This is a technical report. A modified version of this report will be published in: *Handbook of Algebra*, Vol.2, edited by M. Hazewinkel

Some Aspects of Categories in Computer Science

P. J. Scott*

Dept. of Mathematics
University of Ottawa
Ottawa, Ontario CANADA

Sept,1998

Contents

*Research partially supported by NSERC, Canada

1 Introduction

Over the past 25 years, category theory has become an increasingly significant conceptual and practical tool in many areas of computer science. There are major conferences and journals devoted wholly or partially to applying categorical methods to computing. At the same time, the close connections of computer science to logic have seen categorical logic (developed in the 1970's) fruitfully applied in significant ways in both theory and practice.

Given the rapid and enormous development of the subject and the availability of suitable graduate texts and specialized survey articles, we shall only examine a few of the areas that appear to the author to have conceptual and mathematical interest to the readers of this Handbook. Along with the many references in the text, the reader is urged to examine the final section (Literature Notes) where we reference omitted important areas, as well as the Bibliography.

We shall begin by discussing the close connections of certain closed categories with typed lambda calculi on the one hand, and with the proof theory of various logics on the other. It cannot be overemphasized that modern computer science heavily uses formal syntax but we shall try to tread lightly. The so-called Curry-Howard isomorphism (which identifies formal proofs with lambda terms, hence with arrows in certain free categories) is the cornerstone of modern programming language semantics and simply cannot be overlooked.

Notation: We often elide composition symbols, writing $gf : A \to C$ for $g \circ f : A \to C$, whenever $f : A \to B$ and $g : B \to C$. To save some space, we have omitted large numbers of routine diagrams, which the reader can find in the sources referenced.

2 Categories, Lambda Calculi, and Formulas-as-Types

2.1 Cartesian Closed Categories

Cartesian closed categories (ccc's) were developed in the 1960's by F. W. Lawvere [Law66, Law69]. Both Lawvere and Lambek [L74] stressed their connections to Church's lambda calculus, as well as to intuitionistic proof theory. In the 1970's, work of Dana Scott and Gordon Plotkin established their fundamental role in the semantics of programming languages. A precise equivalence between these three notions (ccc's, typed lambda calculi, and intuitionistic proof theory) was published in Lambek and Scott [LS86]. We recall the appropriate definitions from [LS86]:

Objects	Distinguished Arrow(s)	Equations
Terminal 1	$A \xrightarrow{!_A} 1$	$!_A = f$, $f : A \to 1$
Products $A \times B$	$\pi_1^{A,B} : A \times B \to A$	$\pi_1 \circ \langle f, g\rangle = f$
	$\pi_2^{A,B} : A \times B \to B$	$\pi_2 \circ \langle f, g\rangle = g$
	$\dfrac{C \xrightarrow{f} A \quad C \xrightarrow{g} B}{C \xrightarrow{\langle f,g\rangle} A \times B}$	$\langle \pi_1 \circ h, \pi_2 \circ h\rangle = h$, $h : C \to A \times B$
Exponentials B^A	$ev_{A,B} : B^A \times A \to B$	$ev \circ \langle f^* \circ \pi_1, \pi_2\rangle = f$
	$\dfrac{C \times A \xrightarrow{f} B}{C \xrightarrow{f^*} B^A}$	$(ev \circ \langle g \circ \pi_1, \pi_2\rangle)^* = g$, $g : C \to B^A$

Figure 1: CCC's Equationally

Definition 2.1

(i) A *cartesian category* $\mathcal{C}$ is a category with distinguished finite products (equivalently, binary products and a terminal object 1.) This says there are isomorphisms (natural in A, B, C)

$$(1) \qquad Hom_{\mathcal{C}}(A, 1) \cong \{*\}$$

$$(2) \qquad Hom_{\mathcal{C}}(C, A \times B) \cong Hom_{\mathcal{C}}(C, A) \times Hom_{\mathcal{C}}(C, B)$$

(ii) A *cartesian closed category* $\mathcal{C}$ is a cartesian category $\mathcal{C}$ such that, for each object $A \in \mathcal{C}$, the functor $(-) \times A : \mathcal{C} \to \mathcal{C}$ has a specified right adjoint, denoted $(-)^A$ .That is, there is an isomorphism (natural in B and C)

$$(3) \qquad Hom_{\mathcal{C}}(C \times A, B) \cong Hom_{\mathcal{C}}(C, B^A)$$

For many purposes in computer science, it is often useful to have categories with explicitly given *strict* structure along with *strict* functors that preserve everything on the nose. We may present such ccc's equationally, in the spirit of multisorted universal algebra. The arrows and equations are summarized in Figure 1. These equations determine the isomorphisms (1), (2), and (3). In this presentation we say the structure is *strict*, meaning there is only one object representing each of the above constructs $1, A \times B, B^A$. The exponential object B^A is often called the *function space* of A and B. In the computer science literature, the function space is often denoted $A \Rightarrow B$, while the arrow $C \xrightarrow{f^*} B^A$ is often called *currying* of f.

Remark 2.2 Following most categorical logic and computer science literature, we do not assume ccc's have finite limits [Law69, LS86, AC98, Mit96]), in order to keep the correspondence with simply typed lambda calculi, cf Theorem 2.20 below. Earlier books (cf. [Mac71]) do not always follow this convention.

Let us list some useful examples of cartesian closed categories: for details see [LS86, Mit96, Mac71]

Example 2.3 The category **Set** of sets and functions. Here $A \times B$ is a chosen cartesian product and B^A is the set of functions from A to B. The map $B^A \times A \xrightarrow{ev} B$ is the usual evaluation map, while currying $C \xrightarrow{f^*} B^A$ is the map $c \mapsto (a \mapsto f(c, a))$.

An important subfamily of examples are *Henkin models* which are ccc's in which the terminal object 1 is a generator ([Mit96], Theorem 7.2.41). More concretely, for a lambda calculus signature with freely generated types (cf Section 2.13 below), a *Henkin model* $\mathcal{A}$ is a type-indexed family of sets $\mathcal{A} = \{A_\sigma \mid \sigma \text{ a type }\}$ where $A_1 = \{*\}$, $A_{\sigma\times\tau} = A_\sigma \times A_\tau$, $A_{\sigma\Rightarrow\tau} \subseteq A_\tau^{A_\sigma}$ which forms a ccc with respect to restriction of the usual ccc structure of **Set**. In the case of atomic base sorts b, $\mathcal{A}_b$ is some fixed but arbitrary set. A *full type hierarchy* is a Henkin model with full function spaces, i.e. $A_{\sigma\Rightarrow\tau} = A_\tau^{A_\sigma}$.

Example 2.4 More generally, the functor category $\mathbf{Set}^{\mathcal{C}^{op}}$ of presheaves on $\mathcal{C}$ is cartesian closed. Its objects are (contravariant) functors from $\mathcal{C}$ to **Set**, and its arrows are natural transformations between them. We sketch the ccc structure: given $F, G \in \mathbf{Set}^{\mathcal{C}^{op}}$, define $F \times G$ pointwise on objects and arrows. Motivated by Yoneda's Lemma, define $G^F(A) = Nat(h^A \times F, G)$, where $h^A = Hom(A, -)$. This easily extends to a functor. Finally if $H \times F \xrightarrow{\theta} G$, define $H \xrightarrow{\theta^*} G^F$ by: $\theta^*_A(a)_C(h, c) = \theta_C(H(h)(a), c)$.

Functor categories have been used in studying problematic semantical issues in Algol-like languages [Rey81, Ol85, OHT92, Ten94], as well as recently in concurrency theory and models of π-calculus [CSW, CaWi]. Special cases of presheaves have been studied extensively [Mit96, LS86]:

- Let $\mathcal{C}$ be a poset (qua trivial category). Then $\mathbf{Set}^{\mathcal{C}^{op}}$, the category of Kripke models over $\mathcal{C}$, may be identified with sets indexed (or graded) by the poset $\mathcal{C}$. Such models are fundamental in intuitionistic logic [LS86, TrvD88] and also arise in Kripke Logical Relations, an important tool in semantics of programming languages [Mit96, OHT93, OHRi].

- Let $\mathcal{C} = \mathcal{O}(X)$, the poset of opens of the topological space X. The subcategory $Sh(X)$ of sheaves on X is cartesian closed.

- Let $\mathcal{C}$ be a monoid M (qua category with one object). Then $\mathbf{Set}^{\mathcal{C}^{op}}$ is the category of *M-sets*, i.e. sets X equipped with a left action; equivalently, a monoid homomorphism $M \to End(X)$, where $End(X)$ is the monoid of endomaps of X. Morphisms of M-sets X and Y are equivariant maps (i.e. functions commuting with the action.) A special case of this is when M is actually a group G (qua category with one object, where all maps are isos). In that case $\mathbf{Set}^{\mathcal{C}^{op}}$ is the category of G-sets, the category of permutational representations of G. Its objects are sets X equipped with left actions $G \to Sym(X)$ and whose morphisms are equivariant maps. We shall return to these examples when we speak of Laüchli semantics and Full Completeness, Section 5.2

Example 2.5 ***ω-CPO***. Objects are posets P such that countable ascending chains $a_0 \leq a_1 \leq a_2 \leq \cdots$ have suprema. Morphisms are maps which preserve suprema of countable ascending chains (in particular, are order preserving.) This category is a ccc, with products $P \times Q$ ordered pointwise and $Q^P = Hom(P, Q)$, ordered pointwise. In this case, the categories are ω-**CPO**-enriched–i.e. the hom-sets themselves form an ω-**CPO**, compatible with composition. An important subccc is ω-$\mathbf{CPO}_\perp$, in which all objects have a distinguished minimal element $\perp$ (but morphisms need not preserve it).

The category ω-**CPO** is the most basic example in a vast research area, *domain theory*, which has arisen since 1970. This area concerns the denotational semantics of programming languages and models of untyped lambda calculi (cf. Section 2.5 below.) See also the survey article [AbJu94].

Example 2.6 *Coherent Spaces and Stable Maps.* A *Coherent Space* $\mathcal{A}$ is a family of sets satisfying: (i) $a \in \mathcal{A}$ and $b \subseteq a$ implies $b \in \mathcal{A}$, and (ii) if $B \subseteq \mathcal{A}$ and if $\forall c, c' \in B (c \cup c' \in \mathcal{A})$ then $\cup B \in \mathcal{A}$. In particular, $\emptyset \in \mathcal{A}$. Morphisms are *stable maps*, i.e. monotone maps preserving pullbacks and filtered colimits. That is, $f : \mathcal{A} \to \mathcal{B}$ is a stable map if (i) $b \subseteq a \in \mathcal{A}$ implies $f(b) \subseteq f(a)$, (ii) $f(\cup_{i\in I} a_i) = \cup_{i\in I} f(a_i)$, for I directed, and (iii) $a \cup b \in \mathcal{A}$ implies $f(a \cap b) = f(a) \cap f(b)$. This gives a category **Stab**. Every coherent space $\mathcal{A}$ yields a reflexive-symmetric (undirected) graph $(|\mathcal{A}|, \frown)$ where $|\mathcal{A}| = \{a \mid \{a\} \in \mathcal{A}\}$ and $a \frown b$ iff $\{a, b\} \in \mathcal{A}$. Moreover, there is a bijective correspondence between such graphs and coherent spaces. Given two coherent spaces $\mathcal{A}, \mathcal{B}$ their product $\mathcal{A} \times \mathcal{B}$ is defined via the associated graphs as follows: $(|\mathcal{A} \times \mathcal{B}|, \frown_{\mathcal{A}\times\mathcal{B}})$, with $|\mathcal{A} \times \mathcal{B}| = |\mathcal{A}| \uplus |\mathcal{B}| = (\{1\} \times |\mathcal{A}|) \cup (\{2\} \times |\mathcal{B}|)$ where $(1, a) \frown_{\mathcal{A}\times\mathcal{B}} (1, a')$ iff $a \frown_{\mathcal{A}} a'$, $(2, b) \frown_{\mathcal{A}\times\mathcal{B}} (2, b')$ iff $b \frown_{\mathcal{B}} b'$, and $(1, a) \frown_{\mathcal{A}\times\mathcal{B}} (2, b)$ for all $a \in |\mathcal{A}|, b \in |\mathcal{B}|$. The function space $\mathcal{B}^{\mathcal{A}} = Stab(\mathcal{A}, \mathcal{B})$ of stable maps can be given the structure of a coherent space, ordered by Berry's order: $f \preceq g$ iff for all $a, a' \in \mathcal{A}$, $a' \subseteq a$ implies $f(a') = f(a) \cap g(a')$. For details, see [GLT, Tr92]. This class of domains led to the discovery of linear logic (Section 4.2).

Example 2.7 *Per Models.* A *partial equivalence relation* (per) is a symmetric, transitive relation $\sim_A \subseteq A^2$. Thus $\sim_A$ is an equivalence relation on the subset $Dom_A = \{x \in A \mid x \sim_A x\}$. A $\mathcal{P}$-set is a pair $(A, \sim_A)$ where A is a set and $\sim_A$ is a per on A. Given two $\mathcal{P}$-sets $(A, \sim_A)$ and $(B, \sim_B)$ a *morphism of $\mathcal{P}$-sets* is a function $f : A \to B$ such that $a \sim_A a'$ implies $f(a) \sim_B f(a')$ for all $a, a' \in A$. That is, f induces a map of quotients $Dom_A/\sim_A \to Dom_B/\sim_B$ which preserves the associated partitions.

$\mathcal{P}$**Set**, the category of $\mathcal{P}$-sets and morphisms is a ccc, with structure induced from *Set*: we define $(A \times B, \sim_{A\times B})$, where $(a, b) \sim_{A\times B} (a', b')$ iff $a \sim_A a'$ and $b \sim_B b'$ and $(B^A, \sim_{B^A})$, where $f \sim_{B^A} g$ iff for all $a, a' \in A$, $a \sim_A a'$ implies $f(a) \sim_B g(a')$. We shall discuss variants of the ccc structure of $\mathcal{P}$**Set** in Section 2.7 below, with respect to reduction-free normalization.

Other classes of *Per* models are obtained by considering pers on a fixed (functionally complete) partial combinatory algebra, for example built over a model of untyped lambda calculus (cf Section 2.5 below). The prototypical example is the following category $Per(N)$ of pers on the natural numbers. The objects are pers on N. Morphisms $R \xrightarrow{f} S$ are (equivalence classes of) partial recursive functions (= Turing-machine computable partial functions) $N \rightharpoonup N$ which induce a total map on the induced partitions, i.e. for all $m, n \in N$, mRn implies $f(m), f(n)$ are defined and $f(m)Sf(n)$. Here we define equivalence of maps $f, g : R \to S$ by: $f \sim g$ iff $\forall m, n$, mRn implies $f(m), g(n)$ are defined and $f(m)Sg(n)$. The fact that $Per(N)$ is a ccc uses some elementary recursion theory [BFSS90, Mit96, AL91]. (See also Section 2.4.1).

Example 2.8 *Free CCC's.* Given a set of basic objects $\mathcal{X}$, we can form $\mathcal{F}_{\mathcal{X}}$, the *free ccc generated by $\mathcal{X}$*. Its objects are freely generated from $\mathcal{X}$ and 1 using $\times$ and $(-)^{(-)}$, its arrows are freely generated using identities and composition plus the structure in Figure 1, and we impose the minimal equations required to have a ccc. More generally, we may build $\mathcal{F}_{\mathcal{G}}$, the free ccc generated by a directed multigraph (or even a small category) $\mathcal{G}$, by freely generating from the vertices (resp. objects) and edges (resp. arrows) of $\mathcal{G}$ and then–in the case of categories $\mathcal{G}$–imposing the appropriate equations. The sense that this is free is related to Definition 2.9 and discussed in Example 2.23.

Cartesian closed categories can themselves be made into a category in many ways. This depends, to some extent, on how much 2- , bi-, enriched- , etc. structure one wishes to impose. The following elementary notions have proved useful. We shall mention a comparison between strict and nonstrict ccc's with coproducts in Remark 2.28. More general notions of monoidal functors, etc. will be mentioned in Section 4.1.

Definition 2.9 $\mathbf{CART}_{st}$ is the category of strictly structured cartesian closed categories with functors that preserve the structure on the nose. $\mathbf{2\text{-}CART}_{st}$ is the 2-category whose 0-cells are cartesian closed categories, whose 1-cells are strict cartesian closed functors, and whose 2-cells are natural isomorphisms.

As pointed out by Lambek [L74, LS86], given a ccc $\mathcal{A}$, we may adjoin an indeterminate arrow $1 \xrightarrow{x} A$ to $\mathcal{A}$ to form a *polynomial* cartesian closed category $\mathcal{A}[x]$ over $\mathcal{A}$, with the expected universal property in $\mathbf{CART}_{st}$. The objects of $\mathcal{A}[x]$ are the same as those of $\mathcal{A}$, while the arrows are "polynomials", i.e. formal expressions built from the symbol x using the arrow-forming operations of $\mathcal{A}$. The key fact about such polynomial expressions is a normal form theorem, stated here for ccc's, although it applies more generally (see [LS86], p.61):

Proposition 2.10 (Functional Completeness) *For every polynomial $\varphi(x)$ in an indeterminate $1 \xrightarrow{x} A$ over a ccc $\mathcal{A}$, there is a unique arrow $1 \xrightarrow{h} C^A \in \mathcal{A}$ such that $ev{\circ}\langle h, x\rangle \underset{x}{=} \varphi(x)$, where $\underset{x}{=}$ is equality in $\mathcal{A}[x]$.*

Looking ahead to lambda calculus notation in the next section, we write $h \equiv \lambda_{x:A}.\varphi(x)$, so the equation above becomes $ev{\circ}\langle \lambda_{x:A}.\varphi(x), x\rangle \underset{x}{=} \varphi(x)$. The universal property of polynomial algebras guarantees a notion of *substitution of constants* $1 \xrightarrow{a} A \in \mathcal{A}$ for indeterminates x in $\varphi(x)$. We obtain the following:

Corollary 2.11 (The β rule) *In the situation above, for any arrow $1 \xrightarrow{a} A \in \mathcal{A}$*

$$ev{\circ}\langle \lambda_{x:A}.\varphi(x), a\rangle = \varphi(a) \tag{4}$$

holds in $\mathcal{A}$.

The β-rule is the foundation of the *lambda calculus*, fundamental in programming language theory. It says the following: we think of $\lambda_{x:A}.\varphi(x)$ as the function $x \mapsto \varphi(x)$. Equation 4 says: evaluating the function $\lambda_{x:A}.\varphi(x)$ at argument a is just substitution of the constant a for each occurrence of x in $\varphi(x)$. However this process is far more sophisticated than simple polynomial substitution in algebra. In our situation, the argument a may itself be a lambda term, which in turn may contain other lambda terms applied to various arguments, etc. After substitution, the right hand side $\varphi(a)$ of Equation 4 may be far more complex than the left hand side, with many new possibilities for evaluations created by the substitution. Thus, if we think of *computation* as oriented rewriting from the LHS to the RHS, it is not at all obvious the process ever halts. The fact that it does is a basic theorem in the so-called Operational Semantics of typed lambda calculus. Indeed, the Strong Normalization Theorem (cf. [LS86], p. 81) says *every* sequence of ordered rewrites (from left to right) eventually halts at an irreducible term (cf. Remark 2.49 and Section 2.7 below).

Remark 2.12 We may also form polynomial ccc's $\mathcal{A}[x_1, \cdots, x_n]$ by adjoining a finite set of indeterminates $1 \xrightarrow{x_i} A_i$. Using product types, one may show $\mathcal{A}[x_1, \cdots, x_n] \cong \mathcal{A}[z]$, for an indeterminate $1 \xrightarrow{z} A_1 \times \cdots \times A_n$.

Polynomial cartesian or cartesian closed categories $\mathcal{A}[x]$ may be constructed directly, showing they are the Kleisli category of an appropriate comonad on $\mathcal{A}$ (see [LS86], p.56). Extensions of this technique to allow adjoining indeterminates to fibrations, using 2-categorical machinery are considered in [HJ95].

2.2 Simply Typed Lambda Calculi

Lambda Calculus is an abstract theory of functions developed by Alonzo Church in the 1930's. Originally arising in the foundations of logic and computability theory, more recently it has become an essential tool in the mathematical foundations of programming languages [Mit96]. The calculus itself, to be described below, encompasses the process of building functions from variables and constants, using application and functional abstraction.

Actually, there are many "lambda calculi"–typed and untyped–with various elaborate structures of types, terms, and equations. Let us give the basic typed one. We shall follow an algebraic syntax as in [LS86].

Definition 2.13 (Typed λ-calculus) Let Sorts be a set of sorts (or atomic types). The *typed λ-calculus generated by* Sorts is a formal system consisting of three classes: Types, Terms and Equations between terms. We write $a : A$ for "a is a term of type A".

Types: This is the class obtained from the collection of Sorts using the following rules: Sorts are types, 1 is a type, and if A and B are types then so are $A \times B$ and B^A. We allow the possibility of other types or type-forming operations and possible identifications between types.

Terms: To every type A we assign a denumerable set of typed variables $x_i^A : A, \; i = 0, 1, 2, \ldots$. We write $x : A$ or x^A for a typical variable x of type A. Terms are freely generated from variables, constants, and term-forming operations. We require at least the following distinguished generators:

1. $* : 1$,
2. If $a : A, b : B, c : A \times B$, then $\langle a, b\rangle : A \times B$, $\pi_1^{A,B}(c) : A, \pi_2^{A,B}(c) : B$,
3. If $a : A, f : B^A, \varphi : B$ then $ev_{A,B}(f, a) : B, \lambda_{x:A}.\varphi : B^A$.

There may be additional constants and term-forming operations besides those specified.

We shall abbreviate $ev_{A,B}(f, a)$ by $f\text{`}a$, read "*f of a*", omitting types when clear. Intuitively, $ev_{A,B}$ denotes evaluation, $\langle -, -\rangle$ denotes pairing, and $\lambda_{x:A}.\varphi$ denotes the function $x \mapsto \varphi$, where φ is some term expression possibly containing x. The operator $\lambda_{x:A}$ acts like a quantifier, so the variable x in $\lambda_{x:A}.\varphi$ is a bound (or dummy) variable, just like the x in $\forall_{x:A}\varphi$ or in $\int f(x)dx$. We inductively define the sets of free and bound variables in a term t, denoted $FV(t), BV(t)$, resp. (cf. [Bar84], p. 24). We shall always identify terms up to renaming of bound variables. The expression $\varphi[a/x]$ denotes the result of substituting the term $a : A$ for each occurrence of $x : A$ in φ, if necessary renaming bound variables in φ so that no clashes occur (cf. [Bar84].) Terms without free variables are called *closed*; otherwise, *open*.

Equations between terms: A *context* Γ is a finite set of (typed) variables. An *equation in context* Γ is an expression $a \underset{\Gamma}{=} a'$, where a, a' are terms of the same type A whose free variables are contained in Γ.

The equality relation between terms (in context) of the same type is generated using (at least) the following axioms and closure under the following rules:

(i) $\underset{\Gamma}{=}$ is an equivalence relation,

(ii) $\dfrac{a \underset{\Gamma}{=} b}{a \underset{\Delta}{=} b}$, whenever $\Gamma \subseteq \Delta$

(iii) $\underset{\Gamma}{=}$ must be a congruence relation with respect to all term-forming operations. It suffices to consider closure under the following two rules (cf. [LS86])

$$\frac{a \underset{\Gamma}{=} b}{f\text{`}a \underset{\Gamma}{=} f\text{`}b} \qquad \frac{\varphi \underset{\Gamma \cup \{x^A\}}{=} \varphi'}{\lambda_{x:A}.\varphi \underset{\Gamma}{=} \lambda_{x:A}.\varphi'}$$

(iv) The following specific axioms (we omit subscripts on terms, when the types are obvious):

Products

(a) $a \underset{\Gamma}{=} *$ for all $a : 1$,

(b) $\pi_1(\langle a,b\rangle) \underset{\Gamma}{=} a$ for all $a : A, b : B$,

(c) $\pi_2(\langle a,b\rangle) \underset{\Gamma}{=} b$ for all $a : A, b : B$,

(d) $\langle \pi_1(c), \pi_2(c)\rangle \underset{\Gamma}{=} c$ for all $c : C$,

Lambda Calculus

β-Rule $(\lambda_{x:A}.\varphi)`a \underset{\Gamma}{=} \varphi[a/x]$,

η-Rule $\lambda_{x:A}.(f`x) \underset{\Gamma}{=} f$, where $f : B^A$ and x is not a free variable of f.

Remark 2.14 There may be additional types, terms, or equations. Following standard conventions, we equate terms which only differ by change of bound variables–this is called α- conversion in the literature [Bar84]. Equations are *in context*–i.e. occur within a declared set of free variables. This allows the possibility of *empty types*, i.e types without closed terms (of that type). This view is fundamental in recent approaches to functional languages [Mit96] and necessary for interpreting such theories in presheaf categories, for example. However, if there happen to be closed terms $a : A$ of each type, we may omit the subscript Γ on equations, because of the following derivable rule (cf. [LS86], Prop. 10.1, p.75): for $x \notin \Gamma$ and if all free variables of a are contained in Γ,

$$\frac{\varphi(x) \underset{\Gamma\cup\{x\}}{=} \psi(x)}{\varphi[a/x] \underset{\Gamma}{=} \psi[a/x]}$$

Example 2.15 *Freely generated simply typed lambda calculi.* These are freely generated from specified sorts, terms, and/or equations. In the minimal case (no additional assumptions) we obtain the simply typed lambda calculus with finite products freely generated by Sorts. Typically, however, we assume that among the Sorts are distinguished *datatypes* and associated terms, possibly with specified equations. For example, basic universal algebra would be modelled by sorts A with distinguished n-ary operations given by terms $t : A^n \Rightarrow A$ and constants $c : 1 \to A$. Any specified term equations are added to the theory as (nonlogical) axioms.

Example 2.16 *The internal language of a ccc $\mathcal{A}$.* Here the types are the objects of $\mathcal{A}$, where $\times, (-)^{(-)}, 1$ have the obvious meanings. Terms with free variables $x_1 : A_1, \cdots, x_n : A_n$ are polynomials in $\mathcal{A}[x_1, \cdots, x_n]$, where $1 \xrightarrow{x_i} A_i$ is an indeterminate, lambda abstraction is given by functional completeness, as in Proposition 2.10, and we define $a \underset{X}{=} b$ to hold iff $a \underset{X}{=} b$ as polynomials in $\mathcal{A}[X]$, where $X = \{x_1, \cdots, x_n\}$.

Remark 2.17

(i) Historically, typed lambda calculi were often presented with *only* exponential types B^A (no products) and the associated machinery [Bar84, Bar92]. This permits certain simplifications in inductive arguments, athough it is categorically less “natural” (cf. also Remark 2.24).

(ii) It is a fundamental property that lambda calculus is a *higher-order* functional language: terms of type B^A can use an arbitrary term of type A as an argument, and A and B themselves may be very complex. Thus, typed lambda calculus is often referred to as a theory of *functionals of higher type.*

2.3 Formulas-as-Types: the fundamental equivalence

Let us describe the third component of the trio: cartesian closed categories, typed lambda calculi, and formulas-as-types. The *Formulas-as-Types* view, sometimes called the Curry-Howard

Formulas	$A ::= \top \mid Atoms \mid A_1 \wedge A_2 \mid A_1 \Rightarrow A_2$
Provability	$\vdash$ is a reflexive, transitive relation such that, for arbitrary formulas A, B, C
	$A \vdash \top \quad , \quad A \wedge B \vdash A \quad , \quad A \wedge B \vdash B$
	$C \vdash A \wedge B$ iff $C \vdash A$ and $C \vdash B$
	$C \wedge A \vdash B$ iff $C \vdash A \Rightarrow B$

Figure 2: Intuitionistic $\top, \wedge, \Rightarrow$ Logic

isomorphism, is playing an increasingly influential role in the logical foundations of computing, especially in the foundations of functional programming languages. Its historical roots lie in the so-called Brouwer-Heyting-Kolmogorov (BHK) interpretation of intuitionistic logic from the 1920's[GLT, TrvD88]. The idea is based on modelling proofs (which are programs) by functions, i.e. lambda terms. Since proofs can be modelled by lambda terms and the latter are themselves arrows in certain free categories, it follows that functional programs can be modelled categorically.

In modern guise, the Curry-Howard analysis says the following. Proofs in a constructive logic $\mathcal{L}$ may be identified as terms of an appropriate typed lambda calculus $\lambda_{\mathcal{L}}$, where:

- types = formulas of $\mathcal{L}$,
- lambda terms = proofs (i.e. annotations of Natural Deduction proof trees),
- provable equality of lambda terms corresponds to the equivalence relation on proofs generated by Gentzen's normalization algorithm.

Often researchers impose additional equations between lambda terms, motivated from categorical considerations (e.g. to force traditional datatypes to have a strong universal mapping property).

Remark 2.18 (Formulas = Specifications) More generally, the Curry-Howard view identifies types of a programming language with formulas of some logic, and programs of type A as proofs within the logic of formula A. Constructing proofs of formula A may then be interpreted as building programs that meet the specification A.

For example, consider the intuitionistic $\{\top, \wedge, \Rightarrow\}$-fragment of propositional calculus, as in Figure 2. This logic closely follows the presentation of ccc's in Definition 2.1 and Figure 1. We now identity (= Formulas-as-Types) the propositional symbols $\top, \wedge, \Rightarrow$ with the type constructors $1, \times, \Rightarrow$, respectively. We assign lambda terms inductively. To a proof of $A \vdash B$ we assign λ-terms $x : A \vdash t(x) : B$, where $t(x)$ is a term of type B with at most the free variable $x : A$ (i.e. in context $\{x : A\}$) as follows:

$$x : A \vdash x : A \qquad , \qquad \frac{x : A \vdash s(x) : B \quad y : B \vdash t(y) : C}{x : A \vdash t[s(x)/y] : C} \qquad ,$$

$$x : A \vdash * : \top \qquad , \qquad x : A \wedge B \vdash \pi_1(x) : A \qquad , \qquad x : A \wedge B \vdash \pi_2(x) : B \qquad ,$$

$$\frac{x : C \vdash a : A \quad x : C \vdash b : B}{x : C \vdash \langle a, b \rangle : A \wedge B} \qquad , \qquad \frac{z : C \wedge A \vdash t(z) : B}{y : C \vdash \lambda_{x:A}.t[\langle y, x \rangle / z] : A \Rightarrow B} \qquad ,$$

$$\frac{y : C \vdash t(y) : A \Rightarrow B}{z : C \wedge A \vdash t[\pi_1(z)/y]`\pi_2(z) : B}$$

We can now refer to entire proof trees by the associated lambda terms. We wish to put an equivalence relation on proofs, according to the equations of typed lambda calculus. Given two proofs of an entailment $A \vdash B$, say $x : A \vdash s(x) : B$ and $x : A \vdash t(x) : B$, we say they are *equivalent* if we can derive $s \underset{\{x\}}{=} t$ in the appropriate typed lambda calculus.

Definition 2.19 Let λ-**Calc** denote the category whose objects are typed lambda calculi and whose morphisms are *translations*, i.e. maps Φ which send types to types, terms to terms (including mapping the ith variable of type A to the ith variable of type $\Phi(A)$), preserve all the specified operations on types and terms on the nose, and preserve equations.

Theorem 2.20 *There are a pair of functors $C : \lambda\text{-}\boldsymbol{Calc} \rightarrow Cart_{st}$ and $L : Cart_{st} \rightarrow \lambda\text{-}\boldsymbol{Calc}$ which set up an equivalence of categories $Cart_{st} \cong \lambda\text{-}\boldsymbol{Calc}$.*

The functor L associates to ccc $\mathcal{A}$ its internal language, while the functor C associates to any lambda calculus $\mathcal{L}$, a syntactically generated ccc $C(\mathcal{L})$, whose objects are types of $\mathcal{L}$ and whose arrows $A \rightarrow B$ are denoted by (equivalence classes of) lambda terms $t(x)$ representing proofs $x : A \vdash t(x) : B$ as above (see [LS86]).

This leads to a kind of Soundness Theorem for diagrammatic reasoning which is important in categorical logic.

Corollary 2.21 *Verifying that a diagram commutes in a ccc $\mathcal{C}$ is equivalent to proving an equation in the internal language of $\mathcal{C}$.*

The above result includes allowing algebraic theories modelled in the cartesian fragment [Mac82, Cr93], as well as extensions with categorical data types (like weak natural numbers objects, see Section 2.3.1.) Theorem 2.20 also leads to concrete syntactic presentations of free ccc's [LS86, Tay98]. Let **Graph** be the category of directed multi-graphs [ST96].

Corollary 2.22 *The forgetful functor $\mathcal{U} : Cart_{st} \rightarrow Graph$ has a left adjoint $\mathcal{F} : Graph \rightarrow Cart_{st}$. Let $\mathcal{F}_{\mathcal{G}}$ denote the image of graph $\mathcal{G}$ under $\mathcal{F}$. We call $\mathcal{F}_{\mathcal{G}}$ the free ccc generated by $\mathcal{G}$.*

Example 2.23 Given a discrete graph $\mathcal{G}_0$ considered as a a set, $\mathcal{F}_{\mathcal{G}_0}$ = the free ccc generated by the set of sorts $\mathcal{G}_0$. It has the following universal property: for any ccc $\mathcal{C}$ and for any graph morphism $F : \mathcal{G}_0 \rightarrow \mathcal{C}$, there is a unique extension to a (strict) ccc-functor $[\![-]\!]_F : \mathcal{F}_{\mathcal{G}_0} \rightarrow \mathcal{C}$.

$$\begin{tikzcd}
\mathcal{F}_{\mathcal{G}_0} \arrow[r, "{[\![-]\!]_F}"] & \mathcal{C} \\
\mathcal{G}_0 \arrow[u] \arrow[ur, "F"'] &
\end{tikzcd}$$

This says: given any interpretation F of basic atomic types (= nodes of $\mathcal{G}_0$) as objects of $\mathcal{C}$, there is a unique extension to an interpretation $[\![-]\!]_F$ in $\mathcal{C}$ of the entire simply typed lambda calculus generated by $\mathcal{G}_0$ (identifying the free ccc $\mathcal{F}_{\mathcal{G}_0}$ with this lambda calculus.)

Remark 2.24 A Pitts [Pi9?] has shown how to construct free ccc's syntactically, using lambda calculi without product types. The idea is to take objects to be *sequences* of types and arrows

Objects	Distinguished Arrow(s)	Equations
Initial 0	$0 \xrightarrow{O_A} A$	$O_A = f\ ,$ $f : 0 \to A$
Coproducts $A+B$	$in_1^{A,B} : A \to A+B$ $in_2^{A,B} : B \to A+B$	$[f,g]{\circ}in_1 = f$ $[f,g]{\circ}in_2 = g$
	$\dfrac{A \xrightarrow{f} C \quad B \xrightarrow{g} C}{A+B \xrightarrow{[f,g]} C}$	$[h{\circ}in_1, h{\circ}in_2] = h\ ,$ $h : A+B \to C$

Figure 3: Coproducts

to be sequences of terms. The terminal object is the empty sequence, while products are given by concatenation of sequences. For a full discussion, see [CDS97] This is useful in reduction-free normalization (see Section 2.7 below).

Remark 2.25 There are more advanced 2- and bi-categorical versions of the above results. We shall mention more structure in the case of cartesian closed categories with coproducts, in the next section.

2.3.1 Some Datatypes

Computing requires datatypes, for example natural numbers, lists, arrays, etc. The categorical development of such datatypes is an old and established area. The reader is referred to any of the standard texts for discussion of the basics, e.g. [MA86, BW95, Mit96, Ten94]. General categorical treatments of abstract datatypes abound in the literature. The standard treatment is to use initial T-algebras (cf. Section 2.4.2 below) or final T-coalgebras for "definable" or "polynomial" endofunctors T. There are interesting common generalizations to lambda calculi with functorial type constructors [Ha87, Wr89], categories with datatypes determined by strong monads[Mo91, CSp91], and using enriched categorical structures [K82]. There is recent discussion of datatypes in distributive categories, [Co93, W92] and the use of the categorical theory of sketches [BW95, Bor94].

We shall merely illustrate a few elementary algebraic structures commonly added to a cartesian or cartesian closed category (or the associated term calculi).

Definition 2.26 A category $\mathcal{C}$ *has finite coproducts* (equivalently, binary coproducts and an initial object 0) if for every $A, B \in \mathcal{C}$ there is a distinguished object $A + B$, together with isomorphisms (natural in $A, B, C \in \mathcal{C}$)

$$Hom_{\mathcal{C}}(0, A) \cong \{*\} \tag{5}$$

$$Hom_{\mathcal{C}}(A + B, C) \cong Hom_{\mathcal{C}}(A, C) \times Hom_{\mathcal{C}}(B, C) \tag{6}$$

We say $\mathcal{C}$ is *bicartesian closed* (= biccc) if it is a ccc with finite coproducts [1].

Just as in the case of products (cf. Figure 1), we may present coproducts equationally, as in Figure 3, and speak of *strict* structure, etc. In programming language semantics, coproducts correspond to *variant types*, set-theoretically they correspond to disjoint union, while from the logical viewpoint coproducts correspond to disjunction. Thus a biccc corresponds to intuitionistic $\{\bot, \top, \wedge, \vee, \Rightarrow\}$-logic. We add to the logic of Figure 2 formulas $\bot$ and $A_1 \vee A_2$, together with the

[1] Not to be confused with *bicategories*, cf. [Bor94]

rules

$$\perp \vdash A$$
$$A \vee B \vdash C \qquad \text{iff} \qquad A \vdash C \text{ and } B \vdash C$$

corresponding to equations (5),(6). The associated typed lambda calculus with coproducts is rather subtle to formulate [Mit96, GLT]. The problem is with the copairing operator $A + B \xrightarrow{[f,g]} C$ which in *Sets* corresponds to a *definition-by-cases* operator:

$$[f,g](x) = \begin{cases} f(x) & \text{if } x \in A \\ g(x) & \text{if } x \in B \end{cases}$$

The correct lambda calculus formalism for coproduct types corresponds to the logicians' natural deduction rules for strong sums. The issue is not trivial, since the word problem for free biccc's (and the associated type isomorphism problem [DiCo95]) is among the most difficult of this type of question, and—at least for the current state of the art—depends heavily on technical subtleties of syntax for its solution (see [Gh96]).

Just as for ccc's, we may introduce various 2-categories of biccc's (cf. [Cu93]). For example

Definition 2.27 The 2-category $2 - BiCART_{st}$ has 0-cells strict bicartesian closed categories, 1-cells functors preserving the structure on the nose, and 2-cells natural isomorphisms.

One may similarly define a non-strict version $2 - BiCART$.

Remark 2.28 Every bicartesian closed category is equivalent to a strict one. Indeed, this is part of a general 2-categorical adjointness between the above 2-categories, from a theorem of Blackwell, Kelly, and Power. (See Čubrić [Cu93] for applications to lambda calculi.).

Definition 2.29 In a biccc, define **Boole** $= 1 + 1$, the type of booleans.

Boole's most salient feature is that it has two distinguished global elements (boolean values) $T, F : 1 \to Boole$, corresponding to the two injections in_1, in_2, together with the universal property of coproducts. In *Set* we interpret *Boole* as a set of cardinality 2; similarly, in typed lambda calculus, it corresponds to a type with two distinguished constants $T, F : Boole$ and an appropriate notion of definition by cases . In any biccc, we can define all of the classical n-ary propositional logic connectives as arrows $Boole^n \to Boole$ (see [LS86], I.8). A weaker notion of booleans in the category ω-CPO$_\perp$ is illustrated in Figure 4.

Definition 2.30 A *natural numbers object* in a ccc $\mathcal{C}$ is an object N with arrows $1 \xrightarrow{0} N \xrightarrow{S} N$ which is initial among diagrams of that shape. That is, for any object A and arrows $1 \xrightarrow{a} A \xrightarrow{h} A$, there is a unique *iterator* $\mathcal{I}_{ah} : N \to A$ making the following diagram commute:

$$\begin{tikzcd}
1 \arrow[r, "0"] \arrow[dr, "a"'] & N \arrow[r, "S"] \arrow[d, "\mathcal{I}_{ah}"] & N \arrow[d, "\mathcal{I}_{ah}"] \\
 & A \arrow[r, "h"] & A
\end{tikzcd}$$

A *weak natural numbers object* is defined as above, but just assuming existence and not necessarily uniqueness of $\mathcal{I}_{ah}$.

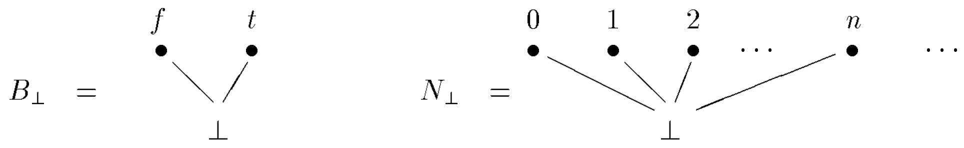


Figure 4: Flat Datatypes in ω-CPO$_\perp$

In the category Set, the natural numbers $(N, 0, S)$ is a natural numbers object, where $Sn = n{+}1$. In functor categories $Set^{\mathcal{C}}$, a natural numbers object is given by the constant functor K_N, where $K_N(A) = N$, and $K_N(f) = id_N$, with obvious natural transformations $1 \xrightarrow{0} K_N \xrightarrow{S} K_N$. In ω-**CPO** there are numerous *weak* natural numbers objects: for example the *flat pointed* natural numbers $N_\perp = N \uplus \{\perp\}$, ordered as follows: $a \leq b$ iff $a = b$ or $a =\perp$, where $S(n) = n + 1$ and $S(\perp) =\perp$, pictured in Figure 4.

Natural numbers objects—when they exist—are unique up to isomorphism; however weak ones are far from unique. Typical programming languages and typed lambda calculi in logic assume only weak natural numbers objects.

If a ccc $\mathcal{C}$ has a natural numbers object N, we can construct parametrized versions of iteration, using products and exponentiation in $\mathcal{C}$ [LS86, FrSc]. For example, in Set: given functions $g : A \to B$ and $f : N \times A \times B \to B$, there exists a unique *primitive recursor* $\mathcal{R}_{gf} : N \times A \to B$ satisfying: (i) $\mathcal{R}_{gf}(0, a) = g(a)$ and $\mathcal{R}_{gf}(Sn, a) = f(n, a, \mathcal{R}_{gf}(n, a))$. These equations are easily represented in any ccc with N, or in the associated typed lambda calculus (e.g. the number $n \in N$ being identified with $S^n 0$). In the case $\mathcal{C}$ has only a weak natural numbers object, we may prove the existence but not necessarily the uniqueness of $\mathcal{R}_{gf}$.

An important datatype in Computer Science is the type of finite lists of elements of some type A. This is defined analogously to (weak) natural numbers objects:

Definition 2.31 Given an object A in a ccc $\mathcal{C}$, we define the object $\ell ist(A)$ of *finite lists on A* with the following distinguished structure: arrows $\mathsf{nil} : 1 \to \ell ist(A)$, $\mathsf{cons} : A \times \ell ist(A) \to \ell ist(A)$ satisfying the following (weak) universal property: for any object B and arrows $b : 1 \to B$ and $h : A \times B \to B$, there exists an "iterator" $\mathcal{I}_{bh} : \ell ist(A) \to B$ satisfying (in the internal language):

$$\mathcal{I}_{bh}\mathsf{nil} = b \qquad \mathcal{I}_{bh}\mathsf{cons}\langle a, w\rangle = h\langle a, \mathcal{I}_{bh} w\rangle.$$

Here nil corresponds to the empty list, and cons takes an element of A and a list and concatenates the element onto the head of the list.

Analogously to (weak) natural numbers objects N, we can use product types and exponentiation to extend iteration on $\ell ist(A)$ to *primitive recursion with parameters* (cf. [GLT], p. 92).

What n-ary numerical Set functions are represented by arrows $N^n \to N$ in a ccc? The answer, of course, depends on the ccc. In general, the best we could expect is the following (cf. [LS86], Part III, Section 2):

Proposition 2.32 *Let $\mathcal{F}_N$ be the free ccc with weak natural numbers object. The class of numerical total functions representable therein is properly contained between the primitive recursive and the Turing-machine computable functions.*

In general, such fast-growing functions as the Ackermann function are representable in any ccc with weak natural numbers object (see [LS86]). Analogous results hold for symmetric monoidal and monoidal closed categories, [PR89].

The question of *strong* versus *weak* datatypes is of some interest. For example, although we can define addition $+ : N \times N \to N$ by primitive recursion on a weak natural numbers type, *commutativity* of addition follows from having a strong natural numbers object; a weak parametrized primitive recursor would only allow us to derive $x+n = n+x$ for each closed numeral n but we cannot then extend this to *variables* (this is similar to consequences of Gödel's incompleteness theorem, cf. [LS86], p. 263). Notice that, on the face of it, the definition of a natural numbers object appears not to be equational: informally, uniqueness of the arrow $\mathcal{I}_{ah}$ requires an *implication*: *for all* $f : N \to A$ *(if* $f0 = a$ *and* $fS = hf$ *) then* $f = \mathcal{I}_{ah}$.

Here we remark on a curious observation of Lambek [L88]. Let us recall from universal algebra that a Mal'cev operator on an algebra A is a function $m_A : A^3 \to A$ satisfying $m_A xxz = z$ and $m_A xzz = x$. For example, if A were a group, $m_A = xy^{-1}z$ is such an operator. Similarly, the definition of a Mal'cev operator on an object A makes sense in any ccc (e.g. as an arrow $A^3 \xrightarrow{m_A} A$ satisfying some diagrams) or, equivalently, in any typed lambda calculus (e.g. as a closed term $m_A : A^3 \Rightarrow A$ satisfying some equations) .

Theorem 2.33 (Lambek) *Let $\mathcal{C}$ be a ccc with weak natural numbers $(N, 0, S)$ in which each object A has a Mal'cev operator m_A. Then the fact that $(N, 0, S)$ is a natural numbers object is equationally definable using the family $\{m_A \mid A \in \mathcal{C}\}$. In particular, if $\mathcal{C} = \mathcal{F}_N$, the free ccc with weak natural numbers object, there are a finite number of additional equations (as schema) that, when added to the original data, guarantee that every type has a Mal'cev operator and N is a natural numbers object.*

2.4 Polymorphism

"The perplexing subject of polymorphism."
C. Darwin *Life & Lett*, 1887

Although Darwin was speaking of biology, he might very well have been discussing computer science 100 years later. Christopher Strachey in the 1960's introduced various notions of polymorphism into programming language design (see [Rey83, Mit96]). Perhaps the most influential was his notion of *parametric polymorphism*. Intuitively, a parametric polymorphic function is one which has a *uniformly given algorithm at all types*. Imagine a "generic" algorithm capable of being instantiated at any arbitrary type, but which is the "same algorithm" at each type instance. It is this idea of the "plurality of form" which inspired the biological metaphor.

Example 2.34 (Reverse) Consider a simple algorithm that takes a finite list and reverses it. Here "lists" could mean: lists of natural numbers, lists of reals, lists of arrays, indeed lists of lists of .... The point is, the types do not matter: we have a uniform algorithm for all types. Let $list(\alpha)$ denote the type of finite lists of entities of type α.We thus might type this algorithm

$$rev_\alpha : list(\alpha) \Rightarrow list(\alpha)$$

where $rev_\alpha(a_1, \cdots, a_n) = (a_n, \cdots, a_1)$.
A second example, discussed by Strachey, is
Example 2.35 (Map-list) This algorithm begins with a function of type $\alpha \Rightarrow \beta$ and a finite α-list, applies the function to each element of the list, and then makes a β-list of the subsequent values. We might represent it as:

$$map_{\alpha,\beta} : (\alpha \Rightarrow \beta) \Rightarrow (list(\alpha) \Rightarrow list(\beta))$$

where $map_{\alpha,\beta}(f)(a_1, \cdots, a_n) = (f(a_1), \cdots, f(a_n))$.

Formulas	$A ::= vbl \mid A_1 \Rightarrow A_2 \mid \forall\alpha.A$
Provability	$\vdash$ is a relation between finite sets of formulas and formulas
	$\Gamma \vdash A$ if $A \in \Gamma$
	$\dfrac{\Gamma \cup \{A\} \vdash B}{\Gamma \vdash A \Rightarrow B}$, $\dfrac{\Gamma \vdash A \quad \Delta \vdash A \Rightarrow B}{\Gamma \cup \Delta \vdash B}$
	$\dfrac{\Gamma \vdash A(\alpha)}{\Gamma \vdash \forall\alpha A(\alpha)}$, $\dfrac{\Gamma \vdash \forall\alpha A(\alpha)}{\Gamma \vdash A[B/\alpha]}$
	where $\alpha \notin FV(\Gamma)$ for any formula B .

Figure 5: Second Order Intuitionistic Propositional Calculus

Many recent programming languages (e.g. ML, Ada) support sophisticated uses of generic types and polymorphism. The mathematical foundations of such languages were a major challenge in the past decade and category theory played a fundamental role. We shall briefly recall the issues.

2.4.1 Polymorphic lambda calculi

The logician J-Y. Girard[Gi71, Gi72] in a series of important works examined higher-order logic from the Curry-Howard viewpoint. He developed formal calculi of variable types, the so-called polymorphic lambda calculi, which correspond to proofs in higher-order logics. At the same time he developed the proof theory of such systems . J. Reynolds[Rey74] independently discovered the second-order fragment of Girard's system, and proposed it as a syntax representing Strachey's parametric polymorphism.

Let us briefly examine Girard's *System $\mathcal{F}$*, second order polymorphic lambda calculus. The underlying logical system is intuitionistic second order propositional calculus. The latter theory is similar to ordinary propositional calculus, except we can universally quantify over propositional variables.

The syntax of second order propositional calculus is presented in Figure 5.

The usual notions of free and bound variables in formulas are assumed. For example, in $\forall\alpha(\alpha \Rightarrow \beta)$, α is a bound variable, while β is free. $A[B/\alpha]$ denotes A with formula B substituted for free α, changing bound variables if necessary to avoid clashes. Notice in the quantifier rules that when we instantiate a universally quantified formula to obtain, say, $\Gamma \vdash A[B/\alpha]$, the formula B may be of arbitrary logical complexity. Thus inductive proof techniques based on the complexity of subformulas are not available in higher-order logic. This is the essence of the problem of *impredicativity* in polymorphism.

We now introduce Girard's second order lambda calculus. We use the notation $FV(t)$ and $BV(t)$ for the set of free and bound variables of term t, respectively. We write $FTV(A)$ and $BTV(A)$ for the set of free type variables and bound type variables of formula A, respectively.

Definition 2.36 (Girard's System $\mathcal{F}$)

Types: Freely generated from type variables $\alpha, \beta, \cdots$ by the rules: if A, B are types, so are $A \Rightarrow B$ and $\forall\alpha.A$.

Terms: Freely generated from variables x_i^A of every type A by

(1) First-order lambda calculus rules: if $f : A \Rightarrow B, a : A, \varphi : B$ then $f`a : B$ and $\lambda_{x:A}.\varphi : A \Rightarrow B$.

(2) Specifically second-order rules:

  (a) If $t : A(\alpha)$, then $\Lambda\alpha.t : \forall\alpha A(\alpha)$ where $\alpha \notin FTV(FV(t))$,

  (b) If $t : \forall\alpha A(\alpha)$ then $t[B] : A[B/\alpha]$ for any type B.

Equations: Equality is the smallest congruence relation closed under β and η for both lambdas, that is:

(3) $(\lambda_{x:A}.\varphi)`a \;=_{\beta^1}\; \varphi[a/x]$ and $\lambda_{x:A}(f`x) \;=_{\eta^1}\; f$, where $x \notin FV(f)$.

(4) $(\Lambda\alpha.\psi)[B] \;=_{\beta^2}\; \psi[B/\alpha]$ and $\Lambda\alpha.t[\alpha] \;=_{\eta^2}\; t$, where $\alpha \notin FTV(t)$.

Equations (3) are the first order $\beta\eta$ equations, while equations (4) are second order $\beta\eta$.

From the Curry-Howard viewpoint, the types of $\mathcal{F}$ are precisely the formulas of second order propositional calculus (Fig. 5), while terms denote proofs. For example, to annotate second order rules we have:

$$\frac{\vec{x} : \Gamma \vdash t : A(\alpha)}{\vec{x} : \Gamma \vdash \Lambda\alpha.t : \forall\alpha A(\alpha)} \;, \qquad \frac{\vec{x} : \Gamma \vdash t : \forall\alpha A(\alpha)}{\vec{x} : \Gamma \vdash t[B] : A[B/\alpha]}$$

The $\beta\eta$ equations of course express equality of proof trees.

What about polymorphism? Suppose we think of a term $t : \forall\alpha A(\alpha)$ as an algorithm of type $A(\alpha)$ varying uniformly over all types α. Then $t[B] : A[B/\alpha]$ is the instantiation of t at the specific type B. Moreover, B may be arbitrarily complex. Thus the type variable acts as a parameter.

In System $\mathcal{F}$ we can internally represent common inductive data types within the syntax as *weak* T-algebras, for covariant definable functors T. Weakness refers to the categorical fact that these structures satisfy existence but not uniqueness of the mediating arrow in the universal mapping property. Thus, for any types A, B we are able to define the types $1, Nat, List(A), A \times B, A + B, \exists\alpha.A$, etc. (see [GLT] for a full treatment) .

Let us give two examples and at the same time illustrate polymorphic instantiation.

Example 2.37 The type of booleans is given by

$$Boole = \forall\alpha.(\alpha \Rightarrow (\alpha \Rightarrow \alpha))$$

It has two distinguished elements $T, F : Boole$ given by $T = \Lambda\alpha.\lambda_{x:\alpha}.\lambda_{y:\alpha}.x$ and $F = \Lambda\alpha.\lambda_{x:\alpha}.\lambda_{y:\alpha}.y$, together with a *Definition by Cases* operator (for each type A) $D_A : A \Rightarrow (A \Rightarrow (Boole \Rightarrow A))$ defined by $D_A uvt = (t[A]`u)`v$ where $u, v : A$, $t : Boole$. One may easily verify that $D_A uvT =_\beta u$ and $D_A uvF =_\beta v$. (where β stands for $\beta^1 \cup \beta^2$).

Example 2.38 The type of (Church) numerals

$$Nat = \forall\alpha(\,(\alpha \Rightarrow \alpha) \Rightarrow (\alpha \Rightarrow \alpha)\,)\ .$$

The numeral $n : Nat$ corresponds at each α to n-fold composition $f \mapsto f^n$, where $f^n = f \circ f \circ \cdots \circ f$ (n times) and $f^0 = Id_\alpha = \lambda x_\alpha.x$. Formally, it is the closed term $n = \Lambda\alpha.\lambda_{f:\alpha\Rightarrow\alpha}.f^n : Nat$. Thus for *any* type B we have a uniform algorithm: $n[B] = \lambda_{f:B\Rightarrow B}.f^n : (B \Rightarrow B) \Rightarrow (B \Rightarrow B)$. Successor is given by $S = \lambda_{n:Nat}.n{+}1$, where $n{+}1 = \Lambda\alpha.\lambda_{f:\alpha\Rightarrow\alpha}.f^{n+1} = \Lambda\alpha.\lambda_{f:\alpha\Rightarrow\alpha}.f \circ f^n = \Lambda\alpha.\lambda_{f:\alpha\Rightarrow\alpha} f \circ (n[\alpha]`f)$. Finally, iteration is given by: if $a : A, h : A \Rightarrow A$, $\mathcal{I}_{ah} = \lambda_{x:Nat}.(x[A]`h)`a : Nat \Rightarrow A$. The reader may easily calculate that $\mathcal{I}_{ah}0 =_\beta a$ and $\mathcal{I}_{ah}(n+1) =_\beta h`(\mathcal{I}_{ah}n)$ for numerals n.

Let us illustrate impredicativity in this situation. Recall the discussion of Church vs. Curry typing, Section 2.5.3. Notice that for any type B, $n[B \Rightarrow B]`n[B]$ makes perfectly good sense. In particular, let $B = Nat$, the type of $n$ itself. This is a well-defined term and if we erase all its types we obtain the *untyped* expression $n`n = \lambda f.f^n$. This latter untyped term is *not* typable in simply typed lambda calculus.

Formal systems describing far more powerful versions of polymorphism have been developed. For example, Girard's thesis described the typed lambda calculus corresponding to ω-order intuitionist type theory, so-called $\mathcal{F}_\omega$. Programming in the various levels of Girard's theories $\{\mathcal{F}_n\}$, $n = 1, 2, \cdots, \omega$ is described in [PDM89]. Other systems include Coquand-Huet's Calculus of Constructions and its extensions [Luo94]. These theories include not only Girard's $\mathcal{F}_\omega$ but also Martin-Löf's dependent type theories [H97a]. Indeed, these theories are among the most powerful logics known, yet form the basis of various proof-development systems (e.g. LEGO and Coq) [LP92, D$^+$93].

2.4.2 What is a Model of System $\mathcal{F}$?

The problem of finding—and indeed defining precisely—a model of System $\mathcal{F}$ was difficult. Cartesian closedness is not the issue. The problem, of course, is the universal quantifier: clearly in $\forall\alpha.A$ the α is to range over all the objects of the model, and at the same time $\forall$ should be interpreted as some kind of product (over *all* objects). Such "large" products create havoc, as foreshadowed in the following theorem of Freyd (cf. [Mac71], Proposition 3, p. 110)

Theorem 2.39 (Freyd) *A small category which is small complete is a preorder.*

Cartesian closed preorders (e.g. complete Heyting algebras) are of no interest for modelling proofs; we seek "nontrivial" categories.

Suppose instead we try to define a naive "set-theoretic" model of System $\mathcal{F}$, in which $\times, \Rightarrow$ have their usual meaning, and $\forall\alpha$ is interpreted as a "large" product. Such models are defined in detail in [RP, Pi87]. John Reynolds proved the following

Theorem 2.40 *There is no Set model for System $\mathcal{F}$.*

There is an elegant categorical proof in Reynolds and Plotkin [RP]. Let us sketch the proof, which applies to somewhat more general categories than Set.

Let $\mathcal{C}$ be a category with an endofunctor $T : \mathcal{C} \to \mathcal{C}$. A *$T$-algebra* is an object A together with an arrow $TA \xrightarrow{a} A$. A morphism of T-algebras is a commutative square:

$$\begin{tikzcd} TA \arrow[r, "Tf"] \arrow[d, "a"] & TB \arrow[d, "b"] \\ A \arrow[r, "f"] & B \end{tikzcd}$$

An *initial T-algebra* (resp. *weakly initial T-algebra*) is one for which there exists a unique morphism (resp. there exists a morphism) to any other T-algebra.

We shall be interested in objects and arrows of the model category $\mathcal{C}$ which are "definable", i.e. denoted by types and terms of System $\mathcal{F}$. There are simple covariant endofunctors T on $\mathcal{C}$ whose action on objects is definable by types and whose actions on arrows is definable by terms (of System $\mathcal{F}$). For example, the identity functor $T(\alpha) = \alpha$ and the functor $T(\alpha) = (\alpha \Rightarrow B) \Rightarrow B$, for any fixed B, have this property.

Now it may be shown (see [RP]) that for any definable functor T, the System $\mathcal{F}$ expression $P = \forall\alpha.(T(\alpha) \Rightarrow \alpha) \Rightarrow \alpha$ is a weakly initial T-algebra . *Suppose the ambient model category $\mathcal{C}$ has equalizers of all subsets of arrows* (e.g. Set has this property). Essentially by taking a large equalizer (cf. the Solution Set Condition in Freyd's Adjoint Functor Theorem, [Mac71], p. 116) we could then construct a subalgebra of P which is an initial T-algebra. Call this initial T-algebra $\mathcal{I}$. We then use the following important observation of Lambek :

Proposition 2.41 (Lambek) *If* $T(\mathcal{I}) \xrightarrow{f} \mathcal{I}$ *is an initial T-algebra, then f is an isomorphism.*

Applying this to the definable functor $T(\alpha) = (\alpha \Rightarrow B) \Rightarrow B$, we observe that $T(\mathcal{I}) \cong \mathcal{I}$. In particular, let $\mathcal{C} = Set$ and $B = Boole$, and take the usual Set interpretation of $\times$ as cartesian product and $\Rightarrow$ as the full function space. Notice $card(B) \geq 2$ (since there are always the two distinct closed terms T and F). Hence we obtain a bijection $B^{B^{\mathcal{I}}} \cong \mathcal{I}$, for some set $\mathcal{I}$, which is impossible for cardinality reasons. $\Box$

The search for models of System $\mathcal{F}$ led to some extraordinary phenomena that had considerable influence in semantics of programming languages. Let us just briefly mention the history. Notice that the Reynolds-Plotkin proof depends on a simple cardinality argument, which itself depends on classical set theory. Similarly, the proof of Freyd's result, Theorem 2.39, depends on using classical (i.e. non-intuitionistic) logical reasoning in the metalanguage. This suggests that it is really the non-constructive nature of the category **Sets** that is at fault; if we were to work within a non-classical universe—say within a model of intuitionistic set theory—there is still a chance that we could escape the above problems but still have a "set-theoretical" model of System $\mathcal{F}$. And, from one point of view, that is exactly what happened.

These ambient categories, called toposes [LS86, MM92], are in general models of intuitionistic higher-order logic (or set-theory), and include such categories as functor categories and sheaves on a topological space, as well as **Sets**. Moggi suggested constructing models of System $\mathcal{F}$ based on an internally complete internal full subcategory of a suitable ambient topos. This ambient topos would serve as our constructive set-theory, and function types would still be interpreted as the full "set-theoretical" space of total functions. M. Hyland [Hy88] proved that the Realizability (or Effective) Topos had (non-trivial) such internal category objects. The difficult development and clarification of these internal models was undertaken by many researchers, e.g. D. Scott, M. Hyland, E. Robinson, P. Rosolini, A. Carboni, P. Freyd, A. Scedrov,A. Pitts et. al. (e.g. [HRR, Rob89, Ros90, CPS88, Pi87]).

In a separate development, R. Seely [See87] gave the first general categorical definition of a so-called *external model* of System $\mathcal{F}$, and more generally $\mathcal{F}_\omega$. The definition was based on the theory of *indexed* or *fibred* categories. This view of logic was pioneered by Lawvere[Law69] who emphasized that quantifiers were interpretable as adjoint functors. Pitts[Pi87] clarified the relationship between Seely's models and internal-category models within ambient toposes of presheaves. Moreover, he showed that there are enough such internal models for a Completeness Theorem. It is worth remarking that Pitts' work uses properties of Yoneda embeddings. For general expositions see [AL91]. Extensions of "set-theoretical" models to cases where function spaces include *partial functions* (i.e. non-termination) is in [RR90].

One can *externalize* these internal category models [Hy88, AL91] to obtain ordinary categories. And one such internal category in the Realizability Topos, the *modest sets*, when externalized is precisely the ccc category $Per(N)$ discussed in Section 2.1.

Proposition 2.42 *$Per(N)$ is a model of System $\mathcal{F}$.*

The idea is that in addition to the ccc structure of $Per(N)$, we interpret $\forall$ as a large intersection (the intersection of an arbitrary family of pers is again a per). We shall return to this example in

Section 3.2.

Ironically, in essence this model was already in Girard's original Phd thesis [Gi72]. Later, domain-theoretic models of System $\mathcal{F}$ were considered by Girard in [Gi86] and were instrumental in his development of linear logic.

2.5 The Untyped World

The advantages of types in programming languages are familiar and well-documented (e.g. [Mit96]). Nonetheless, there is an underlying *untyped* aspect of computation, already going back to the original work on lambda calculus and combinatory logic in the 1930's, which often underlies concrete machine implementations. In this early view, developed by Church, Curry, and Schönfinkel, functions were understood in the old-fashioned (pre-Cantor) sense of "rules", as a computational process of going from an argument to a value. Such a functional process could take anything, even itself, as an argument. Let us just briefly mention some key directions (see [Bar84, AGM]).

2.5.1 Models and Denotational Semantics

From the viewpoint of ccc's, untypedness amounts to finding a ccc $\mathcal{C}$ with an object $D \ncong 1$ satisfying the isomorphism

$$D^D \cong D \tag{7}$$

Thus function spaces and elements are "on the same level". It then makes sense to define formal application $f`g$ for constants $f, g : D^D$ by $f`g = ev(f, \varphi(g))$, where $\varphi : D^D \xrightarrow{\cong} D$ is the isomorphism above. In particular, self-application $f`f$ makes perfectly good sense.

Dana Scott found the first semantical (topological) models of the untyped lambda calculus in 1970 [Sc72]; i.e. non-trivial solutions D to "equations" of the form (7) in various ccc's, perhaps the simplest being in ω-**CPO**. This was part of his general investigations (with Christopher Strachey) into the foundations of programming languages , culminating in the so-called Scott-Strachey approach to the semantics of programming languages. Arguably, this has been one of the major arenas in the use of category theory in Computer Science, with an enormous literature. For an introduction, see [AbJu94, Gun92, Ten94].

More generally, one seeks to find non-trivial domains D satisfying certain so-called "recursive domain equations", of the form

$$D \cong \cdots D \cdots \tag{8}$$

where $\cdots D \cdots$ is some expression built from type constructors. The difficulty is that the variable D may appear both co- and contravariantly. Such recursive defining "equations" are used to specify the semantics of numerous notions in computer science, from datatypes in functional programming languages, to modelling nondeterminism, concurrency, etc. (cf. also [DiCo95]).

The seminal early paper on categorical solutions of domain equations is the paper of Smyth and Plotkin [SP82]. More recent work has focussed on Axiomatic and Synthetic Domain Theory (e.g. [AbJu94, FiPl96, ReSt97]) and use of bisimulations and relation-theoretic methods for reasoning about recursive domains [Pi96a]. These methods rely on fundamental work of Peter Freyd on recursive types (e.g [Fre92]).

2.5.2 C-Monoids and Categorical Combinators

On a more algebraic level, a model of untyped lambda calculus is a ccc with (up to isomorphism) one non-trivial object. That is, a ccc $\mathcal{C}$ with an object $D \ncong 1$ satisfying the domain equations:

$$D \cong D^D \cong D \times D\ . \tag{9}$$

An example of such a D in ω-**CPO** is given in [LS86], using the constructions of D. Scott and Smyth-Plotkin mentioned above. An interesting axiomatization of such D's comes from simply considering $Hom_{\mathcal{C}}(D, D) \cong Hom_{\mathcal{C}}(1, D^D)$ as an abstract monoid. It turns out that the axioms are easy to obtain: take the axioms of a ccc, remove the terminal object, *and erase all the types!*. That is (following the treatment in [LS86], p. 93):

Definition 2.43 A *C-monoid* (C for Curry, Church, Combinatory, or CCC) is a monoid $(\mathcal{M}, \circ, id)$ together with extra structure structure $(\pi_1, \pi_2, \varepsilon, (-)^*, \langle -, - \rangle\ )$ where π_i, ε are elements of $\mathcal{M}$, $(-)^*$ is a unary operation on $\mathcal{M}$, and $\langle -, - \rangle$ is a binary operation on $\mathcal{M}$, satisfying untyped versions of the equations of a ccc (cf. Figure 1):

$$\begin{array}{ll} \pi_1\langle a, b\rangle = a & \varepsilon\langle h^*\pi_1, \pi_2\rangle = h \\ \pi_2\langle a, b\rangle = b & (\varepsilon\langle k\pi_1, \pi_2\rangle)^* = k \quad . \\ \langle \pi_1 c, \pi_2 c\rangle = c & \end{array}$$

for any $a, b, c, h, k \in \mathcal{M}$ (where we elide the monoid operation $\circ$).

C-monoids were first discovered independently by D. Scott and J. Lambek around 1980. The elementary algebraic theory and connections with untyped lambda calculus were developed in [LS86] (and independently in [Cur93], where they were called *categorical combinators*). Obviously C-monoids form an equational class; thus, just like for general ccc's, we may form free algebras, polynomial algebras, prove Functional Completeness, etc. The associated internal language is *untyped lambda calculus with pairing operators.* As above, this language is obtained from simply typed lambda calculus by omitting the type **1** and erasing all the types from terms.

The rewriting theory of categorical combinators has been discussed by Curien, Hardin, et. al. (e.g. see [Har93]). Categorical combinators form a particularly efficient mechanism for implementing functional languages; for example, the language CAML is a version of the functional language ML based on categorical combinators (see [Hu90], Part 1).

The deepest mathematical results to date on *the cartesian fragment of* C-monoids were obtained by R. Statman [St96]. Statman characterizes the free cartesian monoid F (in terms of a representation into certain continuous shift operators on Cantor space), as well as characterizing the finitely generated submonoids of F and the recursively enumerable subsets of F. The latter two results are based on projections of (suitably encoded) unification problems.

2.5.3 Church vs Curry Typing

The fundamental feature of the untyped lambda calculus is self-application. The β-rule $\lambda x.\varphi(x)`a = \varphi[a/x]$ is now totally unrestricted with respect to typing constraints. This permits non-halting computations: for example, the term $\Omega =_{def} (\lambda x.x`x)`(\lambda x.x`x)$ has no normal form and only β-reduces to itself, while the fixed point combinator $\mathsf{Y} =_{def} \lambda f.(\lambda x.f`(x`x))`(\lambda x.f`(x`x))$ satisfies $f`(\mathsf{Y}`f) =_{\beta\eta} \mathsf{Y}`f$, hence $\mathsf{Y}`f$ is a fixed point of f, for any f. Hence we immediately obtain: *all terms of the untyped lambda calculus have a fixed point.*

Untyped lambda calculus suggests a different approach to the typed world: "typing" untyped terms. The Church view (which we have adopted here) insists all terms be explicitly typed, starting with the variables. On the other hand Curry, the founder of the related but older subject of Combinatory Logic, had a different view: start with untyped terms, but add "type inference rules" to algorithmically infer appropriate types (when possible). Many modern typed programming languages (e.g. ML) essentially follow this Curry view and use "typing rules" to assign appropriate type schema to untyped terms. This leads to the so-called Type Inference Problem: given an explicitly typed language $\mathcal{L}$ and type erasure function $\mathcal{L} \stackrel{Erase}{\longrightarrow} \mathcal{U}$ (where $\mathcal{U}$ is untyped lambda calculus), decide if an untyped term t satisfies $t = Erase(M)$ for some $M \in \mathcal{L}$. It turns out that a problem of type inference is essentially equivalent to a so-called unification problem, familiar from Logic Programming (cf. [Mit96]). Fortunately in the case of ML and other typed programming languages there are known type inference algorithms; however in general (e.g. for System $\mathcal{F}, \mathcal{F}_\omega, \cdots$) the problem is undecidable. To the best of our knowledge, the Church-vs-Curry view of typed languages has not yet been systematically analyzed categorically.

2.6 Logical Relations and Logical Permutations

Logical relations play an important role in the recent proof theory and semantics of typed lambda calculi [Mit96, Plo80, St85]. Recall the notion of *Henkin model* (Section 2.1) as a subccc of **Set**.

Definition 2.44 Given two Henkin models $\mathcal{A}$ and $\mathcal{B}$, a *logical relation* from $\mathcal{A}$ to $\mathcal{B}$ is a family of binary relations $\mathcal{R} = \{R_\sigma \subseteq A_\sigma \times B_\sigma \mid \sigma \text{ a type }\}$ satisfying (we write $aR_\sigma b$ for $(a,b) \in R_\sigma$) :

1. $*R_1*$

2. $(a,a')R_{\sigma\times\tau}(b,b')$ if and only if $aR_\sigma b$ and $a'R_\tau b'$, for any $(a,a') \in A_{\sigma\times\tau}, (b,b') \in B_{\sigma\times\tau}$, *i.e. ordered pairs are related exactly when their components are.*

3. For any $f \in A_{\sigma\Rightarrow\tau}$, $g \in B_{\sigma\Rightarrow\tau}$, $fR_{\sigma\Rightarrow\tau}\ g$ if and only if for all $a \in A_\sigma, b \in B_\sigma$ ($aR_\sigma b$ implies $fa\ R_\tau\ gb$), *i.e. functions are related when they map related inputs to related outputs.*

For each (atomic) base type b, fix a binary relation $R_b \subseteq A_b \times B_b$. Then: there is a smallest family of binary relations $\mathcal{R} = \{R_\sigma \subseteq A_\sigma \times B_\sigma \mid \sigma \text{ a type }\}$ defined inductively from the R_b's by 1,2,3 above. That is, any property (relation) at base-types can be inductively lifted to a family $\mathcal{R}$ at all higher types, satisfying 1, 2, 3 above. We write $a\mathcal{R}b$ to denote $aR_\sigma b$ for some σ. If $\mathcal{A} = \mathcal{B}$ and $\mathcal{R}$ is a logical relation from $\mathcal{A}$ to itself, we say an element $a \in \mathcal{A}$ is *invariant* under $\mathcal{R}$ if $a\mathcal{R}a$.

The fundamental property of logical relations is the Soundness Theorem [Mit96, St85]. Let $\vec{x} : \Gamma \vdash M : \sigma$ denote that M is a term of type σ with free variables $\vec{x}$ in context Γ. Consider Henkin models $\mathcal{A}$ with $\eta_{\mathcal{A}}$ an assignment function, assigning variables to elements in $\mathcal{A}$. Let $[\![M]\!]_{\eta_{\mathcal{A}}}$ denote the meaning of term M in model $\mathcal{A}$ w.r.t. the given variable assignment (following [Mit96], we only consider assignments η such that $\eta_{\mathcal{A}}(x_i) \in A_\sigma$ if $x_i : \sigma \in \Gamma$). The following is proved by induction on the form of M:

Theorem 2.45 (Soundness) *Let $\mathcal{R} \subseteq \mathcal{A} \times \mathcal{B}$ be a logical relation between Henkin models $\mathcal{A}, \mathcal{B}$. Let $\vec{x} : \Gamma \vdash M : \sigma$. Suppose assignments $\eta_{\mathcal{A}}, \eta_{\mathcal{B}}$ of the variables are related, i.e. for all $x_i, \mathcal{R}(\eta_{\mathcal{A}}(x_i), \eta_{\mathcal{B}}(x_i))$. Then $\mathcal{R}([\![M]\!]_{\eta_{\mathcal{A}}}, [\![M]\!]_{\eta_{\mathcal{B}}})$.*

In particular, if $\mathcal{A} = \mathcal{B}$ and M is a closed term (i.e. contains no free variables), its meaning $[\![M]\!]$ in a model $\mathcal{A}$ is invariant under all logical relations. This holds also for languages which have constants at base types, by assuming $[\![c]\!]$ is invariant, for all such constants c.

This result has been used by Plotkin, Statman, Sieber , et. al [Plo80, Sie92, St85] to show certain elements (of models) are *not* lambda definable: it suffices to find some logical relation on $\mathcal{A}$ for which the element in question is not invariant.

There is no reason to restrict ourselves to binary logical relations: one may speak of n-ary logical relations, which relate n Henkin models [St85]. Indeed, since Henkin models are closed under products, it suffices to consider unary logical relations, known as *logical predicates.*

Example 2.46 (Hereditary Permutations) Consider a Henkin model $\mathcal{A}$, with a specified permutation $\pi_b : A_b \to A_b$ at each base type b. We extend π to all types as follows: (i) on product types we extend componentwise: $\pi_{\sigma\times\tau} = \pi_\sigma \times \pi_\tau : A_{\sigma\times\tau} \to A_{\sigma\times\tau}$; (ii) on function spaces, extend by conjugation: $\pi_{\sigma\Rightarrow\tau}(f) = \pi_\tau \circ f \circ \pi_\sigma^{-1}$, where $f \in A_{\sigma\Rightarrow\tau}$. We build a logical relation $\mathcal{R}$ on $\mathcal{A}$ by letting $R_\sigma =$ the graph of permutation $\pi_\sigma : A_\sigma \to A_\sigma$, i.e. $R_\sigma(a,b) \Leftrightarrow \pi_\sigma(a) = b$. Members of $\mathcal{R}$ will be called *hereditary permutations.* $\mathcal{R}-$invariant elements $a \in A_\sigma$ are simply fixed points of the permutation: $\pi_\sigma(a) = a$.

Hereditary permutations and invariant elements also arise categorically by interpretation into Set^Z:

Proposition 2.47 *The category Set^Z of (left) Z-sets is equivalent to the category whose objects are sets equipped with a permutation and whose maps (= equivariant maps) are functions commuting with the distinguished permutations. Invariant elements of A are arrows $1 \to A \in Set^Z$.*

Alas, Set^Z is not a Henkin model (1 is not a generator). In the next section we shall slightly generalize the notion of logical relation to work on a larger class of structures.

2.6.1 Logical Relations and Syntax

Logical predicates (also called *computability predicates*) originally arose in proof theory as a technique for proving normalization and other syntactical properties of typed lambda calculi [LS86, GLT]. Later, Plotkin, Statman, and Mitchell [Plo80, St85, Mit96] constructed logical relations on various kinds of structures more general than Henkin models. Following Statman and Mitchell, we extend the notion of logical relation to certain *applicative typed structures* $\mathcal{A}$ for which (i) appropriate meaning functions on the syntax of typed lambda calculus, $[\![M]\!]_{\eta_{\mathcal{A}}}$, are well-defined, and (ii) all logical relations $\mathcal{R}$ are (in a suitable sense) congruence relations with respect to the syntax. This guarantees that the meanings of lambda abstraction and application behave appropriately under these logical relations. Following [Mit96, St85] we call them *admissible* logical relations.

The Soundness Theorem still holds in this more general setting, now using admissible logical relations on applicative typed structures (see [Mit96], Lemma 8.2.10).

Example 2.48 Let $\mathcal{A}$ be the hereditary permutations in Example 2.46. Consider a free simply typed lambda calculus, without constants. Then as a corollary of Soundness we have: *the meaning of any closed term M is invariant under all hereditary permutations* . This conclusion is itself a consequence of the universal property of free cartesian closed categories when interpreted in Set^Z (cf. Corollary 2.22 and Läuchli's Theorem 5.4).

Remark 2.49 The rewriting theory of lambda calculi is a prototype for Operational Semantics of many programming languages (recall the discussion after the β-rule, Corollary 2.11). See also Section 2.8.1 below on PCF. Logical Predicates (so-called *computability predicates*) were first introduced to prove strong normalization for simply typed lambda calculi (with natural numbers types) by W. Tait in the 1960's. Highly sophisticated computability predicates for polymorphically typed systems like $\mathcal{F}$ and $\mathcal{F}_\omega$ were first introduced by Girard in his thesis [Gi71, Gi72]. For a particularly clear presentation, see his book [GLT]. These techniques were later revisited by Statman and Mitchell–using more general logical relations–to also prove Church-Rosser and a host of other syntactic and semantic results for such calculi (see [Mit96]).

For general categorical treatments of logical relations, see [Mit96, MitSce, MaRey] and references there. Uses of logical relations in operational semantics of typed lambda calculi are covered in [AC98, Mit96]. A categorical theory of logical relations applied to data refinement is in [KOPTT]. Use of operationally-based logical relations in programming language semantics is in [Pi96b, Pi97]. For techniques of categorical rewriting applied to lambda calculus, see [JGh95].

2.7 Example 1: Reduction-Free Normalization

The operational semantics of λ-calculi have traditionally been based on rewriting theory or proof theory, e.g. normalization or cut-elimination, Church-Rosser, etc. More recently, Berger and Schwichtenberg [BS91] gave a model-theoretic extraction of normal forms–a kind of "inverting" of the canonical set-theoretic interpretation used in Friedman's Completeness Theorem (cf. 5.2 below).

In this section we sketch the use of categorical methods (essentially from Yoneda's Lemma, cf. Theorem 5.1) to obtain the Berger-Schwichtenberg analysis. A first version of this technique was developed by Altenkirch, Hoffman, and Streicher [AHS95, AHS96]. The analysis given here comes from the article [CDS97], which also mentions intriguing analogues to the Joyal-Gordon-Power-Street techniques for proving coherence in various structured (bi-)categories. The essential idea common to these coherence theorems is to use a version of Yoneda's lemma to embed into a "stricter" presheaf category.

To actually extract a normalization algorithm from these observations requires us to constructively reinterpret the categorical setting in $\mathcal{P}\mathbf{Set}$, as explained below. This leads to a non-trivial example of program extraction from a structured proof, in a manner advocated by Martin-Löf and his school [ML82, Dy95, H97a, CD97] The reader is referred to [CDS97] for the fine details of the proof.

In a certain sense, the results sketched below are "dual" to Lambek's original goal of categorical proof theory [L68, L69], in which he used cut-elimination to study categorical coherence problems. Here, we use a method inspired from categorical coherence proofs to normalize simply typed lambda terms (and thus intuitionistic proofs.)

2.7.1 Categorical Normal Forms

Let $\mathcal{L}$ be a language, $\mathcal{T}$ the set of $\mathcal{L}$-terms and $\sim$ a congruence relation on $\mathcal{T}$. One way to decide whether two terms are $\sim$-congruent is to find an *abstract normal form function*, i.e. a computable function $nf : \mathcal{T} \to \mathcal{T}$ satisfying the following conditions for some (finer) congruence relation $\equiv$:

NF1 $nf(f) \sim f$

NF2 $f \sim g \Rightarrow nf(f) \equiv nf(g)$

NF3 $\equiv \subseteq \sim$

NF4 $\equiv$ is decidable.

From (NF1), (NF2) and (NF3) we see that $f \sim g \Leftrightarrow nf(f) \equiv nf(g)$. This clearly permits a decision procedure: to decide if two terms are $\sim$-related, compute nf of each one, and see if they are $\equiv$ related, using (NF4). The normal form function **nf** essentially "reduces" the decision problem of $\sim$ to that of $\equiv$. This view is inspired from [CD97].

Here we let $\mathcal{L}$ be typed lambda calculus, $\mathcal{T}$ the set of λ-terms, $\sim$ be $\beta\eta$-conversion, and $\equiv$ be α-congruence. Let us see heuristically how category theory can be used to give simply typed λ-calculus a normal form function nf.

Recall 2.22 that λ-terms modulo $\beta\eta$-conversion $\sim_{\beta\eta}$ determine the free ccc $\mathcal{F}_{\mathcal{X}}$ on the set of sorts (atoms) $\mathcal{X}$. [2] By the universal property 2.22, for any ccc $\mathcal{C}$ and any interpretation of the atoms $\mathcal{X}$ in $ob(\mathcal{C})$, there is a unique (up to iso) ccc-functor $[\![-]\!] : \mathcal{F}_{\mathcal{X}} \to \mathcal{C}$ freely extending this interpretation. Let $\mathcal{C}$ be the presheaf category $Set^{\mathcal{F}_{\mathcal{X}}^{op}}$. There are two obvious ccc-functors: (i) the Yoneda embedding $\mathcal{Y} : \mathcal{F}_{\mathcal{X}} \to Set^{\mathcal{F}_{\mathcal{X}}^{op}}$ (cf. 5.1) and (ii) if we interpret the atoms by Yoneda, there

[2] We actually use the free ccc of sequences of λ-terms as defined by Pitts [Pi9?]

is also the free extension to the ccc-functor $[\![-]\!] : \mathcal{F}_{\mathcal{X}} \to Set^{\mathcal{F}_{\mathcal{X}}{}^{op}}$. By the universal property, there is a natural isomorphism $q : \mathcal{Y} \to [\![-]\!]$. By the Yoneda lemma we shall invert the interpretation $[\![-]\!]$ on each hom-set, according to the following commutative diagram

$$
\begin{tikzcd}
\mathcal{F}_{\mathcal{X}}(A,B) \arrow[rr, "{[\![-]\!]}"] \arrow[dr, shift left, "\mathcal{Y}"] & & Set^{\mathcal{F}_{\mathcal{X}}^{op}}([\![A]\!],[\![B]\!]) \arrow[dl, "q_B \circ - \circ q_A^{-1}"] \\
& Set^{\mathcal{F}_{\mathcal{X}}^{op}}(\mathcal{Y}A,\mathcal{Y}B) \arrow[ul, shift left, "-_A(1_A)"] &
\end{tikzcd}
$$

That is, for any $f \in \mathcal{F}_{\mathcal{X}}(A,B)$, we obtain natural transformations $\mathcal{Y}A \xrightarrow{q_A^{-1}} [\![A]\!] \xrightarrow{[\![f]\!]} [\![B]\!] \xrightarrow{q_B} \mathcal{Y}B$. Then evaluating these transformations at A gives the **Set** functions: $\mathcal{F}_{\mathcal{X}}(A,A) \xrightarrow{q_{A,A}^{-1}} [\![A]\!]A \xrightarrow{[\![f]\!]_A} [\![B]\!]A \xrightarrow{q_{B,A}} \mathcal{F}_{\mathcal{X}}(A,B)$. Hence starting with $1_A \in \mathcal{F}_{\mathcal{X}}(A,A)$, we can *define* an **nf** function by:

$$nf(f) =_{def} q_{B,A}\ ([\![f]\!]_A(q_{A,A}^{-1}(1_A))) \tag{10}$$

Clearly $nf(f) \in \mathcal{F}_{\mathcal{X}}(A,B)$. But, alas, by Yoneda's Lemma, $nf(f) = f$! Indeed, this is just a restatement of part of the Yoneda isomorphism. But all is not lost: recall NF1 says $nf(f) \sim f$. This suggests we should reinterpret the entire categorical argument, including the use of functor categories and Yoneda's Lemma, in a setting where "=" becomes "$\sim$", a (partial) equivalence relation. Diagrams which previously commuted now should commute "up to $\sim$".

This viewpoint has a long history in constructive mathematics, where it is common to use sets $(X,\sim)$ equipped with explicit equivalence relations in place of quotients $X/\sim$ (because of problems with the Axiom of Choice). Thus, along with specifying elements of a set, one must also say what it means for two elements to be "equal" (see [Bee85]).

2.7.2 $\mathcal{P}$-category theory and normalization algorithms

Motivated by enriched category theory [K82, Bor94], this view leads to the setting of *$\mathcal{P}$-category theory* in [CDS97]. In $\mathcal{P}$-category theory (i) hom-sets are $\mathcal{P}$**Sets**, i.e. sets equipped with a partial equivalence relation (per) $\sim$, (ii) all operations on arrows are $\mathcal{P}$**Set** maps, i.e. preserve $\sim$, (iii) functors are $\sim$-versions of enriched functors, (iv) $\mathcal{P}$-functor categories and $\mathcal{P}$-natural transformations are $\sim$-versions of appropriate enriched structure, etc. One then proves a $\mathcal{P}$-version of Yoneda's lemma. In essence, $\mathcal{P}$-category theory is the development of ordinary (enriched) category theory in a constructive setting, where equality of arrows is systematically replaced by explicit pers, making sure every operation on arrows is a congruence with respect to the given pers. For an example, see Figure 6.

Now consider the free ccc $(\mathcal{F}_{\mathcal{X}},\sim)$ as a $\mathcal{P}$-category, where the arrows are actually sequences of λ-terms and the per $\sim$ on arrows is $\beta\eta$-equality $=_{\beta\eta}$. Analogously to the above, freeness in the $\mathcal{P}$-setting yields a unique $\mathcal{P}$-ccc functor

$$[\![-]\!] : (\mathcal{F}_{\mathcal{X}},\sim) \to \mathcal{P}Set^{(\mathcal{F}_{\mathcal{X}},\sim)^{op}}$$

where atoms $X \in \mathcal{X}$ are interpreted by $\mathcal{P}$-Yoneda, i.e. as $Hom_{(\mathcal{F}_{\mathcal{X}},\sim)}(-,X)$. Just as in the ordinary case, the $\mathcal{P}$-Yoneda functor $\mathcal{Y}$ is a $\mathcal{P}$-ccc functor, so we have a $\mathcal{P}$-natural isomorphism $q : [\![-]\!] \to \mathcal{Y}$ of $\mathcal{P}$-ccc $\mathcal{P}$-functors:

$\mathcal{P}$-Products

- $c \sim c$ for any constant $c \in \{!_A, \pi_1, \pi_2\}$,
- $f \sim f'$ for any $f, f' : A \to 1$,
- $f_i \sim g_i$ implies $\langle f_1, f_2\rangle \sim \langle g_1, g_2\rangle$, $\pi_i\langle f_1, f_2\rangle \sim f_i$ where $f_i, g_i : C \to A_i$,
- $\langle \pi_1 k, \pi_2 k\rangle \sim k$, for $k : C \to A_1 \times A_2$

$\mathcal{P}$-Exponentials

- $ev \sim ev$ for $B^A \times A \xrightarrow{ev} B$,
- $h \sim h'$ implies $h^* \sim h'^* : C \to B^A$, where $h, h' : C \times A \to B$
- $h \sim h'$ implies $ev\langle h^*\pi_1, \pi_2\rangle \sim h'$,
- $l \sim l'$ implies $(ev\langle l\pi_1, \pi_2\rangle)^* \sim l' : C \to B^A$.

$\mathcal{P}$-Functor

- $f \sim f'$ implies $Ff \sim Ff'$ for all f, f',
- $f \sim f'$, $g \sim g'$ implies $F(gf) \sim Fg'Ff'$ for all composable f, g and f', g',
- $F(id_A) \sim id_{F(A)}$,
- Specified $\mathcal{P}$-isomorphisms $1 \xrightarrow{\cong} F1$, $FA \times FB \xrightarrow{\cong} F(A \times B)$, $(FB)^{FA} \xrightarrow{\cong} F(B^A)$

Figure 6: $\mathcal{P}$-ccc's and $\mathcal{P}$-ccc Functors

$$\begin{tikzcd} (\mathcal{F}_\mathcal{X}, \sim) \arrow[r, shift left=3, "{[\![-]\!]}"] \arrow[r, shift right=3, "\mathcal{Y}"'] \arrow[r, phantom, "q\downarrow\ \uparrow q^{-1}"] & \mathcal{P}\mathbf{Set}^{(\mathcal{F}_\mathcal{X}, \sim)^{op}} \end{tikzcd}$$

In this setting nf as defined by Equation (10) will be a per-preserving function *on terms themselves* and not just on $\beta\eta$-equivalence classes of terms (recall that, classically, a free ccc has for its arrows equivalence classes of terms modulo the appropriate equations). Arguing as before, but now using the $\mathcal{P}$-Yoneda isomorphism, it follows immediately that nf is *an identity $\mathcal{P}$-function.* But this means $nf(f) \sim f$, which is precisely the statement of NF1. Moreover, the part of $\mathcal{P}$-category theory that we use is constructive in the sense that all functions we construct are algorithms. Therefore nf is computable.

It remains to prove NF2: $f \sim g \Rightarrow nf(f) \equiv nf(g)$. This is the most subtle point. Here too the $\mathcal{P}$-version of a general categorical fact will help us (cf 2.4): the $\mathcal{P}$-presheaf category $\mathcal{P}\mathbf{Set}^{\mathcal{C}^{op}}$ is a $\mathcal{P}$-ccc for any $\mathcal{P}$-category $\mathcal{C}$. In particular, let $\mathcal{C}$ be the $\mathcal{P}$-category $(\mathcal{F}_\mathcal{X}, \equiv)$ of sequences of λ-terms up to "change of bound variable" $\equiv$. This is a trivially decidable equivalence relation on terms (called *α-congruence* in the literature) and obviously $\equiv \subseteq =_{\beta\eta}$. Note that this $\mathcal{P}$-category has the same objects and arrows as $(\mathcal{F}_\mathcal{X}, \sim)$, but the pers on arrows are different.

By the freeness of $(\mathcal{F}_\mathcal{X}, \sim)$, we have another interpretation $\mathcal{P}$-functor

$$[\![-]\!]^{\equiv} : (\mathcal{F}_\mathcal{X}, \sim) \to \mathcal{P}\mathbf{Set}^{(\mathcal{F}_\mathcal{X}, \equiv)^{op}}$$

where we interpret atoms $X \in \mathcal{X}$ by the $\mathcal{P}$-presheaf $Hom_{(\mathcal{F}_\mathcal{X}, \equiv)}(-, X)$. The key fact is that this $\mathcal{P}$-functor $[\![-]\!]^{\equiv}$ has exactly the same set-theoretic effect on objects and arrows as $[\![-]\!]$. That is, one proves by induction:

Lemma 2.50 *For all objects C and arrows f in $(\mathcal{F}_\mathcal{X}, \sim)$, $|[\![C]\!]| = |[\![C]\!]^{\equiv}|$ and similarly $|[\![f]\!]| = |[\![f]\!]^{\equiv}|$, where $|-|$ means taking the underlying set-theoretic structure.*

Hence, we can conclude that $f \sim g$ implies $[\![f]\!] \equiv [\![g]\!]$ (here $\equiv$ refers to the per on arrows in $\mathcal{P}Set^{(\mathcal{F}_\mathcal{X},\equiv)^{op}}$). We can show that q_B and q_B^{-1} are $\equiv$-natural, in particular $q_{B,A}$ and $q_{B,A}^{-1}$ preserve $\equiv$. It then follows that $nf(f) \equiv nf(g)$, as desired.

Remark 2.51

(i) The normal forms obtained by this method can be shown to coincide with the so-called *long $\beta\eta$ normal forms* used in lambda calculus [CDS97].

(ii) The direct inductive proofs used above correspond more naturally to a more-involved bicategorical definition of freeness ([CDS97], Remark 3.17).

Finally, in [CDS97] it is shown how to apply the method to the word problem for typed λ-calculi with additional axioms and operations, i.e. to freely-generated ccc's modulo certain theories. This employs appropriate free $\mathcal{P}$-ccc's (over a $\mathcal{P}$-category, a $\mathcal{P}$-cartesian category, etc.) These are generated by various notions of λ-theory, which are determined not only by a set of atomic types, but also by a set of basic typed constants as well as a set of equations between terms. Although the Yoneda methods always yield an algorithm nf it does not necessarily satisfy NF4 (the decidability of $\equiv$). What is obtained in these cases is a reduction of the word problems for such free ccc's to those of the underlying generating categories.

2.8 Example 2: PCF

The language PCF, due to Dana Scott in 1969, has deeply influenced recent programming language theory. Much of this influence arises from seminal work of Gordon Plotkin in the 1970's on operational and denotational semantics for PCF. We shall briefly outline the syntax and basic semantical issues of the language following from Plotkin's work. We follow the treatment in [AC98, Sie92], although the original paper ([Plo77]) is highly recommended.

2.8.1 PCF

The language PCF is an explicitly typed lambda calculus with the following structure:

Types: Generated from $nat, boole$ by $\Rightarrow$.

Lambda Terms: generated from typed variables using the following specified constants:

$n : nat$, for each $n \in N$	$zero? : nat \Rightarrow boole$
$T : boole$	$cond_{nat} : boole \Rightarrow (nat \Rightarrow (nat \Rightarrow nat))$
$F : boole$	$cond_{boole} : boole \Rightarrow (boole \Rightarrow (boole \Rightarrow boole))$
$succ : nat \Rightarrow nat$	$\perp_\sigma : \sigma$
$pred : nat \Rightarrow nat$	$\mathsf{Y}_\sigma : (\sigma \Rightarrow \sigma) \Rightarrow \sigma.$

Categorical Models

The *standard model* of PCF is defined in the ccc ω-CPO$_\perp$ as follows: interpret the base types as in Figure 4: $[\![nat]\!] = N_\perp$, $[\![boole]\!] = B_\perp$, $[\![\sigma \Rightarrow \tau]\!] = [\![\sigma]\!] \Rightarrow [\![\tau]\!]$ (= function space in ω-CPO). Interpret constants as follows (for clarity, we omit writing $[\![-]\!]$): $succ, pred : Nat_\perp \to Nat_\perp$, $zero? : Nat_\perp \to B_\perp$, $cond_\sigma : boole \times \sigma^2 \to \sigma$, $\sigma \in \{nat, boole\}$ (*cond* is for *conditional*, sometimes called *if then else*), $[\![T]\!] = t$, $[\![F]\!] = f$, $[\![n]\!] = n$,

$$
\begin{array}{rcl@{\qquad}rcl}
(\lambda x.\varphi)\text{`}a & \to & \varphi[a/x] & zero?\text{`}(0) & \to & t \\
Y\text{`}f & \to & f\text{`}(Y\text{`}f) & zero?\text{`}(n+1) & \to & f \\
succ\text{`}n & \to & n+1 & cond\,tab & \to & a \\
pred\text{`}(n+1) & \to & n & cond\,fab & \to & b
\end{array}
$$

$$
\frac{f \to g}{f\text{`}a \to g\text{`}a} \qquad \frac{f \to g}{u\text{`}f \to u\text{`}g} \quad \text{where } u \in \{succ, pred, zero?\}
$$

$$
\frac{f \to g}{cond\,fab \to cond\,gab} \qquad \text{where } cond\,xyz = ((cond\text{`}x)\text{`}y)\text{`}z
$$

Figure 7: Operational Semantics for PCF

$$
succ(x) = \begin{cases} x+1 & \text{if } x \neq \perp \\ \perp & \text{if } x = \perp \end{cases}
\qquad
cond_\sigma(p, y, z) = \begin{cases} y & \text{if } p = t \\ z & \text{if } p = f \\ \perp & \text{if } p = \perp \end{cases}
$$

$$
pred(x) = \begin{cases} x-1 & \text{if } x \neq \perp, 0 \\ \perp & \text{else} \end{cases}
$$

$$
zero?(x) = \begin{cases} t & \text{if } x = 0 \\ f & \text{if } x \neq 0, \perp \\ \perp & \text{if } x = \perp \end{cases}
$$

$\perp_\sigma$ is the least element of $[\![\sigma]\!]$ (denoting "non-termination" or "divergent").

Y_σ is the least fixed point operator
$\mathsf{Y}_\sigma(f) = \bigvee\{f^n(\perp_\sigma) \mid n \geq 0\}$ (see Example 3.5).

More generally, a *standard model of PCF* is an ω-CPO-enriched ccc $\mathcal{C}$, in which each homset $\mathcal{C}(A, B)$ has a smallest element $\perp_{AB}\colon A \to B$ with the following properties: (i) pairing and currying are monotonic, (ii) $\perp \circ f = f$ and $ev\circ\langle \perp, f\rangle = \perp$ for all f of appropriate type, (iii) there are objects *nat* and *boole* whose sets of global elements satisfy $\mathcal{C}(1, nat) \cong N_\perp$ and $\mathcal{C}(1, boole) \cong B_\perp$ and in which the constants are all interpreted in the internal language of $\mathcal{C}$ as in the standard model above (e.g. interpreting $succ(x)$ by $ev\circ\langle succ, x\rangle$, etc.). A model is *order-extensional* if 1 is a generator and the order on hom-sets coincides with the pointwise ordering.

The operational semantics of PCF is given by a set of rewriting rules, displayed in Figure 7. This is intended to describe the dynamic evaluation of a term, as a sequence of 1-step transitions. The fixed-point combinator Y guarantees some computations may not terminate. It is important to emphasize that in operational semantics with partially-defined (i.e. possibly non-terminating) computations, different orders of evaluation (e.g. left-most outermost vs innermost) may lead to non-termination in some cases and may also effect efficiency, etc. (See [Mit96], Chapter 2). We have chosen a simple operational semantics for PCF, given by a deterministic evaluation relation, following [AC98].

2.8.2 Adequacy

A *PCF program* is a closed term of base type (i.e. either *nat* or *boole*). The *observable behaviour* of a PCF program $P : nat$ is the set $Beh(P) = \{n \in N \mid P \stackrel{*}{\longrightarrow} n\}$, and similarly for $P : boole$. The set $Beh(P)$ is either empty if P diverges or a singleton if P converges to a (necessarily unique) normal form. The following theorem is proved using a logical relations argument.

Theorem 2.52 (Computational Adequacy) *Let $\mathcal{C}$ be any standard model of PCF. Then for all programs $P : nat$ and $n \in N$,*

$$P \xrightarrow{*} n \quad \textit{iff} \quad [\![P]\!] = n$$

and similarly for $P : boole$. Hence $[\![P]\!] = [\![Q]\!]$ iff their sets of behaviours are equal.

We are interested in a notion of "observational equivalence" arising from the operational semantics. A program (a closed term of base type) can be observed to converge to a specific numeral or boolean value. More generally, what can we observe about arbitrary terms of any type? The idea is to plug them into arbitrary program code, and observe the behaviour. More precisely, a *program context* $C[-]$ is a program with a hole in it (the hole has a specified type) which is such that if we formally plug a PCF term M into (the hole of) $C[-]$ (we don't care about possible clashes of bound variables) we obtain a program $C[M]$. We are looking at the convergence behaviour (with respect to the operational semantics) of the resulting program. (cf. [AC98]).

Definition 2.53 Two PCF terms M, N of the same type are *observationally equivalent* (denoted $M \approx N$) iff $Beh(C[M]) = Beh(C[N])$ for every program context $C[-]$.

That is, $M \approx N$ means that for all program contexts $C[-]$, $C[M] \xrightarrow{*} c$ iff $C[N] \xrightarrow{*} c$ (for c either a boolean value or a numeral). Thus, by the previous Theorem, $M \approx N$ iff $[\![C[M]]\!] = [\![C[N]]\!]$. In order to prove observational equivalence of two PCF terms, R. Milner showed it suffices to pick *applicative contexts*, i.e.

Lemma 2.54 (Milner) *Two closed PCF expressions $M, N : \sigma_1 \Rightarrow (\sigma_2 \Rightarrow \cdots (\cdots (\sigma_n \Rightarrow nat) \cdots)$ are observationally equivalent iff $[\![M P_1 \dots P_n]\!] = [\![N P_1 \dots P_n]\!]$ for all closed $P_i : \sigma_i$, $1 \leq i \leq n$.*

Finally, the main definition of the subject is:

Definition 2.55 (Full Abstraction) A model is called *fully abstract* if observational equivalence coincides with denotational equality in the model, i.e. for any two PCF terms M, N

$$M \approx N \quad \text{iff} \quad [\![M]\!] = [\![N]\!]$$

Which models are fully abstract? There are two main theorems, due to Milner and Plotkin. First we introduce the "parallel-or" function $por : B_\perp \times B_\perp \to B_\perp$ on the standard model of PCF:

$$por(a, b) = \begin{cases} t & \text{if } a = t \text{ or } b = t \\ f & \text{if } a = b = f \\ \perp & \text{else} \end{cases}$$

Theorem 2.56 (Plotkin)

- *por is not definable in PCF.*
- *The standard model is not fully abstract.*
- *The standard model is fully abstract for the language PCF + por.*

The proof of the first two parts of the theorem use logical relations ([Sie92, AC98, Gun92]) . In 1977, R. Milner proved the following [Mil77]:

Theorem 2.57 (Milner) *There is a unique (up to isomorphism) fully abstract order-extensional model of PCF.*

Milner's construction is syntactical, so the question became: find a more " mathematical" (i.e. not explicitly syntactical) characterization of the unique fully abstract model. This is related to the Full Completeness Problems discussed in Section 5.2. A satisfying solution to the Full Abstraction Problem for PCF was recently given by S. Abramsky, R. Jagadeesan, and P. Malacaria and also M. Hyland and L. Ong who use various monoidal categories of games. This has recently led to highly active subject of *games semantics* for programming languages (see Section 4.3.2 and the articles mentioned there).

3 Parametricity

What is parametricity in polymorphism? We have already seen such notions as

- Uniformity of algorithms across types.
- Passing types as parameters in programs.

But the problem is that a type like $\forall\alpha(\alpha \Rightarrow \alpha)$, when interpreted in a model as a large product over all types, may contain in Strachey's words unintended *ad hoc* elements. In addition to removing some entities, we may wish to include yet others. For example, should we consider closure of parametric functions under isomorphism of types?

We have already mentioned the idea of types being functors, in Section 2.4.2. Indeed, this suggests an obvious kind of modelling

- Types = functors
- Terms (programs) = natural transformations

all defined over some ccc $\mathcal{C}$. This view of categorical program semantics has had a fruitful history. Reynolds, Oles, and later O'Hearn and Tennent have used functor categories to develop semantics of local variables, block structure, non-interference, etc. in Algol-like languages (see [OHT92, Ten94] and references there).

In the case of polymorphism this is also not such a far-fetched idea. Imagine a term $t : \forall\alpha.\alpha \Rightarrow \alpha$. We know that for each type A, $t[A] : A \Rightarrow A$. Thus, from our Curry-Howard viewpoint, we think of this as an object-indexed family of arrows. Combining this idea with the mild parametricity condition of *naturality* then seems reasonable. In the mid 1980's, Girard gave functor category models of System $\mathcal{F}$ [Gi86]. However to handle the functorial problem of co/contravariance in an expression like $\alpha \Rightarrow \beta$ (or worse, in $\alpha \Rightarrow \alpha$, which is not a functor at all) he introduced categories of embedding-projection pairs (as in domain theory, Section 2.5). Below we shall consider dinaturality, a multivariant notion of naturality which takes into account such problems.

Reynolds[Rey83] also proposed an analysis of parametricity using the notion of *logical relations*, a fundamental tool in the theory of typed lambda calculi. The paper [BFSS90] studied the above two frameworks for parametricity: Reynolds' relational approach and the dinaturality approach. This work was extended and formalized in [ACC93, BAC95, PlAb93].

3.1 Dinaturality

One attempt to understand parametric polymorphism is to require certain *naturality* conditions on families interpreting universal types. In this view we begin with some appropriate ccc $\mathcal{C}$ of values and interpret polymorphic type expressions $A(\alpha_1, \ldots, \alpha_n)$, with type variables α_i, as certain kinds of multivariant "definable" functors $F : (\mathcal{C}^{op})^n \times \mathcal{C}^n \longrightarrow \mathcal{C}$. Terms or programs t are then interpreted as certain multivariant (= *dinatural*) transformations between (the interpretations of) the types. We need to account for naturality not only in positive (covariant) positions, but also in negative (contravariant) ones. As we shall see, the difficulty will be compositionality.

Definition 3.1 Let $\mathcal{C}$ be a category, and $F, G : (\mathcal{C}^{op})^n \times \mathcal{C}^n \to \mathcal{C}$ functors. A *dinatural transformation* $\theta : F \to G$ is a family of $\mathcal{C}$-morphisms $\theta = \{\theta_A : FAA \to GAA \mid A \in \mathcal{C}^n\}$ satisfying (for any n-tuple $f : A \to B \in \mathcal{C}^n$):

$$\begin{tikzcd}
& FAA \arrow[r, "\theta_A"] & GAA \arrow[dr, "GAf"] & \\
FBA \arrow[ur, "FfA"] \arrow[dr, "FBf"'] & & & GAB \\
& FBB \arrow[r, "\theta_B"] & GBB \arrow[ur, "GfB"'] &
\end{tikzcd}$$

For a history of this notion, see [Mac71]. Dinatural transformations include ordinary natural transformations as a special case (e.g. construe covariant F, G as bifunctors, dummy in the contravariant variable), as well as transformations between co- and contravariant functors. The parametric aspect of naturality here is that θ_A may be varied along an arbitrary map $f : A \to B$ in both the co- and contravariant positions.

In the following examples, K_A denotes the constant functor with value A (where $K_A(f) = id_A$). We use set-theoretic notation, but the examples make sense in any ccc (e.g. using the internal language). We follow the treatment in [BFSS90].

Example 3.2 (Polymorphic Identity) Let $F = K_1$, let $G(-,-) = (-)^{(-)}$. Consider the family $I = \{I_A : 1 \to A^A \mid A \in \mathcal{C}\}$ where $I_A(*) = \lambda_{x:A}.x = (1 \times A \xrightarrow{\pi_2} A)^*$. Definition 3.1 reduces to the following commuting square:

$$\begin{tikzcd}
& A^A \arrow[dr, "f^A"] & \\
1 \arrow[ur, "I_A"] \arrow[dr, "I_B"'] & & B^A \\
& B^B \arrow[ur, "B^f"'] &
\end{tikzcd}$$

which essentially says $f \circ id_A = id_B \circ f$. This equation is true (in **Set** or in the internal language of any ccc) since both sides equal f.

Example 3.3 (Evaluation) Fix an object $D \in \mathcal{C}$. Let $F(-) = D^{(-)}$ and $G = K_D$. The family $Ev = \{ev_A : (D^A) \times A \to D \mid A \in \mathcal{C}\} : F \to G$ is a dinatural transformation, where ev_A is the usual evaluation in any ccc. Definition 3.1 reduces to the following commuting square, for any $f : A \to B$

$$\begin{tikzcd}
& D^A \times A \arrow[dr, "ev_A"] & \\
D^B \times A \arrow[ur, "D^f \times A"] \arrow[dr, "D^B \times f"'] & & D \\
& D^B \times B \arrow[ur, "ev_B"'] &
\end{tikzcd}$$

This says, for any $g : D^B, a : A$, $ev_A(g \circ f, a) = ev_B(g, f(a))$. More informally, $(g \circ f)(a) = g(f(a))$. Again, this is a truism in any ccc.

Extending the above example, *generalized evaluation* $EV = \{ev_{A,A'} : A'^A \times A \to A' \mid A, A' \in \mathcal{C}\}$ determines a dinatural transformation between appropriate functors (cf [BFSS90]). Dinaturality corresponds to the true equation $f'((g \circ f)(a)) = (f' \circ g)(f(a))$ for $g : A'^B$, $a : A$, and any $f : A \to B, f' : A' \to B'$.

Example 3.4 (Church Numerals) Define $n : (-)^{(-)} \to (-)^{(-)}$ to be the family where $n_A : A^A \to A^A$ is given by mapping $h \mapsto h^n$, with $h^n = h \circ h \circ h \cdots \circ h$ (n times). Dinaturality corresponds to the diagram (for any $f : A \to B$)

$$\begin{tikzcd}
& A^A \arrow[r, "n_A"] & A^A \arrow[dr, "f^A"] & \\
A^B \arrow[ur, "A^f"] \arrow[dr, "f^B"'] & & & B^A \\
& B^B \arrow[r, "n_B"] & B^B \arrow[ur, "B^f"'] &
\end{tikzcd}$$

i.e. if $g : A^B$, $(f \circ g)^n \circ f = f \circ (g \circ f)^n$, an instance of associativity.

We shall see dinaturality again in Section 6.1 . Observe that each of the families $\theta = \{\theta_A \mid A \in \mathcal{C}\}$ above—which in essence arise from the syntax of ccc's—have *uniform algorithms* θ_A across all types A. For example, $n_A = \lambda_{h:A^A}.h^n$, uniform in each type A.

We end with an operator which is fundamental to denotational semantics.

Example 3.5 (Fixed Point Combinators) In many ccc's $\mathcal{C}$ used in programming language semantics, e.g. certain subcategories of ω-CPO, there is a dinatural fixed point combinator $\mathsf{Y}_{(-)} : (-)^{(-)} \to (-)$. That is, we have a family $\{\mathsf{Y}_A : A^A \to A \mid A \in \mathcal{C}\}$ making the following diagram commute, for any $f : A \to B$:

$$\begin{tikzcd}
& A^A \arrow[r, "\mathsf{Y}_A"] & A \arrow[dr, "f"] & \\
A^B \arrow[ur, "A^f"] \arrow[dr, "f^B"'] & & & B \\
& B^B \arrow[r, "\mathsf{Y}_B"] & B \arrow[ur, "id_B"'] &
\end{tikzcd}$$

This says, using informal set-theoretic notation, if $g : A^B, f(\mathsf{Y}_A(g \circ f)) = \mathsf{Y}_B(f \circ g)$. In particular, setting $B = A$ and letting $g = id_B$, we have the fixed-point equation $f(\mathsf{Y}_A(f)) = \mathsf{Y}_A(f)$.

For example, consider ω-CPO$_\perp$–the subccc of ω-CPO whose objects A have a least element $\perp_A$ but the morphisms need not preserve it. It may be shown that the family given by $\mathsf{Y}_A(f) =$ *the least fixed-point of* $f = \bigvee\{f^n(\perp_A) \mid n \geq 0\}$ is dinatural (see [BFSS90, Mul91, Si93]).

There is a calculus of multivariant functors $F, G : (\mathcal{C}^{op})^n \times \mathcal{C}^n \to \mathcal{C}$ functors. For example basic type constructors may be defined (using products and exponentials in $\mathcal{C}$) by setting

$$(F \times G)AB = FAB \times GAB \tag{11}$$

$$(G^F)AB = GAB^{FBA} \tag{12}$$

Here A is the list of n contravariant and B the list of n covariant arguments. Note the twist of the arguments in the definition of exponentiation. Much of the structure of cartesian closedness (e.g. evaluation maps, currying, projections, pairing, etc.) exists within the world of dinatural transformations and there is a kind of abstract functorial calculus (cf. [BFSS90], Appendix A6, [Fre93]).

Unfortunately, there is a serious problem: in general, dinaturals do not compose. That is, given dinatural families $\{FAA \xrightarrow{\theta_A} GAA \mid A \in \mathcal{C}\}$ and $\{GAA \xrightarrow{\psi_A} HAA \mid A \in \mathcal{C}\}$, the composite $\{FAA \xrightarrow{\psi_A \circ \theta_A} HAA \mid A \in \mathcal{C}\}$ does not always make the appropriate hexagon commute. However, with respect to the original question of closure of parametric functions under isomorphisms of types, we note that families dinatural with respect to isomorphisms f do in fact compose. But this class is too weak for a general modelling . Detailed studies of such phenomena have been done in [BFSS90, FRRa, FRRb].

Remarkably, there are certain categories $\mathcal{C}$ over which there are large classes of multivariant functors and dinatural transformation which provide a compositional semantics:

- In [BFSS90] it is shown that if $\mathcal{C} = Per(N)$, that so-called *realizable* dinatural transformations between *realizable* functors compose. Realizable functors include almost any functors that arise in practice, (e.g. those definable from the syntax of System $\mathcal{F}$) while realizable dinatural transformations are families of per morphisms whose action is given uniformly by a single Turing machine. This semantics also has a kind of universal quantifier modelling System $\mathcal{F}$(see below).

- In [GSS91] it is shown that the syntax of simply typed lambda calculus with type variables—i.e. $\mathcal{C}$ = a free ccc—admits a compositional dinatural semantics (between logically definable functors and dinatural families). This uses the cut-elimination theorem from proof theory. This work was extended to Linear Logic by R. Blute [Blu93].

- In [BS96] there is a compositional dinatural semantics for the multiplicative fragment of linear logic (generated by atoms). Here $\mathcal{C} = \mathcal{RTVEC}$, a category of reflexive topological vector spaces first studied by Lefschetz [Lef63], with functors being syntactically definable. In [BS96b, BS98] this was extended to a compositional dinatural semantics for Yetter's noncommutative *cyclic* linear logic. In both cases, one demands certain uniformity conditions on dinatural families, involving equivariance w.r.t. continuous group (respectively Hopf-algebra) actions induced from actions on the atoms (see also Section 5.2 below). For the non-cyclic fragment, this is automatic.

Associated with dinaturality is a kind of "parametric" universal quantifier first described by Yoneda and which plays a fundamental role in modern category theory [Mac71].

Definition 3.6 An *end* of a multivariant functor G on a category $\mathcal{C}$ is an object $E = \int_A GAA$ and a dinatural transformation $K_E \to G$, universal among all such dinatural transformations.

In more elementary terms, there is a family of arrows $\{\int_A GAA \xrightarrow{\theta_X} GXX \mid X \in \mathcal{C}\}$ making the main square in the following diagram commute, for any $X \xrightarrow{f} Y$, and such that given any other family $u = \{u_X \mid X \in \mathcal{C}\}$ such that $GXf \circ u_X = GfY \circ u_Y$, there is a unique Λu making the appropriate triangles commute:

$$
\begin{tikzcd}
D \arrow[dr, "u_X"] \arrow[ddr, "u_Y"'] \arrow[dd, dotted, "\Lambda u"] & & \\
 & GXX \arrow[dr, "GXf"] & \\
\int_A GAA \arrow[ur, "\theta_X"] \arrow[dr, "\theta_Y"'] & & GXY \\
 & GYY \arrow[ur, "GfY"'] &
\end{tikzcd}
$$

One may think of $\int_A GAA$ as a subset of $\Pi_A GAA$ (note, this is a "large" product over all $A \in \mathcal{C}$, so $\mathcal{C}$ must have appropriate limits for this to exist)

$$\int_A GAA = \{g \in \Pi_A GAA \mid GXf \circ \theta_X = GfY \circ \theta_Y, \text{ for all } X, Y, f : X \to Y \in \mathcal{C}\}$$

In [BFSS90] versions of such ends over $Per(N)$ are discussed with respect to parametric modelling of System $\mathcal{F}$.

In a somewhat different direction, *co-ends* (dual to ends) are a kind of sum or existential quantifier. Their use in categorical computer science was strongly emphasized in early work of Bainbridge [B72, B76] on duality theory for machines in categories. A useful observation is that we may consider functors $R : \mathcal{C} \times \mathcal{D} \to Set$ and $S : \mathcal{D} \times \mathcal{E} \to Set$ as generalized relations, with relational composition being determined by the coend formula $R; S(C, E) = \int^D (R(C, D) \times S(D, E))$. This view has recently been applied to relational semantics of dataflow languages in [HPW].

We should mention that dinaturality is also intimately connected with categorical coherence theorems and geometrical properties of proofs [Blu93, So87]. It also seems to be hidden in deeper aspects of Cut-Elimination [GSS91], although here there is still much to understand. We shall meet dinaturality again in several places (e.g. in Traced Monoidal Categories, Section 6.1).

3.2 Reynolds Parametricity

Reynolds [Rey83] analyzed Strachey's notion of parametric polymorphism using a relational model of types. Although his original idea of using a Set-based model was later shown by Reynolds himself to be untenable, the framework has greatly influenced subsequent studies. As a concrete illustration, following [BFSS90] we shall sketch a relational model over $Per(N)$. Related results in more general frameworks were obtained by Hasegawa [RHas94, RHas95] and Ma and Reynolds [MaRey]. Although originally Reynolds' work was semantical, general logics for reasoning about formal parametricity, supported by such Per models, were developed in [BAC95, PlAb93, ACC93].

Given pers $A, A' \in Per(N)$, a *saturated relation* $R : A \nrightarrow A'$ is a relation $R \subset dom_A \times dom_{A'}$ satisfying $R = A; R; A'$, where ; denotes relational composition. For all pers A, A', B, B', saturated relations $R : A \nrightarrow A'$ and $S : B \nrightarrow B'$ and elements $a \in dom_A, a' \in dom_{A'}, b \in dom_B, b' \in dom_{B'}$ we define *a relational* System $\mathcal{F}$ type structure as follows:

- $R \times S : (A \times B) \nrightarrow (A' \times B')$, given componentwise by $(a, b) R \times S (a', b')$ iff aRa' and bSb'.

- $R \Rightarrow S : (A \Rightarrow B) \nrightarrow (A' \Rightarrow B')$, where $f(R \Rightarrow S)g$ iff f, g are (codes of) Turing computable functions satisfying aRa' implies $faSga'$ for any a, a' as above.

- $\forall\alpha.\tau(\alpha, S) : \forall\alpha.\tau(\alpha, B) \nrightarrow \forall\alpha.\tau(\alpha, B')$ is defined by a simultaneous inductive definition based on the formation of type expression $\tau(\alpha, \beta)$. We shall omit the technical construction (see [BFSS90], p.49) but the key idea is to redefine the Per-interpretation of $\forall\alpha.\tau(\alpha)$ by triming down the intersection $\cap_A \tau(A)$ to only those elements invariant under all saturated relations, while $\forall\alpha.\tau(\alpha, S) = \cap_R \tau(R, S)$, the intersection being over all pers A, A' and saturated $R : A \nrightarrow A'$.

The somewhat involved construction of $\forall\alpha.\tau(\alpha)$ ensures the type expressions $\tau(-)$ act like functors with respect to saturated relations. More precisely, Reynolds' parametricity entails:

- If $R : A \nrightarrow A'$ is a saturated relation, then for any polymorphic type τ, $\tau(R) : \tau(A) \nrightarrow \tau(A')$ is a saturated relation.

- *Identity Extension Lemma:* τ preserves identity relations, i.e. $\tau(id_A) = id_{\tau(A)}$, as saturated relations $\tau(A) \nrightarrow \tau(A)$ (and similarly for $\tau(id_{A_1}, \ldots, id_{A_n})$)

One obtains a Soundness Theorem, essentially the interpretation of the free term model of System $\mathcal{F}$ into the relational Per model above. For simplicity, consider terms with only one variable. Let $\sigma = \sigma(\alpha_1, \ldots, \alpha_n)$ and $\tau = \tau(\alpha_1, \ldots, \alpha_n)$ denote polymorphic types with free type variables $\subseteq \{\alpha_1, \ldots, \alpha_n\}$. Let $x : \sigma \vdash t : \tau$ denote term $t : \sigma$ with free variable $x : \sigma$. Associated to every System $\mathcal{F}$ term t is a Turing computable numerical function f_t, obtained by essentially erasing all types and considering the result as an untyped lambda term, qua computable partial function (see [BFSS90], Appendix A.1).

Theorem 3.7 (Soundness) *Let A_i, A'_i be pers and $R_i : A_i \nrightarrow A'_i$ saturated relations. Then if $m\sigma(\vec{R})m'$ then $f_t(m)\tau(\vec{R})f_t(m')$. Also, if $t = t'$ in System $\mathcal{F}$, then $f_t = f_{t'}$ as $Per(N)$ maps $\sigma(\vec{A}) \to \tau(\vec{A})$.*

Thus terms (programs) become "relation transformers" $\sigma(\vec{R}) \to \tau(\vec{R})$ (cf. [MitSce, Fre93]) of the form

$$\begin{tikzcd}
& \sigma(\vec{A}) \arrow[rr, "f_t"] & & \tau(\vec{A}) \\
\sigma(\vec{R}) \arrow[ur] \arrow[dr] \arrow[rr, dotted, "\exists"] & & \tau(\vec{R}) \arrow[ur] \arrow[dr] & \\
& \sigma(\vec{A'}) \arrow[rr, "f_t"'] & & \tau(\vec{A'})
\end{tikzcd}$$

In particular this exemplifies Reynolds' interpretation of Strachey's parametricity: if one instantiates an element of polymorphic type at two related types, then the two values obtained must be related themselves.

Reynolds parametricity has the following interesting consequence [BFSS90, RHas94]. Recall the category of T-algebras, for definable functors T (cf. the proof of Theorem 2.40).

Theorem 3.8 *Let T be a System $\mathcal{F}$-definable covariant functor. Then in the parametric Per model, $\forall\alpha.((T\alpha \Rightarrow \alpha) \Rightarrow \alpha)$ is the initial T-algebra.*

This property becomes a general theorem in the formal logics of parametricity (e.g. [PlAb93, RHas95]), and hence would be true in any appropriate parametric model. Thus, although the syntax of second-order logic in general only guarantees *weakly initial* data types as in [GLT], in parametric models of System $\mathcal{F}$ the usual definitions actually yield strong data types.

The reader might rightly enquire: do relational parametricity and dinaturality have anything in common? This is exactly the kind of question that requires a logic for reasoning *about* parametricity. Plotkin and Abadi's logic [PlAb93] extends the equational theory of System $\mathcal{F}$ with quantification over functions and relations, together with a schema expressing Reynolds' relational parametricity. The dinaturality hexagon in Definition 3.1, for definable functors and families, is expressible as a quantified equation in this logic.

Proposition 3.9 *In the formal system above, relational parametricity implies dinaturality.*

Reynolds' work on parametricity continues to inspire fundamental research directions in programming language theory, even beyond polymorphism. For example, O'Hearn and Tennent [OHT92, OHT93] use relational parametricity to examine difficult problems in local-variable declarations in Algol-like languages. Their framework is particularly interesting. They use ccc's of functor categories and natural transformations, à la Oles and Reynolds, but *internal to the category of reflexive directed multigraphs*. The same framework, somewhat generalized, was then used by A. Pitts [Pi96a] in a general relational approach to reasoning about properties of recursively defined domains. Pitts work has led to new approaches to induction and co-induction, etc. (see Section 2.5). The reader is referred to Pitts [Pi96a], p.74 and O'Hearn and Tennent [OHT93] for many examples of these so-called *relational structures over categories* $\mathcal{C}$.

4 Linear Logic

4.1 Monoidal Categories

We briefly recall the relevant definitions. For details, the reader is referred to [Mac71, Bor94].

Definition 4.1 A *monoidal category* is a category $\mathcal{C}$ equipped with a functor $\otimes : \mathcal{C} \times \mathcal{C} \to \mathcal{C}$, an object I, and specified natural isomorphisms:

$$a_{ABC} : (A \otimes B) \otimes C \xrightarrow{\simeq} A \otimes (B \otimes C)$$

$$\ell_A : I \otimes A \xrightarrow{\simeq} A \quad \text{and} \quad r_A : A \otimes I \xrightarrow{\simeq} A$$

satisfying coherence equations: associativity coherence (Mac Lane's Pentagon) and the unit coherence.

A *symmetric monoidal category* is a monoidal category with a natural symmetry isomorphism $s_{AB} : A \otimes B \xrightarrow{\simeq} B \otimes A$ satisfying: $s_{BA}s_{AB} = id_{A \otimes B}$, for all A, B, and (omitting subscripts) $r = \ell s$, $asa = (1 \otimes s)a(s \otimes 1)$.

Symmetric monoidal categories include cartesian categories (with $\otimes = \times$) and cocartesian categories (with $\otimes = +$). However in the two latter cases, the structure is uniquely determined (up to isomorphism)—and similarly for the coherence isomorphisms—by the universal property of products (resp. coproducts). This is not true in the general case–there may be many symmetric monoidal structures on the same category.

We now introduce the monoidal analog of ccc's:

Definition 4.2 A *symmetric monoidal closed category* (= smcc) $(C, \otimes, I, \multimap)$ is a symmetric monoidal category such that for each object $A \in \mathcal{C}$, the functor $- \otimes A : \mathcal{C} \to \mathcal{C}$ has a specified right adjoint $A \multimap -$, i.e. for each A there is an isomorphism, natural in B, C:

$$Hom_{\mathcal{C}}(C \otimes A, B) \cong Hom_{\mathcal{C}}(C, A \multimap B) \quad . \tag{13}$$

As a consequence, in any smcc there are "evaluation" and "coevaluation" maps $(A \multimap B) \otimes A \xrightarrow{ev_{AB}} B$ and $C \longrightarrow (A \multimap (C \otimes A))$ determined by the adjointness (13). We shall try to keep close to our ccc notation, Section 2.1. In particular the analog of Currying arising from (13) is denoted $C \xrightarrow{f^*} (A \multimap B)$ Moreover, this data actually determines a (bi)functor $- \multimap - : \mathcal{C}^{op} \times \mathcal{C} \to \mathcal{C}$. No special coherences have to be supposed for $- \multimap -$: they follow from coherence for $\otimes$ and adjointness..

For the purposes of studying linear logic below, we need (among other things) a notion of an smcc, equipped with an involutive negation or duality, reminiscent of finite dimensional vector spaces. The general theory of such categories, due to M. Barr [Barr79], was developed in the mid 1970's, some ten years before linear logic.

Consider an smcc $\mathcal{C}$, with a distinguished object $\perp$. Consider the map $ev_{A\perp} \circ s_{A\perp} : A \otimes (A \multimap \perp) \longrightarrow \perp$. By (13) this corresponds to a map $\mu_A : A \to (A \multimap \perp) \multimap \perp$. Let us write $A^\perp$ for $A \multimap \perp$. Thus we have a morphism $\mu_A : A \to A^{\perp\perp}$. Objects A for which μ_A is an isomorphism are called *reflexive*, or more precisely reflexive with respect to $\perp$.

Definition 4.3 A $*$-*autonomous category* $(\mathcal{C}, \otimes, I, \multimap, \perp)$ is an smcc $\mathcal{C}$ with a distinguished object $\perp$ such that all objects are reflexive, i.e. the canonical map $\mu_A : A \to A^{\perp\perp}$ is an isomorphism for all $A \in \mathcal{C}$. The object $\perp$ is called the *dualizing object* .

It may be shown that a $*$-autonomous category $\mathcal{C}$ has a contravariant dualizing functor $(-)^\perp : \mathcal{C}^{op} \to \mathcal{C}$, defined on objects by $A \mapsto A^\perp$. There is a natural isomorphism : $Hom_{\mathcal{C}}(A, B) \cong Hom_{\mathcal{C}}(B^\perp, A^\perp)$.

In any $*$-autonomous category $\mathcal{C}$ there are isomorphisms

$$\begin{aligned} (A \multimap B)^\perp &\cong A \otimes B^\perp \\ I &\cong \perp^\perp \end{aligned}$$

The reader is referred to [Barr79] for many examples. Let us mention the obvious one:

Example 4.4 The category Vec_{fd} of finite dimensional vector spaces over a field k is $*$-autonomous. Here $A \multimap B = Lin(A, B)$, the space of linear maps from A to B and the dualizing object $\perp = k$. In particular $A^\perp = A^*$ is the usual dual space. More generally, within the smcc category Vec of k-vector spaces (with $\perp = k$), an object is reflexive iff it is finite dimensional.

In a $*$-autonomous category, we may define the *cotensor* $⅋$ by de Morgan duality: $A ⅋ B = (A^\perp \otimes B^\perp)^\perp$. The above example Vec_{fd} is somewhat "degenerate" since $\otimes$ and $⅋$ are identified (see the Definition 4.10 of *compact* category) . In a typical $*$-autonomous category this is not the case; indeed in linear logic one does not want to identify tensor and cotensor.

To obtain more general $*$-autonomous categories of vector spaces, we add a topological structure, due to Lefschetz [Lef63]. The following discussion is primarily based on work of M. Barr [Barr79], following the treatment in Blute [Blu96].

Definition 4.5 Let V be a vector space. A topology τ on V is *linear* if it satisfies the following three properties:

- Addition and scalar multiplication are continuous, when the field k is given the discrete topology.
- τ is hausdorff
- $0 \in V$ has a neighborhood basis of open linear subspaces.

Let $\mathcal{TVEC}$ denote the category whose objects are vector spaces equipped with linear topologies, and whose morphisms are linear continuous maps.

Barr showed that $\mathcal{TVEC}$ is a symmetric monoidal closed category, when $V \multimap W$ is defined to be the vector space of linear continuous maps, topologized with the topology of pointwise convergence. (It is shown in [Barr96] that the forgetful functor $\mathcal{TVEC} \to \mathcal{VEC}$ is tensor-preserving) . Let $V^{\perp}$ denote $V \multimap k$. Lefschetz proved that the embedding $V \to V^{\perp\perp}$ is always a bijection, but need not be an isomorphism. This is analogous to Dana Scott's method of solving domain equations in denotational semantics, using the topology to cut down the size of the function spaces.

Theorem 4.6 (Barr) *$\mathcal{RTVEC}$, the full subcategory of reflexive objects in $\mathcal{TVEC}$, is a complete, cocomplete $*$-autonomous category, with $I^{\perp} = I = k$ the dualizing object.*

Moreover, in $\mathcal{RTVEC}$, $\otimes$ and $⅋$ are not equated. More generally, other classes of $*$-autonomous categories arise by taking a linear analog of G-sets, namely categories of group representations.

Definition 4.7 Let G be a group. A *continuous G-module* is a linear action of G on a space V in $\mathcal{TVEC}$, such that for all $g \in G$, the induced map $g.(\ ) : V \to V$ is continuous. Let $\mathcal{TMOD}(G)$ denote the category of continuous G-modules and continuous equivariant maps. Let $\mathcal{RTMOD}(G)$ denote the full subcategory of reflexive objects.

We have the following result, which in fact holds in the more general context of Hopf algebras (see below).

Theorem 4.8 *The category $\mathcal{TMOD}(G)$ is symmetric monoidal closed. The category $\mathcal{RTMOD}(G)$ is $*$-autonomous, and a reflective subcategory of $\mathcal{TMOD}(G)$ via the functor $(\ \ )^{\perp\perp}$. Furthermore the forgetful functor $|\ \ | : \mathcal{RTMOD}(G) \to \mathcal{RTVEC}$ preserves the $*$-autonomous structure.*

Still more general classes of $*$-autonomous categories may be obtained from categories of modules of cocommutative Hopf algebras. Given a Hopf algebra H , a *module* over H is a linear action $\rho : \mathsf{H} \otimes V \to V$ satisfying the appropriate diagrams, analogous to the notion of G-module. Let $\mathcal{MOD}(\mathsf{H})$ denote the category of H -modules and equivariant maps. Similarly, $\mathcal{TMOD}(\mathsf{H})$, the category of continuous H -modules, is the linearly topologized version of $\mathcal{MOD}(\mathsf{H})$ where H is given the discrete topology and all vector spaces and maps are in $\mathcal{TVEC}$.

Proposition 4.9 *If H is a cocommutative Hopf algebra, $\mathcal{MOD}(\mathsf{H})$ and $\mathcal{TMOD}(\mathsf{H})$ are symmetric monoidal categories.*

We then obtain precisely the same statement as Theorem 4.8 by replacing the group G by a cocommutative Hopf algebra H . Later we shall mention noncommutative Hopf algebra models for linear logic, with respect to full completeness theorems, Section 5.2.

The case where we do identify $\otimes$ and $⅋$ is an important class of monoidal categories:

Definition 4.10 A $*$-autonomous category is *compact* if $(A \otimes B)^\perp \cong A^\perp \otimes B^\perp$ (i.e. equivalently if $A \multimap B \cong A^\perp \otimes B$).

In addition to the obvious example of Vec_{fd}, there are compact categories of relations, which have considerable importance in computer science. One such is:

Example 4.11 $Rel_\times$ is the category whose objects are sets and whose maps $R : X \to Y$ are (binary) relations $R \subseteq X \times Y$. Composition is relational composition, etc. This is a compact category, with $X \otimes Y = X \multimap Y =_{def} X \times Y$, the ordinary set-theoretic cartesian product. Define $\perp = \{*\}$, a one-element set; hence $X^\perp = X \multimap \perp = X$. On maps we have $R^\perp = R^{op}$, where $yR^{op}x$ iff xRy.

Given two smcc's $\mathcal{C}$ and $\mathcal{D}$ (we shall not distinguish the structure) what are the morphisms between them?

Definition 4.12 A *symmetric monoidal functor* is a functor $F : \mathcal{C} \to \mathcal{D}$ together with two natural transformations $m_I : I \to F(I)$ and $m_{UV} : F(U) \otimes F(V) \to F(U \otimes V)$ such that three coherence diagrams commute. In the case of the closed structure, we can define another natural transformation $\hat{m}_{UV} : F(U \multimap V) \to (FU \multimap FV)$ by $\hat{m}_{UV} = (F(ev_{U \multimap V, U}) \circ m_{U \multimap V, U})^*$. A symmetric monoidal functor is *strong* (resp. *strict*) if m_I and m_{UV} are natural isomorphisms (resp. identities) for all U, V. A symmetric monoidal functor is *strong closed* (resp. *strict closed*) if m_I and $\hat{m}_{UV}$ are natural isomorphisms (resp. identities) for all U, V. Similarly, one defines $*$-*autonomous* functors.

Finally, we need an appropriate notion of natural transformation for monoidal functors.

Definition 4.13 A natural transformation $\alpha : F \to G$ is *monoidal* if it is compatible with both m_I and m_{UV}, for all U, V, in the sense that the following equations hold:

$$\begin{aligned} \alpha_I \circ m_I &= m_I \\ m_{UV} \circ (\alpha_U \otimes \alpha_V) &= \alpha_{U \otimes V} \circ m_{UV} \quad . \end{aligned}$$

4.2 Gentzen's proof theory

Gentzen's proof theory [GLT], especially his *sequent calculi* and his fundamental theorem on Cut-Elimination, have had a profound influence not only in logic, but in category theory and computer science as well.

In the case of category theory, J. Lambek[L68, L69] introduced Gentzen's techniques to study *coherence theorems* in various free monoidal and residuated categories. This logical approach to coherence for such categories was greatly extended by G. Mints, S. Soloviev, B. Jay, et al [Min81, So87, So95, J90] For a comparison of Mints' work with more traditional Kelly-Mac Lane coherence theory see [Mac82] . More recently, coherence for large classes of structured monoidal categories arising in linear logic has been established in a series of papers by Blute, Cockett, Seely et al. This is based on Girard's extensions of Gentzen's methods. (see [BCST96, BCS96, BCS97, CS91, CS96b].)

Recent coherence theorems of Gordon-Power-Street, Joyal-Street, et. al. [GPS96, JS91, JS93] have made extensive use of higher dimensional category theory techniques and Yoneda methods, rather than logical methods. Related Yoneda techniques are now being introduced, in the reverse direction into proof theory, as we outlined in Section2.7 above.

In computer science, entire research areas: proof search (AI, Logic Programming), operational semantics, type inference algorithms, logical frameworks, etc. are a testimonial to Gentzen's work. Indeed, Gentzen's Natural Deduction and Sequent Calculi are fundamental methodological as well as mathematical tools.

A profound and exciting analysis of Gentzen's work has arisen recently in the rapidly growing area of Linear Logic (=LL), developed by J-Y. Girard in 1986. While classical logic is about universal truth, and intuitionistic logic is about constructive proofs, LL is a logic of resources and their management and reuse. (e.g. see [Gi87, Gi89, GLR, Sc93, Sc95, Tr92])

4.2.1 Gentzen sequents

Gentzen's analysis of Hilbert's proof theory begins with a fundamental reformulation of the syntax. We follow the presentation in [GLT].

A *sequent* for a logical language $\mathcal{L}$ is an expression

$$A_1, A_2, \cdots, A_m \vdash B_1, B_2, \cdots, B_n \tag{14}$$

where $A_1, A_2, \cdots, A_m$ and $B_1, B_2, \cdots, B_n$ are finite lists (possibly empty) of formulas of $\mathcal{L}$. Sequents are denoted $\Gamma \vdash \Delta$, for lists of formulas Γ and Δ. Gentzen introduced formal rules for generating sequents, the so-called *derivable* sequents. Gentzen's rules analyze the deep structure and implicit symmetries hidden in logical syntax . Computation in this setting arises from one of two methods:

- The traditional method is Gentzen's Cut-Elimination Algorithm, which allows us to take a formal sequent calculus proof and reduce it to cut-free form. This is closely related to both normalization of lambda terms (cf. Sections 2.7) as well as the operational semantics of such programming languages as PROLOG.

- More recent is the proof search paradigm , which is the bottom-up, goal-directed view of building sequent proofs and is the basis of the discipline of Logic Programming [MNPS, HM94, Mill].

Categorically, the cut elimination algorithm is at the heart of the proof-theoretic approach to coherence theorems previously mentioned. On the other hand, Logic Programming and the proof-search paradigm have only recently attracted the attention of categorists (cf. [FiFrL, PK96]).

Lambek pointed out that Gentzen's sequent calculus was analogous to Bourbaki's method of bilinear maps. For example, given sequences $\Gamma = A_1 \cdots A_m$ and $\Delta = B_1 B_2 \cdots B_n$ of $R{-}R$ bimodules of a given ring R, there is a natural isomorphism

$$Mult(\Gamma A B \Delta, C) \cong Mult(\Gamma A \otimes B \Delta, D) \tag{15}$$

between $m+n+2$-linear and $m+n+1$-linear maps. Bourbaki derived many aspects of tensor products just from this universal property. Such a formal bijection is at the heart of Linear Logic (e.g. [L93]).

Traditional logicians think of sequent (14) as saying : *the conjunction of the A_i entails the disjunction of the B_j*. More generally, following Lambek and Lawvere (cf. Section2.1), categorists interpret such sequents (modulo equivalence of proofs) as *arrows* in appropriate categories. For example, in the case of logics similar to linear logic [CS91], the sequent (14) determines an arrow of the form

$$A_1 \otimes A_2 \otimes \cdots \otimes A_m \longrightarrow B_1 \parr B_2 \cdots \parr B_n \tag{16}$$

in a symmetric monoidal category with a "cotensor" $\parr$ (see Section 4.1 below).

4.2.2 Girard's Analysis of the Structural Rules

Gentzen broke down the manipulations of logic into two classes of rules applied to sequents: *structural rules* and *logical rules*. All rules come in pairs (left/right) applying to the left (resp. right) side of a sequent.

Gentzen's Structural Rules (Left/Right)

$$
\begin{array}{llll}
\textit{Permutation} & \dfrac{\Gamma \vdash \Delta}{\sigma(\Gamma) \vdash \Delta} & \dfrac{\Gamma \vdash \Delta}{\Gamma \vdash \tau(\Delta)} & \sigma, \tau \text{ permutations.} \\[2ex]
\textit{Contraction} & \dfrac{\Gamma, A, A \vdash \Delta}{\Gamma, A \vdash \Delta} & \dfrac{\Gamma \vdash \Delta, B, B}{\Gamma \vdash \Delta, B} & \\[2ex]
\textit{Weakening} & \dfrac{\Gamma \vdash \Delta}{\Gamma, A \vdash \Delta} & \dfrac{\Gamma \vdash \Delta}{\Gamma \vdash \Delta, B} &
\end{array}
$$

For simplicity, consider *intuitionistic* sequents , i.e. those of the form $A_1, A_2, \cdots A_m \vdash B$ with one conclusion. So the right rules disappear and we consider the *left* rules above. We can give a Curry-Howard-style analysis to Gentzen's intuitionistic sequents (cf. Section 2.3), assigning lambda terms (qua functions) to sequents, e.g. $x_1 : A_1, \cdots, x_m : A_m \vdash t(\vec{x}) : B$. The structural rules say the following: *Permutation* says that the class of functions is closed under permutations of arguments; *Contraction* says that the class of functions is closed under duplicating arguments–i.e. setting two input variables equal; and *Weakening* says the class of functions is closed under adding dummy arguments. In the absence of such rules, we obtain the so called *linear* lambda terms, terms where all variables occur exactly once. (see [GSS91, Abr93, L89]):

By removing these traditional structural rules, logic takes on a completely different character[3] (see Figure 8). Previously equivalent notions now split into subtle variants based on resource allocation. For example, the rules for *Multiplicative* connectives simply concatenate their input hypotheses Γ and Γ', whereas the rules for *Additive* connectives merge two input hypotheses Γ into one. The situation is analogous for outputs Δ and Δ' (see Figure 8). The resultant logical connectives can represent linguistic distinctions related to resource use which are simply impossible to formulate in traditional logic (see [Gi86, Abr93, Sc93, Sc95]).

Remark 4.14 First we should remark that on the controversial subject of notation in LL, we have chosen a reasonable categorical notation, somewhere between [Gi87] and [See89]. Observe that in CLL, two-sided sequents can be replaced by one-sided sequents, since $\Gamma \vdash \Delta$ is equivalent to $\vdash \Gamma^\perp, \Delta$, with $\Gamma^\perp$ the list $A_1^\perp, \cdots, A_n^\perp$, where Γ is $A_1, \cdots, A_n$.

Thus the key aspect of linear logic proofs is their resource sensitivity. We think of a linear entailment $A_1, \cdots, A_m \vdash B$ not as an ordinary function, but as an *action*–a kind of process that in a single step consumes the inputs A_i and produces output B. For example, this permits representing in a natural manner the step-by-step behaviour of various abstract machines, certain models of concurrency like Petri Nets, etc. Thus, linear logic permits us to describe the instantaneous state of a system, and its step-wise evolution, intrinsically within the logic itself (e.g. with no need for explicit time parameters, etc.)

But linear logic is not about simply removing Gentzen's structural rules, but rather modulating their use. To this end, Girard introduces a new connective $!A$, which indicates that contraction and weakening may be applied to formula A. This yields the *Exponential* connectives in Figure 8. From a resource viewpoint, an hypothesis $!A$ is one which can be reused arbitrarily. Moreover, this permits decomposing "$\Rightarrow$" (categorically, the ccc function space) into more basic notions:

$$A \Rightarrow B = (!A) \multimap B$$

[3]Formulas of LL are generated from literals $p, q, r, \cdots, p^\perp, q^\perp, r^\perp, \cdots$ and constants $I, \perp, 1, 0$ using binary operations $\otimes, \parr, \times, \oplus$ and unary $!, ?$. Negation $(-)^\perp$ is defined inductively: $I^\perp = \perp, \perp^\perp = I, 1^\perp = 0, 0^\perp = 1, p^{\perp\perp} = p, (A \otimes B)^\perp = A^\perp \parr B^\perp, (A \parr B)^\perp = A^\perp \otimes B^\perp, (A \times B)^\perp = (A^\perp \oplus B^\perp), (A \oplus B)^\perp = A^\perp \times B^\perp, (!A)^\perp = ?(A^\perp), (?A)^\perp = !(A^\perp)$. Also we define $A \multimap B = A^\perp \parr B$.

Structural	*Perm*	$\dfrac{\Gamma \vdash \Delta}{\sigma(\Gamma) \vdash \tau(\Delta)}$	σ, τ permutations.
Axiom&Cut	*Axiom*	$A \vdash A$	
	Cut	$\dfrac{\Gamma \vdash A, \Delta \quad \Gamma', A \vdash \Delta'}{\Gamma, \Gamma' \vdash \Delta, \Delta'}$	
Negation		$\dfrac{\Gamma \vdash A, \Delta}{\Gamma, A^{\perp} \vdash \Delta}$	$\dfrac{\Gamma, A \vdash \Delta}{\Gamma \vdash A^{\perp}, \Delta}$
Multiplicatives	*Tensor*	$\dfrac{\Gamma, A, B \vdash \Delta}{\Gamma, A \otimes B \vdash \Delta}$	$\dfrac{\Gamma \vdash A, \Delta \quad \Gamma' \vdash B, \Delta'}{\Gamma, \Gamma' \vdash A \otimes B, \Delta, \Delta'}$
	Par	$\dfrac{\Gamma, A \vdash \Delta \quad \Gamma', B \vdash \Delta'}{\Gamma, \Gamma', A \parr B \vdash \Delta, \Delta'}$	$\dfrac{\Gamma \vdash A, B, \Delta}{\Gamma \vdash A \parr B, \Delta}$
	Units	$\dfrac{\Gamma \vdash \Delta}{\Gamma, I \vdash \Delta}$	$\vdash I$
		$\bot \ \vdash$	$\dfrac{\Gamma \vdash \Delta}{\Gamma \vdash \bot, \Delta}$
	Implication	$\dfrac{\Gamma \vdash A, \Delta \quad \Gamma', B \vdash \Delta'}{\Gamma, \Gamma', A \multimap B \vdash \Delta, \Delta'}$	$\dfrac{\Gamma, A \vdash B, \Delta}{\Gamma \vdash A \multimap B, \Delta}$
Additives	*Product*	$\dfrac{\Gamma, A \vdash \Delta}{\Gamma, A \times B \vdash \Delta} \quad \dfrac{\Gamma, B \vdash \Delta}{\Gamma, A \times B \vdash \Delta}$	$\dfrac{\Gamma \vdash A, \Delta \quad \Gamma \vdash B, \Delta}{\Gamma \vdash A \times B, \Delta}$
	Coproduct	$\dfrac{\Gamma, A \vdash \Delta \quad \Gamma, B \vdash \Delta}{\Gamma, A + B \vdash \Delta}$	$\dfrac{\Gamma \vdash A, \Delta}{\Gamma \vdash A + B, \Delta} \quad \dfrac{\Gamma \vdash B, \Delta}{\Gamma \vdash A + B, \Delta}$
	Units	$\Gamma, 0 \ \vdash \Delta$	$\Gamma \vdash 1, \Delta$
Exponentials	*Weakening*	$\dfrac{\Gamma \vdash \Delta}{\Gamma, !A \vdash \Delta}$	*Contraction* $\dfrac{\Gamma, !A, !A \vdash \Delta}{\Gamma, !A \vdash \Delta}$
	Storage	$\dfrac{!\Gamma \ \vdash A}{!\Gamma \ \vdash !A}$	*Dereliction* $\dfrac{\Gamma, A \vdash \Delta}{\Gamma, !A \vdash \Delta}$

Figure 8: Rules for Classical Propositional LL

Finally, nothing is lost: classical (as well as intuitionistic) logic can be faithfully translated into CLL. [Gi87, Tr92].

4.2.3 Fragments and Exotic Extensions

The richness of LL permits many natural subtheories (cf.[Gi87, Gi95a]). For a survey of the surprisingly intricate complexity-theoretic structure of many of the fragments of LL see Lincoln [Li95]. These results often involve direct and natural simulation of various kinds of abstract computing machines within the logic [Sc95, Ka95]. Of course there are specific fragments corresponding to various subcategories of categorical models, in the next section. There are also fragments directly connected with classifying complexity classes in computing [GSS92, Gi97] but these latter have not been the object of categorical analysis.

More exotic "noncommutative" fragments of LL are obtained by eliminating or modifying the permutation rule; i.e. one no longer assumes $\otimes$ is symmetric. One such precursor to LL is the work of J. Lambek in the 1950's on categorial grammars in mathematical linguistics (for recent surveys, see [L93, L95]). Here the language becomes yet more involved, since there are two implications $\multimap$ and $\circ\!-$ and two negations $A^{\perp}$ and ${}^{\perp}A$. It has been proven by Pentus [Pen93] that Lambek grammars are equivalent to context-free grammars. In [L89] there is a formulation of Lambek grammars using the notion of multicategory, an idea currently of some interest in higher-dimensional category theory and higher dimensional rewriting theory [HMP98].

D.Yetter [Y90] considered *cyclic* linear logic, a version of LL in which the permutation rule is modified to only allow *cyclic* permutations. This will be discussed briefly below in Section 5.2 with respect to Full Completeness. A proposed classification of different fragments of LL, including braided versions, based on Hopf-algebraic models is in Blute [Blu96], see also Section 5.2.

4.2.4 Topology of Proofs

Let us briefly mention one of the main novelties of linear logic. Traditional Gentzen proof theory writes proofs as trees. In order to give a Curry-Howard isomorphism to arbitrary sequents $\Gamma \vdash \Delta$, Girard introduced multiple-conclusion graphical networks to interpret proofs. These *proof nets* use graph rewriting moves for their operational semantics. It is here that one sees the dynamic aspects of cut-elimination. In essense these networks are the "lambda terms" of linear logic. There are known mathematical criteria to classify which (among arbitrary) networks arise from Gentzen sequent proofs, i.e. in a sense which of these parallel networks are "sequentializable" into a Gentzen proof tree. Homological aspects of proof nets are studied in [Mét94]

The technology of proof nets has grown into an intricate subject. In addition to their uses in linear logic, proof nets are now used in category theory, as a technical tool in graphical approaches to coherence theorems for structured monoidal categories (e.g. [Blu93, BCST96, CS91, CS96b]). There are proof net theories for numerous non-commutative, cyclic, and braided linear logics, e.g. [Abru91, Blu93, Fl96, FR94, Y90].

The method of proof nets has been extended by Y. Lafont [Laf95] to a general graphical language of computation, his *interaction nets*. These latter provide a simple model of parallel computation with, at the same time, new insights into sequential computation.

4.3 What is a categorical model of LL?

4.3.1 General Models

As in Section 2.1, we are interested in finding the categories appropriate to modelling linear logic proofs (just as cartesian closed categories modelled intuitionistic $\wedge, \Rightarrow, \top$ proofs). The basic equations we postulate arise from the operational semantics–that is *normalization of proofs*. In the case of sequent calculi, this is Gentzen's Cut-Elimination process [GLT]. However, there are sometimes natural categorical equations which are not decided by traditional proof theory. The problem is further compounded in linear logic (and monoidal categories) in that there may be several (non-canonical) candidates for appropriate monoidal structure.

The first categorical semantics of LL is in Seely's paper [See89], which is still perhaps the most readable account. Subsequent development of appropriate term calculi [Abr93, Bie95, BBPH, W94, BCST96, CS91, CS96b] have modified and enlarged the scope, but not essentially changed the original analysis for the case of classical linear logic (= CLL). We impose the following equations between CLL proofs, in order to form a category $\mathcal{C}$, where sequents are interpreted as (equivalence classes of) arrows according to formula (16), based on the rules in Figure 8.

- $\mathcal{C}$ is a *symmetric monoidal closed category with products, coproducts, and units* (from the rules: Axiom, Cut, Perm, the Multiplicatives and the Additives).
- $\mathcal{C}$ is $*$-*autonomous* (from the Negation rule) with $\otimes$ and $⅋$ related by de Morgan duality.
- $! : \mathcal{C} \to \mathcal{C}$ is an endofunctor, with associated monoidal transformations $\varepsilon : ! \to id_{\mathcal{C}}$ and $\delta : ! \to !!$ satisfying:
  1. $(!, \delta, \varepsilon)$ forms a *monoidal* comonad on $\mathcal{C}$.
  2. There are natural isomorphisms
  $$I \cong !1 \text{ and } !A \otimes !B \cong !(A \times B) \quad .$$
  making $! : (\mathcal{C}, \times, 1) \to (\mathcal{C}, \otimes, I)$ a symmetric monoidal functor.
  3. In particular, $I \xleftarrow{e_A} !A \xrightarrow{d_A} !A \otimes !A$ is a cocommutative comonoid, for all A in $\mathcal{C}$ and the coalgebra maps $\varepsilon_A : !A \to A$ and $\delta_A : !A \to !!A$ are comonoid maps. In fact, these conditions are a consequence of (2), but are required explicitly in weaker settings.

For modifications appropriate to more general situations (e.g. various fragments of LL without products, linearly distributive categories, etc.) see [Bie95, BCS96].

The essence of Girard's translation of intuitionistic logic into LL is the following easy result (cf [See89, Bie95]).

Proposition 4.15 *If $\mathcal{C}$ is a categorical model of CLL, as above, then the Kleisli category $\mathcal{K}_{\mathcal{C}}$ of the comonad $(!, \delta, \varepsilon)$ is a ccc. Moreover finite products in $\mathcal{K}_{\mathcal{C}}$ and $\mathcal{C}$ coincide, while exponentials in $\mathcal{K}_{\mathcal{C}}$ are given by: $A \Rightarrow B = (!A) \multimap B$.*

We should mention one formal rule, MIX, which appears frequently in the literature. To express it, we use one-sided sequents:

$$Mix \qquad \frac{\vdash \Gamma \quad \vdash \Delta}{\vdash \Gamma, \Delta}$$

Categorically, MIX entails there is a map $A \otimes B \to A ⅋ B$. This rule seems to be valid in most models, certainly so in ones based on $\mathcal{RTVEC}$.

Remark 4.16 The categorical comonad approach to models of linear logic has been put to use by Asperti in clarifying optimal graph reduction techniques in the untyped lambda calculus [Asp] (see also [GAL92]).

4.3.2 Concrete Models

There are by now many categorical models of LL and its interesting fragments. Let us just mention a few [Gi95a, Tr92]:

- *Posetal Models* or *Girard's Phase semantics*. These are $*$-autonous posets with additional structure. This gives an algebraic semantics analogous to Boolean or Heyting algebras for classical (resp. intuitionistic) logic. As categories they are trivial (each hom set has at most one element); hence they do not model proofs but rather provability. There is associated a traditional Tarski semantics, with Soundness and Completeness Theorems. Recently, these models have been applied in Linear Concurrent Constraint Programming, for proving "safety" properties of programs [FRS98].

- *Domain-Theoretic Models*. The category LIN = coherent spaces and linear maps gave the first non-trivial model of LL proofs. This model arose from Girard's analysis of the ccc STAB, realizing that there were many other logical operations available. Indeed, STAB is the Kleisli category of an approproate comonad $(!, \delta, \varepsilon)$ on LIN (cf. Proposition 4.15). In the model LIN, $!A$ is a minimal solution of the domain equation $!A \cong I \times A \times (!A \otimes !A)$, indeed is a cofree comonoid.

- *Relational Models*. As discussed in Barr [Barr91], many compact categories are complete enough to interpret
$$!A \cong \perp \times A \times E_2(A) \times \cdots \times E_n(A) \times \cdots$$
where $E_n(A)$ is the equalizer of the $n!$ permutations of the nth tensor power $A^{\otimes n}$, for $n \geq 2$. For example, Barr proves $Rel_\times$ has that property. More generally, Barr [Barr91] constructs models based on the *Chu-space* construction in [Barr79]. Chu spaces are themselves an interesting class of models of LL and have been the subject of intensive investigation by M. Barr and by Vaughn Pratt [Pra95]

- *Games Models*. Categories of Games now provide some of the most exciting new semantics for LL and Programming Languages. This so-called *intensional semantics* provides a finer-grained analysis of computation than traditional (categorical) models, taking into account the dynamic or interactive aspects of computation. For example, such games can be used to model interactions between a System and its Environment and provided the first syntax-free fully abstract models of PCF, answering a long-standing open problem. Games categories have been extended to handle programming languages with many additional properties, e.g. control features, call-by-value languages, languages with side-effects and store, etc. as well as modern logics like LL, System $\mathcal{F}$, etc. For basic introductions, see [Abr97, Hy97]. For a small sample of more recent work, see [Mc97, AHMc98, AMc98, BDER97].

- *GoI and Functional Analytic Models*: The Geometry of Interaction Program (see e.g. [Gi88, Gi90, Gi95b, DR95]) aims to model the dynamics of cut-elimination by interpreting proofs as objects of a certain C^* algebra, with logical rules corresponding to certain $*$-isomorphisms. The essence of Gentzen's cut-elimination theorem is summarized by the so-called *execution formula*. We shall look at an abstract form of the GoI program (in traced monoidal categories) in Section 6.1. The GoI program itself has influenced both game semantics and work on optimal reduction.

- Finally, as the name suggests, linear logic is roughly inspired from linear algebra. Thus $!A$ is analogous to the Grassmann algebra. Indeed, in categories of Hilbert or Banach spaces, one is reminded of the symmetric and antisymmetric Fock space construction [Ge85]. For a (non-categorical) Banach space interpretation of LL, see Girard [Gi96].

5 Full Completeness

5.1 Representation Theorems

The most basic representation theorem of all is the Yoneda embedding:

Theorem 5.1 (Yoneda) *If $\mathcal{A}$ is locally small, the Yoneda functor $\mathcal{Y} : \mathcal{A} \to Set^{\mathcal{A}^{op}}$, where $\mathcal{Y}(A) = Hom_{\mathcal{A}}(-, A)$, is a fully faithful embedding.*

Indeed, Yoneda preserves limits as well as cartesian closedness.

We seek mathematical models which describe the behaviour of programs. From the viewpoint of the Curry-Howard isomorphism (which identifies proofs with programs) we seek representation theorems for proofs–i.e. mathematical models which fully and faithfully represent proofs. From the viewpoint of a logician, these are Completeness Theorems, but now at the level of proofs rather than provability.

One of the first such theorems was proved by H. Friedman [Frie73]. Friedman showed completeness of typed lambda calculus with respect to ordinary set-theoretic reasoning. Consider the pure typed lambda calculus $\mathcal{L}^{\Rightarrow}$, whose types are generated from some base sorts by $\Rightarrow$ only. We interpret $\mathcal{L}^{\Rightarrow}$ set-theoretically in a full type hierarchy $\mathcal{A}$ (see Example 2.3).

Theorem 5.2 (Friedman) *Let $\mathcal{A}$ be a full type hierarchy with base sorts interpreted as infinite sets. Then for any pure typed lambda terms M, N, $M \underset{\Gamma}{=} N$ is true in $\mathcal{A}$ iff $M \underset{\Gamma}{=} N$ is provable using the rules of typed lambda calculus.*

Similar results but using instead full type hierarchies over ω-CPO or Per-based models have been given by Plotkin and by Mitchell using logical relations (see [Mit96]). Friedman's original **Set**-based theorem has been extended by Cubric to the entire ccc language $\Rightarrow, \times, 1$ [Cu93] to yield the following

Theorem 5.3 (Cubric) *Let $\mathcal{C}$ be a free ccc generated by a graph. Then there exists a faithful ccc functor $F : \mathcal{C} \to Set$.*

Alas this representation is not full.

Let $\mathcal{B}_{\mathcal{G}}$ = the free ccc with binary coproducts generated by discrete graph $\mathcal{G}$, given syntactically by types and terms of typed lambda calculus. For any group G, the functor category Set^G is a ccc with coproducts. So according to the universal property, if F is an initial assignment of G-sets to atomic types then a proof of formula σ, qua closed term $M : \sigma$, qua $\mathcal{F}_{\mathcal{G}}$-arrow $M : 1 \to \sigma$, corresponds to a G-set (= equivariant) map $[\![M]\!]_F : 1 \to [\![\sigma]\!]_F$. Such maps are fixed points under the action. In particular, letting $G = Z$, we obtain the easy half of the following theorem, due to Läuchli [Lau]:

Theorem 5.4 (Läuchli) *A $\{\top, \wedge, \Rightarrow, \vee\}$-formula σ of intuitionistic propositional calculus is provable if and only if for every interpretation F of the base types, its Set^Z-interpretation $[\![\sigma]\!]_F$ has an invariant element.*

Indeed, Harnik and Makkai extend Läuchli's theorem to a representation theorem. Recall, a functor Φ is *weakly full* if $Hom(A, B) = \emptyset$ implies $Hom(\Phi(A), \Phi(B)) = \emptyset$.

Theorem 5.5 (Harnik, Makkai [HM92]) *Let $\mathcal{B}$ be a countable free ccc with binary coproducts. There is a weakly full representation Φ of $\mathcal{B}$ into a countable power of Set^Z. If in addition the* *terminal object* 1 *is indecomposable, then there is a weakly full representation into* Set^Z.

A weakly full representation of $\mathcal{B}$ corresponds to *completeness with respect to provability*: i.e. $Hom_{Set^Z}(1, \Phi(B)) \neq \emptyset$ implies $Hom_{\mathcal{B}}(1, B) \neq \emptyset$, so B is provable. We shall give stronger representation theorems still based on the idea of invariant elements.

5.2 Full Completeness Theorems

A recent topic of considerable interest is *full completeness* theorems. Suppose we have a free category $\mathcal{F}$. We shall say that a model category $\mathcal{M}$ is *fully complete for* $\mathcal{F}$ or that we have *full completeness of* $\mathcal{F}$ *with respect to* $\mathcal{M}$ if the unique free functor (with respect to any interpretation of the generators) $[\![-]\!] : \mathcal{F} \to \mathcal{M}$ is full. It is even better to demand that $[\![-]\!]$ is a fully faithful representation.

For example, suppose $\mathcal{F}$ is a free ccc generated by the typed lambda calculus (cf. Example 2.22). To say a ccc $\mathcal{M}$ is fully complete for $\mathcal{F}$ means the following: given any interpretation of the generators as objects of $\mathcal{M}$, any arrow $[\![A]\!] \to [\![B]\!] \in \mathcal{M}$ between definable objects is itself definable, i.e. of the form $[\![f]\!]$ for $f : A \to B$. If the representation is fully faithful, f is unique. Thus, by Curry-Howard, any morphism in the model between definable objects is itself the image of a proof (or program); indeed of a unique proof if the representation is fully faithful. Thus, such models $\mathcal{M}$, while being semantical, really capture aspects of the syntax of the language.

Such results are mainly of interest when the models $\mathcal{M}$ are "genuine" mathematical models not apparently connected to the syntax. In that case Full Completeness results are more surprising (and interesting). For example, an explicit use of the Yoneda embedding $\mathcal{Y} : \mathcal{C} \to Set^{\mathcal{C}^{op}}$ is not what we want, since $Set^{\mathcal{C}^{op}}$ depends too much on $\mathcal{C}$.

Probably the first full completeness results for free ccc's were by Plotkin [Plo80], using categories of logical relations. In the case of simply typed lambda calculus generated from a fixed base type (= the free ccc on one object), Plotkin proved the following result. Consider the Henkin model T_B = *the full type hierarchy over a set* B, i.e. the full sub-ccc of Sets generated by some set B. The Soundness Theorem for logical relations says that if a term f is lambda definable, it is invariant under all logical relations. We ask for the converse.

The *rank* of a type is defined inductively: rank$(b) = 0$, where b is a base type, rank$(\sigma \Rightarrow \tau)$ = max $\{$rank$(\sigma) + 1$, rank$(\tau)\}$, rank$(\sigma \times \tau)$ = max $\{$rank(σ), rank$(\tau)\}$. The rank of an element $f \in B_\sigma$ in T_B is the rank of the type σ.

Theorem 5.6 (Plotkin, [Plo80]) *In the full type hierarchy* T_B *over an infinite set* B*, all elements* f *of rank* ≤ 2 *satisfy: if* f *is invariant under all logical relations, then* f *is lambda definable.*

This result has been extended by Statman [St85], but the same question for terms of arbitrary rank is still open. However Plotkin [Plo80] did prove the above result for lambda terms of arbitrary rank, by moving to *Kripke Logical Relations* rather than **Set**-based logical relations. Kripke relations occur essentially by replacing Set by a functor category $Set^{P^{op}}$, P a poset, i.e. by looking at P-indexed families of sets and relations. Extensions, with new characterizations of lambda definability, are in work of Jung and Tiuryn [JT93]. A clear categorical treatment of their work, and logical-relations-based full completeness theorems, is in Alimohamed [Ali95] (cf. also [Mit96]).

The name "Full Completeness" first arose in Game Semantics, where the fundamental paper of Abramsky and Jagadeesan [AJ94b] proved full completeness for multiplicative linear logic (+ the Mix rule), using categories of games with history-free winning strategies as morphisms. It is shown there that "uniform" history-free winning strategies are the denotations of unique proof nets. A more involved notion of game, developed by Hyland and Ong (see [Hy97]), permits eliminating the Mix rule in proofs of full completeness for the multiplicatives. These results paved the way for the most spectacular application of these game-theoretic methods: the solution of the Full Abstraction problem for PCF, by Abramsky,Jagadeesan, and Malacaria and by Hyland and Ong , referred to in Section 4.3.2.

In [BS96, BS96b, BS98] Full Completeness for MLL + Mix and for Yetter's Cyclic Linear Logic were also proved using dinaturality and a generalization of Laüchli semantics. Let us briefly recall that view.

5.2.1 Linear Läuchli Semantics

Let $\mathcal{C}$ be a $*$-autonomous category. Given an MLL formula $\varphi(\alpha_1, \ldots, \alpha_n)$ built from $\otimes, \multimap, (\ )^\perp$, with type variables $\alpha_1, \ldots, \alpha_n$, we inductively define its *functorial interpretation* $[\![\varphi(\alpha_1, \ldots, \alpha_n)]\!] : (\mathcal{C}^{op})^n \times \mathcal{C}^n \to \mathcal{C}$ as follows:

- $[\![\varphi]\!](AB) = \begin{cases} B_i & \text{if } \varphi(\alpha_1, \ldots, \alpha_n) \equiv \alpha_i \\ A_i^\perp & \text{if } \varphi(\alpha_1, \ldots, \alpha_n) \equiv \alpha_i^\perp \end{cases}$
- $[\![\varphi_1 \otimes \varphi_2]\!](AB) = [\![\varphi_1]\!](AB) \otimes [\![\varphi_2]\!](AB)$.
- $[\![\varphi_1 \multimap \varphi_2]\!](AB) = [\![\varphi_1]\!](BA) \multimap [\![\varphi_2]\!](AB)$.

The last two clauses correspond to Equations 11 and 12 (following Example 3.5 in Section 3.1). It is readily verified that $[\![\varphi^\perp]\!] = [\![\varphi]\!]^\perp$. Also recall that in MLL, $A \multimap B$ is defined as $A^\perp ⅋ B$. From now on, let $\mathcal{C} = \mathcal{RTVEC}$.

The set $Dinat(F, G)$ of dinatural transformations from F to G is a vector space, under pointwise operations. We call it the *space of proofs* associated to the sequent $F \vdash G$ (where we identify formulas with definable functors.) If $\vdash \Gamma$ is a one-sided sequent, then $Dinat(\Gamma)$ denotes the set of dinaturals from $\mathbf{k}$ to $[\![⅋\Gamma]\!]$. In such sequents, we sometimes abbreviate $[\![⅋\Gamma]\!]$ to $[\![\Gamma]\!]$.

The following is proved in [BS96, BS98]. A binary sequent is one where each atom appears exactly twice, with opposite variances.

Theorem 5.7 (Full Completeness for Binary Sequents) *Let F and G be formulas in multiplicative linear logic, interpreted as definable multivariant functors on $\mathcal{RTVEC}$. Given a binary sequent $F \vdash G$, then $Dinat(F, G)$ is zero or 1-dimensional, depending on whether or not $F \vdash G$ is provable. If it is provable, every dinatural is a scalar multiple of the denotation of the unique cut-free proof (qua Girard proof-net).*

A *diadditive dinatural transformation* is one which is a linear combination of substitution instances of binary dinaturals. Under the same hypotheses as above we obtain:

Theorem 5.8 (Full Completeness) *The proof space $Dinat(F, G)$ of diadditive dinatural transformations has as basis the denotations of cut-free proofs in the theory MLL+MIX.*

Example 5.9 *The proof space of the sequent*

$$\alpha_1, \alpha_1 \multimap \alpha_2, \alpha_2 \multimap \alpha_3, \ldots, \alpha_{n-1} \multimap \alpha_n \vdash \alpha_n$$

has dimension 1, generated by the evaluation dinatural.

The proofs of the above results actually yield a fully faithful representation theorem for a free $*$-autonomous category with MIX, canonically enriched over vector spaces ([BS98]).

In [BS98], a similar Full Completeness Theorem and fully faithful representation theorem is given for Yetter's Cyclic Linear Logic. In this case one employs the category $\mathcal{RTMOD}(\mathsf{H})$ for a Hopf algebra H . This is based on the following observation [Blu96]:

Proposition 5.10 *If* H *is a hopf algebra with an involutive antipode, i.e.* $S^2 = id$ *then* $\mathcal{RTMOD}(\mathsf{H})$ *is a cyclic* $*$*-autonomous category, i.e. a model of Yetter's cyclic linear logic.*

The particular Hopf algebra used is the shuffle Hopf algebra, described in [Ben, Haz, BS98]. Once again we consider formulas as multivariant functors on $\mathcal{RTVEC}$, but restrict the dinaturals to so-called *uniform dinaturals* $\theta_{|V_1|,\cdots,|V_n|}$, i.e. those which are equivariant with respect to the H -action induced from the atoms, for H -modules $V_i \in \mathcal{RTMOD}(\mathsf{H})$. This is completely analogous to the techniques used in logical relations.

Related results using Chu spaces are in [Pra97]. [4]

6 Feedback and Trace

6.1 Traced Monoidal Categories

This new class of categories, introduced by Joyal, Street, and Verity [JSV96], have shown surprising connections to models of computation and iteration. The original versions were very general, including braided and tortile categories arising in several branches of mathematics. At the moment, most of the applications to computing omit any braided structure. But even at the abstract level of [JSV96], the authors illustrate a computational, geometric calculus somewhat akin to Girard's proof nets in linear logic [Gi87], and indeed some precise connections have been made [BCS98]. Moreover, the main construction in [JSV96] has been shown by Abramsky [Abr96] to have fascinating connections with Girard's GoI program, as already hinted by Joyal, Street, and Verity.

We now give a version of traced symmetric monoidal categories. For ease of readability and without loss of generality, we consider strict monoidal categories (recall, from Mac Lane's coherence theorem, that every monoidal category is equivalent to a strict one).

Definition 6.1 A *traced symmetric monoidal category (= tmc)* is a symmetric monoidal category $(\mathcal{C}, \otimes, I, s)$ (where $s_{X,Y} : X \otimes Y \longrightarrow Y \otimes X$ is the symmetry morphism) with a family of functions $Tr^U_{X,Y} : \mathcal{C}(X \otimes U, Y \otimes U) \longrightarrow \mathcal{C}(X, Y)$, called a *trace*, subject to the following conditions:

- **Natural** in X, $Tr^U_{X,Y}(f)g = Tr^U_{X',Y}(f(g \otimes 1_U))$, where $f : X \otimes U \longrightarrow Y \otimes U$, $g : X' \longrightarrow X$,
- **Natural** in Y, $gTr^U_{X,Y}(f) = Tr^U_{X,Y'}((g \otimes 1_U)f)$, where $f : X \otimes U \longrightarrow Y \otimes U$, $g : Y \longrightarrow Y'$,
- **Dinatural** in U, $Tr^U_{X,Y}((1_Y \otimes g)f) = Tr^{U'}_{X,Y}(f(1_X \otimes g))$, where $f : X \otimes U \longrightarrow Y \otimes U'$, $g : U' \longrightarrow U$,
- **Vanishing**, $Tr^I_{X,Y}(f) = f$ and $Tr^{U \otimes V}_{X,Y}(g) = Tr^U_{X,Y}(Tr^V_{X \otimes U, Y \otimes U}(g))$ for $f : X \otimes I \longrightarrow Y \otimes I$ and $g : X \otimes U \otimes V \longrightarrow Y \otimes U \otimes V$.
- **Superposing**, $g \otimes Tr^U_{X,Y}(f) = Tr^U_{W \otimes X, Z \otimes Y}(g \otimes f)$
- **Yanking**, $Tr^U_{U,U}(s_{U,U}) = 1_U$.

[4] Added in proof: there has been recent progress on the above work. Masahiro Hamano (JAIST) has managed to eliminate the use of dinaturals in the full completeness proofs in both of the Blute-Scott papers [BS96, BS98], giving a direct denotational interpretation of MLL + Mix-full completeness in the categories of reflexive topological vector spaces above. This paper will appear in *Ann. Pure and Applied Logic.* Also Hamano has proved MLL (without Mix) full completeness in Barr's category of Reflexive Topological Abelian Groups, using Pontrjagin duality and the dinatural framework above. This will appear in *Math. Struc. in Computer Science*, in a volume dedicated to the 75th birthday of J. Lambek.

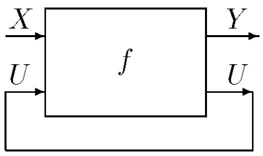


Figure 9: The trace $Tr^U_{X,Y}(f)$

From a computer science viewpoint, the essential feature is to think of $Tr^U_{X,Y}(f)$ as "feedback along U", as in Figure 9. The axiomatization given here differs slightly from those in [Abr96, JSV96], although it can be shown to be equivalent. We shall leave it to the reader to draw the diagrams for the trace axioms. We note however that Vanishing expresses trace along a tensor $U \otimes V$ in terms of iterated traces along U and V. This is related to the so-called Bekič Lemma in Domain Theory.

The above notion is really a *parametrized trace.* The usual notion from linear algebra is when $X = Y = I$ (see Example 6.3) below.

Example 6.2 ***Rel***$_\times$*:* The objects are sets, $\otimes = \times$ (cartesian product), and maps are binary relations. Composition means composition of relations, and $x\, Tr^U_{X,Y}\, y$ iff there exists a u such that $(x,u)R(y,u)$.

Example 6.3 Vec_{fd} : Given $f : X \otimes U \to Y \otimes U$, define $Tr^U_{X,Y}(f)(x_i) = \sum_{j,k} \alpha^{kj}_{ij} y_k$, where $f(x_i \otimes u_j) = \sum_{k,m} \alpha^{km}_{ij} y_k \otimes u_m$, where $(u_i), (x_j), (y_k)$ are bases for U, X, Y, resp. In the case that X, Y are one-dimensional, this reduces to the usual trace of a linear map $f : U \to U$, i.e. the usual trace determines a function $Tr_U : Hom(U,U) \to Hom(I,I)$, where $I = k$.

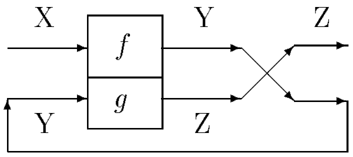
Example 6.4 More generally, any compact category has a canonical trace $Tr^U_{X,Y}(f) = X \cong X \otimes I \xrightarrow{id\otimes\eta} X \otimes U \otimes U^\perp \xrightarrow{f\otimes id} Y \otimes U \otimes U^\perp \xrightarrow{id\otimes ev'} Y \otimes I \cong Y$, where $ev' = ev \circ s$.

Example 6.5 **ω-CPO$_\perp$**: with $\otimes = \times$, $I = \{\perp\}$. In this case the dinatural least-fixed-point combinator $\mathsf{Y}_- : (-)^{(-)} \to (-)$ induces a trace, given as follows (using informal lambda calculus notation): for any $f : X \times U \to Y \times U$, $Tr^U_{X,Y}(f)(x) = f_1(x, \mathsf{Y}_U(\lambda u. f_2(x,u)))$, where $f_1 = \pi_1 \circ f : X \times U \to Y$, $f_2 = \pi_2 \circ f : X \times U \to U$. Hence $Tr^U_{X,Y}(f)(x) = f_1(x, u')$, where u' is the smallest element of U such that $f_2(x,u') = u'$. A generalization of this idea to *traced cartesian categories* is in [MHas97] and mentioned in Remark 6.16 in the next Section.

Unfortunately, these examples do not really illustrate the notion of feedback as data flow: the movement of tokens through a network. More natural examples of traced monoidal categories in the next section, given by partially additive and similar iterative categories, more fully illustrate this aspect.

Example 6.6 *Bicategories of Processes:* The paper of Katis, Sabadini, and Walters [KSW95] develops a general theory of processes with feedback circuits in symmetric monoidal bicategories. They prove their bicategories $Circ(\mathcal{C})$ have a parametrized trace operator. A small difference with the above treatment is that their feedback is given by a family of *partially-defined* functors $fb^U_{X,Y} : Circ(\mathcal{C})(X \otimes U, Y \otimes U) \longrightarrow Circ(\mathcal{C})(X,Y)$.

Remark 6.7 The paper [ABP97] develops a general theory of traced ideals in tensored $*$-categories. The category $\mathcal{HILB}$, the tensored $*$-category of Hilbert spaces and bounded linear maps, illustrates the difficulty. In passing from the finite dimensional case (cf. Example 6.3 above) to the infinite dimensional one, not all endomorphisms have a trace; for example, the identity on an infinite dimensional space. However Tr_U may be defined on *traced ideals* of maps, and this extends to parametrized traces. See [ABP97] for many examples.



$$= \quad X \xrightarrow{f} Y \xrightarrow{g} Z$$

Figure 10: Generalized Yanking

An amusing folklore about traced monoidal categories is that general composition is actually definable using traces of simple compositions:

Proposition 6.8 (Generalized Yanking) *Let $\mathcal{C}$ be a traced symmetric monoidal category, with arrows $f : X \to Y$ and $g : Y \to Z$. Then $g \circ f = Tr^{Y}_{X,Z}(s_{Y,Z} \circ (f \otimes g))$.*

Although a fairly short algebraic proof is possible, the reader may wish to stare at the diagram in Figure 10 , and do a "string-pulling" argument (cf. [JSV96]). Similar calculations are in [KSW95, Mil94]

Definition 6.9 Let $\mathcal{C}$ and $\mathcal{D}$ be traced symmetric monoidal categories. A strong monoidal functor $F : \mathcal{C} \longrightarrow \mathcal{D}$ is *traced* if it is symmetric and satisfies

$$Tr^{FU}_{FA,FB}(\phi^{-1}_{B,U}(Ff)\phi_{A,U}) \quad = \quad F(Tr^{U}_{A,B}(f))$$

where $A \otimes U \xrightarrow{f} B \otimes U$ and $FA \otimes FU \xrightarrow{\phi_{A,U}} F(A \otimes U) \xrightarrow{Ff} F(B \otimes U) \xrightarrow{\phi^{-1}_{B,U}} FB \otimes FU$. In the case of *strict* monoidal functors, they are traced if they preserve the trace on the nose.

We define **TraMon** and $\mathbf{TraMon}_{st}$ to be the 2-categories whose 0-cells are traced monoidal categories (resp. strict traced monoidal categories) , whose 1-cells are traced monoidal functors (resp. strict traced monoidal functors), and whose 2-cells are monoidal natural transformations.

6.2 Partially Additive Categories

We shall be interested in special kinds of traced monoidal categories: those whose homsets are enriched with certain partially-defined infinite sums, which permits canonical calculation of iteration and traces (see formulas 17 and 18 below). A useful example is Manes and Arbib's *partially additive categories*, which first arose in their categorical analysis of iterative and flowchart schema [MA86]). Categories with similar additive structure on the hom-sets had already been considered in the 1950's by Kuroš [Ku63] with regards to categorical Krull-Schmidt-Remak theorems.

Definition 6.10 A *partially additive monoid* is a pair (M, Σ), where M is a nonempty set and Σ is a partial function which maps countable families in M to elements of M (we say that $(x_i | i \in I)$ is *summable* if $\Sigma(x_i | i \in I)$ is defined)[5] subject to the following:

1. *Partition-Associativity Axiom.* If $(x_i | i \in I)$ is a countable family and if $(I_j \,|\, j \in J)$ is a (countable) partition of I , then $(x_i \,|\, i \in I)$ is summable if and only if $(x_i \,|\, i \in I_j)$ is summable for every $j \in J$ and $(\Sigma(x_i \,|\, i \in I_j) \,|\, j \in J)$ is summable. In that case,

$$\Sigma(x_i \,|\, i \in I) = \Sigma(\Sigma(x_i \,|\, i \in I_j) \,|\, j \in J)$$

[5]We sometimes abbreviate $\Sigma(x_i \,|\, i \in I)$ by $\Sigma_{i \in I} x_i$. Throughout, "countable" means finite or denumerable. All index sets are countable. A *partition* $\{I_j \,|\, j \in J\}$ of I satisfies: $I_j \subseteq I$, $I_i \cap I_j = \emptyset$ if $i \neq j$, and $\cup\{I_j \,|\, j \in J\} = I$. But we also allow $I_j = \emptyset$ for countably many j.

2. *Unary Sum Axiom.* Any family $(x_i \mid i \in I)$ in which I is a singleton is summable and $\Sigma(x_i \mid i \in I) = x_j$ if $I = \{j\}$.

3.* *Limit Axiom.* If $(x_i \mid i \in I)$ is a countable family and if $(x_i \mid i \in F)$ is summable for every finite subset F of I then $(x_i \mid i \in I)$ is summable.

We observe the following facts about partially additive monoids:

(i) Axioms 1 and 2 imply that the empty family is summable. We denote $\Sigma(x_i \mid i \in \emptyset)$ by 0, which is an additive identity for summation.

(ii) Axiom 1 implies the obvious equations of commutativity and associativity for the sum (when defined).

(iii) Although Manes and Arbib use the Limit Axiom to prove existence of Elgot-style iteration (see below), Kuroš did not have it. And for many aspects of the theory below, it is not needed.

Definition 6.11 The category of *partially additive monoids*, **PAMon**, is defined as follows. Its objects are partially additive monoids (M, Σ). Its arrows $(M, \Sigma_M) \xrightarrow{f} (N, \Sigma_N)$ are maps from M to N which preserve the sum, in the sense that: $f(\Sigma_M(x_i \mid i \in I)) = \Sigma_N(f(x_i) \mid i \in I)$ for all summable families $(x_i \mid i \in I)$ in M. Composition and identities are inherited from **Sets**.

A **PAMon**-category $\mathcal{C}$ is a category enriched in **PAMon**. This means the hom-sets carry a **PAMon**-structure, compatible with composition. In particular, in each homset $Hom_{\mathcal{C}}(X, Y)$ there is a zero morphism $0_{XY} : X \longrightarrow Y$, the sum of the empty family.

Remark 6.12 In a **PAMon**-category $\mathcal{C}$

1. The family of zero morphisms $\{0_{XY}\}_{X,Y \in \mathcal{C}}$ satisfies: $g0_{WZ} = 0_{WY} = 0_{XY}f$ for any $f : W \to X$ and $g : Z \to Y$.

2. If $\Sigma_{i \in I} f_i = 0_{XY}$ then all summands $f_i = 0_{XY}$ in $Hom_{\mathcal{C}}(X, Y)$.

Definition 6.13 Let $\mathcal{C}$ be a **PAMon**-category with countable coproducts $\bigoplus_{i \in I} X_i$. For any $j \in I$ we define *quasi projections* $PR_j : \bigoplus_{i \in I} X_i \longrightarrow X_j$ as follows:

$$PR_j in_k = \begin{cases} id_{X_j} & \text{if } k = j \\ 0_{X_k X_j} & \text{else} \end{cases}$$

Definition 6.14 A *partially additive category* (pac) $\mathcal{C}$ is a **PAMon**-category with countable coproducts which satisfies the following axioms:

1. *Compatible Sum Axiom*: If $(f_i \mid i \in I) \in \mathcal{C}(X, Y)$ is a countable family and there exists $f : X \longrightarrow I.Y$ such that $PR_i f = f_i$ (we say the f_i are *compatible*), then $\sum f_i$ exists.

2. *Untying Axiom*: If $f + g : X \longrightarrow Y$ exists then so does $in_1 f + in_2 g : X \longrightarrow Y + Y$.

The following facts about partially additive categories follow from Manes and Arbib [MA86]:

- *Matrix Representation of maps*: For any map $f : \bigoplus_{i \in I} X_i \to \bigoplus_{j \in J} Y_j$ there is a unique family $\{f_{ij} : X_i \to Y_j\}_{i \in I j \in J}$ with $f = \sum_{i \in I j \in J} in_j f_{ij} PR_i$ and $PR_j f in_i = f_{ij}$. *Notation:* we write $f_{X_i Y_j}$ for f_{ij}.

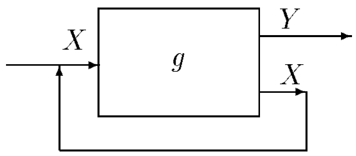


Figure 11: Elgot Dagger $g^\dagger$

- *Elgot Iteration* Given $X \xrightarrow{g} Y+X$, there exists $X \xrightarrow{g^\dagger} Y$ where $g^\dagger = \sum_{n=0}^{\infty} g_{XY} g_{XX}^n$, satisfying the *fixed-point identity*
  (17) $$[1_Y, g^\dagger]g = g^\dagger.$$
  This corresponds to the flowchart scheme in Figure 11. The proof of Elgot iteration ([MA86],p.83) uses the Limit Axiom.

Proposition 6.15 *A partially additive category $\mathcal{C}$ is traced monoidal with $\otimes$ = coproduct and if $f : X \oplus U \longrightarrow Y \oplus U$,*

(18) $$Tr^U_{X,Y}(f) = [1_Y, f_2^\dagger]f_1 = f_{XY} + \sum_{n=0}^{\infty} f_{UY} f_{UU}^n f_{XU}$$

where $f = [f_1, f_2]$ with $f_1 : X \longrightarrow Y \oplus U$, $f_2 : U \longrightarrow Y \oplus U$.

Formula (18) corresponds to the data flow interpretation of trace-as-feedback in Figure 9 : see the examples below. We should also remark that Formula (18) corresponds closely to Girard's *Execution Formula* [Abr96] and is related to a construction of Geometry of Interaction categories in the next section.

Remark 6.16 Conversely, a traced monoidal category where $\otimes$ is coproduct has an Elgot iterator $g^\dagger = Tr^X_{X,Y}([g, g])$, where $g : X \to Y + X$. An axiomatization of the *opposite* of such categories, which correspond to categories with a parametrized Y combinator, is considered in Hasegawa [MHas97]. More generally, Hasegawa considers traced monoidal categories built over cartesian categories and it is shown how various typed lambda calculi with *cyclic sharing* are Sound and Complete for such categorical models.

Finally, we should mention the general notion of *Iteration Theories.* These general categorical theories of feedback and iteration, their axiomatization and equational logics have been studied in detail by S.L. Bloom and Z. Ésik in their book [BE93]. A more recent 2-categorical study of iteration theories is in [BELM].

We shall now give a few important examples of pac's:

Example 6.17 $\mathbf{Rel}_+$, the category of sets and relations. Objects are sets and maps are binary relations. Composition means relational composition. The identity is the identity relation, and the zero morphism 0_{XY} is the empty set $\emptyset \subseteq X \times Y$. Coproducts $\bigoplus_{i\in I} X_i$ are as in **Set**, i.e. disjoint union. *All* countable families are summable where $\Sigma_{i\in I}(R_i) = \cup_{i\in I} R_i$. Finally, let R^* be the reflexive, transitive closure of a relation R. Suppose
$R : X + U \twoheadrightarrow Y + U$. Then formula (18) becomes:

(19) $$\begin{aligned} Tr^U_{X,Y}(R) &= R_{XY} \cup \bigcup_{n\geq 0} R_{UY} \circ R_{UU}^n \circ R_{XU} \\ &= R_{XY} \cup R_{UY} \circ R_{UU}^* \circ R_{XU}. \end{aligned}$$

Example 6.18 Pfn , the category of sets and partial functions. The objects are sets, the maps are partial functions. Composition means the usual composition of partial functions. The zero map 0_{XY} is the empty partial function. A family $\{f_i \mid i \in I\}$ is said to be summable iff $\forall i, j \in I, i \neq j$, $Dom(f_i) \cap Dom(f_j) = \emptyset$. $\Sigma_{i\in I} f_i$ is the partial function with domain $\cup_i Dom(f_i)$ and where

$$(\Sigma_{i\in I} f_i)(x) = \begin{cases} f_j(x) & \text{if } x \in Dom(f_j) \\ \text{undefined} & \text{else} \end{cases}$$

The following example comes from Giry [Giry] (inspired from Lawvere) and is mentioned in [Abr96]. The fact that this is a pac follows from work of P. Panangaden and E. Haghverdi.

Example 6.19 SRel, the category of Stochastic Relations. Objects are measurable spaces (X, Σ_X) where X is a set and Σ_X is a σ-algebra of subsets of X. An arrow $f : (X, \Sigma_X) \longrightarrow (Y, \Sigma_Y)$ is a transition probability, i.e., $f : X \times \Sigma_Y \longrightarrow [0,1]$ such that $f(\cdot, B) : X \longrightarrow [0,1]$ is a measurable function for fixed $B \in \Sigma_Y$ and $f(x, \cdot) : \Sigma_Y \longrightarrow [0,1]$ is a subprobability measure (i.e., a σ-additive set function satisfying $f(x, \emptyset) = 0$ and $f(x, Y) \leq 1$). The identity morphism $id_X : (X, \Sigma_X) \longrightarrow (X, \Sigma_X)$ is a map $id_X : X \times \Sigma_X \longrightarrow [0,1]$, with $id_X(x, A) = \delta(x, A)$, where for A fixed, $\delta(x, A)$ is the characteristic function of A and for x fixed, $\delta(x, A)$ is the Dirac distribution.

Composition is defined as follows: given $f : (X, \Sigma_X) \longrightarrow (Y, \Sigma_Y)$ and $g : (Y, \Sigma_Y) \longrightarrow (Z, \Sigma_Z)$, $g \circ f : (X, \Sigma_X) \longrightarrow (Z, \Sigma_Z)$ is $g \circ f(x, C) = \int_Y g(y, C) d\{f(x, \cdot)\}$, where the notation $d\{f(x, \cdot)\}$ means that we are fixing x and using $f(x, \cdot)$ as the measure for the integration, the function being integrated is the measurable function $g(\cdot, C)$.

Given (X, Σ_X) and (Y, Σ_Y), the *zero morphism* $0_{XY} : (X, \Sigma_X) \longrightarrow (Y, \Sigma_Y)$, is given by $0_{XY}(x, B) = 0$ for all $x \in X$ and $B \in \Sigma_Y$.

The partially additive structure on the homsets of **SRel** is as follows: we say an I-indexed family of morphisms $\{f_i | i \in I\}$ is *summable* if for all $x \in X$ we have $\sum_{i \in I} f_i(x, Y) \leq 1$. Since we are dealing with bounded, positive measures it is easy to verify that the sum so defined is a subprobability measure. Note that we would have only trivial additive structure (only singleton families summable) if we had used probability distributions rather than subprobability distributions.

Finally, let $\{X_i | i \in I\}$ be a countable family of objects. We define the coproduct $\bigoplus_{i\in I} X_i$ as follows. We take the disjoint union of the sets X_i, equipped with the evident σ-algebra. Thus a measurable subset will look like the disjoint union of measurable subsets of each of the X_i, say $\uplus A_i$ (of course some of the A_i may be empty, and a point will be a pair (x, i) where $i \in I$ and $x \in X_i$). The canonical injections $in_j : X_j \longrightarrow \bigoplus_{i\in I} X_i$ are $in_j(x, \uplus A_i) = \delta(x, A_j)$. Given Y and $\forall i \in I$, arrows $h_i : X_i \longrightarrow Y$, we obtain the mediating morphism $h : \bigoplus_{i\in I} X_i \longrightarrow Y$ by the formula $h((x, j), B) = h_j(x, B)$. The verifications are all routine.

The next example, while not a pac, is essentially similar.

Example 6.20 *Pinj*, the category of sets and injective partial functions. This is a fundamental example that arises in Girard's Geometry of Interaction program. Although this category is traced monoidal, with an iterative trace formula given in Abramsky [Abr96], it does not have coproducts. However its pac-like aspects may be captured in a Kuroš-style presentation via a generalization of partially additive categories, in which countable coproducts are replaced by countable tensors, and in which suitable axioms guarantee (analogously to pacs): a matrix representation of maps $\bigotimes_{i\in I} X_i \to \bigotimes_{j\in J} Y_j$ and a trace formula as in (18).

6.3 GoI Categories

Girard's Geometry of Interaction (GoI) program introduces some profound new twists into computation theory. In particular, the idea that proofs are like dynamical systems, interacting locally.

The dynamics of information flow in composition, via cut-elimination, is then related to tracing out paths in certain algebraic structures (Girard originally used operator algebras but the results can be expressed without them [Gi95b]). The connection of Girard's functional analytic methods in GoI with lambda calculus and proof nets is further explored in [DR95, MaRe91].

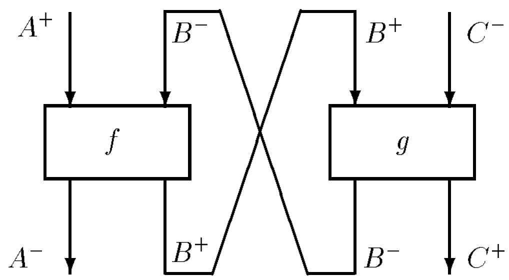
Starting with a traced monoidal category $\mathcal{C}$, we now describe a compact closed category $\mathcal{G}(\mathcal{C})$ (called $Int(\mathcal{C})$ in [JSV96]) which captures in abstract form many of the features of Girard's Geometry of Interaction program, as well as the general ideas behind game semantics. We follow the evocative treatment in Abramsky [Abr96]. The idea is to create a category whose composition is given by an iterative feedback formula, using the trace.

Definition 6.21 (The *Geometry of Interaction* construction) Given a traced monoidal category $\mathcal{C}$ we define a compact closed category, $\mathcal{G}(\mathcal{C})$, as follows [JSV96, Abr96]:

- Objects: Pairs of objects (A^+, A^-) where A^+ and A^- are objects of $\mathcal{C}$.
- Arrows: An arrow $f : (A^+, A^-) \longrightarrow (B^+, B^-)$ in $\mathcal{G}(\mathcal{C})$ is an arrow $f : A^+ \otimes B^- \longrightarrow A^- \otimes B^+$ in $\mathcal{C}$.
- Identity: $1_{(A^+,A^-)} = \sigma_{A^+,A^-}$.
- Composition: given by symmetric feedback. Arrows $f : (A^+, A^-) \longrightarrow (B^+, B^-)$ and $g : (B^+, B^-) \longrightarrow (C^+, C^-)$ have composite $g \circ f : (A^+, A^-) \longrightarrow (C^+, C^-)$ given by:

$$g \circ f = Tr^{B^- \otimes B^+}_{A^+ \otimes C^-, A^- \otimes C^+}(\beta(f \otimes g)\alpha)$$

  where $\alpha = (1_{A^+} \otimes 1_{B^-} \otimes \sigma_{C^-,B^+})(1_{A^+} \otimes \sigma_{C^-,B^-} \otimes 1_{B^+})$ and $\beta = (1_{A^-} \otimes 1_{C^+} \otimes \sigma_{B^+,B^-})(1_{A^-} \otimes \sigma_{B^+,C^+} \otimes 1_{B^-})(1_{A^-} \otimes 1_{B^+} \otimes \sigma_{B^-,C^+})$. An informal picture displaying $g \circ f$ is given below.



- Tensor: $(A^+, A^-) \otimes (B^+, B^-) = (A^+ \otimes B^+, A^- \otimes B^-)$ and for $(A^+, A^-) \longrightarrow (B^+, B^-)$ and $g : (C^+, C^-) \longrightarrow (D^+, D^-)$, $f \otimes g = (1_{A^-} \otimes \sigma_{B^+,C^-} \otimes 1_{D^+})(f \otimes g)(1_{A^+} \otimes \sigma_{C^+,B^-} \otimes 1_{D^-})$
- Unit: (I, I).
- Duality: The dual of (A^+, A^-) is given by $(A^+, A^-)^\perp = (A^-, A^+)$ where the unit $\eta : (I, I) \longrightarrow (A^+, A^-) \otimes (A^+, A^-)^\perp =_{def} \sigma_{A^-,A^+}$ and counit $\epsilon : (A^+, A^-)^\perp \otimes (A^+, A^-) \longrightarrow (I, I) =_{def} \sigma_{A^-,A^+}$.
- Internal Homs: As usual, $(A^+, A^-) \multimap (B^+, B^-) = (A^+, A^-)^\perp \otimes (B^+, B^-) = (A^- \otimes B^+, A^+ \otimes B^-)$.

Remark 6.22 We have used a specific definition for α and β above; however, any other permutations $A^+ \otimes C^- \otimes B^- \otimes B^+ \xrightarrow{\cong} A^+ \otimes B^- \otimes B^+ \otimes C^-$ and $A^- \otimes B^+ \otimes B^- \otimes C^+ \xrightarrow{\cong} A^- \otimes C^+ \otimes B^- \otimes B^+$ for α and β respectively will yield the same result for $g \circ f$ due to coherence.

Translating the work of [JSV96] in our setting we obtain that $\mathcal{G}(\mathcal{C})$ is a kind of "free compact closure" of $\mathcal{C}$:

Proposition 6.23 *Let $\mathcal{C}$ be a traced symmetric monoidal category*

- *$\mathcal{G}(\mathcal{C})$ defined as in Definition 6.21 is a compact closed category. Moreover, $F_{\mathcal{C}} : \mathcal{C} \longrightarrow \mathcal{G}(\mathcal{C})$ defined by $F_{\mathcal{C}}(A) = (A, I)$ and $F_{\mathcal{C}}(f) = f$ is a full and faithful embedding.*
- *The inclusion of 2-categories $CompCl \hookrightarrow TraMon$ has a left biadjoint with unit having component at $\mathcal{C}$ given by $F_{\mathcal{C}}$.*

Following Abramsky [Abr96], we interpret the objects of $\mathcal{G}(\mathcal{C})$ in a game-theoretic manner: A^+ is the type of "moves by Player (the System)" and A^- is the type of "moves by Opponent (the Environment)". The composition of morphisms in $\mathcal{G}(\mathcal{C})$ is the key to Girard's Execution formula, especially for pac-like traces. In [Abr96] it is pointed out that $\mathcal{G}(Pinj)$ is essentially the original Girard formalism, while $\mathcal{G}(\omega$-CPO) is the data-flow model of GoI given in [AJ94a]. [6]

7 Literature Notes

In the above we have merely touched on the large and varied literature. The journals *Mathematical Structures in Computer Science* (Camb. Univ. Press) and *Theoretical Computer Science* (Elsevier) are standard venues for categorical computer science. Two recent graduate texts emphasizing categorical aspects are J. Mitchell [Mit96] and R. Amadio and P.-L. Curien [AC98]. Mitchell's book has an encyclopedic coverage of the major areas and recent results in programming language theory. The Amadio-Curien book covers many recent topics in domain theory and lambda calculi, including full abstraction results, foundations of linear logic, and sequentiality.

We regret that there are many important topics in categorical computer science which we barely mentioned. We particularly recommend the compendia [PD97, FJP, AGM]. Let us give a few pointers with sample papers:

- *Operational and Denotational Semantics:* See the surveys in the Handbook[AGM]. The classical paper on solutions of domain equations is [SP82]. For some recent directions in domain theory, see [FiPl96, ReSt97]. For recent categorical aspects of Operational Semantics, see [Pi97, TP96] . Higher-dimensional category theory has also generated considerable theoretical interest (e.g. [Ba97, HMP98]). Coalgebraic and coinductive methods are a fundamental technique and have considerable influence (e.g. see [AbJu94, Pi96a, Mul91, CSp91, JR, Mil89]).
- *Fibrations and Indexed Category Models* This important area arising from categorical logic is fundamental in treating dependent types, System $\mathcal{F}, \mathcal{F}_\omega, \cdots$ models, and general variable-binding (quantifier-like) operations, for example "hiding" in certain process calculi. For fibred category models of dependent type-theories, see the survey by M. Hofmann [H97a] (cf. also [PowTh97, HJ95, Pi9?, See87]). Indexed category models for *Concurrent Constraint Logic Programming* are given in [PSSS, MPSS95] (see also [FRS98] for connections of this latter paradigm to LL).

[6] Added in proof: recent progress on these matters has been achieved in the PhD thesis of Esfan Haghverdi, Dept. of Mathematics, U. Ottawa, Feb. 2000, and in a paper, to appear in the Lambek Festschrift, *Math. Structures in Computer Science*. See also *http://aix1.uottawa.ca/~ehaghver*

- *Computational Monads*: E. Moggi greatly influenced programming language semantics and associated logics using the categorists' notion of monads and comonads [Mac71]. Moggi's approach permits a modular treatment of such important programming features as: exceptions, side-effects, non-determinism, resumptions, dynamic allocation, etc, as well as their associated logics [Mo97, Mo91]. Practical uses of monads in functional programming languages are discussed in P. Wadler ([W92]). More recently, E. Manes ([M98]) showed how to use monads to implement collection classes in Object-Oriented languages. Alternative category-theoretic perspectives on Moggi's work are in Power and Robinson's [PowRob97].

- *Concurrency Theory and Categorical Bisimulation:* This large and important area is surveyed in Winskel and Nielsen[WN97]. In particular, the fundamental notion of bisimulation via open maps, is introduced in Joyal, Nielsen, and Winskel [JNW]. Presheaf models for Milner's π-calculus ([Mil93a]) and other concurrent calculi are in [CaWi, CSW]. For categorical work on Milner's recent Action Calculi, see [GaHa, Mil93b, Mil94, P96].

- *Complexity Theory:* Characterizing feasible (e.g. polynomial-time) computation is a major area of theoretical computer science (e.g. [Cob64, Cook75]). Typed lambda calculi for feasible higher-order computation have recently been the subjects of intense work, e.g. [CoKa, CU93]. Versions of linear logic have been developed to analyze the fine structure of feasible computation ([GSS92, Gi97]). Although there are some known models ([KOS97]), general categorical treatments for these versions of LL are not yet known. Recently, M. Hofmann (e.g. [H97a]) has analyzed the work of Cook and Urquhart [CU93] as well as giving higher-order extensions of work of Bellantoni and Cook , using presheaf and sheaf categories.

Three volumes [MR97, FJP, GS89] are conferences specializing in applications of categories in computer science (note [MR97] is the 7th Biennial such meeting). Similarly, see the biennial meetings of MFPS (Mathematical Foundations of Programming Semantics)–published in either Springer Lecture Notes in Computer Science or the journal *Theoretical Computer Science.* There is currently an electronic website of categorical logic and computer science, HYPATIA.

Other books covering categorical aspects of computer science and/or some of the topics covered here include[AL91, BW95, Cu93, DiCo95, Gun92, MA86, Tay98]. In categorical logic and proof theory, we should mention our own book with J. Lambek [LS86] which became popular in theoretical computer science. The category theory book of Freyd and Scedrov [FrSc] is a source book for Representation Theorems and categories of relations.

References

[ACC93] M. Abadi, L. Cardelli, and P.-L. Curien, Formal parametric polymorphism. *Theoretical Comp. Science* **121** , 1993, pp. 9-58.

[Abr93] S. Abramsky, Computational Interpretations of Linear Logic, *Theoretical Comp. Science* **111** , 1993, pp. 3-57.

[AbJu94] S. Abramsky and A. Jung, Domain Theory, in [AGM], pp. 1–168.

[Abr96] S. Abramsky, Retracing some paths in process algebra, in CONCUR '96, U. Montanari and V. Sassone, eds., *Springer Lecture Notes in Computer Science* **1119**, 1996, pp. 1–17.

[Abr97] S. Abramsky. Semantics of Interaction: An Introduction to Game Semantics, in [PD97], pp. 1-31.

[ABP97] S. Abramsky, R. Blute, and P. Panangaden, Nuclear and trace ideals in tensored $*$-categories, preprint, 1997. (To appear in *Journal of Pure and Applied Algebra.*)

[AGM] S. Abramsky, D. Gabbay, T. Maibaum, eds. *Handbook of Logic in Computer Science*, Vol 3, Oxford, 1994.

[AGN] S. Abramsky, S. Gay, R. Nagarajan, A Type-Theoretic Approach to Deadlock-Freedom of Asynchronous Systems, in *Theoretical Aspects of Computer Software, TACS'97*, M. Abadi, T. Ito Eds. *Springer Lecture Notes in Computer Science* **1281**, 1997.

[AHMc98] S. Abramsky, K. Honda, G. McCusker. A fully abstract game semantics for general references, *13th Annual IEEE Symp. on Logic in Computer Science (LICS)* , IEEE Press, 1998, pp. 334–344.

[AJ94a] S. Abramsky and R. Jagadeesan, New Foundations for the Geometry of Interaction, *Information and Computation* **111**, Number 1, 1994, pp. 53-119. Preliminary version, in *7th Annual IEEE Symp. on Logic in Computer Science (LICS)* , 1992, IEEE Publications, pp. 211-222.

[AJ94b] S. Abramsky and R. Jagadeesan, Games and Full Completeness Theorem for Multiplicative Linear Logic, *J. Symbolic Logic*, Vol.59, No.2, 1994, pp. 543-574.

[AMc98] S. Abramsky and G. McCusker, Full Abstraction for Idealized Algol with Passive Expressions. To appear in *Theoretical Computer Science.*

[Abru91] V.M. Abrusci, Phase Semantics and Sequent Calculus for Pure Noncommutative Classical Linear Propositional Logic, *J. Symbolic Logic* Vol. 56, 1991, pp. 1403–1456.

[Ali95] M. Alimohamed, A characterization of lambda definability in categorical models of implicit polymorphism, *Theoretical Comp. Science* **146** , 1995, pp. 5-23.

[AHS95] T. Altenkirch, M. Hofmann, T. Streicher, Categorical reconstruction of a reduction-free normalisation proof, *Proc. CTCS '95 Springer Lecture Notes in Computer Science* **953**, pp. 182–199.

[AHS96] T. Altenkirch, M. Hofmann, T. Streicher, Reduction- free normalisation for a polymorphic system, *Proc.of the 11th Annual IEEE Symposium of Logic in Computer Science*, 1996, pp. 98-106.

[AC98] R. M. Amadio, P-L. Curien, *Selected Domains and Lambda Calculi*, Camb. Univ. Press, 1998.

[Asp] A. Asperti, Linear Logic, Comonads, and Optimal Reductions, to appear in *Fund. Informatica.*

[AL91] A. Asperti and G. Longo. *Categories, Types, and Structures.* MIT Press, 1991. (Currently out of print but publicly available on the Web).

[Ba97] J.C. Baez, An Introduction to n-Categories, in *Category Theory and Computer Science, CTCS'97*, E. Moggi and G. Rosolini Eds., *Springer Lecture Notes in Computer Science* **1290**, 1997.

[BDER97] P. Baillot, V. Danos, T. Ehrhard, L. Regnier, Believe it or not: AJM games model is a model of classical linear logic, in *12th Annual IEEE Symp. on Logic in Computer Science (LICS)* , 1997, IEEE Press.

[B72] E.S. Bainbridge. *A Unified Minimal Realization Theory , with Duality.* Dissertation, Department of Computer & Communication Sciences, University of Michigan, 1972.

[B75] E.S. Bainbridge, Addressed Machines and Duality, in *Category Theory Applied to Computation and Control*, E. Manes, Ed., Springer Lecture Notes in Computer Science **25**, 1975, pp. 93–98.

[B76] E.S. Bainbridge, Feedback and Generalized Logic, *Information and Control* **31**, 1976, pp.75–96.

[B77] E.S. Bainbridge, The Fundamental Duality of System Theory, in *Systems: Approaches, Theories, Applications*, W.E. Hartnett Ed., D. Reidel Publishing Company, 1977, pp. 45–61.

[BFSS90] E.S. Bainbridge, P. Freyd , A. Scedrov, and P. J. Scott, Functorial Polymorphism, *Theoretical Computer Science* **70** (1990), pp. 35–64.

[Bar84] H. P. Barendregt. *The Lambda Calculus*, Studies in Logic, Vol. 103, North- Holland, 1984.

[Bar92] H. P. Barendregt, Lambda Calculus with Types,*Handbook of Logic in Computer Science*, Vol. 2, ed. by S. Abramsky, D. Gabbay, T. Maibaum. Oxford U. Press, 1992.

[Barr79] M. Barr, $*$-*Autonomous Categories*, *Springer Lecture Notes in Mathematics* **752** , 1979.

[Barr91] M. Barr, $*$- Autonomous categories and linear logic, *Math. Structures in Compter Science* **1** , 1991, pp. 159–178.

[Barr95] M. Barr, Non-symmetric $*$- autonomous categories, *Theoretical Comp. Science* **139 139**, 1995, pp. 115–130.

[Barr96] M. Barr, Appendix to [Blu96], 1994.

[BW95] M. Barr, C. Wells. *Category Theory for Computing Science*, second edition, Prentice-Hall, 1995.

[Bee85] M. Beeson. *Foundations of Constructive Mathematics*, Springer-Verlag, Berlin, Heidelberg, New York, 1985.

[BAC95] R. Bellucci, M. Abadi, P.-L. Curien, A model for Formal Parametric Polymorphism: a PER Interpretation for System $\mathcal{R}$, in *Springer Lecture Notes in Computer Science* **902**, TLCA'95 , 1995, pp. 32-46.

[Ben] D.B. Benson, Bialgebras: Some Foundations for Distributed and Concurrent Computation, *Fundamenta Informaticae* **12**, 1989, pp. 427-486.

[BBPH] Benton B.N., G. Bierman, V. de Paiva, M. Hyland, Term assignment for intuitionistic linear logic, M. Bezem and J. F. Groote, eds, *Proceedings of the International Conference on Typed Lambda Calculi and Applications*, *Springer Lecture Notes in Computer Science* **664**, 1992, pp. 75–90.

[BS91] U. Berger and H. Schwichtenberg, An inverse to the evaluation functional for typed λ-calculus,*Proc. of the 6th Annual IEEE Symposium of Logic in ComputerScience*, 1991, pp. 203-211.

[BD95] I. Beylin and P. Dybjer, Extracting a proof of coherence for monoidal categories from a proof of normalization for monoids, in: *Types for Proofs and Programs, TYPES'95*, S. Berardi, M. Coppo Eds., *Springer Lecture Notes in Computer Science* **1158**, 1995.

[Bie95] G.M. Bierman, What is a categorical model of intuitionistic linear logic?. In: TLCA'95, M. Dezani-Ciancaglini, G. Plotkin, eds, 1995, pp. 78–93.

[Bla92] A. Blass. A game semantics for linear logic. *Annals of Pure and Applied Logic*, **56**, 1992 , pp. 183-220.

[BE93] S.L. Bloom, Z. Ésik. *Iteration theories; the equational logic of iterative processes,* Berlin, Springer-Verlag, 1993.

[BELM] S.L. Bloom, Z. Ésik, A. Labella, E. Manes, Iteration 2-theories: Extended Abstract, in *Algebraic Methodology and Software Technology, AMAST'97*, M. Johnson Ed., *Springer Lecture Notes in Computer Science* **1349**, 1997, pp. 30-44.

[Blu93] R. Blute, Linear Logic, Coherence and Dinaturality, *Theoretical Computer Science* 115, 1993 , pp. 3- 41.

[Blu96] R. Blute, Hopf Algebras and Linear Logic, *Math. Structures in Compter Science* **6** , 1996, pp. 189-217.

[BCS96] Blute, R.F., J.R.B. Cockett, and R.A.G. Seely, **!** and **?** : Storage as tensorial strength, *Math. Structures in Compter Science* **6** , 1996, pp. 313–351.

[BCST96] R.F. Blute, J.R.B. Cockett, R.A.G. Seely, and T.H. Trimble Natural deduction and coherence for weakly distributive categories,*J. Pure and Applied Algebra* **113**, 1996, pp. 229–296.

[BCS97] Blute, R.F., J.R.B. Cockett, and R.A.G. Seely, Categories for computation in context and unified logic,*J. Pure and Applied Algebra* **116**, 1997, pp. 49–98.

[BCS98] R. Blute, J.R.B. Cockett, and R.A.G. Seely, Feedback for linearly dictributive categories: traces and fixpoints, preprint, McGill University, 1998.

[BS96] R.F. Blute, P.J. Scott, Linear Läuchli Semantics, *Annals of Pure and Applied Logic* **77**, 1996, pp.101-142.

[BS96b] R.F. Blute, P.J. Scott, A Noncommutative Full Completeness Theorem (ExtendedAbstract), *Electronic Notes in Theoretical Computer Science 3* 1996, Elsevier Science B.V.

[BS98] R. Blute and P. J. Scott. The Shuffle Hopf Algebra and Noncommutative Full Completeness, *J. Symbolic Logic*, to appear.

[Bor94] F. Borceux. *Handbook of Categorical Algebra*, Cambridge University Press, 1994.

[Brea-T,G] V. Breazu-Tannen and J. Gallier, Polymorphic rewriting conserves algebraic normalization, *Theoretical Comp. Science* **83** .

[CPS88] a. Carboni, P. Freyd, A. Scedrov, A categorical approach to realizability and polymorphic types, in *Springer Lecture Notes in Computer Science* **298** (Proc. 3rd MFPS), pp. 23-42.

[C86] Cartmell, J. Generalized Algebraic Theories and Contextual Categories, *Annals of Pure and Applied Logic*, **32**, 1986, pp. 209–243.

[CSW] G.L. Cattani, I. Stark, G. Winskel, Presheaf Models for the π-Calculus, in *Category Theory and Computer Science, CTCS'97*, E. Moggi and G. Rosolini Eds., *Springer Lecture Notes in Computer Science* **1290**, 1997.

[CaWi] G.L. Cattani and G. Winskel, Presheaf Models for Concurrency, in *Computer Science Logic, CSL'96*, D. van Dalen and M. Bezem Eds., *Springer Lecture Notes in Computer Science* **1258**, 1996.

[Cob64] A. Cobham, The intrinsic computational difficulty of functions. *Proc. of the 1964 Internations Congress for Logic,Methodology and Philosophy of Science* , Y. Bar- Hillel, ed. North-Holland Publishing Co., Amsterdam, 1964, pp. 24-30

[Co93] J.R.B. Cockett, Introduction to distributive categories. *Math. Structures in Compter Science* **3** , 1993, pp. 277–308.

[CS91] J.R.B. Cockett, R.A.G. Seely, Weakly distributive categories, in *J. Pure and Applied Algebra* **114**, 1997, pp. 133-173. (Preliminary version in [FJP], pp. 45–65.)

[CS96a] J.R.B. Cockett, R.A.G. Seely, Linearly distributive functors, preprint,McGill University, 1996.

[CS96b] J.R.B. Cockett, R.A.G. Seely, Proof theory for full intuitionistic linear logic, bilinear logic,and MIX categories, *Theory and Applications of Categories* **3**, 1997, pp. 85–131.

[CSp91] J.R.B. Cockett, D.L. Spencer, Strong categorical datatypes I, in R.A.G. Seely, ed.,*Category Theory 1991, Montreal*, CMS Conference Proceedings, **13**, 1991, pp. 141–169.

[Cook75] S. A. Cook, Feasibly constructive proofs and the propositional calculus, in Proc. 7th Annual ACM Symp. on Theory of Computing , 1975, pp. 83-97.

[CoKa] S. A. Cook and B. M. Kapron, Characterizations of the Basic Feasible Functionals of FiniteType, in: *Feasible Mathematics*, S. Buss and P. Scott, eds., Birkhauser, pp. 71-96.

[CD97] T. Coquand and P. Dybjer, Intuitionistic Model Constructions and Normalization Proofs , *Math. Structures in Compter Science* **7** , 1997, pp. 75-94.

[Coq90] T. Coquand, On the Analogy Between Propositons and Types, in [Hu90], pp. 399-417.

[CU93] S. A. Cook and A. Urquhart. Functional interpretations of feasibly constructive arithmetic, *Ann.Pure & Applied Logic* **63**, No. 2, 1993, pp. 103-200.

[CoGa] A. Corradini and F. Gadducci, A 2-Categorical Presentation of Term Graph Rewriting, in *Category Theory and Computer Science, CTCS'97*, E. Moggi and G. Rosolini Eds., *Springer Lecture Notes in Computer Science* **1290**, 1997.

[Cr93] R. Crole. *Categories for Types*, Cambridge Mathematical Textbooks, 1993.

[Cu93] D. Čubrić, Results in Categorical Proof Theory, Phd thesis, Dept. of Mathematics, McGill University, 1993.

[CDS97] D. Čubrić, P. Dybjer and P.J.Scott. Normalization and the Yoneda Embedding, *Math. Structures in Compter Science* **8** , No.2, 1997, pp. 153-192.

[Cur93] P-L. Curien. *Categorical combinators, sequential algorithms, and functional programming* , Pitman 1986 (revised edition, Birkhäuser, 1993).

[D90] V. Danos. La logique linéaire appliquée àl'étude de divers processus de normalisation et principalement du λ-calcul, Thèse de doctorat, , U. Paris VII, U.F.R. de Mathématiques, 1990.

[DR95] V. Danos and L. Regnier, Proof-nets and the Hilbert space, in [GLR], pp. 307–328.

[DR89] V. Danos and L. Regnier. The structure of Multiplicatives, *Arch. Math. Logic* (1989) **28**, pp. 181-203.

[DJ] N. Dershowitz and J.- P. Jouannaud, Rewrite Systems, *Handbook of Theoretical Computer Science*, Chapter 15, North-Holland, 1990.

[DiCo95] R. Di Cosmo. *Isomorphism of Types:from λ-calculus to information retrieval and language design*, Birkhäus r, 1995.

[D$^+$93] G. Dowek, et. al. The Coq proof assistant user's guide, version 5.8. Technical Report, INRIA-Rocquencourt, Feb. 1993.

[DuRe94] D. Duval and J-C Reynaud, Sketches and computation I: basic definitions and static evaluation, *Math. Structures in Compter Science* **4** , No. 2, 1994, pp. 185-238.

[Dy95] P. Dybjer, Internal Type Theory, in *Types for Proofs and Programs, TYPES'95*, S. Berardi, M. Coppo Eds., *Springer Lecture Notes in Computer Science* **1158**, 1995.

[EK66] S. Eilenberg and G. M Kelly, A generalization of the functorial calculus, *J. Algebra* **3** (1966), 366- 375.

[EsLa] Z. Ésik, A. Labella, Equational Properties of Iteration in Algebraically Complete Categories, in MFCS'96, *Springer Lecture Notes in Computer Science* **1113**, 1996.

[FRS98] F. Fages, P. Ruet, S. Soliman, Phase Semantics and Verification of Concurrent Constraint Programs, in *13th Annual IEEE Symp. on Logic in Computer Science (LICS)* , 1998, IEEE Press, pp. 141-152.

[FiFrL] S. Finkelstein, P. Freyd, J. Lipton, A new Framework for Declarative Programming, *Th. Comp. Science* (to appear).

[FiPl96] M.P. Fiore, G.D. Plotkin. An Extension of Models of Axiomatic Domain Theory to Models of Synthetic Domain Theory, in CSL'96 , *Springer Lecture Notes in Computer Science* **1258**, 1996.

[Fl96] A. Fleury, Thesis, Thèse de doctorat, U. Paris VII, U.F.R. de Mathématiques, 1996.

[FR94] A. Fleury, C. Rétoré, The *MIX* Rule, *Math. Structures in Compter Science* **4** , p. 273-285 (1994).

[FJP] M. P. Fourman, P. T. Johnstone, and A. M. Pitts, eds. *Applications of Categories in Computer Science*, London Math. Soc. Lecture Notes 177, Camb. U. Press, 1992.

[Fre92] P. Freyd. Remarks on algebraically compact categories, in [FJP], pp. 95-107.

[Fre93] P. Freyd. Structural Polymorphism, *Theoretical Comp. Science* **115** , 1993, pp. 107-129.

[FRRa] P. Freyd, E. Robinson, G. Rosolini. Dinaturality for free, in [FJP], pp. 107-118.

[FRRb] P. Freyd, E. Robinson, G. Rosolini, Functorial Parametricity, manuscript.

[FrSc] P. Freyd and A. Scedrov. *Categories, Allegories*, North-Holland, 1990.

[Frie73] H. Friedman, Equality between functionals, *Logic Colloquium '73*, *Springer Lecture Notes in Computer Science* **453**, 1975, pp. 22-37.

[GaHa] P. Gardner, M. Hasegawa, Types and Models for Higher-Order Action Calculi, in TACS'97, *Springer Lecture Notes in Computer Science* **1281**, 1997.

[Ge85] R. Geroch. *Mathematical Physics,* University of Chicago Press, 1985.

[Gh96] N. Ghani, $\beta\eta$-equality for coproducts. in *TLCA '95 Springer Lecture Notes in Computer Science* **902**, 1995, pp. 171-185.

[Gi71] J-Y. Girard, Une Extension de l'Interprétation de Gödel a l'Analyse, et Son Application a l'Elimination des Coupures dans l'Analyse et la Théorie des Types, in: J. E. Fenstad, ed. *Proc. 2nd Scandanavian Logic Symposium* , North-Holland, 1971, pp. 63-92.

[Gi72] J-Y. Girard, *Interprétation Fonctionnelle et Elimination des Coupures dans l'Arithmétique d'Ordre Supérieur.* Thèse de doctorat d'état, U. Paris VII, 1972.

[Gi86] J-Y. Girard. The System **F** of Variables Types, Fifteen Years Later, *Theoretical Comp. Science* **45** , 1986, pp. 159-192.

[Gi87] J-Y. Girard, Linear Logic, *Theoretical Computer Science*, **50**, 1987, pp. 1-102

[Gi88] J-Y.Girard, Geometry of Interaction I: Interpretation of System F, in: *Logic Colloquium '88* , ed. R. Ferro, et al. North-Holland, 1989, pp. 221-260.

[Gi89] J-Y. Girard, Towards a Geometry of Interaction, in: *Categories in Computer Science and Logic*, ed. by J.W. Gray and A. Scedrov, *Contemp. Math*, **92**, AMS , 1989, pp. 69-108.

[Gi90] J-Y.Girard. Geometry of Interaction 2: Deadlock- free Algorithms. COLOG-88 (P. Martin-Lof, G. Mints, eds.) *Springer Lecture Notes in Computer Science* **417**, 1990, pp. 76-93.

[Gi91] J-Y. Girard, A new constructive logic: classical logic. *Math. Structures in Compter Science* **1** , 1991, pp. 255–296.

[Gi93] J.-Y. Girard, On the unity of logic, *Annals of Pure and Applied Logic* **59**, 1993, pp. 201–217.

[Gi95a] J.-Y. Girard, Linear Logic: its syntax and semantics, in [GLR], pp. 1–42.

[Gi95b] J.-Y. Girard, Geometry of Interaction III: Accommodating the Additives, in [GLR], pp. 329–389.

[Gi96] J.-Y. Girard, Coherent Banach Spaces, preprint, (1996) and lectures delived at Keio University, Linear Logic'96, April 1996

[Gi97] J.-Y. Girard, Light linear logic, preprint, 1995, (revised 1997). URL: ftp://lmd.univ-mrs.fr/pub/girard/ LL.ps.Z

[GLR] J.-Y. Girard, Y. Lafont, L. Regnier, eds. *Advances in Linear Logic*, London Math. Soc. Series 222, Camb. Univ. Press, 1995.

[GLT] J.-Y. Girard, Y. Lafont, P.Taylor. *Proofs and Types*, Cambridge Tracts in Theoretical Computer Science 7, 1989.

[GSS91] J. Y. Girard, A. Scedrov, P. Scott, Normal Forms and Cut-free Proofs as Natural Transformations, in : *Logic From Computer Science*, Mathematical Science Research Institute Publications 21, (1991),pp. 217-241. (Also available by anonymous ftp from:theory.doc.ic.ac uk, in: papers/Scott).

[GSS92] Girard, J.-Y., A. Scedrov, and P.J. Scott Bounded linear logic, *Theoretical Comp. Science* **97** , 1992, pp. 1–66.

[Giry] M. Giry A categorical approach to probability theory, in *Springer Lecture Notes in Mathematics* **915**, *Proc. of a Conference on Categorical Aspects of Topology*, 1981, pp. 68-85.

[GAL92] G. Gonthier, M. Abadi, and J.J. Levy, Linear logic without boxes. In: *7th Annual IEEE Symp. on Logic in Computer Science (LICS)* , 1992, IEEE Publications, pp. 223-234.

[GPS96] R. Gordon, A. Power, and R. Street. *Coherence for tricategories*, Memoirs of the American Mathematical Society, 1996.

[GS89] J. Gray and A. Scedrov, eds. *Categories in Computer Science and Logic*, *Contemp. Math* **92**, AMS, 1989.

[Gun92] C. Gunter. *Semantics of Programming Languages*, MIT Press, 1992.

[Gun94] C. Gunter, The semantics of types in programming languages, in [AGM], pp. 395-475.

[Ha87] T. Hagino. A Typed Lambda Calculus with Categorical Type Constructors, in *Category Theory and Computer Science*, *Springer Lecture Notes in Computer Science* **283**, 1987, pp. 140-157.

[Har93] T. Hardin, How to get Confluence for Explicit Substitutions, in *Term Graph Rewriting*, M. Sleep, M. Plasmeijer, M. van Eekelen, eds., Wiley, 1993, pp. 31-45.

[HM92] V. Harnik and M. Makkai, Lambek's categorical proof theory and Läuchli's abstract realizability, *J. Symbolic Logic* **57**, 1992, pp. 200-230.

[MHas97] M. Hasegawa. Recursion from Cyclic Sharing : Traced Monoidal Categories and Models of Cyclic Lambda Calculi, TLCA '97, *Springer Lecture Notes in Computer Science* **1210**, 1997, pp. 196-213.

[RHas94] R. Hasegawa, Categorical data types in parametric polymorphism, *Math. Structures in Compter Science* **4** , no.1, 1994, pp. 71-109.

[RHas95] R. Hasegawa, A Logical Aspect of Parametric Polymorphism, in *Computer Science Logic, CSL'95*, H.K. Büning Ed., *Springer Lecture Notes in Computer Science* **1092**, 1995.

[Haz] M. Hazewinkel, Introductory Recommendations for the Study of Hopf Algebras in Mathematics and Physics, CWI Quarterly, *Centre for Mathematics and Computer Science, Amsterdam* Vol. 4,No. 1, March 1991.

[HJ95] C. Hermida and B. Jacobs, Fibrations with indeterminates: contextual and functional completeness for polymorphic lambda calculi, *Math. Structures in Compter Science* **5** , no. 4, pp. 501–532.

[HMP98] C. Hermida, M. Makkai, J. Power, Higher dimensional multigraphs, *13th Annual IEEE Symp. on Logic in Computer Science (LICS)* , IEEE, 1998, pp. 199-206.

[Hi80] D. Higgs, Axiomatic Infinite Sums–An Algebraic Approach to Integration Theory, *Contemporary Mathematics*, Volume 2, 1980.

[HR89] D.A. Higgs and K.A. Rowe. Nuclearity in the category of complete semilattices, *J. Pure and Applied Algebra* **5**, 1989 pp. 67–78.

[HPW] T. Hildebrandt, P. Panangaden, G. Winskel, A Relational Model of Non-Deterministic Dataflow, to appear in: Concur 98, *Springer Lecture Notes in Computer Science* **?**.

[HM94] J. Hodas and D. Miller. Logic programming in a fragment of intuitionistic linear logic, *Inf. and Computation* **110**(2), 1994, pp. 327-365.

[H97a] M. Hofmann, Syntax and Semantics of Dependent Types, in [PD97], pp. 79-130.

[H97b] M. Hofmann, An application of category-theoretic semantics to the characterisation of complexity classes using higher-order function algebras, *Bull. Symb. Logic* **3** (4), 1997, pp. 469-485.

[HJ97] H. Hu, A. Joyal Coherence completions of categories and their enriched softness, in S. Brookes and M. Mislove (eds.), *Proceedings, Mathematical Foundations of Programming Semantics, Thirteenth Annual Conference*, Electronic Notes in Theoretical Computer Science **6**, 1997.

[Hu90] G. Huet, editor. *Logical Foundations of Functional Programming*, Addison-Wesley, 1990.

[Hy88] M. Hyland, A small complete category, *Ann. Pure and Appl. Logic* **40**, 1988, pp. 135-165.

[Hy97] M. Hyland, Game Semantics, in [PD97], pp. 131-184.

[HRR] M. Hyland, E. Robinson, G. Rosolini. The discrete objects in the effective topos. *Proc. Lond. Math. Soc.* 3, **60**, 1990, pp. 1-36.

[HP93] M. Hyland, V. de Paiva, Full intuitionistic linear logic (Extended Abstract), *Annals of Pure and Applied Logic*, **64**, 3, 1993, pp. 273–291.

[JR] B. Jacobs and J. Rutten, A tutorial on (Co)Algebras and (Co)Induction, EATCS Bulletin **62** , 1997, pp. 222-259.

[J90] C.B. Jay, The structure of free closed categories, *J. Pure and Applied Algebra* **66**, 1990, pp. 271–285.

[JGh95] C. B. Jay and N. Ghani, The virtues of eta expansion, *Journal of Functional Programing*, **5**, no.2, 1995, pp. 135-154.

[JNW] A. Joyal, M. Nielsen, and G. Winskel, Bisimulation from open maps, *Inf. and Computation*, **127**(2), 1996, pp. 164-185.

[JS91b] A. Joyal and R. Street, An introduction to Tannaka duality and quantum groups, in A. Carboni, *et. al.*, eds.,*Category Theory, Proceedings, Como 1990*, *Springer Lecture Notes in Mathematics* **1488**, 1991, pp. 411–492.

[JS91] A. Joyal and R. Street, The geometry of tensor calculus I, *Advances in Mathematics* **88**, 1991, pp. 55–112.

[JS93] A. Joyal and R. Street, Braided tensor categories , *Advances in Mathematics* **102**, no. 1 1993, pp. 20-79.

[JSV96] A. Joyal, R. Street, and D. Verity, Traced monoidal categories, *Math. Proc. Camb. Phil. Soc.* **119**, 1996, pp. 447–468.

[JT93] A. Jung and J. Tiuryn, A new characterization of lambda definability, in TLCA'93 , *Springer Lecture Notes in Computer Science* **664**, pp. 230-244.

[Ka95] M. Kanovich, The direct simulation of Minksky machines in linear logic, in [GLR], pp. 123-146.

[KOS97] M. Kanovich, M. Okada, and A. Scedrov Phase semantics for light linear logic,in S. Brookes and M. Mislove (eds.), *Proceedings, Mathematical Foundations of Programming Semantics, Thirteenth Annual Conference*, Electronic Notes in Theoretical Computer Science **6**, 1997.

[KSW95] P. Katis, N. Sabadini, and R.F.C. Walters, Bicategories of Processes, Manuscript, 1995. Dept. of Mathematics, U. Sydney.

[KSWa] P. Katis, N. Sabadini, and R.F.C. Walters, Span(Graph): A Categorical Algebra of Transition Systems, in AMAST'97, *Springer Lecture Notes in Computer Science* **1349**, 1997.

[K77] G. M. Kelly, Many-variable functorial calculus.I, *Coherence in Categories Springer Lecture Notes in Mathematics* **281**, 1977, pp. 66-105.

[K82] G. M. Kelly. *Basic Concepts of Enriched Category Theory*, London Math. Soc. Lecture Notes 64, Camb. Univ.Press, 1982.

[KM71] G. M. Kelly and S. Mac Lane. Coherence in closed categories, *J. Pure and Applied Algebra* **1** (1971), pp. 97-140.

[KOPTT] Y. Kinoshita, P. W. O'Hearn, A. J. Power, M. Takeyama, R. D. Tennent. An Axiomatic Approach to Binary Logical Relations with Applications to Data Refinement, in *Springer Lecture Notes in Computer Science* **1281**, TACS '97, 1997, pp. 191-212.

[Ku63] A.G. Kurosh, Direct Decomposition in Algebraic Categories, *Amer. Math. Soc. Transl.* Ser. 2, 27 (1963), pp. 231–255.

[Lac95] S.G. Lack. *The algebra of distributive and extensive categories.* PhD thesis, University of Cambridge,1995.

[Laf88] Y. Lafont. Logiques, catégories et machines. Thèse de doctorat, Université Paris VII, U.F.R. de Mathématiques, 1988.

[Laf95] Y. Lafont, From proof nets to interaction nets, in [GLR], pp. 225–247.

[L68] J. Lambek, Deductive Systems and Categories I, *J. Math. Systems Theory* **2**, pp. 278-318.

[L69] J. Lambek, Deductive systems and categories II,*Springer Lecture Notes in Mathematics* **87** Springer-Verlag, Berlin, Heidelberg, New York, 1969.

[L74] J. Lambek, Functional completeness of cartesian catgories. *Ann. Math. Logic* **6**, pp. 259-292.

[L88] J. Lambek, On the Unity of algebra and Logic, *Springer Lecture Notes in Mathematics* **1348**, *Categorical Algebra and its applications,* F. Borceux, ed. Springer-Verlag, 1988.

[L89] J. Lambek, Multicategories Revisited. *Contemp. Math.***92**, pp. 217-239.

[L90] J. Lambek, Logic without structural rules. Manuscript, (Mcgill University), 1990.

[L93] J. Lambek, From categorial grammar to bilinear logic, in K. Došen and P. Schroeder-Heister, eds., *Substructural logics*, Studies in Logic and Computation **2**, Oxford Science Publications, 1993.

[L95] J. Lambek, Bilinear Logic in Algebra and Linguistics, in [GLR].

[LS86] J. Lambek and P. J. Scott. *Introduction to Higher Order Categorical Logic*, Cambridge Studies in Advanced Mathematics **7**, Cambridge University Press, 1986.

[Lau] H. Läuchli, An Abstract Notion of Realizability for which Intuitionistic Predicate Calculus is Complete, *Intuitionism and Proof Theory,*North–Holland, 1970, pp. 227-234.

[Law66] F. W. Lawvere,The category of categories as a foundation for mathematics. In *Proc. Conf. on Categorical Algebra*, ed. by S. Eilenberg, et.al., 1965, Sprinter-Verlag.

[Law69] F.W. Lawvere, Adjointness in Foundations, *Dialectica*, **23**, pp. 281-296.

[Lef63] S. Lefschetz, Algebraic Topology, American Mathematical Society Colloquium Publications 27, (1963).

[Li95] P. Lincoln, Deciding provability of linear logic formulas, in [GLR] pp. 109-122.

[Luo94] Z. Luo. *Computation and Reasoning: a Type Theory for Computer Science*. Oxford U. Press, 1994.

[LP92] Z. Luo and R. Pollack. LEGO Proof Development System: User's Manual. Technical Report ECS-LFCS-92-211, U. Edinburgh, 1992.

[MaRey] Q. Ma, J. C. Reynolds. Types, Abstraction, and Parametric Polymorphism, Part 2. *Springer Lecture Notes in Computer Science* **598**, 1991, pp. 1-40.

[Mac71] S. Mac Lane. *Categories for the Working Mathematician,*Springer Graduate Texts in Mathematics, Springer-Verlag, 1971.

[Mac82] S. Mac Lane, Why commutative diagrams coincide with equivalent proofs, *Contemp. Math.***13**, 1982, pp. 387-401.

[MM92] S. Mac Lane, I. Moerdijk. *Sheaves in Geometry and Logic*, Springer-Verlag Universitext, 1992.

[MaRe91] P. Malacaria and L. Regnier. Some Results on the Interpretation of λ-calculus in Operator Algebras, *6th Annual IEEE Symp. on Logic in Computer Science (LICS)* , 1991, IEEE Press, pp. 63-72.

[MA86] E. Manes and M. Arbib. *Algebraic Approaches to Program Semantics*, Springer-Verlag, 1986.

[M98] E. Manes, Implementing collection classes with monads, *Math. Structures in Compter Science* **8** , Number 3, pp. 231-276.

[ML73] P. Martin-Löf, 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.

[ML82] P. Martin-Löf, Constructive mathematics and computer programming, in *Proc. Sixth International Congress for Logic, Methodology and Philosophy of Science*, North-Holland, 1982.

[Mc97] G. McCusker, Games and definability for FPC. *Bull. Symb. Logic* **3** 3, 1997, pp. 347-362.

[MPSS95] N. Mendler, P. Panangaden, P. Scott, R. Seely, The Logical Structure of Concurrent Constraint Languages. *Nordic Journal of Computing 2* , 1995, pp. 181-220.

[Mét94] F. Métayer, Homology of proof nets, *Arch. Math Logic***33**, 1994. (see also [GLR]).

[Mill] D. Miller, Linear Logic as Logic Programming: An Abstract, in *Logical Aspects of Computational Linguistics, LACL'96* (C. Retoré Ed.), *Lecture Notes in Artificial Intelligence* **1328**, 1996.

[MNPS] D. Miller, G. Nadathur, F. Pfenning, A. Scedrov. Uniform Proofs as a Foundation for Logic Programming, *Ann. Pure and Applied Logic* **51**, 1991, pp. 125-157.

[Mil89] R. Milner. *Communication and Concurrency*, Prentice-Hall, 1989.

[Mil77] R. Milner, Fully abstract models of typed lambda calculi, *Theoretical Comp. Science* **4** , 1977, pp. 1-22.

[Mil93a] R. Milner, The polyadic π- Calculus: a tutorial, in: *Logic and Algebra of Specification, eds. F.L Bauer, W. Brauer, and H. Schwichtenberg* , Springer-Verlag, 1993.

[Mil93b] R. Milner, Action calculi, or concrete action structures, Mathematical Foundations of Computer Science, Springer *Springer Lecture Notes in Computer Science* **711**, 1993, pp. 105–121.

[Mil94] R. Milner. Action Calculi V: Reflexive Molecular Forms, Manuscript, University of Edinburgh, June 1994.

[Min81] G. E. Mints. Closed Categories and the theory of proofs. *J. Soviet Math* **15**, 1981, pp. 45-62.

[Mit96] J. C. Mitchell. *Foundations for Programming Languages*, MIT Press, 1996.

[Mit90] J. C. Mitchell, Type Systems for Programming Languages, in: *Handbook of Theoretical Computer Science*, Vol.B,(*Formal Models and Semantics*), J. Van Leeuwen, ed., North-Holland, 1990 pp. 365-458.

[MitSce] J. C. Mitchell and A. Scedrov, Notes on Sconing and Relators, in *Springer Lecture Notes in Computer Science* **702**, CSL'92, 1992, pp. 352-378.

[MS89] J. C. Mitchell and P. J. Scott, Typed Lambda Models and Cartesian Closed Categories, in [GS89], pp. 301- 316.

[MiVi] J.C. Mitchell, R. Visvanathan, Effective Models of Polymorphism, Subtyping and Recursion (Extended abstract), in *Automata, Languages and Programming, ICALP'96*, F.M. auf der Heide and B. Monien Eds., *Springer Lecture Notes in Computer Science* **1099**, 1996.

[Mo91] E. Moggi, Notions of computation and Monads, *Inf. & Computation* **93**1, 1991, pp. 55-92.

[Mo97] E. Moggi, Metalanguages and Applications, in [PD97], pp. 185-239.

[MR97] E. Moggi, G. Rosolini, eds. *7th International Conference on Category Theory and Computer Science*, *Springer Lecture Notes in Computer Science* **1290**, 1997.

[Mul91] P. Mulry, Strong Monads, Algebras, and Fixed Points, in [FJP], pp. 202-216.

[OHT92] P. O'Hearn and R. D. Tennent, Semantics of Local Variables, in: *Applications of Categories in Computer Science*, London Math. Soc. Lecture Series 177, Camb. U. Press, 1992, pp. 217-236.

[OHT93] P. O'Hearn and R. D. Tennent, Relational Parametricity and Local Variables, in *20th Symp. on Priniciples of Programming Languages*, Assn. Comp. Machinery, 1993, pp. 171-184.

[OHRi] P. O'Hearn and J. Riecke. Kripke Logical Relations and PCF, to appear in *Information and Computation.*

[Ol85] F. J. Oles, Type algebras, functor categories and block structure, in: *Algebraic Methods in Semantics*, M. Nivat and J. Reynolds, eds., Camb. U. Press, 1985.

[PSSS] P. Panangaden, V. Saraswat, P. J. Scott, R. A. G. Seely, A Categorical View of Concurrent Constraint Programming, *Proceedings of REX Workshop*, *Springer Lecture Notes in Computer Science* **666**, 1993, pp. 457-476.

[PR89] R. Paré, and L. Román, Monoidal categories with natural numbers object, *Studia Logica* XLVIII (1989) 361–376.

[P96] D. Pavlović, Categorical logic of names and abstraction in action calculi, *Math. Structures in Compter Science* **7** , No. 6, 1997, pp. 619-637.

[PaAb] D. Pavloviċ, S. Abramsky, Specifying Interaction Categories, in CTCS'97 (E. Moggi and G. Rosolini, eds.) , *Springer Lecture Notes in Computer Science* **1290**, 1997.

[Pen93] M. Pentus, Lambek Grammars Are Context Free, in *8th Annual IEEE Symp. on Logic in Computer Science (LICS)* , Montreal, 1993, pp. 429–433.

[PDM89] B. Pierce, S. Dietzen, S. Michaylov. Programming in Higher-Order Typed Lambda-Calculi, Reprint CMU-CS-89-111, Dept. of Computer Science, Carnegie-Mellon University, 1989.

[Pi87] A. Pitts. Polymorphism is set-theoretic,constructi ely, in: *Category Theory and Computer Science*, *Springer Lecture Notes in Computer Science* **283** (D. H. Pitt, ed.), 1987, pp. 12-39.

[Pi96a] A. Pitts, Relational Properties of Domains, *Inf. and Computation*, **127**, No. 2, 1996, pp. 66-90.

[Pi96b] A. Pitts, Reasoning about Local Variables with Operationally-Based Logical Relations, in *11th Annual IEEE Symp. on Logic in Computer Science (LICS)* , IEEE Press, 1996, pp. 152–163.

[Pi97] A. Pitts, Operationally-Based Theories of Program Equivalence, in [PD97], pp. 241-298.

[Pi9?] A. M. Pitts , Categorical Logic, *Handbook of Logic in Computer Science*, S. Abramsky, D. M. Gabbay and T. S. E. Maibaum, eds. Oxford University Press, 1996, Vol. 6 (to appear)

[PD97] A. Pitts and P. Dybjer, eds. *Semantics and Logics of Computation*, Publications of the Newton Institute, Camb. Univ. Press, 1997.

[Plo77] G. D. Plotkin, LCF Considered as as Programming Language, *Theoretical Comp. Science* **5** , 1977, pp. 223-255.

[Plo80] G. D. Plotkin, Lambda definability in the full type hierarchy, in *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus, and Formalism*, Academic Press, 1980, pp. 363-373.

[PlAb93] G. Plotkin, M. Abadi, A Logic for Parametric Polymorphism, in TLCA'93, *Springer Lecture Notes in Computer Science* **664**,1993, pp. 361–375.

[Pow89] J. Power, A general coherence result , *J. Pure and Applied Algebra* **57**, 1989, pp. 165-173.

[PK96] J. Power and Y. Kinoshita, A new foundation for logic programming, in *Extensions of Logic Programming '96*, Springer-Verlag, 1996.

[PowRob97] Power, A.J. and E.P. Robinson, Premonoidal categories and notions of computation,*Math. Structures in Compter Science* **7** , 5, 1997, pp. 453-468.

[PowTh97] J. Power, H. Thielecke, Environments, Continuation Semantics and Indexed Categories, in TACS'97, *Springer Lecture Notes in Computer Science* **1281**, 1997, pp. 191-212.

[Pra95] V. Pratt, Chu Spaces and their Interpretation as Concurrent Objects, *Springer Lecture Notes in Computer Science* **1000**, *Computer Science Today*, J. van Leeuwen, ed., 1995, pp. 392-405.

[Pra97] V. Pratt, Towards Full Completeness for the Linear Logic of Chu spaces, *Proc. MFPS (MFPS'97)*, Electronic Notes of Theoretical Computer Science, 1997.

[Pr65] D. Prawitz. *Natural Deduction* , Almquist& Wilksell, Stockhom, 1965.

[Rgn92] L. Regnier. Lambda-calcul et réseaux. Thèse de doctorat, U. Paris VII, U.F.R. Mathématiques,1992.

[ReSt97] B. Reus, Th. Streicher, General Synthetic Domain Theory– A Logical Approach (Extended abstract), in CTCS'97, *Springer Lecture Notes in Computer Science* **1290**, 1997.

[Rey74] J. Reynolds, Towards a theory of type structure, *Programming Symposium*, *Springer Lecture Notes in Computer Science* **19**, 1974, pp. 408-425.

[Rey81] J. C. Reynolds. The essence of Algol, in: *Algorithmic Languages*, J. W. de Bakker and J. C. van Vliet, eds., North-Holland, 1981, pp. 345–372.

[Rey83] J.C. Reynolds. Types, Abstraction, and Parametric Polymorphism, in: *Information Processing '83* , R.E.A. Mason, ed. North-Holland, 1983, pp. 513-523.

[RP] J. Reynolds and G. Plotkin. On Functors Expressible in the Polymorphic Typed Lambda Calculus, *Inf. and Computation* , reprinted in [Hu90], pp. 127-152.

[Rob89] E. Robinson, How complete is PER? *4th Annual IEEE Symp. on Logic in Computer Science (LICS)* , 1989, IEEE Publications, pp. 106-111.

[RR90] E. Robinson and G. Rosolini, Polymorphism, Set Theory, and Call-by-Value, *5th Annual IEEE Symp. on Logic in Computer Science (LICS)* ,1990, IEEE Publications, pp. 12-18.

[Rom89] L. Roman, Cartesian Categories with Natural Numbers Object, *J. Pure and Applied Algebra* **58**, 1989, pp. 267–278.

[Ros90] G. Rosolini, About Modest Sets, in *Int. J. Found. Comp. Sci* **1**, 1990, pp. 341-353.

[Sc93] A. Scedrov. A Brief Guide to Linear Logic. In: *Current Trends in Theoretical Computer Science*, G. Rozenberg and A. Salomaa Eds., World Scientific Publishing Co., 1993, pp. 377–394.

[Sc95] A. Scedrov. Linear Logic and Computation: A Survey, In: *Proof and Computation*, H. Schwichtenberg Ed., NATO Advanced Science Institutes, Series F, Volume 139, Springer–Verlag, Berlin, 1995, pp. 379–395.

[Sc72] D. Scott, Continuous lattices *Springer Lecture Notes in Mathematics* **274**, pp.97-136.

[See87] R. A. G. Seely. Categorical Semantics for Higher Order Polymorphic Lambda Calculus. *J. Symb. Logic* **52**, 1987, pp. 969–989.

[See89] R.A.G. Seely, Linear logic, $*$–autonomous categories and cofree coalgebras, in J. Gray and A. Scedrov (eds.), *Categories in Computer Science and Logic*, Contemporary Mathematics **92** (Am. Math. Soc. 1989).

[Sie92] K. Sieber, Reasoning about sequential functions via logical relations, in [FJP], pp. 258-269.

[Si93] A.K. Simpson A characterization of least-fixed-point operator by dinaturality,*Theoretical Comp. Science* **118** , 1993, pp. 301–314.

[SP82] M. B. Smyth and G. D. Plotkin. The category-theoretic solution of recursive domain equations. *SIAM J. Computing* **11**, 1982, pp. 761-783.

[So87] S. Soloviev, On natural transformations of distinguished functors and their superpositions in certain closed categories, *J. Pure and Applied Algebra* **47**, 1987, pp. 181–204.

[So95] S. Soloviev, Proof of a S. Mac Lane Conjecture (Extended Abstract), in CTCS'95 *Springer Lecture Notes in Computer Science* **953**, pp. 59-80.

[St96] R.F. Stärk, Call-by-Value, Call-by-Name and the Logic of Values, in *computer Science Logic, CSL'96*, D. van Dalen and M. Bezem Eds., *Springer Lecture Notes in Computer Science* **1258**, 1996.

[St85] R. Statman, Logical Relations and the Typed Lambda Calculus, *Inf. and Control* **65**, 1985, pp. 85-97.

[St96] R. Statman, On Cartesian Monoids, in: CSL'96 , *Springer Lecture Notes in Computer Science* **1258**, 1996.

[ST96] R. Street. Categorical Structures, in *Handbook of Algebra, Vol. 1* (Edited by M. Hazewinkel), Elsevier Science B.V., 1996.

[Tay98] P. Taylor. *Practical Foundations*, Camb. U. Press, 1998 (to appear).

[Ten94] R. D. Tennent, Denotational Semantics, in [AGM], pp. 169–322.

[TP96] D. Turi. G. Plotkin, Towards a Mathematical Operational Semantics, *12th Annual IEEE Symp. on Logic in Computer Science (LICS)* , 1996, pp. 280-291.

[Tr92] A. Troelstra. *Lectures on Linear Logic*, CSLI Lecture Notes **29** (CSLI 1992).

[TrvD88] A. S. Troelstra and D. van Dalen. *Constructivism in Mathematics*, Vols. I and II. North-Holland, 1988.

[W92] P. Wadler, The essence of functional programming, *Symp. on Principles of Programming Languages* (POPL'92), ACM Press, pp. 1-14.

[W94] P. Wadler, A Syntax for Linear Logic, *Springer Lecture Notes in Computer Science* **802** Springer-Verlag, Berlin, Heidelberg, New York, 1994, pp. 513–528.

[WW86] K.Wagner and G. Wechsung. *Computational Complexity*, D. Reidel, 1986.

[W92] R.F.C. Walters. *Categories and Computer Science*, Camb. U. Press, 1992.

[WN97] G. Winskel and M. Nielsen, Categories in Concurrency, in [PD97], pp. 299-354.

[Wr89] G. Wraith. A Note on Categorical Datatypes, in *Category Theory in Computer Science,Springer Lecture Notes in Computer Science* **389**, 1989, pp. 118-127.

[Y90] D. Yetter, Quantales and (Noncommutative) Linear Logic, *Journal of Symbolic Logic* **55**, 1990, pp. 41-64.