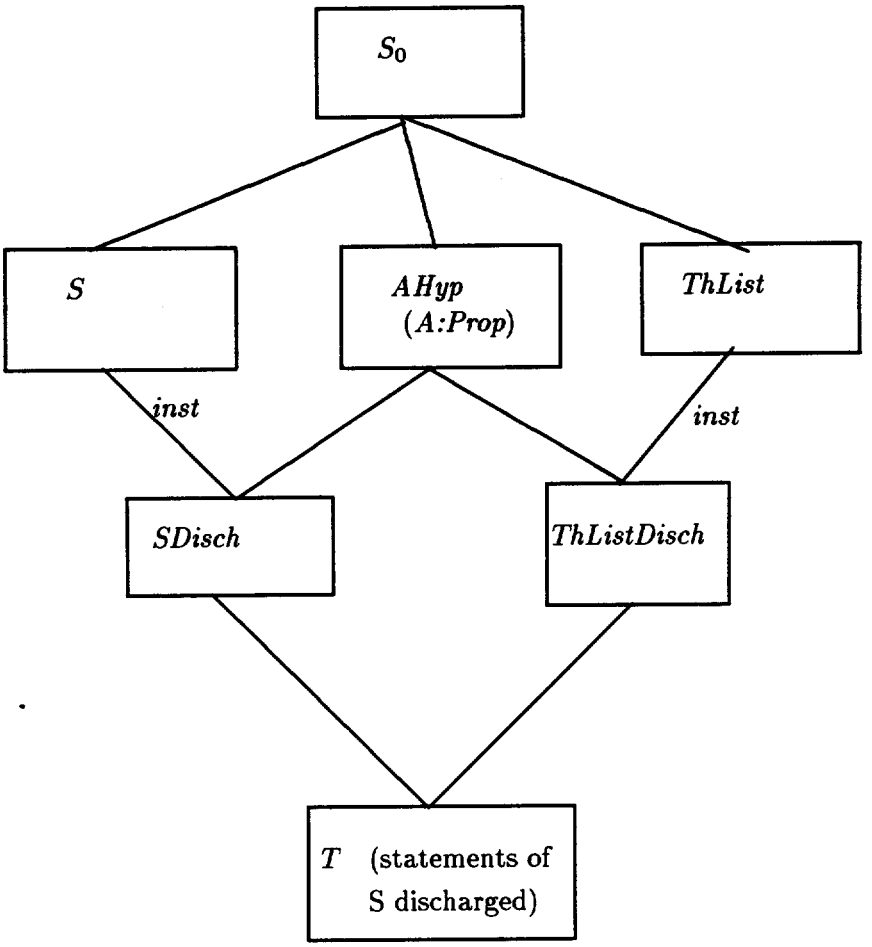

Figure 3.6: Instantiation with a Common Dependency

S is different from a theory that is specifically dependent on a particular list theory: it can be used with a variety of theory presentations with LF statements of the appropriate types. The resulting theory presentation T has each of the statements of S specialized to the witnesses provided by the theory *ThList* via *ThListDisch*. The introduced statements of *ThListDisch* are:

Alist	(*list* (*FeatureA*))
ACons	(*Cons* (*FeatureA*))
ANil	(*Nil* (*FeatureA*))

The introduced statements of *SDisch* are:

aXone	(*Xone* (*FeatureA*))
aXtwo	(*Xtwo* (*FeatureA*))
aXthree	(*Xthree* (*FeatureA*))

The introduced statements of T are:

nXone	(*aXone* (*Alist Acons ANil*))
nXtwo	(*aXtwo* (*Alist Acons ANil*))
nXthree	(*aXthree* (*Alist Acons ANil*))

3.4.4 *Generalize*

Lastly, we may construct a series of theory presentations whose structure is too specific. It may be that we did not analyze the structure of our presentation well enough to identify all the common subcomponents and ended up with a hierarchy that does not represent the commonalities between theory presentations that are related semantically. As an example, using the theory presentations *Proplogic* and *Booleans*, we might construct the theory presentation *PropBools* as shown earlier. But perhaps we want to extend *PropBools* with some proofs about the behavior of $\wedge$ and $\vee$, for instance DeMorgan's laws, calling this new theory presentation PB'. We might also wish to introduce a set of statements capturing resolution (PB''). Ideally, we would like a structure that captures them both as extensions of a common theory presentation, as in Figure 3.7.

However, if we constructed the three theory presentations via the following sequence

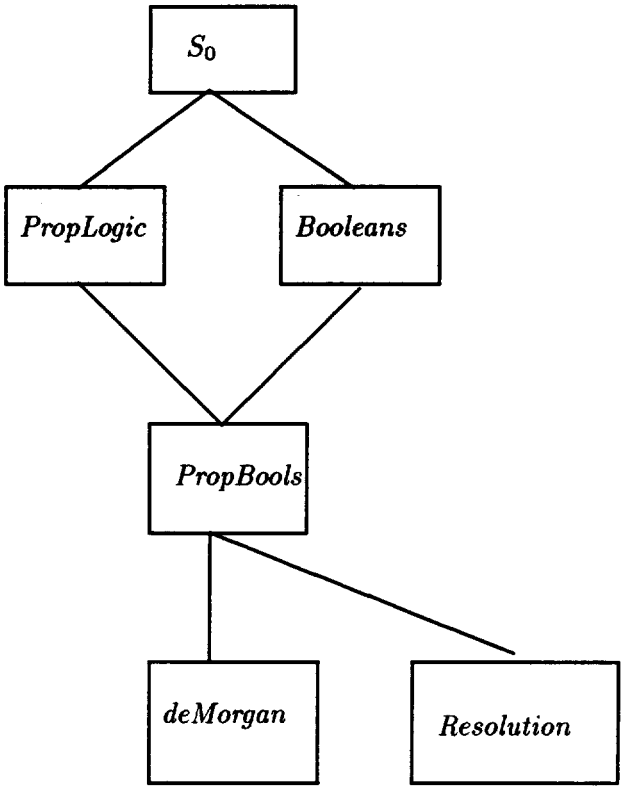

Figure 3.7: Ideal Presentation Base Structure

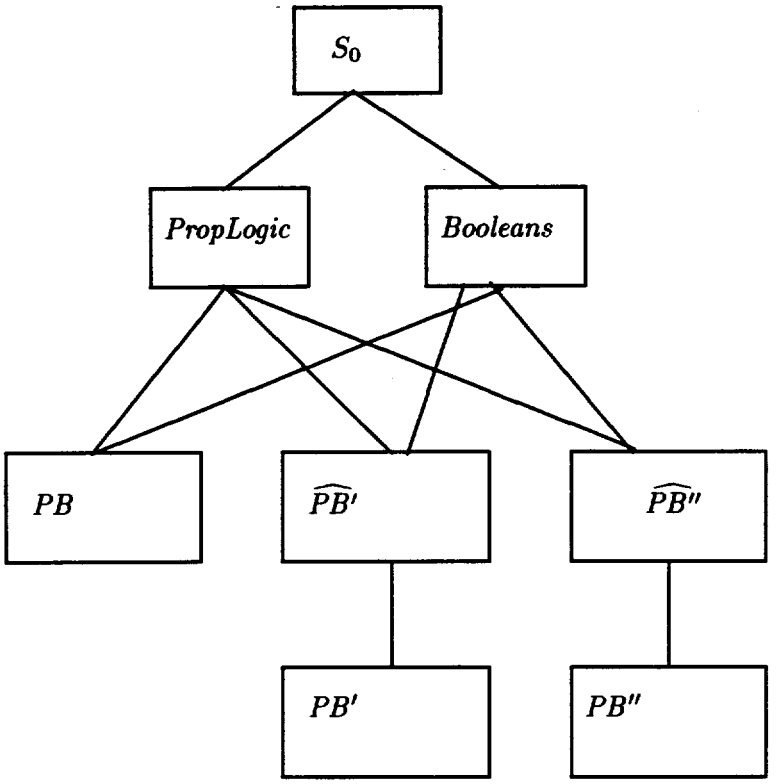

Figure 3.8: Presentation Base with Multiple Extensions

of presentation commands:

$$
\begin{aligned}
PB &\equiv \textbf{instantiate}\, Proplogic Booleans(\cdots)(\cdots) \\
\widehat{PB'} &\equiv \textbf{instantiate}\, Proplogic Booleans(\cdots)(\cdots) \\
PB' &\equiv \textbf{extend}\widehat{PB'}(DeMorgan's Laws) \\
\widehat{PB''} &\equiv \textbf{instantiate}\, Proplogic Booleans(\cdots)(\cdots) \\
PB'' &\equiv \textbf{extend}\widehat{PB''}(Resolutionstatements)
\end{aligned}
$$

We would get a presentation base that looks like the one shown in Figure 3.8.

In this second case, there are three distinct identifiers ($PB, \widehat{PB}', \widehat{PB}''$) that all denote the same semantic element. It is now much harder to see that PB' and PB'' are both extensions of the same central theory.

Having constructed this somewhat non-ideal structure, is there any way to recapture the more general structure? The *MF* constructor **generalize** allows us to construct a new

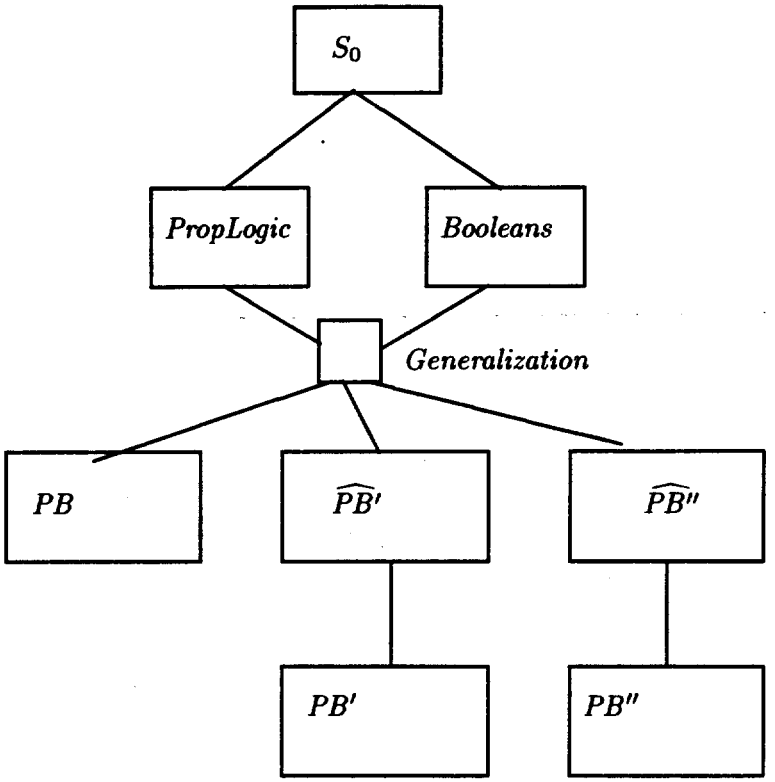

Figure 3.9: Presentation Base after Generalize

theory presentation that is the intersection of the two structural environments.

$$\textit{Generalization} \equiv \textbf{generalize}\ PB\ \widehat{PB'}\ \widehat{PB''}$$

This new theory presentation represents a presentation that is composed of all the components that two (or more) theories have in common. A generalization adds no new information (no new LF statements are added); rather it adds a structural element that clarifies in what way the theories are related. See Figure 3.9.

3.5 Querying the Presentation Base

Besides providing a repository for theory presentations for purposes of construction, a presentation base also serves as a (semi-)permanent library of theory presentations. To support reuse of theory presentations, we need to be able to retrieve theory presentations

easily. We have already seen that we can retrieve a theory presentation by accessing its identifier. However, there are several ways in which a theory presentation may be described:

- By its theory presentation identifier.
- By the way it was constructed.
- By the features it includes.

These presentation expressions are interpreted in the structure defined by the presentation base. When interpreted as a query, a presentation expression really asks the question: are there any theory presentations that have been constructed (named) that match this description. Because the presentation base primarily records extensions, the answer to a query is really a set: the set of all theory presentations that are consistent extensions of the given presentation expression.

In our presentation base constructed in this chapter (see Figure 3.3), the presentation expression **combine** *ThArity ThList* matches the set: { *ThGr0, ThGroupoid, ThSGr0, ThSemigroup, ThMonoid0, ThMonoid* }. In the presentation base, this set has a *LUB*: the theory presentation *ThGr0*. This theory presentation is *representative* of the set: all other members of the set are consistent extensions of *ThGr0*. This same set can be described by the features it includes. The expression

$$[(Arity, c_{15}), (ZeroAr, c_{16}), (SuccAr, c_{17}), (list, c_{18}), (Nil, c_{19}), (Cons, c_{20})$$

lists all the features and their values as determined by the Level 1 identifier and asks for the set of theory presentations in which all these features are defined. (Not all the features need be listed.)

When constructing new theory presentations, the structural environment can be specified using any presentation expression. So in a new theory presentation, if a dependency requires both a list theory and an arity theory, then the presentation command

$$N \equiv \textbf{extend } (\textbf{combine} ThList\ ThArity)(\cdots)$$

will evaluate the expression (**combine** *ThList ThArity*) returning the representative member (if it exists) of the set of the consistent extensions of the theory denoted by the expression **combine** *ThList ThArity* if it exists. In our presentation base, this would be the set *ThGro, ThGroupoid, ...*, with the representative member being *ThGr0.* This would position N as an extension of *ThGr0.* While it would be feasible to construct the combination "on the fly". as it were, *MF* does not do so. Because of the persistent nature and the requirement that constructed objects have names, unnamed "local" objects are not defined.

Chapter 4

MF Definition and Semantics

This chapter presents the formalization of *MF*. *MF* has three levels of description. At the base *MF* relies on a logical framework (an *LF*). On top of the *LF*, *MF* describes a space of encodings of *LF* statements (Level 1). Built on this layer, *MF* defines a space of structured sequences, allowing sequences of *LF* statements to be constructed by combining other sequences (Level 2). At the top, *MF* distinguishes between sequences of *LF* statements that are describable from those sequences that have actually been specified, and checked in the *LF* (Level 3). There are three classes of judgements used in the description of *MF*, corresponding to the three levels. The *MSΣterm* judgement identifies Level 1 statements. The *MFsort* judgements identify Level 2 statements. The *MFsort* judgements are divided into four forms, reflecting the specifics of construction. The *MFtheory* judgement identifies Level 3 statements. Figure 4.1 shows an abbreviated sequence of theory presentation definitions: Theory T_1 introduces an extension of the empty theory, S_0, with features ι_1 and ι_2, mapping to *LF* identifiers and values c_1, x_1 and c_2, x_2 respectively. This results in the Level 2 environment being extended with the value for T_1 (defined in Section 4.2) and the Level 1 environment extended with the mapping between the *LF* identifiers and the *LF* values.

In this chapter we define the syntax and semantics of Levels 1, 2, and 3, and show the relationship between the levels.

The full set of judgement forms of *MF* are:

1. *MSΣterm* These terms are the Level 1 terms that directly represent *LF* statements, or applications of *LF* statements. *MSΣterms* mediate between the *LF* and Level 2

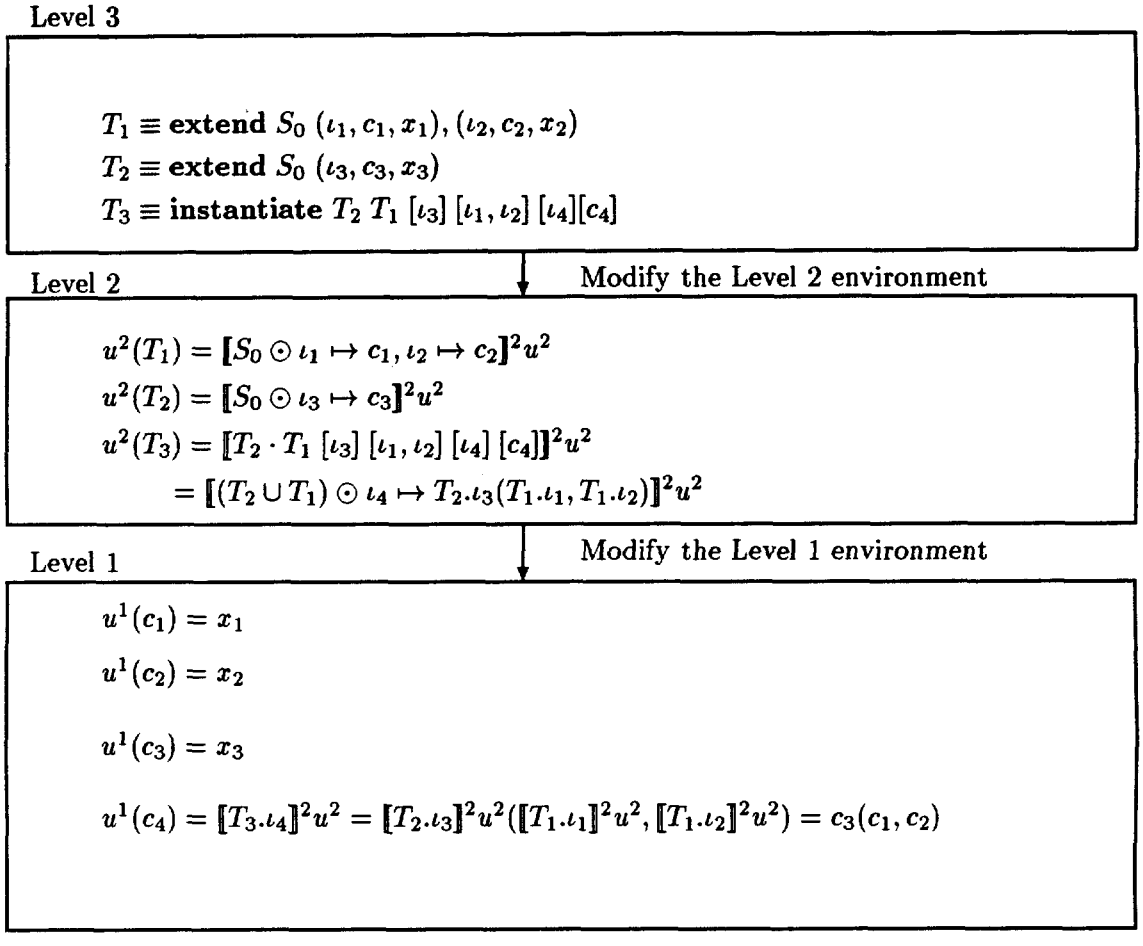

Figure 4.1: The Three Levels of *MF*

statements and are described in Section 4.1.

2. *MFsort* These terms define the structured space of theory presentations (Level 2). The four forms of *MFsorts* discriminate among constant theories, theories adding only structural information, and theories adding LF statement information (either directly through extension or indirectly through instantiation). These four forms are described in Section 4.2. The Level 2 terms represent the *algebra* of operations over an abstract value space of constructible Level 1 terms.

3. *MFtheory* These terms represent theories that have been constructed according to the formation rules of Level 2 and been bound to an identifier. An MF theory presentation distinguishes between theories that have been constructed (i.e. given a name bound to a construction) as opposed to those that are constructible (defined by the *MFsorts*). Currently, no distinction is maintained based on the type-checking status of a theory: an MF theory describes both checked and unchecked theory presentations. Level 3 is composed of MF theories and is described in Section 4.3.

We use the phrases $MS\Sigma$term and Level 1 term, *MFsort* and Level 2 term, and *MFTheory* and Level 3 term interchangeably.

MF judgements are specified using the operator ::. For example, to express that A is an $MS\Sigma$term we write A:: $MS\Sigma$ term. We also use :: as the MF type judgement within a level (for example, $A :: \theta \times \theta$ in Level 2). The notation : is used to identify type judgements in the LF and in the meta-notation used in describing the semantics of MF. The notation $\vdash_{LF}$ is used as a meta-judgement, stating that an LF type judgement is valid. The notation $\vdash_{MF}$ is used to indicate that an MF judgement is valid.

MF has at its base a set of statements in the logical framework that represents all possible statements. In one form or another, all three levels need to know about these underlying LF statements. While each level (in introducing its own language and semantics) might introduce a new identifier to refer to the same LF value, we allow the same meta-variable to refer to the LF identifier: c_i.

(1)	$(c_i, (Context, LFval, LFtyp))$	An explicit LF define.
(2)	$(c_i, (c_{j_1}(c_{j_2}, \cdots, c_{j_n})))$	An LF define constructed via an MF instantiation using LF application. Each c_{j_i} is assumed to have been introduced in a preceding definition in the context and c_i is assumed to be fresh in the LF context.

Figure 4.2: Abstract Values

4.1 $MS\Sigma$ Terms - Level 1

LF statements are the set of legal propositions, types, or theorems valid in the LF. These statements may be bound to an LF identifier for later use in other LF statements, as discussed in Section 3.1. The only statements that MF may refer to are *named* LF statements. We assume an infinite set $[\tau]$ of identifier symbols, ranged over by c_i and type symbols $\hat{c_i}$, where each c_i maps to an encoding of a statement in the logical framework and $\hat{c_i}$ is its type in MF. Each c_i represents a specific statement in the logical framework, and is used both as the MF identifier and as the LF identifier for the associated statement. The set $[AbstrVals]$ is the set of legal (unnamed) LF statements. There are two forms of supported LF statements: those introduced as a proposition, type, or theorem and those introduced by LF application (β reduction). The association of the LF statement and its identifier is represented by a pair, the LF identifier and the value to be bound to the LF identifier: $[\tau] \times [AbstrVals]$, with the meanings shown in Figure 4.2.

These two forms of abstract values represent the judgements in LF:

$$(1) \quad LF \text{ defines} \qquad c_i \quad \Gamma \quad = \quad x_i \quad : \quad a_i$$

The identifier c_i identifies the LF value x_i that, given witnesses for the context Γ, has the LF type a_i.

$$(2) \quad LF \text{ applications} \qquad c_i \quad = \quad c_{j_1}(c_{j_2}, \cdots, c_{j_n})$$

The identifier c_i identifies the LF value specified by the application. The LF type is derived from the types of the $c_{j_1}(c_{j_2}, \cdots, c_{j_n})$ according to the LF typing rules.

The sets [*defines*] and [*apps*] refer to the two forms of [*AbstrVals*] when a distinction must be made.

4.1.1 Syntax

*MSΣ*terms consist of identifiers in the set $[\tau]$ and applications of identifiers in the set $[\tau]$, providing the interface between *MF* and the logical framework. For each identifier $c \in [\tau]$ we have an *MF* type symbol $\hat{c} \in [\hat{\tau}]$, its corresponding type.

MSΣtermintro
$$\frac{}{c :: MS\Sigma term}\ c \in [\tau] \qquad (MS\Sigma term1)$$

MSΣapplicintro
$$\frac{c :: MS\Sigma term}{c_1(c_2, \cdots, c_n) :: MS\Sigma term}\ 1 \leq i \leq n \qquad (MS\Sigma term2)$$

MSΣtermtype
$$\frac{c :: MS\Sigma term}{c :: \hat{c}} \qquad (MS\Sigma term3)$$

MSΣapplictype
$$\frac{c_i :: MS\Sigma \text{ term}}{c_1(c_2, \cdots, c_n) :: \hat{c_1}(\hat{c_2}, \cdots, \hat{c_n})}\ 1 \leq i \leq n \qquad (MS\Sigma term4)$$

4.1.2 Semantics

The language of Level 1 has no assignment or means of affecting an environment: it's just the space of constructible values. The environment is specified in Level 3, and within Level 1 the type environment and valuation environment are assumed to be predefined.

The set of *LF* statements, $[\![\hat{\tau}]\!]$, is represented either as an introduced define or as an application: $\Sigma([\tau] \times [AbstrVals]) + [apps]$ [1], associating the *LF* identifiers to the unnamed *LF* statements or performing an application. The set $[\tau]^1_\pi$ is the set of *MSΣ* terms constructible via the above rules. We let π^1 be the type environment, where $c_i \in dom(\pi^1)$ for all c_i. The set $[\![\pi^1]\!]$ is the set of all valuation environments u^1, such that $u^1(c_i) \in [\![\pi^1(c_i)]\!]$. For Level 1 types, the meaning of an identifier is its associated encoding of an *LF* statement. The valuation environment has the type: $u^1 : [\tau] \rightarrow [AbstrVals]$. We shorten π^1 to π when the level is obvious, as in $[\tau]^1_\pi$ instead of $[\tau]^1_{\pi^1}$.

[1] The notation Σ represents the set of the sums. So $\Sigma([\tau] \times [AbstrVals]) + [apps]$ is the set all *LF* statements, each statement being either an introduced define or an application.

The semantic function has the type:

$$[\![\cdot]\!]^1 : [\tau]^1_\pi \rightarrow ([\![\pi^1]\!] \rightarrow [\![\hat{\tau}]\!])$$

The definition of the semantic function is:

$$[\![c_i]\!]^1\ u^1 = (c_i, u^1(c_i))$$

$$[\![c_1(c_2, \cdots, c_n)]\!]^1\ u^1 = c_1(c_2, \cdots, c_n)$$

Note that in the equations above, the use of an apparent application is syntactic: no functional application is performed. This is a textual term that will be evaluated by the underlying *LF*, assuming that it understands β reduction. Each c_i in an application is assumed to already be defined in the *LF* environment. Level 3 is responsible for maintaining such an environment.

We extend this semantic evaluation to sequences of $MS\Sigma$ terms:

$$[\![c_1, \cdots, c_n]\!]^1\ u^1 = [\![c_1]\!]^1 u^1\ ,\ \cdots\ ,\ [\![c_n]\!]^1 u^1$$

We will use only the abstract syntax pairs defined above in Figure 4.2, assuming a translator of these pairs to the concrete syntax of a particular logical framework, so that $[\![c_i]\!]^1 u^1$ has meaning if the corresponding *LF* statement has meaning in a suitable context in the *LF*.

4.1.3 Properties

The soundness of *MF* is based on soundness in the underlying *LF*. The intent is that *MF* manages and structures correct sequences of *LF* statements. The *LF* acts as the model for *MF*. Every *MF* Level 1 term denotes a statement in the *LF*, identified by an *LF* identifier c or application of *LF* identifiers.

A sequence of *LF* statements $c_1, \cdots, c_n$ represents the following judgement:

$$[\![c_1 \cdots c_{n-1}]\!]^1 u^1 \vdash_{LF} [\![c_n]\!]^1 u^1$$

meaning that in the *LF* context defined by the sequence $c_1, \cdots, c_{n-1}$, the type judgement determined by $[\![c_n]\!]^1 u^1$ is valid in the *LF*. When the above relationship is true for the *MF*

sequence $c_1, \cdots, c_n$ we say that $c_1, \cdots, c_n$ is valid, written $\vdash_{MF} c_1, \cdots, c_n$ *valid.* A sequence of $MS\Sigma$*terms* is sound if the above relation is true.

We define an equivalence class ($=_{LF}$) on *MF* sequences based on their relation in the *LF*. For any sequence of propositions P, context Γ in an *LF*, and valid *MF* sequences M, N,

$$M =_{LF} N \text{ if:}$$
$$(\Gamma, [\![M]\!]u^1 \vdash_{LF} P) \Leftrightarrow (\Gamma, [\![N]\!]u^1 \vdash_{LF} P)$$

This equivalence allows us to identify sequences that differ only in non-essential order. We will make use of this in Levels 2 and 3 to identify when two *LF* contexts are equivalent.

4.2 *MFsorts* - Level 2

The *MFsorts* describe all possible theories that can be derived from the base of encoded *LF* statements using the structured theory operations. This section defines the language of Level 2: the structured theory operations. The term *MFsort* is used to refer to constructible theories.

4.2.1 Syntax

The Level 1 terms are defined by the encoded *LF* statements. In Level 2, these same terms are treated as constants. Level 2 terms describe *sequences* of *LF* statements. Each *LF* statement in a sequence is introduced via an identifier called a feature identifier. Features support three distinct roles in *MF*:

- They capture the aggregate nature of a sequence. In this sense they act like labels of a product type. The product type is composed of all the individual features (fields).
- They record a history of applications. One of the structured operations of theory presentations is instantiation or theory application, which supports applying the statements of one theory to statements of another. Level 2 records this instantiation through application of the indicated features, providing a record of which features are used to discharge which assumptions.

- Lastly, features provide a map from the *MF* structural view to an *LF* identifier (or application of *LF* identifiers), tying the Level 2 theory organization to the underlying *LF*.

We assume a set $[\omega]$ of identifiers for feature identifiers. The type categories for our language are defined as:

$$\begin{array}{lcll} \tau & ::= & \hat{c} \mid \tau_1(\tau_2, \cdots, \tau_n) & LF \text{ statements and applications} \\ \gamma & ::= & \iota_1 \mapsto \tau_1 \& \cdots \& \iota_n \mapsto \tau_n & \text{Sequences of } LF \text{ statements } \iota_i \in [\omega] \\ \theta & ::= & \alpha \mid \theta + \gamma \mid \theta_1 \sqcup \theta_2 | \theta_1 \sqcap \theta_2 & \text{Structured elements} \end{array}$$

The type category τ represents the types of *LF* statements, and specify the Level 1 terms. Level 1 terms are referred to by directly specifying the Level 1 identifier of an encoding of a *LF* statement, or by specifying an application. The type category γ represents the tupling of arbitrary *LF* statements, where each statement is identified by a feature identifier. Tupling is the means by which *LF* statements are associated with a theory and support theory instantiation. Each γ type specifies the feature identifiers of an extension to a theory presentation, mapping each feature identifier to a τ type.

The type category θ represents the phrase types of the language, describing the space of constructible theories (theory presentations). These phrase types define the terms of Level 2, the *MFsorts*. The four forms of θ types represent the four methods used in structuring theories. The phrase type α is the type of the empty theory. The phrase type $\theta + \gamma$ is the type of an extension theory. Any theory may be extended by a sequence (an element of γ type), resulting in a new theory. Theories may be composed of existing theories, adding no new *LF* information. The phrase types $\theta_1 \sqcup \theta_2$ and $\theta_1 \sqcap \theta_2$ capture these composite theories. $\theta_1 \sqcup \theta_2$ is the type of a theory where two theories are combined: a theory of this type has all the sequences denoted by either of the subtheories of type θ_1, θ_2. $\theta_1 \sqcap \theta_2$ is the type of a theory where two theories share one or more sequences: a theory of this type has the sequences that are present in both of the subtheories. Theories may also be constructed by adding new sequences of statements (tuples) through instantiation. The phrase type $\theta + \gamma$ captures these theories as well, where γ is the type of the extension (a sequence), constructed through instantiation.

We use x_i for terms of τ types, ι_i for identifiers of tuple components (feature identifiers), F_i for terms of γ types (the sequences), and A, B for terms of θ types.

$\tau - introI$ $\quad \dfrac{c :: \hat{c}}{c :: \hat{c}}$ Level 2 constants $\quad (LFstmts1)$

$\tau - introII$ $\quad \dfrac{x_i :: \tau_i \; x_{j_1} :: \tau_{j_1}, \cdots, x_{j_n} :: \tau_{j_n}}{x_i(x_{j_1}, \cdots, x_{j_n}) :: \tau_i(\tau_{j_1}, \cdots, \tau_{j_n})}$ $\quad (LFstmts2)$

Feature identifiers identify the sequence components of an *MFsort*. The following three rules are feature tuple introduction and projection.

$\gamma - intro$ $\quad \dfrac{x_i :: \tau_i \;\; i = 1, \cdots, n, \iota_i \in [\omega]fresh}{\iota_1 \mapsto x_1, \cdots, \iota_n \mapsto x_n :: \iota_1 \mapsto \tau_1 \& \cdots \& \iota_n \mapsto \tau_n}$ $\quad (Features1)$

$\gamma - elimI$ $\quad \dfrac{F :: \gamma \;\; \gamma \equiv \iota_1 \mapsto \tau_1 \& \cdots \& \iota_n \mapsto \tau_n}{F.\iota_i :: \tau_i} \; i = 1, \cdots, n$ $\quad (Features2)$

Multiple projections are equivalent to the sequence of individual projections:

$$F.(\iota_1, \cdots, \iota_m) \equiv \iota_1 \mapsto F.\iota_1, \cdots, \iota_m \mapsto F.\iota_m$$

The requirement that the feature identifiers ι be fresh assures that every new statement introduced in an extension is unique in *MF*. Because of sequentiality requirements, not all projections are semantically sound in the *LF* even when they are projected from semantically sound terms. Soundness is handled by restrictions on projections and only guaranteed through checking in the *LF*.

The least theory is the empty theory (the empty tuple or sequence). (This can be augmented by an initial environment and phrase type environment allowing more than one of minimal theories.) [2]

The *MF* Level 2 operators, $\cap$, $\cup$, $\odot$, and $\cdot$ support operations on sequences (not sets) and are left associative and of equal priority. Parentheses may be used to group terms. The sequence operator $\cup$ combines sequences. The operator $\cap$ combines only the common sequences of two (or more) *MFsorts*. The operator $\odot$ adds a new sequence to an existing sequence. The operator $\cdot$ adds a new sequence by instantiating (applying) elements of a sequence with (to) witnesses from another sequence.

[2] In the semantics section, the initial environment could in fact be augmented to allow an arbitrary collection of minimal theories, that would have no structural component, and therefore would not be susceptible to reduction, and would serve the same role as the empty theory (the constant theories).

$\alpha - intro$ $\quad \dfrac{}{S_0 :: \alpha}$ The empty theory $\quad (MFsorts1)$

$\sqcup - intro$ $\quad \dfrac{A :: \theta_1 \quad B :: \theta_2}{A \cup B :: \theta_1 \sqcup \theta_2}$ $\quad (MFsorts2)$

$\sqcap - intro$ $\quad \dfrac{A :: \theta_1 \quad B :: \theta_2}{A \cap B :: \theta_1 \sqcap \theta_2}$ $\quad (MFsorts3)$

$\odot - intro$ $\quad \dfrac{A :: \theta \quad F :: \gamma}{A \odot F :: \theta + \gamma}$ $\quad (MFsorts4)$

We introduce three type coercions, allowing all types to be treated as extensions types. (The symbol $\vdash$ indicates a coercion; $\theta_i \vdash \gamma_j$ states that the type θ_i is coerced to the type γ_j.)

CI $\quad \theta + \iota_1 \mapsto \tau_1 \& \cdots \& \iota_n \mapsto \tau_n \vdash \iota_1 \mapsto \tau_1 \& \cdots \& \iota_n \mapsto \tau_n$ $\quad$ (Coerce1)

CII $\quad \iota_1 \mapsto \tau_1 \& \cdots \& \iota_n \mapsto \tau_n \vdash \iota'_{i_1} \mapsto \tau_{i_1} \& \cdots \& \iota'_{i_m} \mapsto \tau_{i_m}$ where $\iota'_{i_1}, \cdots, \iota'_{i_m} \subseteq \iota_1, \cdots, \iota_m$ $\quad$ (Coerce2)

CIII $\quad \theta_1 \sqcup \theta_2 \vdash \iota_1 \mapsto \tau_1 \& \cdots \& \iota_n \mapsto \tau_n$ If $\theta_1 \vdash \iota_1 \mapsto \tau_1 \& \cdots \& \iota_m \mapsto \tau_m$ and $\theta_2 \vdash \iota_{m+1} \mapsto \tau_{m+1} \& \cdots \& \iota_n \mapsto \tau_n$ $\quad$ (Coerce3)

The conversions corresponding to these coercions are defined in Section 4.2.6.

Coercion I allows us to drop structural information outside of an extension. Coercion II allows us to drop features. Coercion III allows us to view a structured theory presentation as an unstructured flat sequence. Because coercions allow us to drop information, they can only be used where the information dropped is preserved elsewhere in the context.

There is no coercion from a $\cap$ type: theory intersection can be represented by the union of theory presentations. This will be discussed more fully in Section 4.2.5. For now we assume that $A \cap B =_{def} A_1 \cup \cdots \cup A_n$, for some sequence of *MFsorts* A_i.

There is one remaining theory constructor: theory instantiation. Theory instantiation is defined using the property of β reduction in the underlying *LF*. The features of a theory (A) may be specialized using features of another theory (B), by specifying the features of A to be instantiated, the features of B (to be used as witnesses), and new feature identifiers for the newly constructed extension.

Because of order dependencies in the LF, an extension requires a specific order. Projections and coercions (which remove components and thus violate the order determined by the extension) are only allowed during instantiations, which preserve the original information and order in the context.

$$\cdot - intro \quad \frac{\begin{array}{c} A :: \theta_A + \gamma_A \;\; B :: \theta_B + \gamma_B \\ \gamma_A \equiv \iota_{A_1} \mapsto \tau_{A_1} \& \cdots \& \iota_{A_n} \mapsto \tau_{A_n} \\ \gamma_B \equiv \iota_{B_1} \mapsto \tau_{B_1} \& \cdots \& \iota_{B_m} \mapsto \tau_{B_m} \end{array}}{\begin{array}{l} A \cdot B[\iota_{A_{j_1}}, \cdots, \iota_{A_{j_n}}]\,[\iota_{B_{i_1}}, \cdots, \iota_{B_{i_m}}]\,[\iota_{C_{j_1}}, \cdots, \iota_{C_{j_n}}] :: \\ ((\theta_A + \gamma_A) \sqcup (\theta_B + \gamma_B)) \\ \quad + \iota_{C_{j_1}} \mapsto \tau_{A_{j_1}}(\tau_{B_{i_1}}, \cdots, \tau_{B_{i_m}}) \& \cdots \& \\ \quad \iota_{C_{j_n}} \mapsto \tau_{A_{j_n}}(\tau_{B_{i_1}}, \cdots, \tau_{B_{i_m}}) \end{array}} \qquad (MFinst1)$$

The term for the instantiation is:

$$\begin{array}{l} A \cdot B\,[\iota_{A_{j_1}}, \cdots, \iota_{A_{j_n}}]\,[\iota_{B_{i_1}}, \cdots, \iota_{B_{i_m}}]\,[\iota_{C_{j_1}}, \cdots, \iota_{C_{j_n}}] =_{def} \\ (A \cup B) \\ \quad \odot \iota_{C_{j_1}} \mapsto A.\iota_{A_{j_1}}(B.(\iota_{B_{i_1}}, \cdots, \iota_{B_{i_m}})) \& \cdots \& \\ \quad \iota_{C_{j_n}} \mapsto A.\iota_{A_{j_n}}(B.(\iota_{B_{i_1}}, \cdots, \iota_{B_{i_m}})) \end{array}$$

Note that instantiate introduces an application of MF features values in one theory to feature values of another, each feature given a new identifier. This is the rule that lets us build new theories dependent on witnesses provided by other theories. Recall the group example from Chapter 3. While this rule is the most complicated, it is the rule that must keep track of the most structural information.

The type rules above describe the structured space of *MFsorts*.

4.2.2 Formation Trees

We use labeled binary trees to represent *MFsorts*, called *formation trees*. These formation trees give a concrete representation for the *MFsorts*. We first introduce formation trees capturing the structure of *MFsorts*. Then we show that we can construct a reduced formation tree from any formation tree (flattening the structure to a sequence of extensions) that is consistent with the formation tree. Lastly we define a minimal formation tree that

maintains consistency with the formation tree and removes any duplication of context elements introduced by the sequence operators. Formation trees allow us to unambiguously define equality of *MFsorts*, as well as the ordering requirement on the extensions to *MFsorts*, which is used to define a context in the *LF*.

We offer some basic definitions to support formation trees.

Definition 1 (Labeled Trees) *A* **labeled tree** *is a tree T with a function associating some object (the label) with every node.*

Definition 2 (Paths in a Binary Tree) *A* **path** *in a binary tree is a binary sequence where*

1. *The root of the tree has the path $\emptyset$.*
2. *If σ is a path to a node in the tree, then $\sigma|0$ is the path of the root of the left sub-tree and $\sigma|1$ is the path of the root of the right sub-tree of the node at σ. The symbol $|$ denotes concatenation in the path.*

Definition 3 (Formation Tree) *A* **formation tree** *is a finite binary labeled tree T whose nodes are labeled with terms of MFsorts and γ type, satisfying the following conditions:*

1. *The leaves are labeled with terms of γ type or terms of α type.*
2. *If a node at path σ is labeled with an MFsort $A \cup B$ or $A \cap B$, then its immediate successors at paths $\sigma|0$ and $\sigma|1$ are labeled with $A :: \theta_A$ and $B :: \theta_B$ respectively.*
3. *If a node at path σ is labeled with an MFsort $A \odot F$, then its immediate successors at paths $\sigma|0$, $\sigma|1$ are labeled $A :: \theta_A$ and $F :: \gamma$ respectively.*

Figure 4.3 shows the formation tree for the *MFsort* $(A \cup B) \cup (B \cup C)$, where $A = S_0 \odot F_A$, $B = S_0 \odot F_B$, and $C = S_0 \odot F_C$.

We denote the formation tree for a phrase S of type θ by FT(S). We also identify an interior label with its operator, rather than the full term, so a label of $A \cup B$ would be identified with the label $\cup$. For the remainder of this discussion, we assume that formation trees do not include the label $\cap$, because in Section 4.2.3 we define $\cap$ in terms of $\cup$. We

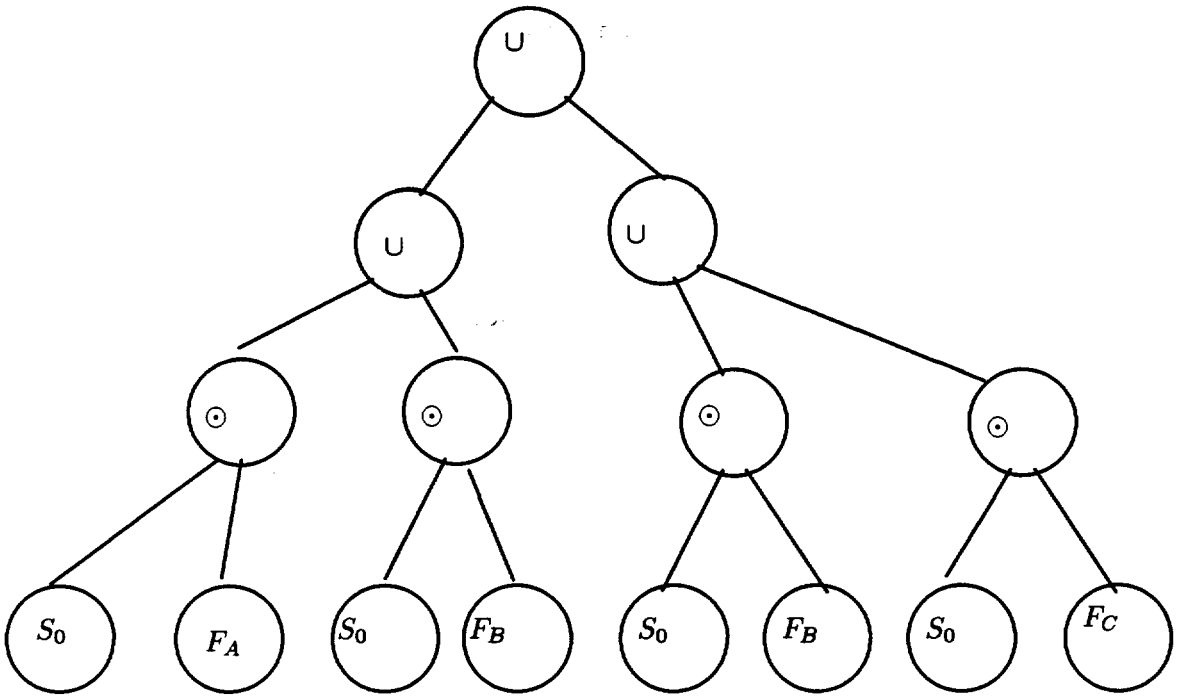

Figure 4.3: Formation Tree for $(A \cup B) \cup (B \cup C))$

use the path symbol σ to refer to both a node in a formation tree and the formation tree rooted at the node σ.

There is a unique formation tree associated with a given *MFsort*. The proof is by induction on the structure of the space of *MFsort* terms. For an expression of the form $A \cup B$, the formation tree will be the tree with the node at path $\emptyset$ labeled with $\cup$, the node at the path $\emptyset|0$ will be the root of the formation tree for A, and the node at the path $\emptyset|1$ will be the root of the formation tree for B. For a formation tree of the form $A \odot F$, the formation tree will be the tree with the node at path $\emptyset$ labeled with $\odot$, the subtree rooted at the path $\emptyset|0$ will be the formation tree for A, and the node at path $\emptyset|1$ will be the node labeled with F.

We define the *support* of an *MFsort*:

Definition 4 (Support of an *MFsort*) *The* **support** *of an MFsort is the set of terms of γ type and α type that occur as labels on the leaves of the associated formation tree. The support can be uniquely represented by the set of binary paths to leaves given in Definition 2.*

For the formation tree given in Figure 4.3, the support given as a set of paths would be:

$$\{\emptyset|0|0|0, \emptyset|0|0|1, \emptyset|0|1|0, \emptyset|0|1|1, \emptyset|1|0|0, \emptyset|1|0|1, \emptyset|1|1|0, \emptyset|1|1|1\}$$

corresponding to the labels:

$$S_0, F_A, S_0, F_B, S_0, F_B, S_0, F_C$$

The extension *MFsorts* requires a particular order. In the *MFsort* $A \odot \iota_1 \mapsto x_1, \cdots, \iota_n \mapsto x_n$, the components of the extension (x_i) define LF statements. These LF statements may refer to other LF statements that are defined by the *MFsort* A. Extensions therefore induce an order dependency on the formation tree.

Definition 5 (Order Dependency) *An* **order dependency** *in a formation tree is defined as follows: If a node at path σ has a label of $\odot$, then the node at path $\sigma|1$ with the label of $F :: \gamma$ has an order dependency on the support of the formation tree rooted at the path $\sigma|0$. If the support of the formation tree rooted at the path $\sigma|0$ is the set of paths $\{\sigma_1, \cdots, \sigma_{i-1}\}$, then this order dependency is written as $\{\sigma_1 < \sigma|1, \sigma_2 < \sigma|1, \cdots, \sigma_{i-1} < \sigma|1\}$ or we can write $\{\sigma_1, \cdots, \sigma_{i-1}\} < \sigma|1$ as a notational shorthand.*

The formation tree for an *MFsort* determines an LF context, by specifying the order of LF statements. If a node at path $\sigma|n$ has an order dependency on the node at path $\sigma|m$, for paths n and m, then the LF meaning denoted by the label of the node at path $\sigma|m$ must be present in the LF context in order for the label of the node at path $\sigma|n$ to have an LF meaning. This order requirement will be discussed further in Section 4.2.4.

The support of a node σ in a formation tree T_i can be constructed based on the structure of the formation tree.

$$\begin{aligned} support(\sigma) = & \\ & label(\sigma) = S_0 :: \alpha \;\Rightarrow\; \{\sigma\} \\ & label(\sigma) = \cup \;\Rightarrow\; support(\sigma|0) \cup support(\sigma|1) \\ & label(\sigma) = \odot \;\Rightarrow\; support(\sigma|0) \cup \{\sigma|1\} \end{aligned}$$

The order dependency is defined similarly. We define $\sigma > \{\sigma_1, \sigma_2, \cdots, \sigma_m\}$ to be the partial order $\{\sigma > \sigma_1, \cdots, \sigma > \sigma_m\}$. If the set S has the form $\{\sigma_1 > \sigma_2, \cdots, \sigma_n > \sigma_{n+1}\}$, then $\sigma > S$ is defined to be the partial order $\{\sigma > \sigma_1, \cdots, \sigma > \sigma_n\} \cup S$.

$$order(\sigma) =$$

$$\begin{aligned} label(\sigma) = S_0 :: \alpha &\Rightarrow \{\sigma\} \\ label(\sigma) = \cup &\Rightarrow order(\sigma|0) \cup order(\sigma|1) \\ label(\sigma) = \odot &\Rightarrow order(\sigma|0) < \{\sigma|1\} \end{aligned}$$

The order dependency for the formation tree of Figure 4.3 is:

$$\{\emptyset|0|0|0 < \emptyset|0|0|1, \emptyset|0|1|0 < \emptyset|0|1|1, \emptyset|1|0|0 < \emptyset|1|0|1, \emptyset|1|1|0 < \emptyset|1|1|1\}$$

corresponding to the order dependency expressed as labels:

$$\{S_0 < F_A, S_0 < F_B, S_0 < F_B, S_0 < F_C\}$$

The support and order dependency for a formation tree T are referred to as $\langle N_T, <_T \rangle$, where N_T is the support of T and $<_T$ is the order dependency, both expressed using labels rather than paths. Two *MFsorts* are equal if they denote the same semantic object, which is sometimes hard to determine. However, we can define a syntactic equality of *MFsorts* based on formation trees, and show that if two terms are syntactically equal then they denote the same semantic object:

Definition 6 (Equality of Formation Trees) *Two formation trees, T_1 and T_2, are* **equal** *($=_{FT}$) if*

1. *They have the same support ($N_{T_1} = N_{T_2}$).*
2. *They obey the same order dependencies: $(F_i <_{T_1} F_j) \in \langle N_{T_1}, <_{T_1} \rangle \Leftrightarrow (F_i <_{T_2} F_j) \in \langle N_{T_2}, <_{T_2} \rangle$.*

For example, the trees associated with $(A \cup B) \cup (B \cup C)$ and $(A \cup B) \cup C$ have equal formation trees.

Because we have the condition that features can only be introduced into an extension if they are unique [3], in the current definition we cannot have an *MFsort* that has a term of

[3]The uniqueness requirement is preserved by the uniqueness of the feature identifiers. It is possible that two distinct labels will refer to identical *LF* values. However, these will not be judged identical in *MF* because the feature identifiers are unique.

γ-type with two different order dependencies. However, to allow for possible future work (discussed in Chapter 6), the definition of equality includes this condition.

Equality of formation trees is too limiting, as there are many contexts that will correspond to the order dependencies of a particular formation tree. So we define consistency on formation trees:

Definition 7 (Consistent Formation Trees) *Formation tree T_1 is* **consistent** *with formation tree T_2 ($T_1 \sqsubseteq T_2$) if:*

- $N_{T_2} \subseteq N_{T_1}$ *and*
- $F_i <_{T_2} F_j \in T_2 \Rightarrow F_i <_{T_1} F_j \in T_1$

For instance, $FT((A \cup B) \cup C) \sqsubseteq FT(A \cup B)$.

MFsorts and their formation trees do not define a unique *LF* context, as mentioned before. Rather, they define a set of contexts, all of which obey the order dependencies of the formation tree. A particular *LF* context has the order of the context elements fixed. We define a *Reduced Formation Tree RFT* that captures this fixed nature and gives a standard form for formation trees.

Definition 8 (RFT) *A* **reduced formation tree** *(RFT) has the form:*

- *The tree with a single node labeled $A :: \alpha$ is an RFT.*
- *If T is an RFT, and $F :: \gamma$, then the tree having $\emptyset$ labeled with $\odot$, $\emptyset|0$ the root of the tree T, and $\emptyset|1$ the noded labeled $F :: \gamma$ is an RFT.*

The reduced formation tree for our example of Figure 4.3 is the tree associated with $((((S_0 \odot F_A) \odot F_B) \odot F_B) \odot F_C)$, shown in Figure 4.4. A reduced formation tree flattens the structure of an *MFsort* expressing any *MFsort* as a sequence of extensions.

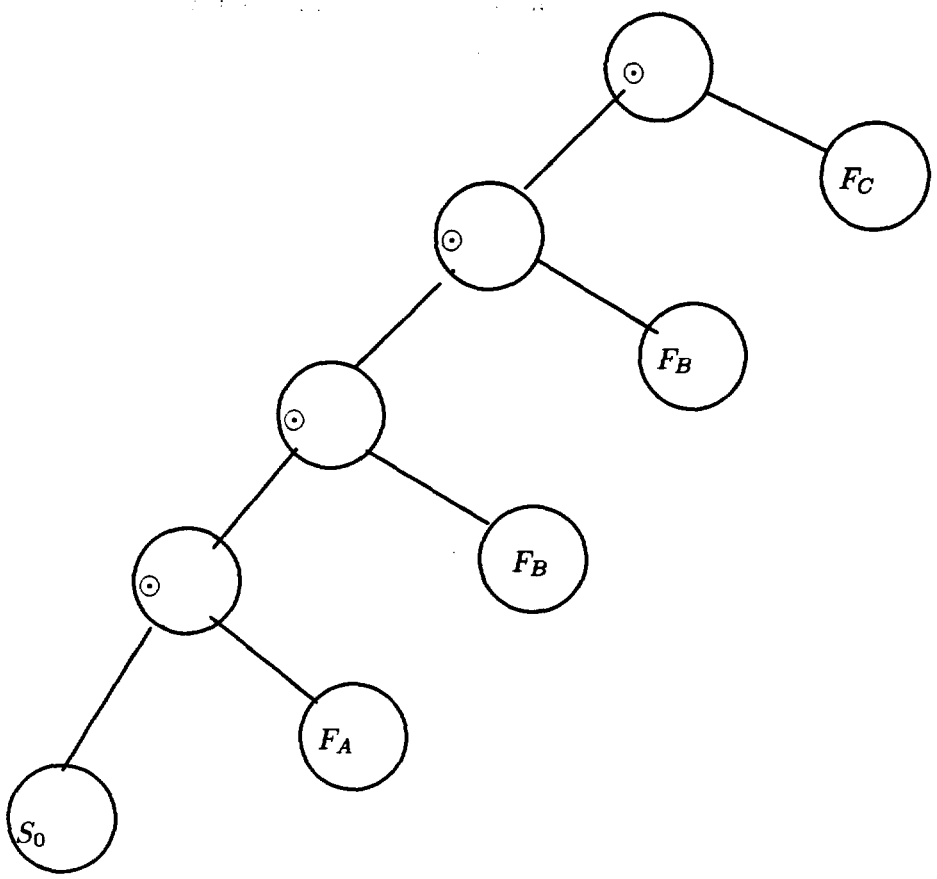

Figure 4.4: Reduced Formation Tree for $(A \cup B) \cup (B \cup C)$

An *RFT* has the properties:

- All the right leaves (all $\sigma|1$ for all paths in the tree) have nodes whose labels are terms of γ types.
- All the internal nodes are labeled with the operator $\odot$.
- There is a unique left hand leaf $\sigma|0$ having the label $A :: \alpha$.[4]

Any formation tree can be converted to an *RFT* using the following rules. Let σ be a node, and $RFT(\sigma)$ be the reduced formation tree of the formation tree rooted at σ.

[4]If *MF* is extended with multiple constant theories, these can all be combined into one large minimal constant theory, because they have no dependencies among them.

1. *RFT-1*: If σ is labeled with a term of type α, the $RFT(\sigma)$ is the node labeled with the same label.

2. *RFT-2*: If σ is labeled with a term of type γ, the $RFT(\sigma)$ is the node labeled with the same label.

3. *RFT-3*: If σ is labeled with a term of $\odot$, the $RFT(\sigma)$ has a root labeled with $\odot$ and a left subtree $RFT(\sigma|0)$ and a right subtree $RFT(\sigma|1)$.

4. *RFT-4*: If σ is labeled with a term $\cup$, the $RFT(\sigma)$ is the $RFT(\sigma|1)$ with the $RFT(\sigma|0)$ replacing the unique left-hand leaf in $RFT(\sigma|1)$.

Proposition 9 *Every formation tree can be reduced to a reduced formation tree using the rules RFT-1 through RFT-4.*

Proof The proof is by induction on the structure of formation trees.

- If the tree is a single node labeled by $A :: \alpha$, then the RFT is just that node. (*RFT-1*)

- The smallest tree including an extension must have the form $S_0 \odot F_i :: \alpha + \gamma_i$. If the root node of the tree is labeled $\odot$, then $\emptyset|1$ is a node with a label of $F_i :: \gamma_i$ (by the definition of formation trees and $\odot$ introduction). If $\emptyset|0$ is labeled with $S_0 :: \alpha$, then $RFT(\emptyset|0)$ is the node with the label $S_0 :: \alpha$ by rule *RFT-1*. $RFT(\emptyset|1)$ is the node with the label $F_i :: \gamma_i$ by rule *RFT-2*. The result is a reduced formation tree.

- We assume that for formation trees of heights less than or equal to $n > 1$, that we can construct a reduced formation tree using rules *RFT1-4*. Let T be a tree of height $n + 1$. T can have the form:

- $T_1 \odot F$. T_1 is a formation trees of height less than or equal to n, and F is a a single node labeled with $F :: \gamma$. We can construct the reduced formation tree for T_1 by the induction hypothesis: $RFT(T_1)$. Then using *RFT-3* we can construct the reduced formation tree for T.

- $T_1 \cup T_2$. T_1 and T_2 are two formation trees of heights less than or equal to n, so they each can be converted to the reduced formation trees $RFT(T_1)$ and $RFT(T_2)$

by the induction hypothesis. $RFT(T_2)$ therefore has a unique leaf with the label $A :: \alpha$. Replacing this unique leaf with the reduced formation tree $RFT(T_1)$ will be a reduced formation tree. using *RFT-4*.

∎

Proposition 10 *For any formation tree T, $RFT(T) \sqsubseteq T$.*

Proof

- $N_T \subseteq N_{RFT(T)}$ by rules *RFT-1* and *RFT-2*, the only rules dealing with the leaves.
- If $F_i > F_j$ in T, then F_i had a label of type γ_i in T occurring at path $\sigma|1$. N_j must be a node with a label having a type of γ_j somewhere in the formation tree $\sigma|0$ (by the construction of formation trees and the definition of the order function) and the node at σ has a label of $\odot$. By *RFT-3*, $RFT(\sigma)$ will also have its root labeled by $\odot$, and the label of $\emptyset|1$ in $RFT(\sigma)$ will be the label of F_i. Similarly, in $RFT(\sigma)$ $\emptyset|0$ will contain the label of F_j. Therefore, $order(RFT(\sigma))$ will generate the order dependency corresponding to $F_i > F_j$.

This proof does not refer to Rule *RFT-4*. The rule *RFT-4* does not generate any order dependencies. Hence if a node σ in the formation tree T_i is labeled with a $\cup$, there will be no dependencies from a node in the subtree $\sigma|1$ on a node in the subtree $\sigma|0$ or vice versa. ∎

Because the formation tree for an *MFsort* expression is unique, we sometimes say A for $FT(A)$ when it is clear from the context if we are looking at a formation tree or an expression.

4.2.3 Canonical Forms

We define a canonical form allowing us to express all the *MFsort* expressions using only the operators $\cup$ and $\odot$.

Definition 11 (Canonical Form) *The* **canonical form** *for an MFsort expression is an expression that uses only* $\cup$ *and* $\odot$ *for constructors. A tree in canonical form has nodes labeled only with* $\cup$, $\odot$, $S_0 :: \alpha$, *or* $F :: \gamma$.

First we define what the intersection of Level 2 terms A and B is.

Definition 12 (Tree Intersection) *A subtree* T_i *in* $FT(A)$ *is in the* **intersection** *of* $FT(A)$ *and* $FT(B)$ *if there is a subtree* T_j *in* $FT(B)$ *such that* $T_i =_{FT} T_j$. *A maximal subtree is the largest tree* T_i *with this property. We define the intersection set of two (or more) formation trees to be the the set of maximal subtrees common to both (or all).*

Definition 13 (Tree Union) *A set of formation trees, all of whose nodes are labeled by* $\cup$ *or* $\odot$ *can be combined into a new formation tree. We can take the* **union** *of a set of formation trees by replacing every pair of formation trees in the set by a new formation tree composed of a root labeled with a* $\cup$ *with* $\emptyset|0$ *identified with one member of the pair, and* $\emptyset|1$ *identified with the other member of the pair. We can continue this until only one formation tree remains in the set.*

There are many formation trees that may result from a tree union, but they are all equal formation trees ($=_{FT}$).

Definition 14 (Tree Subtraction) *We can* **subtract** *a subtree from a formation tree by replacing the subtree with the formation tree labeled by* $S_0 : \alpha$. *If* T_1 *is a subtree of* T, *then* $T - T_1$ *is the formation tree resulting from the subtraction.*

We can simplify the resulting tree: $A \cup S_0 =_{FT} S_0 \cup A -_{FT} A$

The tree resulting from tree subtraction may not be consistent with the original as it may not preserve the order dependencies of the original tree. Tree subtraction must therefore only be used where the dependencies are preserved elsewhere in the tree.

Definition 15 ($A \cap B$) *The intersection of the MFsorts A and B is formed by taking the intersection of their formation trees and then taking the union of all the formation trees in the intersection set.*

$$A \cap B = (A_1 \cup \cdots \cup A_m)$$

where $\{A_1, \cdots, A_m\}$ *is the intersection set of the* $FT(A)$ *and* $FT(B)$

Intuitively, the intersection of two theories denotes all the sequences they have in common. Each sequence is introduced in one theory presentation, so this is equivalent to saying that the intersection of two theories denotes all the theory presentations the two theories have in common.

Proposition 16 *All MFsorts can be expressed using only the theory constructors* $\cup$ *and* $\odot$.

Proof We have already defined instantiation $(\cdot)$ in terms of $\cup$ and $\odot$. So the only case remaining is the constructor $\cap$.

- The smallest tree is the single node labeled with $S_0 :: \alpha$. $S_0 \cap S_0 = S_0$ which has no operators and thus corresponds to an *MFsort* in canonical form.
- We assume that all trees of height less than or equal to $n > 1$ can be expressed in canonical form. Let T be a tree of height $n+1$. If the root of T is labeled by $\cup$ or $\odot$ then this is in canonical form by definition. If the root of T is labeled by $\cap$, then this is a tree intersection. The subtrees of T at paths $\emptyset|0$ and $\emptyset|1$ are in canonical form by the induction hypothesis. Taking the union of the trees in the intersection set of these two subtrees (Definition 15) will yield a formation tree in canonical form.

∎

Proposition 17 *The theory constructors preserve consistency:*

$$\begin{array}{lll} FT(A) & \sqsubseteq & FT(A \cap B) \\ FT(B) & \sqsubseteq & FT(A \cap B) \\ FT(A \cup B) & \sqsubseteq & FT(A) \\ FT(A \cup B) & \sqsubseteq & FT(B) \\ FT(A \odot F) & \sqsubseteq & FT(A) \end{array}$$

Proof

- Case $FT(A) \sqsubseteq FT(A \cap B)$: By Definition 15, $A \cap B = A_1 \cup \cdot \cup A_n$ for all A_i in the intersection set of A and B.

1. $N_{A \cap B} = N_{A_1 \cup \cdots \cup A_n}$. Because each A_i is a subtree of both A and B, the terminal leaves of $A_1 \cup \cdots \cup A_n$ are also in $FT(A)$. So each node in $N_{A \cap B}$ must also be in N_A.

2. If there is an order dependency in $FT(A \cap B)$, then there is a leaf in $FT(A \cap B)$ that generated that dependency. That leaf must therefore occur as a leaf in one of the subtrees in the intersection set of A and B. Hence it is in both A and B, and so $FT(A)$ contains the same order dependency.

- Case $FT(B) \sqsubseteq FT(A \cap B)$: this is the same as the case above.
- Case $FT(A \cup B) \sqsubseteq FT(A)$: Because $FT(A)$ is a subtree of $FT(A \cup B)$ and $\cup$ introduces no additional order dependencies on the nodes in A, the conditions for consistency are satisfied.
- Case $FT(A \cup B) \sqsubseteq FT(B)$: this is the same as the case above.
- Case $FT(A \odot F) \sqsubseteq FT(A)$: The $FT(A)$ is a subtree of $FT(A \odot F)$, and hence the conditions for consistency are satisfied.

∎

4.2.4 Admissible Contexts

We need to characterize what an admissible context is in an LF because LF contexts are used to model theory presentations. Admissible contexts characterize legal contexts and how these contexts are related. We show that formation trees define admissible contexts, and there is an ordering relation on Level 2 terms that is consistent with the ordering relation on admissible contexts. Thus, every theory presentation can be put in the form of an admissible context. In fact, that is what we have to do if we want to discharge assumptions of a theory presentation with LF statements. We let the *MFsort* A be the formation tree associated with A throughout. We also let $[\![\cdot]\!]^2$ be the semantic function for Level 2 terms (defined in Section 4.2.5). In this section on admissible contexts, we assume that we are able to assign some meaning to Level 2 terms, although the actual definition is

not given until the next section. We let u^2 be a Level 2 environment, assigning meanings to pre-defined identifiers. The superscript on the environments and the semantic functions distinguishes among the environments and semantic functions for each Level.

We propose a model suggested by the term model defined by Thomas Streicher [50], and based on the semantics function of Level 2 terms. This model meets the criteria that only provably well-defined contexts, types, and objects are given an interpretation. The advantage of this model is that it incorporates the context for all terms in the elements of the model, which is required for soundness in an *LF*.

Definition 18 (Pre-Context) *A pre-context is a syntactic expression of the form*

$$y_1 : A_1, \cdots, y_n : A_n$$

where y_i are all pairwise distinct variables.

Definition 19 (Admissible Context) *A pre-context is in $\alpha-$normal form if $y_i = v_i, 1 \leq i \leq n$. A pre-context is admissible if Γ is in $\alpha-$normal form and the context Γ is well formed in the LF ($\Gamma \vdash_{LF} *$).*

All the results that follow are developed around formation trees with the understanding that the same results apply isomorphically to the semantic elements.

Definition 20 (The Context for a Formation Tree) *The context of a formation tree is defined on its structure:*

$$\begin{aligned} Ctxt(\sigma, u^2) \quad = & \\ & label(\sigma) = \cup \quad \Rightarrow \quad Ctxt(\sigma|0), Ctxt(\sigma|1) \\ & label(\sigma) = \odot \quad \Rightarrow \quad Ctxt(\sigma|0), [\![label(\sigma|1)]\!]^2 u^2 \\ & label(\sigma) = S_0 \quad \Rightarrow \quad [\![S_0]\!]^2 u^2 \end{aligned}$$

Definition 20 results in a sequence of the meanings given to each γ term occurring on the leaves of the formation tree. We simplify this notation and refer to the context defined for a Level 2 term A as simply $Ctxt(FT(A))$, assuming the existence of the environments u^2 or more simply, $Ctxt(A)$.

Contexts impose a total order on the partial order of dependencies in a formation tree. The context defined above for a formation tree is identical to the context associated with

the reduced formation tree, for any formation tree. This is easily shown by induction on the structure of the reduced formation tree.

Contexts have a sequential order by definition allowing any context element to refer to any preceding context elements. However, not all context elements refer to prior portions of the context.

Definition 21 (Independent Contexts) *Two contexts* Γ *and* Δ *are* **independent** *if* $\Gamma \vdash_{LF} *$ *and* $\Delta \vdash_{LF} *$, *and* $dom(\Gamma) \cap dom(\Delta) = \emptyset$

All contexts can be broken up into closed components, called sections.

Definition 22 (Sections of a Context) *The* **section** *for an identifier* x *in a context* Γ *is defined by*

$$\Delta, x : A$$

where

1. *Each unbound identifier* y *in the term* A *is in the domain of* Δ *(*$dom(\Delta)$*).*
2. *The context* Δ *is a section for each such* $y \in dom(\Gamma)$.

Definition 23 (Minimal Section) *A section* $\Delta, x : A$ *is* **minimal** *for* x *if* $\forall y \in dom(\Delta)$ *either*

1. y *is an unbound identifier in* A *OR*
2. *if* y *is not an unbound identifier in* A, *then there is some* $z : B$ *in* $dom(\Delta)$ *s.t.* y *is unbound in* B.

Definition 24 (Parent of a Context) *Contexts are related to one another by a parent relation, denoted by* $\langle|$. *For any context* Γ, *variable identifier* c_i *not in the domain of* Γ, *and LF type* A_i*:*

$$\Gamma \langle| \ \Gamma, c_i : A_i$$

We abbreviate a sequence of contexts related by $\langle|$ *to* $\Gamma \langle^* | \Gamma, c_1 : A_1, \cdots, c_k : A_k$.

Contexts can be combined under certain conditions. From a purely logical standpoint, any two contexts can be combined by concatenation. If there are any duplicate context entries, the duplication is irrelevant. The definition of a context is generally as a sequence,

and the most rightmost entry for an identifier is taken to be the definition (this definition is used in the COC implementation [15], [31]).

However, from a pragmatic standpoint, different implementations treat contexts differently. For instance, it is ill-defined in the LEGO implementation [32] for a context to introduce a second value or type for an identifier in the context. In *MF*, we do not currently allow an identifier to have more than one definition. Therefore, however many times that identifier is included in the context it must have the same definition. However, both in the interests of providing for future extensions allowing identifiers to be bound to more than one definition in *MF*, and to define precisely how two or more contexts are related, we need to have a precise definition of how contexts can be combined, and how that combination relates to other combinations. Any two contexts that are independent can be freely combined. Two contexts that are not independent may be combined using the following criteria.

Definition 25 (Combining Contexts) *If* Γ *and* Δ *are two LF contexts such that* $\Gamma \vdash_{LF} * $ *and* $\Delta \vdash_{LF} *$, *and if* $dom(\Delta) \cap dom(\Gamma) = y_1, \cdots, y_n$ *and* $y_i : A \in \Gamma \Rightarrow y_i : A \in \Delta$, *then they can be combined:* $\Gamma \circ \Delta$.

Proposition 26 *The result of combining two well formed LF contexts is a well formed LF context.*

Proof If the contexts Γ, Δ are independent they can be combined freely: $\Gamma \circ \Delta = \Gamma, \Delta \vdash_{LF} *$, $\Delta \circ \Gamma = \Delta, \Gamma \vdash_{LF} *$, and $\Gamma, \Delta =_{LF} \Delta, \Gamma$.

If Γ and Δ have overlapping domains, choose in each the minimal section that includes the overlapping context elements, say Γ_i and Δ_i. Under the assumption that each identifier maps to the same type and value then $\Gamma_i = \Delta_i$. By definition, the remaining portions of the two contexts are independent. Let $\Gamma - \Gamma_i$ be the remaining portions of Γ, and $\Delta - \Delta_i$ be the remaining portions of Δ, then

$$\Gamma_i, (\Gamma - \Gamma_i) =_{LF} \Gamma$$

and

$$\Delta_i, (\Delta - \Delta_i) =_{LF} \Delta$$

Now, Γ and Δ can be combined as follows:

$$\begin{aligned}\Gamma \circ \Delta = \quad \Delta \circ \Gamma = \quad & \Gamma_A \equiv \Gamma_i, (\Gamma - \Gamma_i), (\Delta - \Delta_i) \vdash_{LF} * \\ & \Gamma_B \equiv \Gamma_i, (\Delta - \Delta_i), (\Gamma - \Gamma_i) \vdash_{LF} * \\ & \Gamma_C \equiv \Delta_i, (\Gamma - \Gamma_i), (\Delta - \Delta_i) \vdash_{LF} * \\ & \Gamma_D \equiv \Delta_i, (\Delta - \Delta_i), (\Gamma - \Gamma_i) \vdash_{LF} *\end{aligned}$$

and $\Gamma, \Delta =_{LF} \Delta, \Gamma =_{LF} \Gamma_A =_{LF} \Gamma_B =_{LF} \Gamma_C =_{LF} \Gamma_D$. ∎

Definition 27 (Equivalence Class of contexts) *We refer to the* **equivalence class** *defined by this equality relation as* $[\Gamma]$*, for any context* Γ*: the set of all contexts that are equivalent in the LF.*

Definition 28 (Union of admissible contexts) *The* **union** *of admissible contexts* ($\cup_C$) *is defined to be the combination defined in Definition 25.*

Proposition 29 *The union of two admissible contexts is an admissible context* ($\cup_C$).

Proof

1. If the two admissible contexts are independent, then they are admissible.

2. If the two admissible contexts (Δ, Γ) have overlapping domains, then using Definition 25 we can select the minimal section (Δ_i) containing the overlapping definitions. The context Δ_i is admissible, because if not there would be some context element $y : A$ such that y is unbound in some context element $z : B$ in Δ_i. But if so, then it must be in both Δ and Γ, because these are admissible by assumption. Therefore, it must also be in Δ_i.

   The two contexts Δ_i and $\Delta - \Delta_i$ are now independent. The context Δ_i is admissible. But $\Delta - \Delta_i$ may not be admissible, because it may contain unbound references to variables defined in Δ. However, because Definition 25 places Δ_i first in the context, $\Delta - \Delta_i$ is also admissible. ∎

Definition 30 (Intersection of admissible contexts) *The* **intersection** *of admissible contexts* ($\cap_C$) *is defined to be the overlapping section in Definition 25.*

Proposition 31 *The intersection of two admissible contexts is an admissible context* ($\cap_C$).

Proof Context intersection is similar to the union of two admissible contexts, except that we drop the portions of the context that that are not in the overlapping section. Because the overlapping sections are admissible, the intersection is admissible. ∎

The context defined by a formation tree in Definition 20 is not identical to the definition of admissible contexts defined in Definition 19. To see this, consider a new theory constructed using one of the theory operators $\cup, \cap, \odot$. We get a new formation tree that parallels the construction and because it parallels the construction of the theory presentation, it retains any duplication of sequences and as such is not an admissible context.

Proposition 32 *Suppose a formation tree T denotes an admissible context. Then*

- *The contexts defined for the formation trees at the paths $\sigma|0$ and $\sigma|1$ are both admissible contexts if the label of σ is $\cup$.*
- *The contexts for the formation tree at path $\sigma|0$ is an admissible context if the label of σ is $\odot$.*

Proof

- If the label of the node at path σ is $\cup$, both subtrees will define admissible contexts because the node at path σ denotes an admissible context, and there are no order dependencies generated by $\cup$, each subtree denotes an section of the context defined by the formation tree.
- If the label of the node at path σ is $\odot$, then the tree rooted at path $\sigma|0$ is an admissible context, but the node at path $\sigma|1$ has an order dependency on the support of the tree at path $\sigma|0$, so there is only one admissible context: the one denoted by the tree at path σ itself. ∎

We define a minimal formation tree that preserves all the sequences of the original formation tree but removes all duplication, to support the relation of a formation tree to an admissible context.

Definition 33 (Minimal Formation Trees) *A* **minimal formation tree** *(minFT(T)) for the tree T is a formation tree where the intersection set of all subtrees is empty (or is* $= \{S_0\}$*).*

Proposition 34 *We can construct a minimal formation tree for any formation tre.*

- *The tree containing only the node labeled by* $S_0 :: \alpha$ *is minimal.*
- *T has the form that the root is labeled* $\odot$, $\emptyset|0$ *has a label of* $S_0 :: \alpha$, *and* $\emptyset|1$ *has a label of* $F_i :: \gamma_i$, *then* $minFT(T) = T$.
- *We assume that we can construct a minimal formation tree for all trees of height less than or equal to* $n > 1$. *Let T be a tree of height* $n + 1$.
  1. $T = T_1 \cup T_2$. *T_1 and T_2 are minimal formation trees by the induction hypothesis. Let* $\{A_n\}$ *be the intersection set of* T_1, *and* T_2 *(Definition 12). We can remove the subtrees (Definition 14) of* T_1 *and* T_2 *associated with each* $A_i \in \{A_n\}$: $T_j - \{A_n\}$ *is the result of removing each of the subtrees* $\{A_i\}$ *from the tree* T_j. *Then* $minFT(A \cup B) = A_1 \cup A_2 \cup \cdots \cup A_n \cup (T_1 - \{A_n\}) \cup (T_2 - \{A_n\})$. *All the* A_i *in the intersection set are subtrees of a minimal subtree and so are minimal.* $T_1 - \{A_n\} \cap T_2 - \{A_n\} = \{S_0\}$ *by construction, so T is minimal.*
  2. $T = T_1 \cap T_2$. *T_1 and T_2 are minimal formation trees for by the induction hypothesis. Let* $\{A_n\}$ *be the intersection set of* T_1 *and* T_2 *(Definition 12). Then* $minFT(A \cap B) = A_1 \cup A_2 \cup \cdots \cup A_n$.
  3. $T = T_1 \odot F$. *For an extension* $(\odot)$, *because the features are newly introduced corresponding to newly introduced LF constants, if* T_1 *is a minimal formation tree for the MFsort A, then the formation tree for* $A \odot F_i$ *is also a minimal formation tree.*

Proposition 35 *For any formation tree T,* $minFT(T) =_{FT} T$.

Proof

- $N_T = N_{minFT(T)}$:
  - If T has the form of the base case ($\emptyset$ has the label $\odot, \emptyset|0$ has the label $S_0, \emptyset|1$ has the label $F_i :: \gamma_i$), this is immediately true. No nodes are added or removed.

- We assume that for trees T of height less than or equal to $n > 1$ that $N_{minFT(T)} = N_T$. Let T be a tree of height $n+1$. T may have the form:
  - $T_1 \cup T_2$. If the intersection set of T_1 and T_2 is empty then it is trivially true that $N_{T_1 \cup T_2} = N_{minFT(T_1 \cup T_2)}$, as no subtrees are removed. Suppose the intersection set of T_1 and T_2 is $A_1, \cdots, A_n$. If the sequence F is in T_1 and T_2, then it is in one of the A_i in the intersection set, and hence in the minimal tree. If the sequence F is not in either T_1 or T_2, then it is not in any of the A_i in the intersection set, but must be in either $T_1 - \{A_n\}$ or $T_2 - \{A_n\}$.
  - $T_1 \cap T_2$. If the largest subtree of $T_1 \cap T_2$ is the node S_0, then it is trivially true that $N_{T_1 \cap T_2} = N_{minFT(T_1 \cap T_2)}$, as there are is only one subtree in common and it is minimal. By Definition 15, $(T_1 \cap T_2) = A_1 \cup A_2 \cdots A_n$ for the intersection set $\{A_n\}$. So this reduces to the case for $\cup$.
  - $T_1 \odot F$. Because T_1 has the property by assumption, $N_{minFT(T_1)} = N_{FT(T_1)}$, and the newly introduced sequence is not a sequence in T_1 by the definition of $\odot$, so $N_{minFT(T_1 \odot F)} = N_{FT(T_1 \odot F)}$.

- Let $S = minFT(T)$. Then $F_i <_T F_j \Rightarrow F_i <_S F_j$
  - If T is the node S_0 this is trivially true, because there are no dependencies.
  - If T is the tree who root is labeled with $\odot$ and whose tree at $\emptyset|0$ is labeled with $S_0 :: \alpha$ and whose node at $\emptyset|1$ is labeled with $F :: \gamma$, this too is trivial, because $minFT(T) = T$.
  - We assume that for trees of height less than or equal to $n > 1$ that dependencies in such trees are preserved in the minimal tree. Let T be a tree of height $n+1$. Then T may have the form:
    - $T_1 \cup T_2$. Then $F_i < F_j$ must be in either T_1 or in T_2 or both, because the label $\cup$ introduces no new dependencies.
      - $F_i <_{T_1} F_j$. Then it is not in the intersection set but is in $T_1 - \{A_n\}$. So it is in the minimal tree as well. The case for $F_i <_{T_2} F_j$ is similar.

- $F_i < F_j$ is in both T_1 and in T_2. So the nodes that generated this dependency are in a subtree in the intersection set, so it is in the minimal tree.

* $T_1 \cap T_2$. If $F_i < F_j$ is a dependency generated in both T_1 and T_2, then the subtree generating the dependency is in a tree in the intersection set and this case reduces to the case for $\cup$. If it is not in both, then it won't be in any tree in the intersection set in either the tree T or in $minFT(T)$.
* $T_1 \odot F$. If $F_i <_{T_1} F_j$, then it must be in $minFT(T)$ because T_1 is minimal by the induction hypothesis and the construction of the minimal tree for $\odot$ does not alter the the tree.

- Let $S = minFT(T)$. Then $F_i <_S F_j \Rightarrow F_i <_T F_j$.
  - If S has the form $S_0 :: \alpha$ or a root labeled with $\odot$ with $\emptyset|0$ labeled with $S_0 :: \alpha$ and $\emptyset|1$ labeled with $F :: \gamma$, then $S = T$ by construction.
  - We assume that for minimal trees of height less than or equal to $n > 1$ that dependencies in the minimal trees are also in the original tree. Let S be a minimal tree for T of height $n + 1$. Then S may have the form:
    * $S_1 \cup S_2$. If $F_i <_S F_j$ then either it occurs in S_1 or S_2. Because these are minimal trees that preserve dependencies, the dependency is also in the original tree T.
    * $S_1 \odot F$. This argument is similar to the $\odot$ case above.
    * $S_1 \cap S_2$ can not occur (by construction).

∎

Proposition 36 *The minimal formation tree denotes an admissible context if each introduced sequence F_i and its support denote an admissible context .*

Proof

- The formation tree consisting of a single node labeled by $S_0 :: \alpha$ determines an admissible context: the empty context. (If we expand MF to include constant theories, these constant theories would be assumed to determine an admissible context.)

- Assume we have formation trees of height less than or equal to $n > 1$ denoting admissible contexts. Let T be a formation tree of height $n + 1$.

  - T may have the form T_1 of height n extended with a sequence. The path $\emptyset|1$ of the $minFT(T)$ will have the label $F :: \gamma$ for the newly introduced sequence. The path $\emptyset|0$ will have the formation tree $minFT(T_1)$. This is an admissible context if the following condition holds:

    $$\begin{array}{ll} \text{If} & Ctxt(minFT(T_1)) \vdash_{LF} [[F]^2 u^2]^1 u^1 : * \\ \text{then} & Ctxt(minFT(T)) \vdash_{LF} * \end{array}$$

    Meaning, that if the introduced sequence is valid in the indicated context, then the context extended by the sequence is an admissible sequence, assuming the environments u^2 and u^1.

  - T may have the form $T_1 \cup T_2$ $T_1 = minFT(A)$ and $T_2 = minFT(B)$, each of height less than or equal to n If T_1 and T_2 have no sub-trees in common, then $A \cup B$ determines an admissible context, because they are mutually independent and each denotes an admissible context by assumption.

    If $minFT(A)$ and $minFT(B)$ have some sub-trees in common, then these trees will be in the intersection set $\{A_n\}$ of T_1 and T_2. Because T_1 and T_2 both denote admissible contexts, each subtree denotes an admissible context by Proposition 32 except those subtrees labeled with a term of type category γ. Because a subtree F labeled with a term of type category γ introduces an order dependency, if F is in T_1 and in T_2 it must occur at some paths $\sigma_1|1$ and $\sigma_2|1$ in T_1 and T_2, respectively. As each tree in the intersection set is maximal, if F is in some A_k in the intersection set, it must also be the case that the subtree denoted by $\sigma_1|0$ ($\sigma_2|0$) must also be in A_k, as well as the tree denoted by σ_1

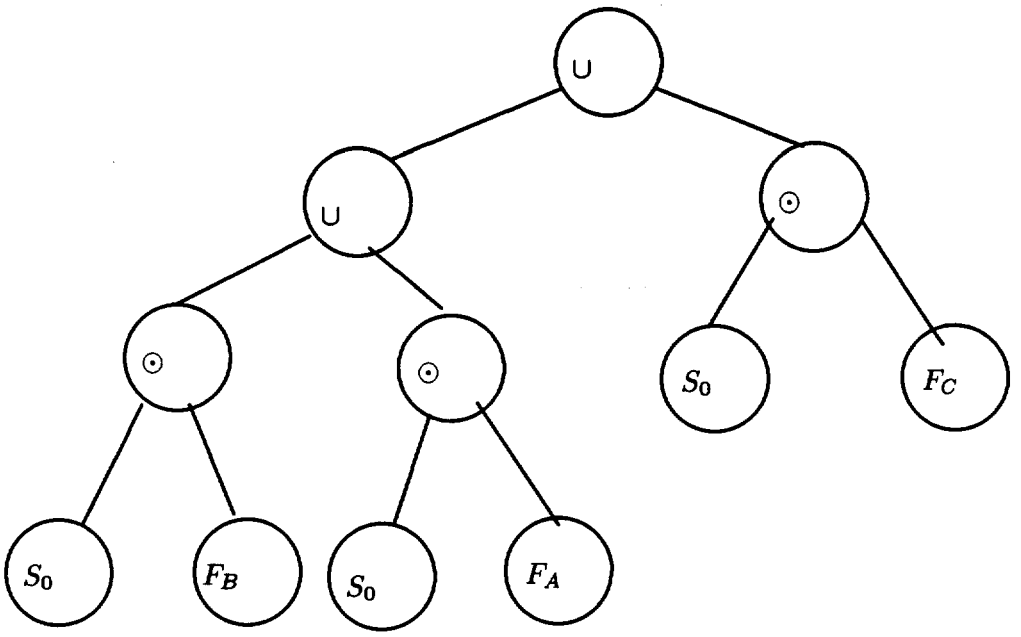

Figure 4.5: Minimal Formation Tree

(σ_2). So all trees in the intersection set are rooted with nodes having a label of $\odot$ or $\cup$, meeting the criteria of Proposition 32.

- The case for the intersection of two *MFsorts* A and B denoting admissible contexts reduces to the case for the union.

∎

We assume from here on that a formation tree denotes a minimal formation tree and hence an admissible context under the above conditions.

Figure 4.5 shows the minimal tree for the formation tree of Figure 4.3.

Definition 37 (Parent of a Level 2 term) *Level 2 terms are related to one another by a parent relation ($<_p$), based on the $\langle|$ relation for are contexts. $A <_p B$ if*

- *$FT(A)$ is a subtree of $FT(B)$ This condition ensures that B is constructed from A.*
- *$Ctxt(A)\langle|^* Ctxt(B)$ This condition states that the context defined for A is a parent of that defined for B.*

While contexts are related to other contexts by context elements (individual declarations of the for $y : A$), theory presentations are related to one another by sequences. Each presentation constructor adds one or more sequences to another.

Proposition 38 *Presentation constructors preserve the $<_p$ relationship such that if A and B denote admissible contexts:*

$$A <_p A \cup B$$
$$B <_p A \cup B$$
$$A \cap B <_p A$$
$$A \cap B <_p B$$
$$A <_p A \odot F$$

Proof Assume that A and B are theory presentations denoting admissible contexts.

- Case 1. If A and B are independent, then $Ctxt(A), Ctxt(B) =_{LF} Ctxt(B), Ctxt(A)$. And by the definition of $Ctxt$ and $\langle|^*$:

$$Ctxt(A)\langle|^* Ctxt(A \cup B)$$
$$Ctxt(B)\langle|^* Ctxt(B \cup A)$$

- Case 2. A and B denote contexts with overlapping domains. Using Definition 20, we can extract the overlapping elements:

$$Ctxt(A \cup B) =_{LF} \Delta, (Ctxt(A) - \Delta), (Ctxt(B) - \Delta)$$
$$Ctxt(A \cup B) =_{LF} \Delta, (Ctxt(B) - \Delta), (Ctxt(A) - \Delta)$$

and by Definition 21, $\Delta = Ctxt(FT(A) \cap FT(B))$, leaving us with:

$$\Delta, Ctxt(A) - \Delta = Ctxt(A)\langle|^* Ctxt(A \cup B)$$
$$\Delta, Ctxt(B) - \Delta = Ctxt(B)\langle|^* Ctxt(B \cup A) = Ctxt(A \cup B)$$

- Case 3. $A \odot F$ denotes an admissible context. $Ctxt(A \odot F) = Ctxt(A), [\![F]\!]$ by the definition of $Ctxt$, so

$$Ctxt(A)\langle|^* Ctxt(A), [\![F]\!].$$

- Case 4. If A and B denote independent contexts, then the intersection is empty. If they have overlapping contexts, then we can form the canonical expression for them, which is Case 1.

∎

4.2.5 Semantics

Before defining the semantics, we give an intuition for what these composite terms mean. *MF* constructs new types by constructing new sequences, with the intent that these sequences will determine an *LF* context. The natural way to represent these sequences is via a $\prod$ type, such that if p is an element of such a product type, $p(1)$ will be the first element in the sequence, etc. When sequences are combined, we again have a $\prod$ type, such that if q is an element of such a product type, $q(1)$ will be the first sequences in the sequences of sequences, etc. (This is essentially the structure of the *RFT* for a formation tree.)

1. Each γ term denotes a sequence of statements in the *LF*, called γ sequences.

2. A constant theory of type α has no structure for *MF* and can be treated as the empty theory (the empty sequence, denoted by $\langle\rangle$).

3. A term $A \odot F_1$ is a concatenation of a γ sequence onto a theory. Each extension adds one γ sequence onto an existing theory. In this sense, a theory is just a sequence of γ sequences. We use a comma to indicate a sequence of γ sequences.

4. $A \cup B$: a union of all the statements in each theory. Each term A, B denotes the sequence of γ sequences(s) that appear as extensions in their construction. The union of the two is the concatenation of these γ sequences, yielding another sequence of γ sequences.

5. $A \cap B$: an intersection of all the statements in each theory. The meaning of the intersection is all the γ sequences common to both theories. Intersection will be represented as a union of theories using the canonical form defined in 4.2.3.

6. $A \cdot B[\iota_{k_1}, \cdots, \iota_{k_m}][\iota_{j_1}, \cdots, \iota_{j_n}][\iota_{i_1}, \cdots, \iota_{i_n}]$: instantiation constructs an extension from applications of the features specified for A to the features specified for B.

So, ultimately, the meaning of *MFsorts* is a structured sequence of sequences. Additionally, we make use of a particular view of *MFsorts*. Every *MFsort* can be viewed as

introducing some extension (i.e. some sequence). The term $A :: \alpha$ denotes a constant theory and has no structure in *MF*, meaning that the theory A introduces the empty sequence (or one presumed empty in *MF*: $\langle\rangle :: \gamma_0$). Similarly, a term $A \cup B :: \theta_A \sqcup \theta_B$ denotes a theory that is composed of the sequences defined by A and by B, but itself introduces no new sequence ($A \cup B$ introduces no additional *LF* content). However, terms $A \odot F :: \theta_A + \gamma_F$ do introduce a new sequence: the sequence denoted by F.

This view of *MFsorts* matches the view presented in a formation tree, which reduces an *MFsort* to the sequence of terms of γ types introduced using the $\odot$ operator.

As in Level 1, Level 2 has no means of introducing or assigning values to identifiers. We assume a pre-defined phrase type environment, π^2, that maps predefined identifiers (defined in Level 3) to types.

We introduce the following sets:

$[\tau]^2_\pi$	The set of phrases of type τ in π^2
$[\gamma]^2_\pi$	The set of phrases of type γ in π^2
$[\theta]^2_\pi$	The set of phrases of type θ in π^2
$[\omega]$	The set of identifiers for features
$[\theta]$	The set of identifiers for θ types

The meanings of the τ types are the Level 1 terms used to identify the abstract values. The sets of possible meaning associated with τ types ($[\![\tau]\!]$) are:

$$[\![\tau]\!] \equiv [\tau]^1_\pi$$

The γ types are sequences of *LF* statements, each statement denoting an element of $[\![\tau]\!]$. Elements of the γ type determine an extension to an *LF* environment. The sets of possible meaning associated with γ types are:

$$[\![\gamma]\!] = \prod_{i=1,n} [\![\tau_i]\!]$$

This set denotes all sequences of Level 1 terms. This set includes a lot of junk, in the sense that not all sequences will be meaningful in *LF*. Constraints on what are meaningful sequences are handled separately, because what is meaningful in an *LF* depends partially on what has already been defined in the *LF*.

The set of possible meanings associated with θ types is:

$$[\theta] = \prod_{i=1,n} [\gamma_i]$$

The possible meanings for the terms of θ type are the set of sequences of sequences of LF statements, corresponding to the context defined for a minimal formation tree.

The set $[\![\pi^2]\!]$ denotes the set of π^2-compatible valuation environments: the set of all functions u^2 from identifiers to values such that for all identifiers $\iota \in [\theta]$, $u^2(\iota) \in [\![\pi^2(\iota)]\!]$.

The semantic function has the type:

$$[\![\cdot]\!]^2_\pi : [\theta]^2_\pi \rightarrow ([\![\pi^2]\!] \rightarrow [\![\theta]\!])$$

$$[\![\cdot]\!]^2_\pi : [\gamma]^2_\pi \rightarrow ([\![\pi^2]\!] \rightarrow [\![\theta]\!])$$

In the following equations, we use the comma (,) to denote context concatenation. The meanings of *MFsorts* are admissible contexts, and we use the $\cup_C$ operator from the previous section to combine admissible contexts. The semantic functions are defined by[5]:

$$[\![S_0]\!]^2_{\pi,\alpha} u^2 = \langle\rangle$$

$$[\![A]\!]^2_{\pi,\theta} u^2 = u^2(A)$$

$$[\![A \cup B]\!]^2_{\pi,\theta_A \cup \theta_B} u^2 = [\![A]\!]^2_\pi u^2 \cup_C [\![B]\!]^2_\pi u^2$$

$$[\![A \odot F]\!]^2_{\pi,\theta_A+\gamma_F} u^2 = [\![A]\!]^2_\pi u^2, [\![F]\!]^2_\pi u^2$$

$$[\![\iota_1 \mapsto x_1, \cdots, \iota_n \mapsto x_n]\!]^2_{\pi,\gamma_i} u^2 = [\![x_i]\!]^2_\pi u^2, 1 \leq i \leq n$$

$$[\![A.\iota]\!]^2_{\pi,\tau_i} u^2 = ([\![A]\!]^2_{\pi,\iota \mapsto \tau_i} u^2)(1)$$

$$[\![A.\iota(B.\iota_1, \cdots, B.\iota_m]\!]^2_{\pi,\tau_i(\tau_{\iota_1},\cdots,\tau_{\iota_m})} u^2 = [\![A.\iota]\!]_{\pi,\tau_i}([\![B.\iota_1]\!]^2_{\pi,\tau_{\iota_1}} u^2, \cdots, [\![B.\iota_m]\!]^2_{\pi,\tau_{\iota_m}} u^2)$$

$$[\![c_i]\!]^2_\pi u^2 = c_i$$

$$[\![c_i(c_{j_1}, \cdots, c_{j_m})]\!]^2_\pi u^2 = c_i(c_{j_1}, \cdots, c_{j_m})$$

[5]Note that in the semantic functions for projection we use positional rather than named projections, corresponding to the semantic sets associated with sequences.

4.2.6 Conversions

We define conversion functions for the coercions defined in Section 4.2.1.

$$\text{ConvertI} \quad [\theta + \iota_1 \mapsto \tau_1 \& \cdots \& \iota_n \mapsto \tau_n \vdash \iota_1 \mapsto \tau_1 \& \cdots \& \iota_n \mapsto \tau_n] p\iota_i = p(2)(i) \quad \text{(Convert1)}$$

$$\text{ConvertII} \quad \begin{array}{l} [\iota_1 \mapsto \tau_1 \& \cdots \& \iota_n \mapsto \tau_n \vdash \iota'_{i_1} \mapsto \tau_{i_1} \& \cdots \& \iota'_{i_m} \mapsto \tau_{i_m}] p\iota'_{i_j} = p(k) \\ \text{when } \iota'_{i_j} = \iota_k \end{array} \quad \text{(Convert2)}$$

$$\text{ConvertIII} \quad \begin{array}{l} [\theta_1 \sqcup \theta_2 \vdash \iota_1 \mapsto \tau_1 \& \cdots \& \iota_n \mapsto \tau_n] p\ \iota_i = \\ ([\theta_1 \vdash \iota_1 \mapsto \tau_1 \& \cdots \& \iota_m \mapsto \tau_m] p(1)\iota_j \text{ if } \iota_i = \iota_j \\ ([\theta_2 \vdash \iota_{m+1} \mapsto \tau_{m+1} \& \cdots \& \iota_n \mapsto \tau_n] p(2)\iota_k \text{ if } \iota_i = \iota_k \end{array} \quad \text{(Convert3)}$$

These conversions allow us to drop information. The first conversion would allow us to coerce $A \odot F$ to F. The second would allow us to select parts of a sequence: $And \mapsto c_0, Or \mapsto c_1, Impl \mapsto c_2 \vdash And \mapsto c_0$. The last conversion allows us to select feature from any part of a structured type: $(S_0 \odot F_1) \cup (S_0 \odot F_2) \vdash (S_0 \odot F_1) \odot F_2$. This last is equivalent to the flattening done when a formation tree is converted to a reduced formation tree.

4.2.7 Properties

This section demonstrates the properties of Level 2, notably soundness and completeness. We define soundness based on the underlying *LF*.

Definition 39 (Soundness) *If an MF sequence is judged valid in MF (meaning that we judge that it denotes an admissible context), then the denotation of that sequence is valid in the underlying LF.*

Definition 40 (Completeness) *MF is complete if it is sound and if an LF sequence is valid in the underlying LF, then it's representation in MF is also valid in MF.*

Proposition 41 (Proof of Equal Terms in Level 2) *If two terms have equal formation trees then they denote contexts in LF that are equal in the LF ($=_{LF}$).*

Proof If two terms have equal formation trees then they obey all the same order dependencies. Therefore, the paths that describe all the possible admissible contexts derived

from these trees all determine contexts in the same equivalence class, and are therefore equal in the LF ($=_{LF}$). ∎

Proposition 42 (Soundness) *MF Level 2 terms are sound provided all the extensions are valid in the LF in the defined context for each extension.*

Proof *MF* Level 2 terms include terms that introduce no additional LF content ($\cap, \cup$) and those that do ($\odot, \cdot$).

1. If the *MFsorts* A, B are valid, meaning that they denote admissible LF contexts, then $A \cap B$ and $A \cup B$ are valid in the LF (provable by the structure of the formation trees and combination of admissible contexts).

2. If an *MF* Level 2 term introduces a γ extension, such as $A \odot F$, then the result is judged valid only if

$$Ctxt(FT(A)) \vdash_{LF} [[F]^2 u^2]^1 u^1$$

∎

MF can record valid relationships, but it relies for its validity on the underlying logical framework.

Proposition 43 (Completeness) *Level 2 is complete.*

Proof It is trivially true that any valid sequence in LF can be represented by a Level 2 term, so *MF* is complete. ∎

4.3 *MFTheory* Commands - Level 3

The purpose of Level 3 is to provide a persistent aspect to *MF*. Level 2 provides descriptive support: a Level 2 term can describe the structure of a theory presentation. However, Level 2 does not support naming of theory presentations nor inspecting a current set of theory presentations, two aspects required for reuse. The persistent nature of the

Level 3 presentation base supports reuse, both in construction (i.e. reusing existing theory presentations to build new ones) and in queries (inspecting the presentation base for its current state).

Level 3 distinguishes those *MFsorts* that have been constructed and named from the space of all possibly constructible theories: these named artifacts and their relation to other named artifacts are kept in a *Presentation Base*, allowing theories to be retained and described by their relation to other theory presentations. There are three principle methods of interacting with a presentation base:

1. We can examine a presentation base for all theories that are consistent extensions of a description.

2. We can reconstruct a theory presentation from a presentation base (i.e., reconstruct a sequence of *LF* statements from a Level 3 theory presentation).

3. We can add to the presentation base by binding an identifier to a Level 2 value and defining the dependencies in the presentation base.

The presentation base is structured as a partial order whose nodes are identifiers of Level 2 terms. The presentation base is extended through presentation commands that are isomorphic to the Level 2 constructors: **combine**, **generalize**, **extend**, and **instantiate** $(\cup, \cap, \odot, \cdot)$, differing in that the result of a presentation command is a modification of the presentation base. The state gives for each theory presentation, the relation that theory presentation has to its constituents (often called a *location* in the presentation base), a Level 3 environment that maps each identifier to the extension introduced by that identifier (a term of γ type), a Level 2 environment binding each Level 2 identifier to its admissible context, and a Level 1 environment binding each Level 1 identifier to an element of the set $[AbstrVals]$. From this state a Level 2 term and environment, and a Level 1 environment can be reconstructed, allowing the Level 3 term to be interpreted at Level 2, and hence in Level 1 and the *LF*. The Level 3 environment retains the textual record of the extensions to provide an actual history of feature introduction.

For example, if we have a presentation as specified in Figure 3.3, we could add to the

presentation base by

$$ListAndEquality \equiv \textbf{combine}\ ThList\ ThEquality$$

which would result in the following changes in the state of the presentation base:

$ThList < ListAndEquality$	Add to the partial order.
$ThEquality < ListAndEquality$	Add to the partial order.
$u^3(ListAndEquality) = \langle\rangle$	The empty sequence.
$u^2(ListAndEquality) = [\![ThList \cup ThEquality]\!]^2 u^2$	The meaning of *ListAndEquality* is the meaning of the corresponding Level 2 term.
	No features added.
	No Level 1 abstract pairs added.

This Level 3 command only has meaning in a presentation base where *ThList* and *ThEquality* are identifiers in the domain of u^2. The Level 2 environment is extended by mapping *ListAndEquality* to the meaning of the equivalent Level 2 term in the indicated environment.

Similarly, if we had

$$ExtendedList \equiv \textbf{extend}\ ThList\ (\iota_i, c_i, ([Context_i], LFval_i, LFtyp_i))$$

we would see:

$ThList < ExtendedList$	Add to the partial order.
$ThList \odot \iota_i \mapsto c_i$	The Level 2 term.
$u^3(ExtendedList) = \langle \iota_i \mapsto c_i \rangle$	The introduced extension: the u^3 environment.
$u^2(ExtendedList) = [\![ThList \odot \iota_i \mapsto c_i]\!]^2 u^2$	The meaning of the new theory presentation.
$u^1(c_i) = ([Context_i], LFval_i, LFtyp_i)$	The abstract value pair: the u^1 environment.

4.3.1 Frames

The presentation base is represented as a frame, similar to Kripke's frames. (The treatment given below is derived from the presentation by Anil Nerode and Richard Shore, [35].)

Frames are a method for organizing sets of related statements in a language. Usually they are defined by specifying a set of worlds, a relation on this set, and a (classical) structure for each set, giving its meaning in terms of the language of the statements in the set. In our case, each world is a presentation base structure: a set of named Level 2 terms, an ordering on these terms, and environments. The language of these worlds is the set of named theory presentations bound in each world. The relation between these worlds is a relation based on set containment: each world extends a previous world by naming a new theory presentation through binding operators known as presentation commands. We don't however have a simple (classical) structure for any set such that for any statement in the language of the set, we can determine if the statement is true or false (valid or invalid). We use instead the model of admissible contexts defined in Section 4.2.4.

We can now define a structure representing the presentation base.

Definition 44 (*PB*-Structure) *A PB structure is a 5-tuple*

$$(N, <_P, u^3, u^2, u^1)$$

where

- N *is a set of nodes, where a node is an identifier of a Level 2 term.*
- $<_P$ *is the partial order on N, defined by $<_p$.*
- u^3 *is a function that assigns to each $n \in N$ the term of γ type indicating what extension was introduced, if any.*
- u^2 *is a function that assigns to each $n \in N$ the admissible context associated with the identifier.*
- u^1 *is a function that assigns to each c_i introduced in an extension for each $n \in N$ the associated value as an LF encoded term.*

The set N is the set of named theory presentations. We let π^3 be the type environment for a presentation base, assigning to each $A \in dom\pi^3$ a type such that $[\![u^3(A)]\!]^2 u^2 :: \pi^3(A)$. This type environment maps all identifiers to a type in the γ category of types from Level 2. This type environment serves to provide a record of what features were introduced for each theory presentation. This type environment π^3, the valuation environment u^3 and the order $<_P$ are sufficient to rebuild a Level 2 term from a presentation base structure that

is semantically equivalent to the original term used in its construction. The Level 3 type environment and valuation environment only specify the extensions (if any) introduced in each theory presentation. The rest of the Level 2 term can be reconstructed from a presentation base structure. Each *PB* structure also implicitly defines type environments π^2 and π^1 such that $u^2(A) :: \pi^2(A)$ and $u^1(c) :: \pi^1(c)$ (or equivalently, $u^2(A) \in [\![\pi^2(A)]\!]$, denoting all the admissible contexts defined by the equivalence class for the formation tree for A and $u^1(c) \in [\![\pi^1(c)]\!]$, denoting all abstract value encodings for an *LF* identifier).

PB structures are related to one another.

Definition 45 ($\leq_{PB}$) *If PB is a set of PB structures, and $p_i, p_j \in PB$, then $p_i \leq p_j$ if for*

$$p_i = (N_i, \leq_{p_i}, u_i^3, u_i^2, u_i^1)$$
$$p_j = (N_j, \leq_{p_j}, u_j^3, u_j^2, u_j^1)$$

then

$N_i \subseteq N_j$

$<_{p_i} \subseteq <_{p_j}$ *meaning:* $n_i <_P n_j \in p_i \Rightarrow n_i <_P n_j \in p_j$

$u_i^3 \subseteq u_j^3$ *meaning:* $u_i^3(n) = u_j^3(n)\ \forall n \in N_i$

$u_i^2 \subseteq u_j^2$ *meaning:* $u_i^2(n) = u_j^2(n)\ \forall n \in N_i$

$u_i^1 \subseteq u_j^1$ *meaning:* $u_i^1(c) = u_j^1(c)\ \forall c \in dom(u^1)$

Definition 46 (Structure M) *A structure M for a language N of identifiers over a set of Level 2 terms consists of a set of admissible contexts such that the admissible contexts form a set of trees related by the $\langle|^*$ relationship.*

The structure M supports a meaning function for any PB structure p with identifiers n, each $n \in N$ mapping to an admissible context, when interpreted in the environment u^2. The set of contexts forms a set of trees $\langle T, \langle| \rangle$, where each T denotes an admissible context, and the contexts determined by the trees are related by the $\langle|$ relation. The definition of the Level 2 formation trees, the semantic functions and the admissible contexts they denote form the required set of set of admissible contexts, each related to one another by $\langle|^*$. The proof that the contexts are related by the $\langle|^*$ relation is by induction on the structure of terms, formation trees, and contexts. These contexts can all be combined via rules for combining contexts defined in Section 4.2.4.

A PB structure p defines a context based on the contexts of its nodes (the Level 2 terms). Each n in a PB structure specifies a context, defined by the formation tree constructed for the identifier n in the PB structure p. We identify this formation tree at node n in a presentation base structure p by $FT(n,p)$. The formation trees for each n can all be combined using the rules for combining formation trees. If each n represents an admissible context, then the result of combining each n will also be an admissible context. We will refer to the context defined by a PB structure p at all nodes as $Ctxt(p)$.

Definition 47 (Frames) *Let* $C = (\mathbf{P}, \leq_{\mathbf{P}}, \{M(p)\}_{p \in bfP})$ *where* $\mathbf{P}$ *is a set of PB structures,* $\leq_{\mathbf{P}}$ *is a relation on* $\mathbf{P}$*, and* $\{M(p)\}_{p \in \mathbf{P}}$ *is a function assigning to each* $p \in \mathbf{P}$ *a structure* $M(p)$ *for the language* N *(the set of identifiers in the PB structure).*

C is a frame if for every p_i, p_j in $\mathbf{P}$, $p_i \leq_{\mathbf{P}} p_j$ implies that $N_{p_i} \subseteq N_{p_j}$, $M(p_i) \subseteq M(p_j)$, and the interpretations of all $n \in N_{p_i}$ is the same in $M(p_i)$ and $M(p_j)$.

$M(p)$ is the model for $p \in \mathbf{P}$. Each element of the set N in a presentation base structure maps to an admissible context.

Note that $M(p)$ is a model for a *PB* structure because it maps each identifier in a presentation base structure to an admissible context that is its meaning in an underlying *LF*. For $p \in \mathbf{P}$ we take $M(p)$ to be the set of admissible contexts constructed for all n in the set of nodes of the *PB* structure p. If $p <_{\mathbf{P}} q$ we say that p is accessible from q. This requires a formalization of what kinds of *PB structures* are accessible from one another.

Proposition 48 (Relation of Contexts for *PB structures*) *If p_i, p_k are PB structures such that $p_i <_{\mathbf{P}} p_k$, then there is some sequence of presentation base structures:*

$$p_i, \cdots, p_j, \cdots, p_k$$

such that for each $p_j, p_i <_{\mathbf{P}} p_j <_{\mathbf{P}} p_k$ adds one node to the preceding PB structure in the sequence (by definition of $<_{\mathbf{P}}$) and where the contexts defined by each PB structure will be related by the context relation: $Ctxt(p_i)\langle|^* Ctxt(p_j)\langle|^* Ctxt(p_k)$.

Proof

- The PB structure p_k may add 1 node n to p_i. So $N_{p_k} = N_{p_i} \cup \{n\}$, and $N_{p_i} \subseteq N_{p_k}$. $M(p_i)$ defines an admissible context for all the nodes in p_i. The new node n may have two cases:

1. The node n adds no new information, so there is no added sequence. When $M(p_k)$ is extended with the admissible context defined for n ($FT(n, p_k)$), the formation tree (context) will already be in the intersection, and hence $Ctxt(p_i) = Ctxt(p_k)$.

2. The node n adds a new sequence to an existing node, so there will be a subtree in $FT(n, p_k)$ whose root has a label of $\odot$ and the label of $\emptyset|1$ is labeled with the extension introduced in n. The subtree at $\emptyset|0$ is already present by assumption. When the trees are combined, the subtree at $\emptyset|0$ will be in the intersection and hence will be prior to the context associated with the introduced extension of n:

$$Ctxt(p_i)\langle|^* Ctxt(p_i), x_1 : \tau_1, \cdots, x_n : \tau_n$$

where $x_1 : \tau_1, \cdots, x_n : \tau_n$ is the context associated with the extension introduced by n.

- We assume that for p_k adding $n > 1$ nodes that we have a sequence:

$$p_i <_{\mathbf{P}} p_{j_1} <_{\mathbf{P}} \cdots <_{\mathbf{P}} p_{j_n} <_{\mathbf{P}} p_k$$

Now if p_k adds $n+1$ nodes to p_i, then there must be a sequence where we have a p_{j_n} adding n nodes to p_i and p_k adds 1 node to p_{j_n}. Then the same reasoning applies as in the case above.

∎

A *Presentation Base* is a *Frame* where the set $\mathbf{P}$ is constructed through presentation commands, each preserving the relationship of PB structures in the set $\mathbf{P}$ through the $\langle|^*$ relation of their associated contexts. Each PB structure in the Frame represents a snapshot of the state of the context, and hence the state of a *LF*. We use the terms Presentation Base and *Frame* interchangeably. [6]

[6]Note that a presentation base structure is not a Presentation Base. A frame which includes a set of presentation base structures is a Presentation Base. This is an unfortunate similarity. A Presentation Base represents a growing state. Each presentation base structure is a snapshot in that growth. However, the Presentation Base (the frame) also includes the history, in the sense of the growth of a development.

4.3.2 Presentation Base Commands, Expressions, and Queries

This section defines the presentation base terms, which are the commands, expressions, and queries that access a *PB structure* and derive a new *PB structure*.

In this section, we use A, B for theory identifiers, e_i for expressions, f_i, g_i, h_i for feature identifiers, p_i for presentation base structures, and **f** for a frame. We abbreviate the list of features $[f_0, \cdots, f_n]$ as $[f_i]$, and let $[f]_A$ be a list of features in the theory presentation A. We use the same identifier space as Level 2: the set $[\theta]$ of structured theory identifiers.

Syntax

MF Level 3 has three kinds of phrases, corresponding to the way in which we interact with a presentation base. We can examine a frame to find all presentation base structures in which a theory presentation is defined (*exp*). We can query a frame to find a value (an *LF* theory) for a theory presentation (*qry*). We can extend a frame by adding a new presentation base structure conforming to the structure specified in Definitions 44 and 47 (*cmd*).

The three phrase types are:

$$\delta \quad ::= \quad exp \mid qry \mid cmd \quad \text{Phrase Types}$$

$$\begin{array}{lll} Fintr & ::= & (\iota, c_i, A\_val_i) \mid (\iota_i, c_i, A\_val_i), Fintr \quad \iota_i \in [\omega], c_i \in [\tau] \\ & & \qquad\qquad\qquad\qquad\qquad\qquad\quad A\_val_i \in [AbstrVals], \\ Flist & ::= & [\iota_1, \cdots, \iota_n] \qquad\qquad\qquad\qquad\quad \iota_i \in [\omega] \\ Clist & ::= & [c_1, \cdots, c_m] \qquad\qquad\qquad\qquad\quad c_i \in [\tau] \\ FClist & ::= & [(\iota_1, c_1), \cdots, (\iota_m, c_m)] \qquad\quad c_i \in [\tau] \;\; \iota_i \in [\omega] \end{array}$$

Identifier
$$\frac{}{P :: exp} \; P \in [\theta] \qquad (Expressions1)$$

Combine
$$\frac{e_1 :: exp \;\; e_2 :: exp}{\textbf{combine } e_1 \; e_2 :: exp} \qquad (Expressions2)$$

Generalize
$$\frac{e_1 :: exp \;\; e_2 :: exp}{\textbf{generalize } e_1 \; e_2 :: exp} \qquad (Expressions3)$$

Instantiate
$$\frac{e_1 :: exp \;\; e_2 :: exp \;\; f :: Flist \;\; g :: Flist \;\; h :: Flist \;\; c :: Clist}{\textbf{instantiate } e_1 \; e_2 \; f \;\; g \;\; h \;\; c \;\; :: exp} \qquad (Expressions4)$$

$$\textit{Extend} \qquad \frac{e :: exp \quad f :: Fintr}{\textbf{extend}\ e\ f :: exp} \qquad (Expressions5)$$

$$\textit{FeatureDescription} \qquad \frac{F :: FClist}{F :: exp} \qquad (Expressions6)$$

$$\textit{Command} \qquad \frac{e :: exp}{P \equiv e :: cmd}\ P \in [\theta] \qquad (Commands1)$$

$$\textit{ExpressionQuery} \qquad \frac{e :: exp}{\mathbf{Q}(e) :: qry} \qquad (Queries1)$$

Not all expressions that are syntactically well formed have a meaning in a presentation base structure. The meaning of an expressions in a presentation base structure is limited to those theory presentations that have been bound to an identifier.

4.3.3 Semantics

Our presentation base is a frame, meaning that it encompasses a set of PB structures in a particular relationship. We can define a maximal and minimal PB structure as being the greatest (least) presentation base structure p_i in a frame according to the presentation base structure order ($\leq_P$) in the frame.

The set *Frame* is the set of frames whose presentation base structures meet the consistency requirements above.

The phrase types for Level 3 are:

$$\begin{aligned} [exp] &= \text{The set of expressions} \\ [cmd] &= \text{The set of commands} \\ [qry] &= \text{The set of queries} \end{aligned}$$

The sets of possible meanings of the phrase types are defined as follows:

$$\begin{aligned} [\![exp]\!] &= Frame \rightarrow Frame \\ [\![cmd]\!] &= PBStructure \rightarrow Frame \rightarrow Frame \\ [\![qry]\!] &= PBStructure \rightarrow Frame \rightarrow [\![\theta]\!] \end{aligned}$$

The sets of meanings are intuitively:

- $[\![exp]\!]$ determines a map from a *Frame* to a *Frame*: restricting a frame to those *PB structures* in which the expression has meaning. The minimal p in the frame

is the least *PB* structure where a node identifier has a meaning with the indicated properties.

- $[\![cmd]\!]$ extends a frame with a new PB structure in which a new identifier is bound to a Level 2 value that has the indicated meaning within the new PB structure in the frame. (We assume the most recent (maximal) PB structure is selected, though there is potential for selecting other structures).

- $[\![qry]\!]$ is map from a *PB* structure to the meaning of a theory presentation identifier in a *PB structure* in the frame.

The semantic functions for the Level 3 phrases have the types:

$$\begin{aligned}
[\![\cdot]\!]_E &: [exp] \rightarrow [\![exp]\!] \\
[\![\cdot]\!]_C &: [cmd] \rightarrow [\![cmd]\!] \\
[\![\cdot]\!]_Q &: [qry] \rightarrow [\![qry]\!] \\
[\![\cdot]\!]_N &: [exp] \rightarrow PBStructure \rightarrow [\theta]
\end{aligned}$$

The semantic function $[\![\cdot]\!]_N$ evaluates an expression to an identifier bound in the indicated PB Structure having the characteristics of the expression. This function uses the ordering defined in Proposition 38. The semantic function $[\![e]\!]_N$ gives a theory presentation identifier C such that C is the least theory presentation in the presentation base structure having the property that $C >_p e$, for any expression e (assuming that e is in canonical form). The value of $[\![\cdot]\!]_N$ will be the theory presentation that has the closest structure to the structure of the expression e, if one exists. The theory presentation having the closest structure to an expression e is defined to be the least theory presentation e' such that $e \leq e'$.

The semantics function $[\![\cdot]\!]_N$ is used by the other semantic functions to restrict meaning to those theory presentations that have been identified through some presentation base command.

4.3.4 Semantic Equations

The equations make use of the coercions defined in Section 4.2.6. The coercion $A \vdash B$ is a filter on a set, selecting only those elements whose types A may be coerced to B. In

the section on the commands, we use a notation extending environments. The notation $\{u^n | u^n(C) = V\}$ specifies that the environment u^n is extended such that the identifier C is now in the domain of u^n having a value of V.

Naming

$$[\![A]\!]_N p \quad = \quad A \text{ if } A \in N_p$$

$$[\![\textbf{extend } e\ [(\iota_1, c_1, x_1), \cdots, (\iota_n, c_n, x_n)]]\!]_N p =$$

$$\begin{array}{lll} & \text{let} & A = [\![e]\!]_N p \\ & \text{and} & \{B \mid B \geq A \in p \\ & & \quad \& \ \pi^2(B) \vdash \iota_1 \mapsto c_1 \& \cdots \& \iota_n \mapsto c_n\} \\ & \text{and} & B' = \text{ the least such } B \\ & \text{in} & B' \end{array}$$

$$[\![\textbf{combine } e_1\ e_2]\!]_N p =$$

$$\begin{array}{lll} & \text{let} & A = [\![e_1]\!]_N\ p \\ & \text{and} & B = [\![e_2]\!]_N\ p \\ & \text{and} & \{C \mid C \geq A \in p\ \&\ C \geq B \in p\} \\ & \text{and} & C' = \text{ the least such } C \\ & \text{in} & C' \end{array}$$

$$[\![\textbf{generalize } e_1\ e_2]\!]_N p =$$

$$\begin{array}{lll} & \text{let} & A = [\![e_1]\!]_N\ p \\ & \text{and} & B = [\![e_2]\!]_N\ p \\ & \text{and} & \{C \mid C \leq A \in p \& C \leq B \in p\} \\ & \text{and} & C' = \text{ the greatest such } C \\ & \text{in} & C' \end{array}$$

$$
\begin{aligned}
&[\![\textbf{instantiate } e_1\ e_2\ [f]_A\ [g]_B\ [h]_C]\!]_N p = \\
&\quad \text{let} \quad A = [\![e_1]\!]_N\ p \\
&\quad \text{and} \quad B = [\![e_2]\!]_N\ p \\
&\quad \text{and} \quad \{C | C \geq A \in p\ \&\ C \geq B \in p\ \& \\
&\qquad \pi^2(C) \vdash \\
&\qquad \iota_{C_1} \mapsto \pi^2(A.\iota_{A_1})(\pi^2(B.\iota_{B_1}), \cdots, \pi^2(B.\iota_{B_n})) \& \cdots \& \\
&\qquad \iota_{C_m} \mapsto \pi^2(A.\iota_{A_m})(\pi^2(B.\iota_{B_1}), \cdots, \pi^2(B.\iota_{B_n}))\} \\
&\qquad\quad \iota_{A_i} \in [f]_A \\
&\qquad\quad \iota_{B_j} \in [g]_B \\
&\qquad\quad \iota_{C_k} \in [h]_C \\
&\quad \text{and} \quad C' = \text{ the least such } C \\
&\quad \text{in} \quad C'
\end{aligned}
$$

$$
\begin{aligned}
&[\![[(\iota_1, c_1), \cdots, (\iota_n, c_n)]]\!]_N\ p = \\
&\quad \text{let} \quad \{B | B \in N_p \& \pi^2(B) \vdash \iota_1 \mapsto c_1 \& \cdots \& \iota_n \mapsto c_n\} \\
&\quad \text{and} \quad B' = \text{ the least such } B \\
&\quad \text{in} \quad B'
\end{aligned}
$$

The definition of the semantic function for extensions can be written differently allowing more flexibility in how a PB structure is examined for consistent types. In the definition above, extensions are identified by their feature identifier and the identifier of the Level 1 term that this feature identifier denotes. This expression is equivalent to asking: are there any theory presentations in the presentation base structure p_i having the features $\iota_1, \cdots, \iota_n$ mapping to values $c_1, \cdots, c_n$. We could drop the c_i, which would equate to the question: are there any theory presentations having the features $\iota_1, \cdots, \iota_n$. Or we could add in the abstract value information, asking: are there any theory presentations having features $\iota_1, \cdots, \iota_n$, mapping to values $c_1, \cdots, c_n$ which in turn map to LF encodings $x_1, \cdots, x_n$.

All expressions that refer to a least or greatest in the set are not defined if there is no least or greatest member, modulo $=_{FT}$. If this happens, it means that the expression is underdetermined, and refers to more than one theory presentation, none of which can be

said to be a better fit than the others. If there has been specifically a theory presentation constructed and hence bound that had precisely those characteristics, there would in fact be a least (or greatest) in the indicated presentation base structure.

Queries

$$[\![\mathbf{Q}(e)]\!]_Q p\mathbf{f} = [[[\![e]\!]_N p]^2 u^2]^1 u^1$$

Queries evaluate expressions to a name that matches the construction specified and evaluate that identifier in the environment denoted by the presentation base structure.

For example, the query **Q**(**combine** A B) will evaluate the presentation expression **combine** A B to find the identifier of a theory presentation in the PB structure p that is an extension of the combination of A and B. It will then evaluate this identifier in the environment specified by p to an admissible context, giving an LF theory presentation including both the statements of A and of B.

Expressions

$$[\![e]\!]_E\mathbf{f} = \{p | p \in \mathbf{f} \ \& \ [\![e]\!]_N p \mathit{defined}\}$$

Expression evaluation selects from the frame all the PB structures where the expression has meaning. For example, $[\![$**combine** A $B]\!]_E\mathbf{f}$ returns a new frame all of whose PB structures have an identifier whose meaning is consistent with (is an extension of) the combination of A and B.

Commands

Commands add a new PB structure to the set **P**, returning a new frame. The new PB structure in the frame has a new node added whose meaning is the admissible context denoted by the expression.

In the following equations, the operator | is used to indicate that a set or function is extended with a new value or mapping. So $\{u^3 | u^3(C) = \langle\rangle\}$ specifies that the environment

u^3 is extended such that it now maps the identifier C to the empty sequence. C is presumed to be fresh.

$$
\begin{aligned}
&[\![C \equiv \textbf{combine } e_1\ e_2]\!]_C\ p\ \mathbf{f} = \\
&\quad \text{let} \quad A = [\![e_1]\!]_N p \\
&\quad \text{and} \quad B = [\![e_2]\!]_N p \\
&\quad \text{and} \quad p' = (N_p \cup \{C\}, \{\leq_p | C \geq A, C \geq B\}, \\
&\qquad\qquad \{u^3 | u^3(C) = \langle\rangle\}, \\
&\qquad\qquad \{u^2 | u^2(C) = [\![A \cup B]\!]^2 u^2\}, u^1) \\
&\quad \text{in} \quad \mathbf{f}' = (\mathbf{P_f} \cup \{p'\}, \{<_{\mathbf{P}} | p <_{\mathbf{P}} p'\}, \\
&\qquad\qquad \{M(p)\}_{p \in \mathbf{P_f}} | M(p') = Ctxt(p'))
\end{aligned}
$$

$$
\begin{aligned}
&[\![C \equiv \textbf{generalize } e_1\ e_2]\!]_C p\ \mathbf{f} = \\
&\quad \text{let} \quad A = [\![e_1]\!]_N p \\
&\quad \text{and} \quad B = [\![e_2]\!]_N p \\
&\quad \text{and} \quad p' = (N_p \cup \{C\}, \{\leq_p | C \leq A, C \leq B\}, \\
&\qquad\qquad \{u^3 | u^3(C) = \langle\rangle\}, \\
&\qquad\qquad \{u^2 | u^2(C) = [\![A \cap B]\!]^2 u^2\}, u^1) \\
&\quad \text{in} \quad f' = (\mathbf{P_f} \cup \{p'\}, \{<_{\mathbf{P}} | p <_{\mathbf{P}} p'\}, \\
&\qquad\qquad \{M(p)\}_{p \in \mathbf{P_f}} | M(p') = Ctxt(p'))
\end{aligned}
$$

$$[\![C \equiv \textbf{extend } e_1[(\iota_1, c_1, x_1), \cdots, \iota_n, c_n, x_n)]]\!]_C p\ \mathbf{f} =$$

$$\begin{array}{ll}
\text{let} & A = [\![e_1]\!]_N p \\
\text{and} & p' = (N_p \cup \{C\}, \{\leq_p | C \geq A, \}, \\
& \quad \{u^3 | u^3(C) = \iota_1 \mapsto c_1, \cdots, \iota_n \mapsto c_n\}, \\
& \quad \{u^2 | u^2(C) = \\
& \quad\quad [\![A \odot [\iota_1 \mapsto c_1, \cdots, \iota_n \mapsto c_n]\!]^2 u^2\}, \\
& \quad \{u^1 | u^1(c_1) = x_1\}) \\
\text{in} & f' = (\mathbf{P_f} \cup \{p'\}, \{<_{\mathbf{P}} | p <_{\mathbf{P}} p'\}, \\
& \quad \{M(p)\}_{p \in \mathbf{P}_f} | M(p') = Ctxt(p'))
\end{array}$$

$$[\![C \equiv \textbf{instantiate } e_1\ e_2\ [f]_A[g]_B[h]_C[c_{i_1}, \cdots, c_{i_n}]]\!]_C p\ \mathbf{f} =$$

$$\begin{array}{ll}
\text{let} & A = [\![e_1]\!]_N p \\
\text{and} & B = [\![e_2]\!]_N p \\
\text{and} & p' = (N_p \cup \{C\}, \{\leq_p | C \geq A, C \geq B\}, \\
& \quad \{u^3 | u^3(C) = \iota_{C_1} \mapsto A.\iota_{A_1}(B.\iota_{B_1}, \cdots, B.\iota_{B_n}), \\
& \quad\quad \cdots, \iota_{C_m} \mapsto A.\iota_{A_m}(B.\iota_{B_1}, \cdots, B.\iota_{B_n})\}, \\
& \quad \{u^2 | u^2(C) = \\
& \quad\quad [\![A \cup B \odot u^3(C)]\!]^2 u^2\}, \\
& \quad \{u^1 | u^1(c_{i_1}) = \\
& \quad\quad [\![[\![A.\iota_{A_1}(B.\iota_{B_1}, \cdots, B.\iota_{B_n})]\!]^2 u^2]\!]^1 u^1, \cdots, \\
& \quad\quad u^1(c_{i_m}) = [\![[\![A.\iota_{A_m}(B.\iota_{B_1}, \cdots, B.\iota_{B_n})]\!]^2 u^2]\!]^1 u^1 \\
\text{in} & f' = (\mathbf{P_f} \cup \{p'\}, \{<_{\mathbf{P}} | p <_{\mathbf{P}} p'\}, \\
& \quad \{M(p)\}_{p \in \mathbf{P_f}} | M(p') = Ctxt(p'))
\end{array}$$

$$
\begin{aligned}
&[\![C \equiv [(\iota_1, c_1), \cdots, (\iota_n, c_n)]]\!]_{C}\mathbf{pf} = \\
&\quad \text{let} \quad A = [\![[(\iota_1, c_1), \cdots, (\iota_n, c_n)]]\!]_{N}p \\
&\quad \text{and} \quad p_{i+1} = (N_p \cup \{C\}, \{C \geq A\}, \\
&\qquad\qquad \{u^3(C) = \langle\rangle\}, \\
&\qquad\qquad \{u^2(C) = [\![A]\!]^2 u^2\}, \\
&\qquad\qquad u^1) \\
&\quad \text{in} \quad f' = (PB_{\mathbf{f}} \cup \{p'\}, \{<_{\mathbf{P}} | p <_{\mathbf{P}} p'\}, \\
&\qquad\qquad \{M(p)\}_{p \in \mathbf{P}_f} | M(p') = Ctxt(p'))
\end{aligned}
$$

4.3.5 Properties

The consistency of Level 3 relies on Level 2. The structure of the frames preserves the Level 2 meanings such that if $p_i <_{\mathbf{P}} p_j$ and p_i and p_j are admissible, then $Ctxt(p_i)\langle|^* Ctxt(p_j)$ and $M(p_i)\langle|^* M(p_j)$.

4.4 Summary of Results

This concludes the formalization of *MF*. We have shown a formalization that limits manipulable objects to named theory presentations and supports persistence through the frame structure of Level 3. Level 3 supports re-use and queries by allowing us to retrieve theory presentations that are consistent with a structural specification as embodied by the presentation expressions. We separate the persistent character of a development from the space of all constructible artifacts, limiting the world of theory presentations to those we have identified.

Level 2 gives a semantics to the structure of theory presentations with little expectation or knowledge of the underlying base language. We relate the theory of sequences to admissible contexts in the underlying language, provide a syntactic approach to manipulating theory presentations (the formation trees), and show a correspondence to the semantic objects (the admissible contexts) under certain restrictions (no collision of identifiers unless they map to the same values).

Feature terms were originally introduced to provide a way to locate theory presentations with certain characteristics through unification. This may still be possible in future developments. However, in the current system they serve a simpler role.

Features support a global name space so there are no duplication of feature identifiers. The meaning of a sequence is given as the list of feature identifiers as a way to describe what elements are introduced in the sequence without reference to the actual values. Feature terms provide a mechanism to identify when elements in a sequence are introduced in one theory presentation as opposed to several unrelated presentations. They provide a level of independence that allows us to provide a different interpretation for features (allowing features to represent the sequence independent of the meaning given to the elements of the sequence). However, the features of *MF* terms, as they exist now, are much closer to record labels than to true feature terms.

In Chapter 6 we discuss some proposed future extensions of *MF* that would allow us to explore more complex maps from one presentation base structure to another.

Chapter 5

Comparisons and Conclusions

As noted earlier, the research presented here draws on several research communities: database and knowledge representation, type theory, and specification languages. Until recently, no one of these areas provided work that was directly comparable to *MF*. The goal of *MF* was to find a way to make better use of the expressivity and the power of the higher-order constructive logics. These logics are used most often to capture specifications (of programs or mathematical theories) and to express properties (of programs or mathematical theories). The research reported here explores the use of structure to achieve this goal: maintaining a strong separation of structure and theory content, deferring commitment of the nature of a base language or the primitives required for theory development, providing a theoretical structure to support persistent storage of and access to this structural information, and, lastly, preserving the semantic conditions (i.e., the necessary context) needed by a particular specification through the structural relationships.

By intent, *MF* does not provide any support for a specific methodology of algorithm development nor does it make use of any automated support tools that aid in refinement. A consequence of a strong separation of structure from content is that specific algorithm development is associated with the content. There are existing tools and systems that focus on this content-oriented development (such as *KIDS* [45] and *NuPRL* [13]). Additionally, *MF* does not provide any support for a realization of a specification. Again, there are systems whose sole goal is to take constructive specifications and extract their computable component (such as Christine Paulin-Mohring's [36]). These different components (structure, algorithm refinement, and program realization) could be used together

to provide support for the different aspects of program development: algorithm design, refinement, implementation, and verification.

I will look at systems that have provided similar or overlapping capabilities: *Automath* [8], *CLEAR* [11], Institutions [21], [41] , consequence relations in LF [24], *ELF* [37], Extended ML [42], a system of *Deliverables* [9], and Specware [49]. Each of these formalizes a particular activity associated with program or theory development, with the Specware system being the closest in terms of its stated goals.

5.1 Automath

The goal of Automath ([8]) was to develop a system for writing mathematical theories such that verification of the correctness could be performed. Some of de Bruijn's identified criteria were to:

1. Define a system that will accommodate all theories involved because many errors occur in the assumptions as to the nature of the theories.

2. Provide a notion of a book of theorems (although the notion of a book appears implicit with respect to the logic).

3. Implement the basic proof methodology as functions, β-reduction, and typing.

4. Allow type inclusion and partial typing (Aut-QE). Aut-QE allows a restricted form of sub-typing: the type expression $T : type$ may be used in place of $T : [u : U]type$ which may in turn be used in place of $T : [u : U][v : V]type$.

5. Distinguish a proposition from its proof(s): for an arbitrary proposition p, the proof of p has the type $proof(p)$. Proofs are therefore not confused with the type or proposition, as may occur in COC.

While their primary goal was to provide a means to fully prove the correctness of theories, the project incorporated many of the themes of *MF*. Their focus on checking or validating a proof obscures some assumptions that the theorist may already have made.

These assumptions span several activities in the specification or theory development process:

1. We can only prove statements that can actually be expressed: the proofs in the system are highly influenced by the language provided and the meta-theory supported. Fixing on a particular language, as Automath does, commits to some extent the kinds of theories that will be explored.

2. The exposition of mathematical theories is dependent on the use of axioms, definitions, lemmas, theorems, and proofs. These meta-logical concepts must be independent of the underlying language if we are to be able to explore their role and their properties.

3. The interpretation of the theories should be expressible as well and independent of the underlying language.

4. Automath distinguished informally the role of a book and the context of mathematical theories. A book is a complete presentation of a mathematical theory or set of theories while a context is defined to be:

   - The set of assumptions considered valid at a point in time.
   - The set of variables active at that point.
   - The set of all theorems, lemmas, assumptions that have been previously developed.

   How this distinction can be captured is left unexplored in Automath.

While the above were concerns discussed in the Automath literature, they were not all equally well explored in the Automath project. In particular, two (related) goals are to maintain a distinction between the object and meta-level and facilitate understanding about how theories are constructed. Level confusion is a very common problem in the domain of theory specification. It is a common pitfall to try and put too much information at one level, obscuring what one is trying to emphasize. *MF* makes a strong distinction

between the structural (the meta-level) and the underlying logical framework (the object level).

As part of the investigation, several Automath languages were developed, each having different capabilities as a base language, such as type variables, λ-expressions, higher-order capabilities, pair extensions, syntactic handling extensions. Some of these modifications were to the underlying language, while some were more in the nature of pre-processor. Automath is an initial definition of a family of languages: its primary aim was to explore to what extent these languages could implement their program of automating mathematical proofs.

Some things that were not included in Automath:

1. The underlying language was not necessarily constructive. So, while theories could be constructed and proved, the resulting objects were not necessarily executable.

2. The part of the context consisting of previously developed notions was implicit in the theory construction. So the order of the required contexts and the resulting context itself was not something that we could make judgements about or manipulate.

3. They did not investigate theory construction as a process, but as a static description.

4. They did not have general higher order types, by intent. This limited their uniformity, although at least one of the Automath languages was higher order, Aut-SL [8].

By using a constructive, higher order language as the base language for theory construction in *MF*, we are able to provide a fully expressive language in which to describe arbitrary theories. By separating the structural information from the statements that compose the theories, we can explore the ways in which theories are related to their components, independent of the interpretation or meaning of the theories. The construction of a theory presentation represents a particular sequence. More than one sequence may map to the same theory. The sequence is represented in *MF* through the $\leq_P$ relationship on theory presentations. In *MF*, the equivalent concept of the Automath book is the downward closure or structural environment defined by a theory in a presentation

base structure. Unlike Automath, multiple books may exist at the same time, sharing parts through their structure. Automath also had no modularity of theories: there was no structure aside from the primitive notions of theorem (lemma, etc). The burden of completeness is still on the theorist.

Automath was a very early system. Many of the issues that are explored in *MF* are a direct outgrowth of the work in Automath. *MF* has taken much further the separation of the structure from the presentation of a theory.

5.2 Clear

Clear ([11]) is a specification language that supports algebraic specification of theories. Clear theories have a well-defined structure of sorts, operations, and equations, which are interpreted in a category of theories. Theories are constructed from other theories by specifying theory morphisms: how the signatures are changed, additional sorts, operations, and equations.

The basic theory operations in *Clear* are:

- introduce a theory: introduce the sorts, operations, and equations of a theory.
- combine two or more theories: combination is interpreted as the coproduct (colimit) of the theories in the category of theories.
- enrich a theory: add new sorts, operations, and equations.
- instantiate a theory, replacing sorts or operations with new ones.
- use pushouts to apply generic theories to their actual arguments, constructing a theory morphism (a procedure).
- use diagrams as environments to keep track of shared theories.

Theory morphisms preserve structure and allow sharing of common structure: If a theory is constructed from two theories that share a common substructure, only one such substructure will be identified. Theories with their supporting substructure are known as *based theories*.

Procedures may be constructed that take theory arguments and produce a method that can produce a new theory when applied to appropriate arguments. The match of actual theory argument to formal theory argument is made via a fitting morphism. The resulting theory has as part of its base the actual theories that were used in the procedure.

Additionally, theories may be constrained to be free with respect to their constructors: the algebras satisfying the theory must produce all their elements only from the specified operations and the elements must all be distinct, except where equality equations force them to be equal. These are called data theories.

In contrast, *MF* theory presentations have no specific structure or type. There are no theory presentation procedures nor are there any data theories. However, both of these facilities are supported in a uniform way by the underlying logical framework. This lack of (internal) structure limits (in some ways) what can be done with presentations at the meta-theoretic level, while allowing the base language the expressive freedom to support a notion of theory appropriate to the problem domain. In *MF* a presentation is constructed from other presentations only by extension, inclusion, generalization, or instantiation. Because a signature of sorts and operations cannot be assumed, nor the presence of equations, the structural operations on presentations are confined to a more abstract notion of structure. Constraints on models, such as those presented by freeness constraints of data theories, are properties of the base language, rather than *MF*. Parameterization (the construction of theory morphisms) is treated similarly.

The *MF* approach presents some advantages. By changing the underlying base language of *MF*, one can change not only how one presents a theory, but also the domain of discourse. As an example, if the underlying base language were a language of deliverables (as discussed in Section 5.7), then the denotation of a presentation would be the deliverables, the structure would correspond to the derivation of the deliverables. If the underlying base language were order-sorted equational algebras, we would have a system very similar to that of *Clear*.

The aspect of *MF* and *Clear* that is most comparable is the role of structure. Because *Clear* is defined in the category of theories, an environment is represented as a diagram in this category. As in Chapter 3, the structural environment of a theory refers

to its environment or context. For theories that introduce no new informational content, such as $A \cup B$, the structural environment is the theory A (and its structural environment) and the theory B (and its structural environment). For theories that introduce new information, such as the extension $A \odot F_A$, the structural environment is the theory A (and its structural environment). A diagram in *Clear* is very similar to the structural environment in *MF*: the structural environment is a description of a (portion of a) lattice in a presentation base structure. In *Clear*, if we had two specifications: *total_order* and *partial_order*, and we intended to specify a topological sort, we might enrich the colimit of *total_order* and *partial_order* with the operations and equations for a topological sort, assuming that the existing specification had the required theory specifications for total order and partial order. In *MF*, we would construct a similar sounding specification: **extend**(**combine** *total_order partial_order*)[*featurelist*], where (**combine** *total_order partial_order*) is the required structural environment and [*featurelist*] are the introduced features: the operations and equations for the topological sort.

In *Clear* the colimit specifies a diagram in the category of theories. In *MF*, the structural environment has no meaning unless the presentations *total_order* and *partial_order* are defined in the specified presentation base structure, and there is some theory presentation A in the presentation base such that A is an extension of both *total_order* and *partial_order*. The new theory will then be an extension of A. This constraint restricts theory presentations in *MF* to those that have been identified by a designer. This identification makes the statement that the theory composed of (*total_order* $\cup$ *partial_order*) is something meaningful that we wish to retain in our structure.

The lattice associated with the structural environment (*total_order* $\cup$ *partial_order*) has a least upper bound: A. The formation tree for A identifies exactly how A was constructed, and the denotation of A would be a sequence of statements in the underlying logical framework. If we hypothesize a map m from one presentation base structure to another, preserving the structural relationships (and the intended meanings), then if A is defined in the presentation base structure p, then $m(A)$ is defined in the presentation base structure $m(p)$, and the structural environment for $m(A)$ in $m(p)$ is consistent structurally with the

structural environment for A in p (meaning that if $A \leq A'$ in p, then $m(A) \leq m(A')$ in $m(p)$). In this sense, the structural environment acts to parameterize a theory over many compatible environments. The actual environment is derived from the presentation base. The structure can be thought of as capturing the interpretation relationship: the map m maps specifications to interpretations. (For some thoughts on extensions to the role of structural environment, see the Chapter 6.)

The example above can be thought of as instantiating an environment, in the sense that the description of a structural environment in a theory presentation construction may be instantiated by more than one actual environment. A related idea is presentation specialization (or theory instantiation). *Clear* has the notion of a theory *apply* that allows a *Clear* procedure to have a parameter. The application of a *Clear* procedure to a theory (under a fitting morphism demonstrating the correctness of the application) is another theory specification. Like environment instantiation in *MF*, a theory may be applied to many different theory parameters, with different fitting morphisms. In *MF* the equivalent to a *Clear* apply is accomplished by theory instantiation. Because the underlying language to *MF* is expected to be a logical framework with β-reduction, *MF* relies on this capacity to support theory application. In the initial design, a theory presentation is assumed to have a common context on which all the sentences are abstracted. (There are various relaxations to this constraint suggested in Chapter 6.) A theory can be specialized to particular values by instantiation. If P_3 is the theory presentation **instantiate** P_1 $P_2[f_1, f_2][g_1, g_2, g_3][h_1, h_2][c_1, c_2]$, then P_3 has a structural environment of $P_1 \cup P_2$, with P_1 having the features f_1 and f_2, and P_2 having the features g_1, g_2, and g_3 and is extended with features h_1, h_2 such that the denotation of h_1 is the denotation of f_1 applied to the denotations of g_1, g_2, g_3, and similarly for h_2. Structurally, this application can be captured as:

$$\begin{aligned} h_1 &\mapsto P_1.f_1(P_2.g_1, P_2.g_2, P_2.g_3) \\ h_2 &\mapsto P_1.f_2(P_2.g_1, P_2.g_2, P_2.g_3) \end{aligned}$$

meaning that the features f_1, f_2 in theory presentation P_1 are applied to the features g_1, g_2, g_3 in theory presentation P_2, discharging the (assumed) context elements in the underlying object theory.

In *Clear*, the dependence of a theory specification on a parameter theory is indicated by specifying a formal parameter X with a theory "metasort". For instance, we might have a procedure Q with a parameter $X : Poset$, meaning that X needs to be a partially ordered set, where *Poset* itself is a theory specification. The result of the application is a *Clear* specification. In *MF*, Q would not be a distinguished linguistic element (a procedure). Q itself would be an *MF* theory presentation whose underlying sentences must have the form that they are abstracted on the required object level witnesses, for instance a $\leq$ operator. Because the burden of fitting the witness to the assumption needs to be checked at the object level, theory instantiation has a component that is structural (which theory presentation and which features in the theory presentation are used to discharge assumptions) as well as an object language level component (proof that the witness is indeed of the correct type to discharge the assumption). Chapter 6 discusses extensions to the theory instantiation model that manipulates the abstraction contexts directly.

The remaining roles of the *MF* environment have no counterpart in *Clear*. The persistent presentation base is an extension of the *Clear* environment: *Clear* environments are not persistent. The structural environment is a mediator between the presentation base and the designer, by constraining what presentations are to be referenced. During construction, it identifies a location (a node in a diagram, a position in the presentation base lattice) in the presentation base. During a query, it identifies possible solutions (diagrams, structural environments). It acts as both a prescription for how the underlying logical system must be extended to support a presentation, and as a descriptor of the logical system produced by these extensions.

The essential differences between *Clear* environments and *MF* environments and presentation base is the persistence and reusability of the presentations and environments in *MF*, the freeing of parameterized specifications from a specific presentation to satisfying structural environments (lattices, diagrams), and the move of the details of presentation instantiation (application) to the object level.

5.3 Institutions

The goal of Goguen and Burstall's work on *Institutions* ([21], [41]) is to:

1. Support as many computer science features as possible independently of the underlying logical system.

2. Facilitate the transfer of results from one logical system to another.

3. Permit the combining of different logical systems naturally.

To accomplish this, they formalize the role of a logical system very abstractly to allow multiple formal systems to be cast in the role of an Institution. They introduce a satisfaction relation between models and sentences that allows a change in notation, focusing on the language features being defined rather than the specific syntactic and semantic details used to define the language feature. An institution is a category of signatures with associated sets of sentences, models, and a satisfaction relationship. In particular, *Clear* is a specification language designed to work with any institution. Because a central feature of specification languages is to facilitate the construction of large theories from available smaller theories, *Institutions* formalize this feature as a colimit of the small theories connected by theory morphisms.

Goguen and Burstall state several main results from this research, two of which are of particular interest in the context of the research presented here. The first main result states that any institution that allows signatures to be fitted together will allow theories to be fitted together, where signatures define the notation, and theories are collections of sentences over a particular signature. Another result shows how theory structures are preserved during institution morphisms.

MF is not really an institution. Institutions are semantically organized, describing generalities of logical systems and supporting morphisms between logical systems. Much as *Clear* is a specification language designed to work with any institution, *MF* is a specification language that can work with a variety of underlying logical systems, and hence with a variety of institutions. But, because *MF* has no innate language of specification, the underlying language to *MF* must itself be an institution. The definition of institution

provides a specific structure to the base language. An institution is defined by giving a signature, a functor mapping each signature to a set of sentences over that signature, a functor associating a model to each signature, and a satisfaction relation between models and sentences that is preserved over signature morphisms. Because *MF* may rely on an institution, if an institution allows signatures to be fitted together, then *MF* will as well. If the *LF* supports the fitting together of signatures through β-reduction, matching operator names to operator names, and functions to functions, then the model of *MF* theory application and the facility provided by the *LF* correspond directly. If we had an institution that supported signature fitting via a different mechanism than β-reduction, assuming the different mechanism were comparable, a map defining the institution's signature fitting mechanism in terms of the *MF* notion of theory application would be sufficient.

The preservation of theory structure during institution morphisms supports the *MF* separation of structure from content, where the content is expressed in the institution and the structure is expressed in an *MF* presentation base structure. If an institution morphism preserves theory structure, then the structure as expressed in *MF* will be preserved as well.

Institutions contrast with the work on structure and representation in *LF* presented in the next section. The *Institution* abstraction is based on model theory, whereas consequence relations are based on proof theory. *MF* is closer to the proof theoretic school in its syntactic treatment of theory structure. What is particularly appealing about the institution concept is its ability to capture what properties are necessary in the institution and to provide for institution morphisms, allowing a change in the base language, while preserving the structure of a theory development.

5.4 Structure and Representation in LF

Harper, Sannella, and Tarlecki have proposed a framework for structured theories in a logical framework by lifting presentations in the object logic to the metalogic of the framework ([24]). Like the work on *Institutions*, it provides an abstract and unifying way of viewing logical systems.

Because one of the goals is to conduct all the inferential activity of the object logic in

the logical framework, LF must represent the object logic in sufficient detail to capture the relevant properties. They study the use of theory structure in order to control behavior in the LF encoding of the object logic. They take this idea further in searching for a structure for logical systems themselves, allowing them to be defined in a similar modular fashion, with the ultimate goal of defining both theories and logics in one logical framework.

This uniform approach to structure is common. It has the advantage (much like Extended ML) that one formal system may be used to describe differing properties so that only one formal system need be examined. The LF type system supports the encoding of various object logic properties, such as all syntactic apparatus, judgement forms, and inference rules. They introduce the abstraction of a logical system as a family of consequence relations indexed by signatures. This is similar to the abstraction proposed in Institutions, except that Institutions use a model-theoretic view of logical systems, while this view uses only the consequence relation.

Harper, Sannella, and Tarlecki's system is very general, categorizing the underlying logical systems in an abstract and complete framework. However the notion of structure is an embedded one: one cannot view the structure of a problem separate from the details of the implementation. Their view is to use structure to aid in the specification of logical systems in the metalogic. I am not concerned with taking over the function of the object logic. In fact, I want to use the object logic properties to do what they are intended to do. My goal is to facilitate structured descriptions in an arbitrary logic, putting the least restrictions on how the logic is used, while allowing the structure to be fully specified. In terms of modular re-use, this kind of generality seems to be mandatory. Much like the Smalltalk system provides a structured hierarchy to facilitate re-use, a structural presentation hierarchy gives a map or derivation of desired components. The work on structure and representation in *LF* intends to use structure to facilitate proof search and other object-level activities that are encoded in the metalogic. *MF* uses structure in an entirely orthogonal way: to organize development, to locate common developments, and to describe structural constraints.

MF is logic independent and *LF* independent. By changing the logical framework used as the base language, one may change the kinds of logical systems that can be described

in *MF*. Questions still to be explored are how *MF* can be used with different *LF*'s and or logical systems, each describing a different aspect, as in Institution changes.

5.5 ELF

ELF, a language for logic definition and verified metaprogramming by Frank Pfenning [37] again attempts to define a meta-language for proof manipulation that is independent of any particular logical system. Independence from particular logical systems is a strong motif, leveraging off of the common factors of logical systems. The goals of *ELF* are disjoint from those of *MF*: it is of interest here because it provides yet another example of the kinds of base languages that might be explored with *MF*.

The goal of ELF is to manipulate proof objects produced by meta-programs such as theorem provers and type inference systems. Using an operational interpretation for types, ELF is able to construct terms that can represent object-logic proofs

ELF provides meta-programming support to be used with logical frameworks. It supports the logician in constructing proofs in a logic, while *LF* supports the ability to describe the logic. *MF* fits in by describing the structure of a development. As well as exploring how structure can be preserved in going from one logical framework to another, I would like to explore the relationships between them. Is the structure of a logical system related to the structure of the ELF description? How would one incorporate a meta-programming aid such as ELF in a structural environment such as *MF*? They both use an *LF* description as a base.

5.6 Extended ML

Donald Sannella and A. Tarlecki have defined an extended version of ML to facilitate the production of correct programs from requirements [42]. Extended ML is a specification framework for program development in Standard ML. The design choices made in Extended ML were influenced by this restricted goal. Rather than aiming at a general specification language, such as *Clear*, Extended ML embodies a particular methodology

and development style for program development for successive decomposition of ML modules until an executable program is reached. The Extended ML language accommodates both specifications and executable modules in a single uniform framework. This process allows a development to exist in mixed-mode: partially specification and partially executable.

Extended ML's methodology is oriented around signatures (with axioms restricting the values given to the signature) and implementations (called structures). Structures may be built from other structures, and signatures from other signatures, yielding a hierarchy of development. This hierarchy however is not accessible as a manipulable object.

MF does not provide a methodology to support the construction of theories nor does it provide support for successive decomposition of specifications into executable modules. What it does provide in the context of program development is a tangible scoping mechanism. Like suggestions for LCF in *Structured Theories in LCF* [44], *MF* environments can be used to limit the context of search (for a solution or a proof) to a problem. Extended ML's use of signatures with axioms provides much of the detail not present in *MF* environments. Combining *MF* with an Extended ML base could provide both the structuring support needed by Extended ML and the interface restrictions needed in the base language.

5.7 Deliverables

James Hugh McKinna in his thesis *Deliverables: A Categorical Approach to Program Development in Type Theory* [9] investigates Burstall's notion of deliverables: a program along with its proof of correctness. The objects in this system are more stringently defined than in ours: an object in *MF* may indeed fit the descriptions of a deliverable, although it need not. But, in general, the aim of *MF* is the construction of correct programs, whether they are delivered in a formal structure of a deliverable or not.

The language of the deliverables could be used as the underlying base language, allowing deliverables to be defined in the structured environment of *MF*. If we could describe a map between an *LF* description and a system of deliverables, *MF* could conceivably be used

both as a development environment and a verification environment, providing different capacities for the same *MF* specification. Changing the underlying base language promotes different features, altering what is expressed. Theory development over a base language that provides a notion of deliverables could be developed in the same way as is currently presented in *MF*, but the result, rather than being a COC presentation, would be the set of deliverables. Allowing a change in deliverables also suggests that several activities such as developing proofs of correctness or extracting computational components might have parallel development environments. The goal of distinguishing propositional from computational information in *Deliverables* is reminiscent of *MF*'s goal of distinguishing structure from content.

5.8 Specware

Specware [49] was developed in the context of the KIDS (Kestrel Interactive Development System) [45] project at Kestrel Institute. Specware is defined in the context of a category of specifications and specification morphisms. Much like *CLEAR*, diagrams in this category describe system structure. A Specware specification is a finite presentation of a theory in a higher-order logic.

Specware has a specification language (*Slang*) and a built-in sort Boolean (with associated operators) as well as universal and existential quantification. *Slang* also has a set of sort constructors supporting N-ary products and coproducts, function sorts, subsorts (as pullbacks), quotient sorts, and sort axioms (restricted to renaming).

At the specification level, there are three structural morphisms: translation of a specification by renaming rules, putting specifications together using a colimit, and importing a specification and extending it with a definitional extension.

In comparing the different systems looked at in this chapter, a common problem is finding the correct level of comparison. In Specware, there are three tangible levels. The meta-level manipulates structure in either the category of Specifications or the category of Interpretations, allowing for naming diagrams. The language *Slang* is the object language for this meta-level. The second level is the language of the theories. *Slang* also forms this

language of theory specifications and interpretations, similar to Clear. There is a third level that corresponds to an implementation language, derived from the specification, such as Lisp, C, Ada, etc.

Dividing *MF* up in a similar way, we too have a meta-level: the language of structural expressions described in Level 2. *MF* Level 3 names presentations, performing the same function as the named diagrams in Specware as well as defining the presentation base. The language of specifications in *MF* is however intentionally sparse. In trying to localize what are true structural effects distinct from object-level information, *MF* has no counterpart to the *Slang* language. Instead, a specification is represented structurally as a collection of features defined in a particular context (structural environment). A feature acts as a tag: a notification that there is some object level informational content present at this point. There is no commitment to what a feature might be. It could be a hypothesis intended to be discharged. It could be a proof. It could be an operator. What can be described in a system such as *MF* is dependent on the logical framework that underlies the structural component. In this case, the assumed logical framework was the Calculus of Constructions, allowing a full range of expression for specifications.

As previously mentioned, because *MF* does not commit to a particular model of specification, some things are more easily expressible in *MF* than in Specware. Similarly, because Specware does have a model of decomposition, some things are more easily expressible in Specware. For instance, Specware supports different kinds of theories: problem specifications, implementations, and interpretations. *MF* does not have a notion of different kinds of theories, although each kind can be represented in *MF*. The more general approach of *MF* gives the user more flexibility while the Specware provides greater methodological support.

Specware maintains a distinction between a specification and an interpretation. Some theories are intended to be specifications: they do not have implementation information and are intended to be refined. Ultimately, a specification is intended to be refined via an interpretation, which gives more detailed information. For example, if we intend to interpret a total order specification as a sequence, we first give a definitional extension to the sequence to include all the total order operations implemented as a sequence, calling

this new theory presentation *total-order-as-seq*. We then give a renaming morphism from the *total order* to *total-order-as-seq*, that will allow us to map each total order operation to the equivalent operation implemented as a sequence. This pair of morphisms (the extension and the renaming) is then seen as an interpretation of total order as a sequence, with *total-order-as-seq* being a mediating specification.

Interpretations can be composed via pushouts, because the definitional extensions are closed under composition, and the renaming morphisms can be combined (assuming they do not conflict). This sequence of pushouts is called vertical composition.

Interpretations can also be combined horizontally, allowing interpretations to be fitted together, assuming that a pullback exists (meaning the common part of two interpretations, if any, is compatible), explicitly indicating how one interpretation specializes another, and using a strong equality for morphisms that allows them to be checked syntactically. This fitting together of interpretations supports the same refinement relationship as in the category of specifications: if S_1 refines S_2 and if I_1 and I_2 are interpretations for $S_1 and S_2$, respectively, then I_1 refines I_2 in the category of interpretations.

Unlike Specware, *MF* does not have a distinct notion of a category of interpretations separate from the specifications. Like other decisions made in the separation of structure from content, the structure determines equally a specification or an interpretation. The meaning of a theory presentation in *MF* is its object-level statements, which are recorded via feature values in a particular context. To accomplish the equivalent an interpretation in *MF*, we would use application. Application is vertical composition in that it is implemented as an extension of two theory presentations. If we had a theory presentation for a *total-order*, where we might intend to implement the total order via different data structures, we would allow for this possibility by providing a context intended to be discharged delineating what components needed a witness. We might then also have a theory presentation for a *sequence*. At some future point, we could apply *total-order* to a *sequence*, providing the necessary feature correspondence, and have a theory that is specialized to sequences behaving as *total-orders*. If we also had a theory presentation for a list, we could similarly apply *total-order* to a *list* (assuming the constructions were type correct in the underlying logical framework). Further, if the sequence theory had a context that could

be discharged allowing it to be interpreted as yet another data structure we could have the following (allowing for some simplification of notation): *totalorder · sequence* (a total order interpreted as a sequence), *totalorder · list* (a total order interpreted as a list), and *sequence·list* (a sequence interpreted as a list). We could also get *totalorder·(sequence·list)* (a total-order interpreted as a list acting as a sequence).

The question of horizontal composition is somewhat different. Suppose that we have *totalorder · sequence*, and a *partialorder · sequence*. We could also say that we had a definitional extension of the colimit of *totalorder* and *partialorder* providing the specification of a *topologicalsort*. Similarly, we could extend *totalorder · sequence* and *partialorder · sequence* to get a *topologicalsortwithsequences*. Is there then a relationship between *topologicalsort* and *topologicalsortwithsequences*? If we can now discharge the required features of *topologicalsort* with the appropriate features we could get an identical theory in the underlying object language, yet these two theory presentations were derived via *MF* combinations that are unrelated. In *MF*, this relationship is not explicit, though it is derivable by defining a relationship between feature specifications and introducing a derivable relationship. This is discussed in more detail in the next chapter on future extensions.

5.9 Contributions of *MF* and Conclusions

Institutions, *Clear*, and (to a less extent) Specware are based on a semantic approach to program development. *Clear* gives a notation for operators on theories, where theories are sentences in a logical system. Institutions generalize the relationship of the theories to the models and gives the conditions necessary for proving that satisfaction of theories by models is preserved under transformations.

A consequence of the semantic approach is that the same semantic theory content is embedded in more than one place. A *Clear* specification includes not only the structural pieces and how they are connected, but also a definition of a theory (in *Clear* a theory is a signature and a set of equations). The denotation of a specification in *Clear* is a set of sentences in some logical system. In this sense, we could say that *Clear* combines a

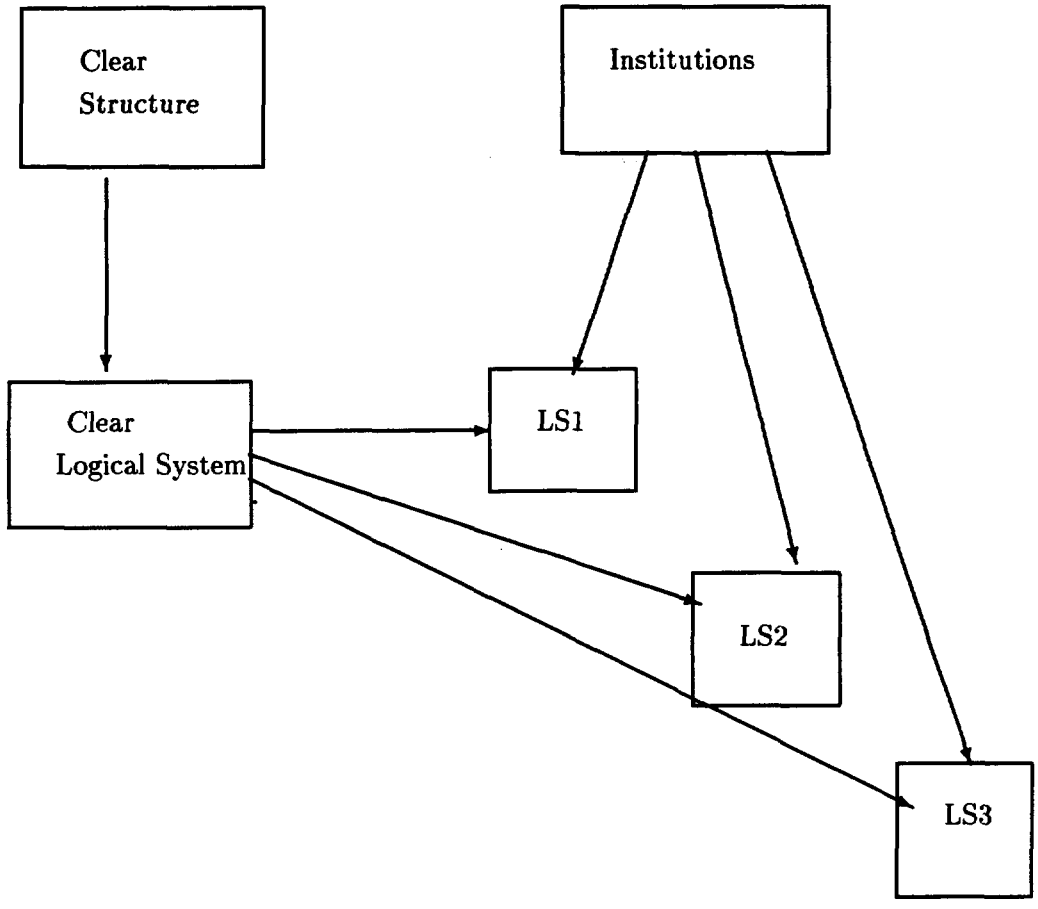

Figure 5.1: Relationship of Clear to Institutions

logical system (the one for defining the operations and equations) as well as a rich set of structured operations for fitting together these sentences. When combined with an institution (or several institutions), we would have the relationship shown in Figure 5.1. (Each of LS1, LS2, and LS3 represents a different logical system.)

By using morphisms from the Clear specification to other logical systems, and from logical systems to other logical systems, a Clear specification can be made to apply to different theories over different logical systems (assuming that semantic conditions are met for the morphisms).

This combining of content and structure in Clear makes it difficult to see what contribution each makes in theory development. Moreover, the question arises as to the nature

of the actual language used in Clear. Is it distinguished in some way? Are the elements provided necessary? Unique?

Specware continues in this tradition. While retaining a similar style of specification language, the Specware developers further refined specifications into the roles that a specification may take (problem type, specification, refinement, interpretation). Structure (diagrams and shapes) is more intrinsic than in Clear, supporting sequential and parallel composition. The question that *MF* explores is to what degree these structural benefits can be realized without the associated semantic content.

In our description of Automath, we mentioned a goal of distinguishing a theory from its interpretation(s). Specware supports this distinction by distinguishing the category of specifications from interpretations, and showing that refinements preserve the structural relationships. *MF* does not support the roles of interpretations and specifications. The relationship of a specification to an interpretation can be captured via theory specialization and β-reduction (and environment instantiation). Using theory specialization has the advantage of allowing any theory presentation to assume a different role at different times. The disadvantage of this approach is that it does not support the distinction of a specification from an interpretation in a meaningful way at the level of the *MF* theory presentation language. This distinction is an aspect that we would like to explore more. Can we answer questions such as determining what are all the interpretations for a specification in a given presentation base? Can we compare interpretations? Are interpretations themselves related if they were not constructed explicitly via one or more of the *MF* commands? Some of these issues are discussed in Chapter 6.

As shown by the survey above, much work centers on unifying a view of some aspect of program development (refinement, proof, specification) over a wide variety of logical systems. Automath, as an early system, looked at a family of logical systems, in an attempt to do a very general task. Subsequent research has looked more to generalizing what these families need look like in order to accomplish a given task (design, specification, proof, etc.) Work such as Clear, Institutions, structure and representation in LF, and ELF all try to abstract out an essential component of theory building, program specification, and proofs of correctness. and formalize that component regardless of the underlying logical

system. Like them, *MF* also tries to abstract out a critical component of theory building. The use of these abstractions categorizing the logical systems themselves are suggestive. I would like to investigate combining *MF* with different base languages providing different facilities, and combining *MF* with a more general system, using the structural component of *MF* to structure the activity of the companion language.

In contrast, systems such as Extended ML and Deliverables attempt to do very specific tasks in a specific language. I would like to push the structural abstraction to these more specific languages to investigate what leverage they provide, both alone and in conjunction with a logical framework.

MF limits its theory combining operations to strictly structural relationships, making it an appropriate model to investigate to what extent structure can be shared independent of particular underlying languages. This limitation allows us to both use *MF* across a wide variety of similar languages as well as with more special-purpose languages such as those suggested in the work on Deliverables.

One of the complexities in designing and thinking about *MF* has been the role of names and identifiers and issues relating to Level 3. Because designers manipulate finite textual realizations or approximations of theories, the design of *MF* has focused quite a bit on handling the relationship of these textual pieces. The focus on named textual artifacts has dictated many of the decisions and interpretations of *MF*.

The nature of ensuring independent name spaces, supporting renaming to allow distinct identifiers to refer to the same values has not been fully formalized. The initial decision to use β-reduction to handle these details seemed a pragmatic choice: it off-loaded the details of tracking what values were assigned to which identifiers, simplifying the nature of the environment at the *MF* level and allowing us to rely on the underlying logical framework, which needed to keep track of these details in any case.

Use of β-reduction as the vehicle to discharge witnesses had the unfortunate consequence of also limiting what could be expressed about the action of discharging assumptions and the subsequent relationship of theory presentations that had been instantiated. The following chapter discusses some of these issues.

Chapter 6

Future Work

Many alternatives were shelved during the design of *MF* that are appropriate to discuss here. Because the goal of *MF* was to explore the structural components independent of the underlying object language, some issues were problematic. The most significant of these is the role of parameterization in specifications development, and to what degree this parameterization can be expressed structurally.

6.1 Theory Specialization

As noted in the previous chapter, there are two views of looking at theory presentation specialization or instantiation. One we have denoted by the instantiation of a structural environment and the other by theory specialization (or theory instantiation). In *MF* the specification of a structural environment in a theory presentation combining operation allows the specification writer to articulate at a theory-presentation level what the requirements of this theory are. So we might say that a theory P requires theory Q. But, this does not let us say what elements of Q are really required or why. In this model of theory instantiation, theories are opaque, highlighting what is purely structural from what is dependent on the internal values of the theory.

The other model of specializing a theory presentation is more dependent on the nature of the underlying object language. A theory itself is a set of sentences that is the denotation of a theory presentation. This set of sentences, when expressed in the object language directly, may have a context of expectations. For instance, if a theory T assumes that it will be provided with values of types X : *xtype*, Y : *ytype*, and Z : *ztype*, the sentences of T will

be well formed assuming that witnesses can be provided. The context for these sentences encapsulates what assumptions the theory has vis a vis witnesses to be provided. These kinds of theories can be checked in a logical system such as the Calculus of Constructions. They will be provisional on providing constructions for the assumptions X, Y, Z.

The manner of supporting this second kind of parameterization in *MF* is dictated by our view of structure and our goal to refrain from putting roles and conditions on kinds of specifications. The current *MF* approach specified in Chapter 4 assumes that this context of expectations is implicit: it is the responsibility of the object language to define and manage the context. In the *Lego* version of the Calculus of Constructions, for instance, this context can be specified on a statement-by-statement basis, or as a global context. Whichever method is chosen, *MF* has (currently) no knowledge of the context per se, outside of the assumption that a theory presentation to be instantiated with witnesses from another theory must all be abstracted uniformly on the same context elements. This means that if $[f_i]$ are features in the presentation P, $[g_j]$ in some presentation Q, and if we expect to be able to discharge assumptions in P with witnesses defined in Q, then the $[f_i]$ must all be abstracted on some $x_1 : x_1 type, \cdots, x_n : x_n type$ such that $f_i(g_1, \cdots, g_n)$ is well formed and $\vdash_{LF} g_j : x_j type$ for $1 \leq j \leq n$.

This approach has several limitations. First, it places a strong constraint on the structure of the sentences in the theory. While some constraints on the structure is reasonable (if the set of sentences is to be discharged in common, they should have the same structure), this approach has the potential for becoming an organizational headache, some of which *MF* is intended to ameliorate. Secondly, if a theory presentation is constructed by a theory presentation combination operator, there is no reason to suppose that each of the constituent theories will have been constructed with the same purpose in mind, so their sentences may not be abstracted in the same order, even if some of the same expectations are present. Because β-reduction is order dependent, it may require some work to discharge the desired elements. This in-depth knowledge of the internal representation of the sentences in a theory is not the way that we would ultimately like to handle theory specialization structurally.

An alternative is to specify (as in *Clear*) an explicit parameter with that parameter

having a theory presentation name as a "metasort". This explicit parameter commits the witnessing values to belong to a particular theory. If we require a $\leq$ operator, must we select a specification for partial orders? In that case, if we had an appropriate $\leq$ operator elsewhere in the presentation hierarchy, we would be unable to use it as a witness. In the end, this approach does not seem to provide any more than the instantiation of a structural environment.

A second kind of parameterization can be specified by supporting the construction a different kind of specification: a *context_object*. In a way, a context object can be thought of as a completely hypothetical specification, having no computational or constructive content. Then, if a specification is to be parameterized in the same sense as the *Clear* specification procedures, it could be defined as an extension of this context object, the context object functioning as the specification parameter. Discharging the elements of the context object would be accomplished via substitution: if Q is a theory whose elements were to be used to discharge the hypotheses in a context object C, we would have to construct a substitution R mapping the identifiers in C to the values Q.

Another alternative is to have the parameterizations be identified as feature terms and use feature term unification to identify appropriate witness for the discharging relationship. This would be another way to specify a context-object. A disadvantage to this approach is that the context-object would never denote an admissible context: It would always have to be instantiated with a non-hypothetical specification.

The disadvantage of the context object approach is that substitutions would end up affecting the object language: they are not really a property of the structure, but rather affect particular references to object level values. Secondly, introducing these distinguished theories requires a commitment to what kind an object is. The position of *MF* is that the decision as to what kind of object a specification is should be undetermined in *MF*: it consists of a role a presentation plays. So a theory might at one point be used as a context for another, while at a different point of development it might be used as a witness.

The goal of this section is to explore ways in which a theory may be parameterized in a way that is compatible with the structural focus of *MF* and give it a semantics that is consistent with those of the existing relationships in *MF*.

In the existing formalization of *MF*, the only relationships between theory presentations is the extension relationship. The different theory presentation combination operators all construct theory presentations that are ultimately extensions of their constituents. This has the advantage that the state of the logical system can be read from the relationships. To extend the underlying logical system with all the elements required for an arbitrary theory P, we would only need to extend the underlying logical system with all the constituents of P in the presentation base: the downward closure.

By loosening this requirement on the kinds of relationships theory presentations may have, we can introduce a different kind of relationship between theories and hence a different kind of theory presentation object. By introducing a *context_object* along with a discharge relationship, we can make explicit the role that a context plays distinct from the sentences in the theory. In Figure 6.1 we show four theory presentation objects: a *context_object* C, a *witnessing_object* D, which may be any suitable theory presentation (suitability is discussed below), a theory presentation E abstracted on C, and a theory E' with the same extension as E but whose assumptions have been discharged with the elements from D.

The relationship between C and D is no longer one of extension. Because we need not discharge all of the assumptions of C at once and we need not use all of the values of D to discharge assumptions, the relationship is somewhat more complex. If we wished to discharge assumption c_2 from C with value d_3 from D, we could do so if this substitution was (type) correct in the logical system. This means that if C is a valid context ($c_1 : c_1 type, c_2 : c_2 type, \cdots, c_{n-1} : c_{n-1} type \vdash_{LF} c_n : c_n type$), and D is a valid context, then the selected value(s) from D may be used to discharge the selected assumptions from C provided that $c_1 : c_1 type, d_1 : d_1 type, d_2 : d_2 type, d_3 : d_3 type \vdash_{LF} d_3 : c_2 type$. I.e., if we assume any context elements preceding both the assumption to be discharged and the witness used to discharge the assumption and we can demonstrate that the proposed witness has the type of the assumption, then the selected item(s) may be used to discharge the indicated assumption and may be used freely in any theory that depends on this assumption. The new context that has been discharged in this way will (or may) have some remaining assumptions, as well as some values that were not used to discharge any

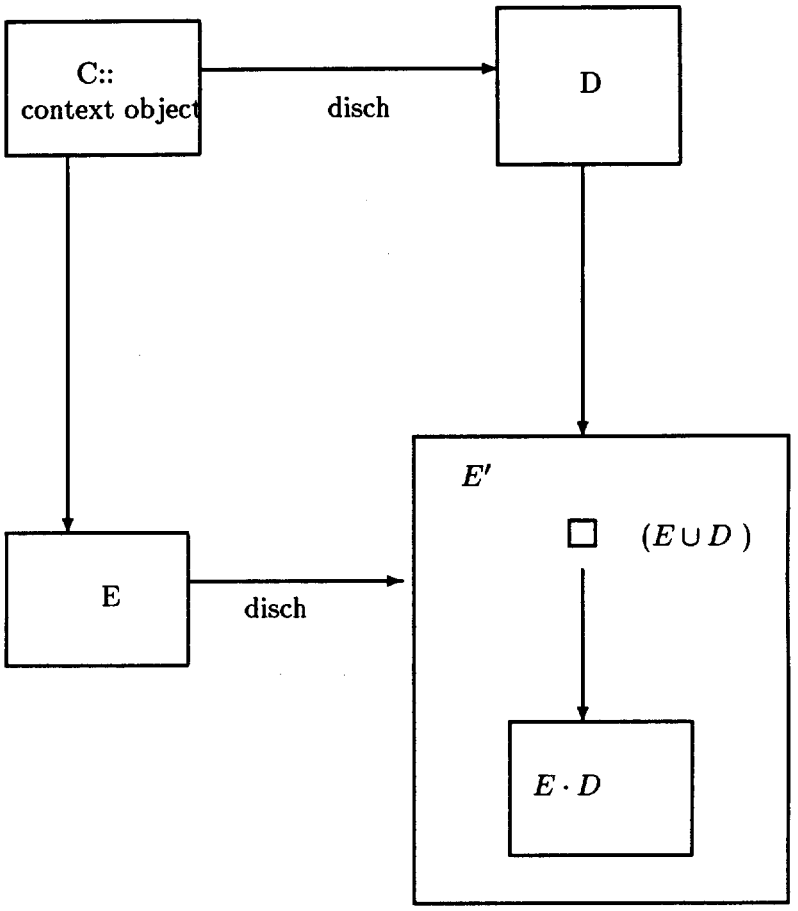

Figure 6.1: The Discharge Relationship

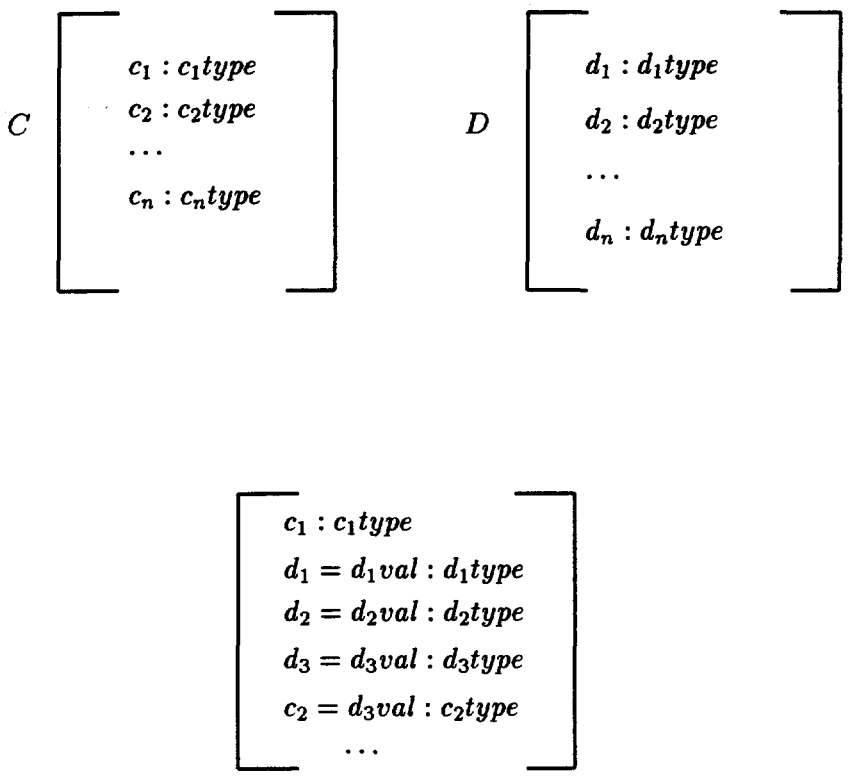

Figure 6.2: Context and Discharging objects

assumptions.

We have two choices here as well. The discharge relationship could be represented as a value replacement: the assumption to be discharged is given the value of the discharging witness (as shown in Figure 6.1). The resulting context is an extension of C in the sense that it extends the name space of C, it adds a value, but makes no other changes. As shown, the resulting context is not an extension of D: it only has those pieces of D that are required for the witnessing element to be well formed in the logical system. This is a problem in that it violates one of our conditions: to preserve the unity of theory presentations. All of D is no longer present, so any constraints on the use of the discharging values of D are not guaranteed to be present in the logical system. Moreover, the extension relationship between D and the new context no longer exists. We can "fix" this by having the discharge relationship in MF include all of the sentences of the discharging theory

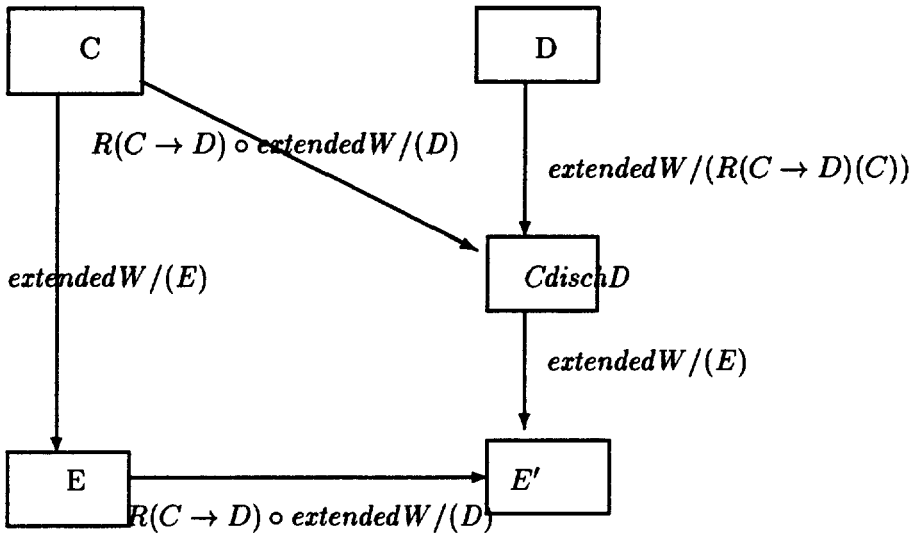

Figure 6.3: Relation of context and discharging theory

presentation. In that case, the new context is an extension of D as well [1].

If we designate $extendedW/(E)$ to be the extension associated with the theory E, $extendedW/(D)$ to be the extension associated with the witnessing theory D, $extendedW/(C)$ to be the extension associated with C and $R(C \rightarrow D)$ to be the replacement (witnessing) substitution generated by the discharge relationship [2], then we can represent the discharged context as $(extendedW/(R(C \rightarrow D)(C)))(D)$: the extension of the witnessing theory D by the extension associated with the context object C, with the appropriate values substituted to establish the witnesses. The actual application of E to D is then just an extension of the discharged context $CdischD$ with the same extension of the original unspecialized theory E.

[1] This argument assumes that the assumption type c_2type and the discharging type d_3type are identical. Otherwise, a proof would be required demonstrating that the two were identical. This is doable, if the types really are provably equal, but considerably more complicated to express. In general, this will expand in complexity, because the proof of the equality of the types may depend on previous values substituted by previous discharged assumptions that were generated along the way.

[2] This notation is deliberately simplified to explicate the structure. In fact, constructing the new discharged context $CdischD$ would require that the contexts be interleaved to preserve ordering requirements. The extension $extendedW/(D)$ extends a context with the elements of D, preserving the context dependencies.

The following table gives the results of the extensions and witnessing substitutions:

$R(C \rightarrow D)$	The witnessing substitution mapping values of D to identifiers of C.
$R(C \rightarrow D)(C)$	A new object C', that performs the substitution indicated. This object is only well formed if the definitions of D are in the environment.
$extendedW/(D)$	The extension introduced in D.
$extendedW/(D)(C)$	A new object that extends C with the extension introduced in D. This object is only well formed if the environment of C has all the definitions required by D.

The discharged context *CdischD* has all the elements of D. It also has all the elements of C: because the name space is preserved, and this is only a substitution of values, the change between the discharged context and the context object is that value information has been added to type information. So the discharged context is an extension of both C and D.

The new theory E' extends both C and *CdischD* (shown in Figure 6.1): the renaming substitution does not affect E, since E only has knowledge of the elements of C, so the extensions of E' and E are identical, the only difference being that E' has the context *CdischD*, which has specific values for some (or all) of the assumptions of C.

This gives us the following relationships:

$$\begin{aligned} CdischD &> C \\ CdischD &> D \\ E &> C \\ E' &> CdischD \\ E' &> C \end{aligned}$$

Additionally, E' is an extension of E in that it has all the same features, it has the same name space, and the only distinction is that is has had its context extended by the features of D and some elements of C have been given values from D. Certainly from a model theoretic point of view, rather than a syntactic point of view, everything provable in E is still provable in E'.

An alternative approach to the discharge relationship is to use name replacement, replacing the context elements in C entirely with the witnesses in D, and providing a renaming substitution for the specialized theories rather than for the context objects. However, this changes the name space and requires the continual application of renaming substitutions to make the same arguments.

6.1.1 What is a suitable theory to discharge a context object C?

The suitability of a theory to discharge a context object relies on the underlying logical system.

As referred to earlier, we can formalize the required relationship between a context object and a witnessing theory through the provability relationship of the underlying logical framework. Assuming that the witnessing theory is independent of the context object, the meaning associated with each can be denoted by a formation tree, which in turn will specify a context. This context can be partitioned into the appropriate independent pieces, allowing a suitable context to be constructed.

The meaning associated with a context object C is then formed in the same way as the meaning of any theory presentation. The meaning of the witnessing theory is formed in the same way (which may be any theory presentation in MF). The meaning is mapped into an admissible context, which may then be partitioned according to the elements to be discharged.

We let the notation $E \cdot Dunder(R(C \rightarrow D))$ indicate the context application of a theory E, abstracted on a context object C, to a witnessing theory D, under the indicated witnessing substitution $R(C \rightarrow D)$. Then the suitability conditions for this application can be expressed by:

$$Ctxt - ap \qquad \frac{Ctxt(C) \vdash_{LF} C : *\ \ Ctxt(C), Ctxt(D) \vdash_{LF} R(C \rightarrow D)(C) : *\ \ E > C}{Ctxt(E), Ctxt(D) \vdash_{LF} E \cdot D\ under\ (R(C \rightarrow D)) : *} \qquad (1)$$

6.2 Additional relationships

There are several additional relationship to be explored among context objects and discharging witnesses. First, there may be more than one discharging witness. This is shown

in Figure 6.4. Each D_i may be a witness for C (indicated by the arrows labeled by d for a discharging relationship). It is possible that each D_i discharges the same assumptions, or they may discharge different assumptions. If they discharge the same assumptions, they are related in a sense (all the theory presentations that can be used to discharge C). The D_i do not have the same extension, and it is not the case that any D_i might be used in place of another. But, any D_i can be used in place of C. This gives a more constrained notion of extension.

In this respect, C acts as a filter, describing what the witnesses must minimally look like. As such, it acts much like a structural environment acts, only with a tighter connection to the underlying logical framework. A structural environment denotes a (sub-)lattice in a presentation base structure. Similarly, C may denote a (sub-) lattice, where each specification in the sub-lattice is one that can be used to discharge the context object.

Alternatively, each D_i may discharge a disjoint set of assumptions. In that case, each D_i represents a different view of what witnesses are being provided. If it were the case that each D_i were totally disjoint and that the collection of D_i were able to discharge all the assumptions from C, we could have the sequence (using a simplified notation for application, ignoring the witnesses and statements being discharged): $(\cdots((E \cdot D_1) \cdot D_2) \cdots D_n)$, where each application discharged a different group of assumptions. Each application would have the correct relationship: $E \cdot D_1$ would have some part of its context undischarged and the resulting theory would be one that had some assumptions remaining. Each application would discharge the indicated assumption. The use of application to discharge assumptions points out the regularity of the discharging relationship: the difference between a context object and a theory presentation is one of role. Any theory presentation may act as a context object for another. *MF* cannot determine the correctness of such an application. Nor can it determine if the indicated assumptions are truly assumptions. The act of discharging a context of assumptions is no different than the *MF* notion of theory specialization. Ultimately they both rely on the correctness of the underlying logical framework. *MF* only records the relationships and maintains correctness given meaning preserving combining operations. We would like to examine the relationships between

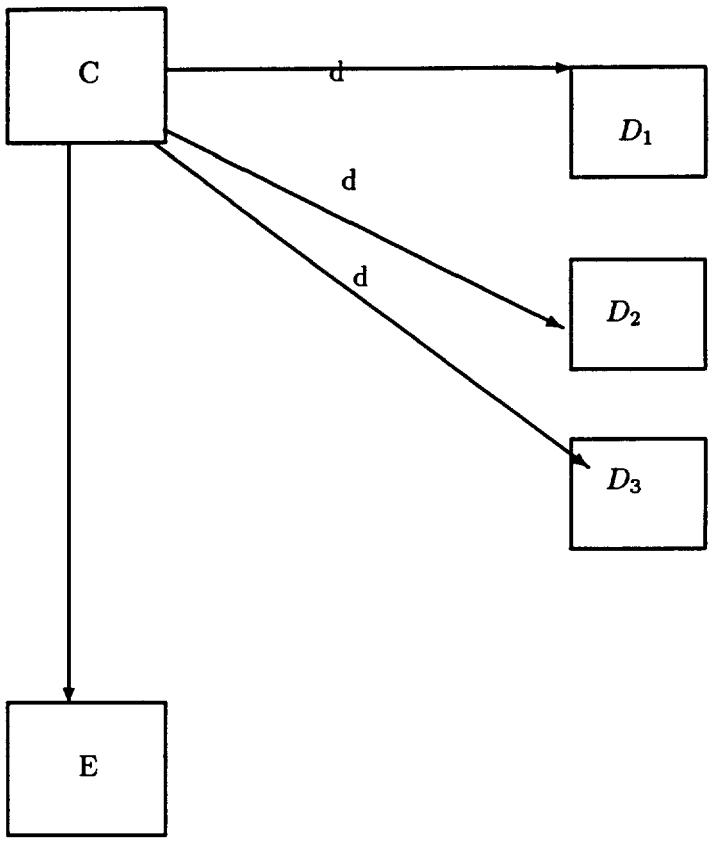

Figure 6.4: Multiple Dischargees Relationship

these sequences of theory specializations. If $E \cdot D_1\ d_1$ is a well formed theory specialization discharging some set of assumptions d_1, and if $(E \cdot D_1\ d_1) \cdot\ d_2$ further discharges some set of assumptions d_2, what is the relationship that *MF* can conclude from the underlying structure?

Specware defines a distributive law stating that if they have 2 diagram refinements, $\Delta_1 : d_1 \rightarrow d_2$ and $\Delta_2 : d_2 \rightarrow d_3$ that can be composed, then they satisfy the distributive law:

$$|\Delta_2| \odot |\Delta_1| = |\Delta_2 \odot \Delta_1|$$

This captures the relationship from refinements in one category to refinements in another (specifications and interpretations). In *MF*, diagram refinements are captured by theory specialization (or structural environment instantiation) with no distinction between a

category of specifications and interpretations.

Specware distinquished the kind of horizontal composition they provide from that defined in Sannella and Tarlecki [43], saying that their "distributive law is a generalization of Sannella and Tarlecki's, which uses parameterization and does not handle colimits; moreover, it's semantically oriented." Similarly, *MF* theory instantiation is not semantically oriented (although it obeys the semantic conditions of Sannella and Tarlecki). To move from parameterization to colimits, one needs to be able to combine the parameterizations appropriately. In the case of *MF*, using the context object-discharge relationship, this means that context objects should be combinable. Indeed, since a context object may be any object (only its role has really changed), it is the discharge relationship that is at issue.

Looking at Figure 6.5, the relationship between the diagram d_1 and d_2, is a discharge (or theory instantiation) relationship as discussed in the previous section. If E' now takes on the role of a context object, and D' is a theory that can discharge assumptions in E', then the relationship between the diagram d_2 and d_3 is also one of a discharge (theory instantiation). If E' is a specialization of E, and E'' is a specialization of E', and if the set of assumptions discharged is disjoint, then E'' is also a specialization of E.

In the next section we explore some of the relationships that can be captured structurally by the composition of theory instantiations by examining a case study.

6.2.1 Horizontal Composition

Horizontal composition as defined by Sanella and Tarlecki [43] expresses the relationship between parameterized theories and their arguments (written in an *MF* style notation): given theory presentations P, P', Q, and Q', if $P > P'$ and $Q > Q'$, then $P(Q) > P'(Q')$.

In the following tables, we give a set of theory instantiations and explore the way in which the different instantiations are related to one another. This section examines the relationship of theory instantiations as they currently exist in *MF*: through theory specialization rather than through a context-object discharge relationship.

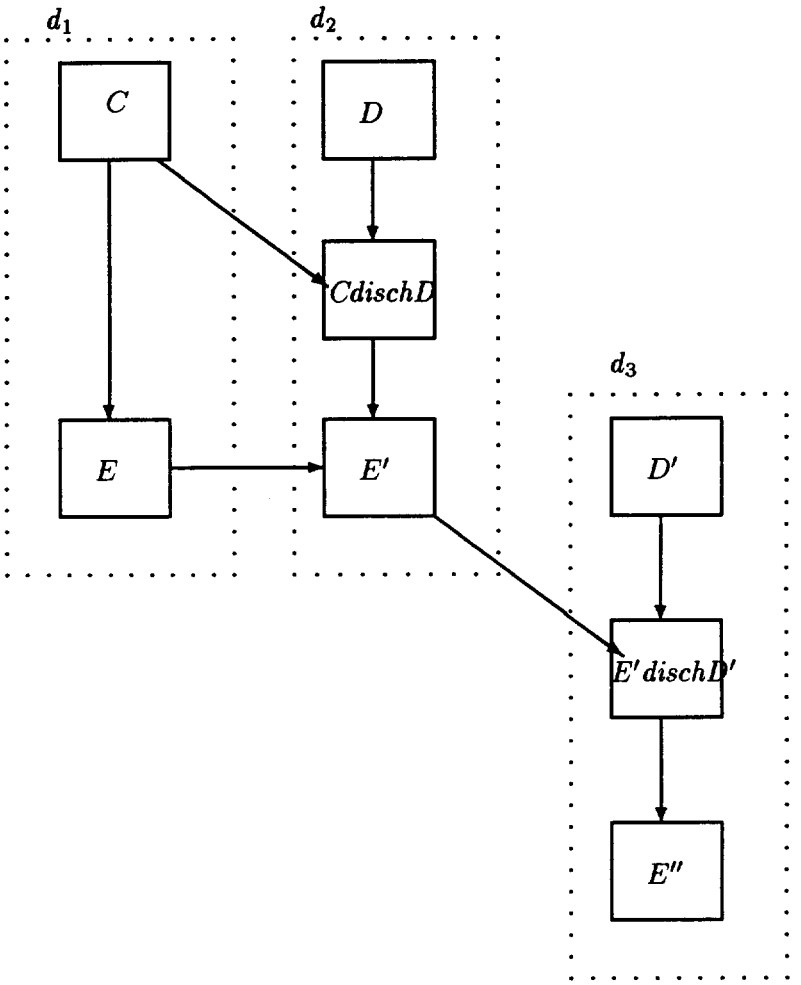

Figure 6.5: Horizontal Composition of Applications

We assume a presentation base structure with the following theory presentations:

A	f_1, f_2	Theory presentation A has features f_1, f_2
A'	g_3	Theory presentation A' extends A with g_3
B	g_1, g_2	Theory presentation B has features g_1, g_2
B'	g_3	Theory presentation B' extends B with g_3

These have the relationships: $A \leq A'$, $B \leq B'$. The theory presentations A and A' may be instantiated with witnesses from B and B' (assuming the witnesses are of correct type in the *LF*):

$C \equiv A \cdot B$	$[f_1, f_2]$	$[g_1]$	$[h_1, h_2]$	$h_1 \mapsto f_1(g_1)$
				$h_2 \mapsto f_2(g_1)$
$D \equiv A \cdot B$	$[f_1, f_2]$	$[g_1, g_2]$	$[h_3, h_4]$	$h_3 \mapsto f_1(g_1, g_2)$
				$h_4 \mapsto f_2(g_1, g_2)$
$E \equiv A \cdot B'$	$[f_1, f_2]$	$[g_1, g_2]$	$[h_5, g_6]$	$h_5 \mapsto f_1(g_1, g_2)$
				$h_6 \mapsto f_2(g_1, g_2)$
$F \equiv A \cdot B'$	$[f_1, f_2]$	$[g_1, g_2, g_3]$	$[h_7, h_8]$	$h_7 \mapsto f_1(g_1, g_2, g_3)$
				$h_8 \mapsto f_2(g_1, g_2, g_3)$
$G \equiv C \cdot B$	$[h_1, h_2]$	$[g_2]$	$[h_9, h_{10}]$	$h_9 \mapsto h_1(g_2) \equiv f_1(g_1)(g_2)$
				$h_{10} \mapsto h_2(g_2) \equiv f_2(g_1)(g_2)$
$H \equiv A' \cdot B$	$[f_1, f_2, f_3]$	$[g_1]$	$[h_{11}, h_{12}, h_{13}]$	$h_{11} \mapsto f_1(g_1)$
				$h_{12} \mapsto f_2(g_1)$
				$h_{13} \mapsto f_3(g_1)$
$I \equiv A' \cdot B$	$[f_1, f_2, f_3]$	$[g_1, g_2]$	$[h_{14}, h_{15}, h_{16}]$	$h_{14} \mapsto f_1(g_1, g_2)$
				$h_{15} \mapsto f_2(g_1, g_2)$
				$h_{16} \mapsto f_3(g_1, g_2)$
$J \equiv A' \cdot B'$	$[f_1, f_2, f_3]$	$[g_1, g_2]$	$[h_{17}, h_{18}, h_{19}]$	$h_{17} \mapsto f_1(g_1, g_2)$
				$h_{18} \mapsto f_2(g_1, g_2)$
				$h_{19} \mapsto f_3(g_1, g_2)$
$K \equiv A' \cdot B'$	$[f_1, f_2, f_3]$	$[g_1, g_2, g_3]$	$[h_{20}, h_{21}, h_{22}]$	$h_{20} \mapsto f_1(g_1, g_2, g_3)$
				$h_{21} \mapsto f_2(g_1, g_2, g_3)$
				$h_{22} \mapsto f_3(g_1, g_2, g_3)$

We propose two different relationships capturing the different cases in the table above.

- Theories where their structural environments are related and the statements to be discharged are related are also related. In the case of theory presentation D and E, $A \cup B \leq_p A \cup B'$. D and E each introduce an identical extension. Similarly, G and D have related structural environments. The structural environment for G includes C which is not present in D. C and H are also related in this way: H has both a larger environment specified and it discharges an additional statement.

- Theories where their structural environments are related but where one discharges more assumptions than the other. This is the case for C and D. They each are composed of A and B, but D has more assumptions discharged than C. Everything true in C is also true in D, since D is quantified over all witnesses having the indicated type.

Neither of these relationship is currently derivable in *MF* because they each have different genealogies: each is constructed via a distinct path. However, by introducing some new rules, we can derive these relationships.

Let A, A', B, B' be theory presentations and F_A, $F_{A'}$, F_B, $F_{B'}$ be feature identifier lists for statements to be discharged and statements to use as witnesses, where $A \cdot BF_AF_B$ is the application of A to B discharging the statements indicated in F_A with witnesses indicated in F_B. We introduce an ordering on feature identifier lists:

FRel $$\frac{f \in F_A \Rightarrow f \in F_B}{F_A \leq F_B} \qquad \text{(DRel1)}$$

The following rules are a first cut at capturing the above relationships:

Ext1 $$\frac{A \cup B \leq_p A' \cup B' \quad F_A \leq F_{A'} \quad F_B = F_{B'}}{A \cdot BF_AF_B \leq_p A' \cdot B'F_{A'}F_{B'}} \qquad \text{(DRel2)}$$

Ext2 $$\frac{A \cup B \leq_p A' \cup B' \quad F_A \leq F_{A'} \quad F_B < F_{B'}}{A \cdot BF_AF_B \prec A' \cdot B'F_{A'}F_{B'}} \qquad \text{(DRel3)}$$

Using these rules we can now conclude that:

$$C \prec D$$
$$D \prec F$$
$$J \prec K$$
$$I \leq J$$
$$H \leq J$$
$$D \leq E$$
$$D \leq G$$

MF theory instantiation includes more information than the horizontal compositions of Sannella and Tarlecki, in that they indicate which statements are to be discharged and

what assumptions are to be used and the relationship of any resulting theories depends on the relationship of these discharging witnesses and statements to be discharged. The case of D and E fit the model of horizontal composition in that they have identical specifications of what is to be discharged and what witnesses are to be used. The refinement relationship ($\prec$) is not an extension relationship but a specialization. Everything true in C is also true in D, since C is universally quantified on the undischarged witnesses, so in some sense it is similar to extension. We would like to be able to use this refinement relationship in the same way that we use extension: if $C \prec D$ and if $E > C$, then we should be able to satisfy any references to C (in the structural environment of E) by D, replacing the meaning of C by the meaning of D, and providing the necessary substitutions or applications. To carry along this information about the discharging witnesses and the statements to be discharged, it seems more appropriate to use the model of the context object (which makes the discharge relationship explicit) rather than β-reduction.

6.3 Extensions to Relate Structural Environments

Most of the work in *MF* has related to the individual theory presentation. The presentation base structure is the framework that can be used to extend this work to the relationship of structural environments. As mentioned several times, each theory presentation A in some presentation base structure p_i defines a presentation base structure, p_j, where each of the theory presentations in p_j is in the structural environment of A. We want to explore issues of mapping between different interpretations, different specifications and different refinements, and how the structure is preserved under these mappings. The presentation base structure should be able to support these relationships.

We also presented the frames as though they were linear developments. However, we can support alternate developments by allowing a presentation base structure to be selected anywhere in the development.

Unlike Specware, there is no commitment to what a presentation base structure means. While some presentation base structure may take on the role of an interpretation for another, this is not part of the defined methodology of *MF*. Exploring the issues of how

such maps between presentation base structures interact, what conditions need to be met, how they compose, are further areas that we wish to investigate.

6.4 Extensions Relating Object Level Statements

MF purposely refrained from making statements about the relationship of the underlying object level statements. We can introduce relationships among these statements as well, so that we can make judgements as to the equality of two theory presentations based not only on their structure, but also on declared or defined equalities of the underlying statements.

6.5 Exploring Different Roles

MF focused on structural relationships through extensions. Often, theory presentations play different roles that can't be distinguished using extension. These include differences such as the role of specifications, interpretations, abstractions, templates, and so on. Now that the basic formalization is in place, we can explore how these other roles can be supported and how they relate to *MF*'s more structural view.

Additionally, now that a structural definition is in place, we might explore how different properties of a base language might be described and then used. This might open up the ability to use specialized syntax appropriate to different domains.

Bibliography

[1] AÏT-KACI, H., AND NASR, R. LOGIN: A logic-programming language with built-in inheritance. *Journal of Logic Programming 3* (1986), 185–215.

[2] BARENDREGT, H. P. The type free lambda calculus. In *Mathematical Logic*, J. Barwise, Ed. North Holland, 1977, pp. 1091–1132.

[3] BARENDREGT, H. P. *The Lambda Calculus: Its Syntax and Semantics*, second ed., vol. II of *Studies in Logic and the Foundations of Mathematics*. North-Holland, 1984.

[4] BARENDREGT, H. P. Introduction to generalized type systems. *Journal of Functional Programming 1*, 2 (1991), 125–154.

[5] BISHOP, E. *Foundations of Constructive Analysis*. McGraw Hill, 1967.

[6] BRACHMAN, R. J., AND SCHMOLZE, J. An overview of the KL-ONE knowledge representation system. *Cognitive Science 9*, 2 (1985), 171–216.

[7] BROUWER, L. E. J. Historical background, principles and methods of intuitionism. *South African J. Sci*, 49 (1952), 139–146.

[8] BRUIJN, N. G. D. A survey of the project AUTOMATH. In *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, J. P. Seldin and J. R. Hindley, Eds. Academic Press, 1980, pp. 579–606.

[9] BURSTALL, R., AND MCKINNA, J. Deliverables: an approach to program development in the calculus of constructions. In *Proceedings of the First Annual Workshop on Logical Frameworks* (Antibes, France, 1990), pp. 113–122.

[10] BURSTALL, R. M., AND GOGUEN, J. A. Putting theories together to make specifications. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence* (Cambridge, Mass., 1977), pp. 1045–1058.

[11] BURSTALL, R. M., AND GOGUEN, J. A. The semantics of Clear, a specification language. In *Lecture Notes in Computer Science*, vol. 86. Springer-Verlag, 1980, pp. 293–329.

[12] CHURCH, A. *The Calculi of Lambda Conversion.* Princeton University Press, 1941.

[13] CONSTABLE, R. L., ET AL. *Implementing Mathematics with the Nuprl Proof Development System.* Prentice Hall, 1986.

[14] COQUAND, T., AND HUET, G. Constructions: A higher order proof system for mechanizing mathematics. In *Lecture Notes in Computer Science*, vol. 203. Springer-Verlag, 1986, pp. 151–184.

[15] COQUAND, T., AND HUET, G. The calculus of constructions. *Information and Computation 76* (1988), 95–120.

[16] CURRY, H. B., AND FEYS, R. *Combinatory Logic*, vol. I. North-Holland, 1958.

[17] DÖRRE, J., AND ROUNDS, W. C. On subsumption and semiunification in feature algebras. In *Proceedings Symposium on Logic in Computer Science* (1990), IEEE Computer Society Press, pp. 300–310.

[18] DUMMETT, M. *Truth and Other Enigmas.* Duckworth, 1978.

[19] GENTZEN, G. Investigations into logical deduction. In *The Collected Papers of Gerhard Gentzen*, M. E. Szabo, Ed. North-Holland, 1969, pp. 68–131.

[20] GIRARD, J.-Y. *Proof Theory and Logical Complexity*, vol. 1. Bibliopolis, 1987.

[21] GOGUEN, J., AND BURSTALL, R. Introducing institutions. In *Lecture Notes in Computer Science*, vol. 164. Springer-Verlag, 1983, pp. 221–256.

[22] GOGUEN, J. A., AND TARDO, J. An introduction to OBJ: A language for writing and testing software. In *Specification of Reliable Software.* IEEE Press, 1979, pp. 170–189.

[23] HARPER, R., HONSELL, F., AND PLOTKIN, G. A framework for defining logics. In *Proceedings Symposium on Logic in Computer Science* (Washington, D.C., 1987), IEEE Computer Society Press, pp. 194–204.

[24] HARPER, R., SANNELLA, D., AND TARLECKI, A. Structure and representation in LF. In *Proceedings Symposium on Logic in Computer Science* (Washington, D.C., 1989), IEEE Computer Society Press, pp. 226–241.

[25] HEYTING, A. *Intuitionism, an Introduction.* North-Holland, 1956.

[26] JEAN-YVEW GIRARD, Y. L., AND TAYLOR, P. *Proofs and Types.* Cambridge University Press, 1989.

[27] KASPER, R. T., AND ROUNDS, W. C. A logical semantics for feature structures. In *Proceedings of the 24th Annual Meeting of the Association for Computational Linguistics* (Columbia University, 1986).

[28] KREISEL, G. Interpretation of analysis by means of constructive functionals of finite type. In *Constructivity in Mathematics*, A. Heyting, Ed. North Holland, 1959, pp. 101–128.

[29] LUO, Z. $cc^{\omega}_{\subset}$ and its meta theory. Tech. Rep. ECS-LFCS-88-58, LFCS Report Series, 1988. University of Edinburgh, Department of Computer Science.

[30] LUO, Z. A higher-order calculus and theory abstraction. Tech. Rep. ECS-LFCS-88-57, LFCS Report Series, 1988. University of Edinburgh, Department of Computer Science.

[31] LUO, Z. ECC, an extended calculus of constructions. Tech. Rep. ECS-LFCS-90-118, LFCS Report Series, 1989. University of Edinburgh, Department of Computer Science.

[32] LUO, Z., POLLACK, R., AND TAYLOR, P. How to use Lego. Tech. Rep. ECS-LFCS-92-211, LFCS Report Series, 1992. University of Edinburgh, Department of Computer Science.

[33] MARTIN-LÖF, P. An intuitionistic theory of types: Predicative part. In *Logic Colloquium '73*, H. E. Rose and J. C. Shepherdson, Eds. North Holland, 1975, pp. 73–118.

[34] MARTIN-LÖF, P. Constructive mathematics and computer programming. In *Sixth International Congress for Logic, Methodology, and Philosophy of Science* (1982), North Holland, pp. 153–175.

[35] NERODE, A., AND SHORE, R. A. *Logic for Applications.* Springer-Verlag, 1993.

[36] PAULIN-MOHRING, C. Extracting F_{ω}'s programs from proofs in the calculus of constructions. In *Conference Record of the Sixteenth Annual ACM Symposium on Principles of Programming Languages.* ACM, 1989, pp. 89–104.

[37] PFENNING, F. ELF: A language for logic definition and verified metaprogramming. In *Proceedings Symposium on Logic in Computer Science* (Washington, D.C., 1989), IEEE Computer Society Press, pp. 313–323.

[38] PRAWITZ, D. *Natural Deduction.* Almqvist & Wiksell, 1965.

[39] PRAWITZ, D. Ideas and results in proof theory. In *Proceedings of the Second Scandinavian Logic Symposium* (Amsterdam, 1971), North-Holland, pp. 235–307.

[40] RUSSELL, B. *Principles of Mathematics.* Norton, 1938.

[41] SANELLA, D., AND TARLECKI, A. Building specifications in an arbitrary institution. In *Lecture Notes in Computer Science*, vol. 173. Springer-Verlag, 1984, pp. 337–356.

[42] SANNELLA, D. Formal program development in extended ML for the working programmer. In *Proceeding 3rd BCS.FACS Workshop on Refinement* (1990).

[43] SANNELLA, D., AND TARLECKI, A. Toward formal development of programs from algebraic specifications: Implementations revisited. *Acta Infrormatica 25* (1988), 233–282.

[44] SANNELLA, D. T., AND BURSTALL, R. M. Structured theories in LCF. In *Proc. of the 8th Colloquium on Algebra and Trees in Programming* (1983), pp. 377–391.

[45] SMITH, D. R. *KIDS - A Knowledge-Based Softwqare Development System.* AAAI/MIT Press, 1991, pp. 483–514.

[46] SMOLKA, G. A feature logic with subsorts. Tech. Rep. 33, LILOG, 1988.

[47] SMOLKA, G., AND AÏT-KACI, H. Inheritance hierarchies: Semantics and unification. *J. Symbolic Computation 7* (1989), 343–370.

[48] SMORYNSKI, C. The incompleteness theorems. In *Mathematical Logic*, J. Barwise, Ed. North Holland, 1977, pp. 821–866.

[49] SRINIVAS, Y. V., AND JÜLLIG, R. Specware: Formal support for composing software. Tech. Rep. KES.U.95.5, The Kestrel Institute, Palo Alto, Ca., 1995.

[50] STREICHER, T. *Semantics of Type Theory.* Birkhäuser, 1991.

[51] TROELSTRA, A. S. Aspects of constructive mathematics. In *Mathematical Logic*, J. Barwise, Ed. North Holland, 1977, pp. 973–1052.

Appendix A

Notational Index

c, c_i	identifier of LF statements
$\hat{c}, \hat{c}_i$	type of LF statements
$[\tau]$	set of identifiers for LF statements, $c_i \in [\tau]$
$[\hat{\tau}]$	set of types of it LF statements, $\hat{c}_i \in [\hat{\tau}]$
$[\hat{\tau}]^1_\pi$	set of Level 1 terms denoting it LF statements,
$[AbstrVals]$	Encodings of LF statements
$[\![\hat{\tau}]\!]$	meanings of LF identifiers
π^1	type environment for Level 1 identifiers
$[\![\pi^1]\!]$	set of environments compatible with π^1
$u^1 \in [\![\pi^1]\!]$	evaluation environment for Level 1
$[\![\cdot]\!]^1 : [\tau] \to ([\![\pi^1]\!] \to [\![\hat{\tau}]\!])$	semantic function for Level 1 terms
ι_i	meta-variable ranging over feature identifiers
$[\omega]$	set of feature identifiers, $\iota_i \in [\omega]$
τ	Level 2 type of an LF statement
γ	Level 2 meta-variable for types of an extension
α	Level 2 type of the empty theory
θ	Level 2 meta-variable for types of structured theories
$\vdash$	Type coercion.
x_i	Level 2 meta-variable for terms of type τ_i
F_i	Level 2 meta-variable for terms of type γ_i
A, B	Level 2 meta-variable for terms of type θ_i
FT	Formation tree for an *MFsort*
T_i	Meta-variable for formation trees
$\sigma, \sigma\|0, \sigma\|1$	A path in a formation tree
$\langle N_{T_i}, <_{T_i} \rangle$	Support and order for a formation tree
$\cong$	Consistency relation between formation trees
$=_{FT}$	Equality of formation trees

$\sqsubseteq$	Formation tree consistency
$RFT(T_i)$	Reduced formation tree
$minFT(T_i)$	Minimal formation tree
$\cup$	Theory combination
$\cap$	Theory intersection
$\odot$	Theory extension
$\cup_C$	Combination of admissible contexts
$\cap_C$	Intersection of admissible contexts
,	Concatentation
π^2	The phrase type environment for Level 2 terms
$<\mid$	The parent relationship on contexts
$<\mid*$	The ancestor relationship on contexts
$<_p$	Ordering relation on Level 2 terms
$Ctxt(T_i)$	The flat LF context defined by a formation tree
Γ, Δ	Meta-variables for LF contexts
$[\tau]^2_\pi$	The set of phrases of type τ in the environment π^2
$[\gamma]^2_\pi$	The set of phrases of type γ in the environment π^2
$[\theta]^2_\pi$	The set of phrases of type θ in the environment π^2
$[\tau]$	The set of identifiers for τ types
$[\gamma]$	The set of identifiers for γ types
$[\omega]$	The set of identifiers for features
$[\theta]$	The set of identifiers for θ types
$[\![\tau]\!]$	The meanings for the Level 2 terms that denote individual LF statements
$[\![\gamma]\!]$	The meanings for the Level 2 terms that denote sequences of LF statements
$[\![\theta]\!]$	The meanings for the Level 2 terms that denote sequences of sequences constructed using the theory presentation constructors
$[\![\cdot]\!]^2$	The semantics function for Level 2 terms
$[exp]$	The set of phrases of Level 3 expressions
$[cmd]$	The set of phrases of Level 3 commands
$[qry]$	The set of phrases of Level 3 queries
$[\![exp]\!]$	The semantic set for expressions
$[\![cmd]\!]$	The semantic set for commands
$[\![qry]\!]$	The semantic set for queries

$[\![\cdot]\!]^3_N$	The meaning of a Level 3 phrase as an identifier
$[\![\cdot]\!]_C$	The meaning of a Level 3 phrase as a command
$[\![\cdot]\!]_Q$	The meaning of a Level 3 phrase as a query
$[\![\cdot]\!]_E$	The meaning of a Level 3 phrase as an expression
S_0	The empty theory
$<_P$	The partial order on theory presentation identifiers
u^3	The Level 3 valuation environment
u^2	The Level 2 valuation environment
u^1	The Level 1 valuation environment
$\mathbf{f}$	The Level 3 meta-variable for frames
p, p_i	The Level 3 meta-variable for presentation base structures
f_i, g_i, h_i	The meta-variable used for features in Level 3
$[\langle N, \leq_N \rangle]$	The set of paths that are consistent with the order dependency of the formation tree
$[\langle N, \leq_N \rangle]u^2, u^1$	The set of contexts for each path in a given environment
$Ctxt(p_i)$	The LF context defined by a presentation base structure
$<_{\mathbf{P}}$	The ordering relation on presentaiton base structures

Biographical Note

Sherri Shulman lives in Portland, Or. with her husband and three children.