Some Topologies for Computations*

Giuseppe Longo
CNRS and Dépt. d'Informatique
Ecole Normale Supérieure
45, rue d'Ulm
75005 Paris, France
www.di.ens.fr/users/longo

September 18, 2003

1 Introduction

1.1 Computations

The applications of topological and order structures in Theory of Computation, a key aspect of Foundations of Mathematics and of Theoretical Computer Science, has various origins and it is largely due to the relevance of these structures for Intuitionistic (constructive) Systems: their Logic underlies theoretical as well as applied Computability.

Computability Theory was born in the '30's as a formalization of the informal notion of "potentially mechanizable" deduction, which was at the core of foundational analysis, since late '800, and resulted from the happy marriage of a Formalist and Intuitionistic understanding of effectiveness. For Peano, Hilbert and many others, foundation and certainty, in Mathematics, had to be found in stepwise deductions within "finitistic" system. For these leading mathematicians, the abstract and symbolic nature of Mathematics, but rigor as well, were identified and all coincided with *formalism* or with the "yet-to-be-specified" notion of *potentially mechanizable process*. At the core of it, there was the newly invented formal Theory of Numbers, Peano Arithmetic (PA), based on Peano-Dedekind formal rule for induction.

Along the lines of Leibniz's philosophical intuitions, Peano's "pasigraphy" is an early example of formal language, where finite strings of symbols were manipulated in a way feasible by an (abstract) machine. Peano, Padoa, Hilbert ... were actually referring to an mathematical machine, as nor the steam engine, nor the petrol engine, but not even the computing devices by Babbage and Power could be the reference for the notion they had in mind. These concrete machines were the paradigms of changing times, but these pioneers

*Invited paper, proceedings of the Conference **Géométrie au XX siècle**, Paris, septembre 2001 (to appear).

were thinking to a mathematical device yet to be invented, an absolute tool for a deductive mind.

Such a tool was fully described in the '30's. Its history is rich of independent inventions. Curry anticipated the notion of computable function by a simple language of combinators [Curry30], proposed for the finite manipulation of strings, on the grounds of an algebraic theory due to Shoenfinkel. Herbrand, in 1930, see [Goldfarb87], had a mathematical definition of a class of functions which were meant to represent logical deduction. Gödel ([Gödel31]) gave another characterization for the analysis of arithmetical deduction (i. e. for the metatheory of PA). And he encoded this metatheory within PA itself by two fundamental steps: he "gödel-numbered" formulas by integer numbers and represented his class of (partial) recursive - or computable - functions into PA (a much harder task than programming in Assembler). By this operation Gödel ended Hilbert's dream of a final solution to all foundational problems: prove formally the consistency of PA, prove that all arithmetic statements can be decided and ... mathematically live in the fregean Paradise of ideal (infinitary) objects.

As a matter of fact, the (formalized) mathematics of ideal objects, Geometry and Analysis, had been encoded into Arithmetic by [Hilbert99] and by Cantor and Dedekind, respectively. Thus, foundations, certainty and knowledge, for Hilbert, could be all transferred into the provable consistency and completeness (decidability) of Arithmetic. But, by encoding the metatheory of PA into the theory, Gödel showed that the encoded metatheoretic proposition of consistency is an undecidable statement of PA. If his class of computable functions were fully general (i.e. it convincingly represented the original, but informal idea of "potentially mechanisable - deductive - process"), then this one fact was enough to negate both Hilbert's conjectures. And it was so. A few years later, in [Turing36], Turing gave the most convincing, abstract notion of formal machine ever possible. Key idea: distinguish between hardware and software. Here is the reading/writing head and the tape, there are the programs. Formal Logic finally had his locus for absolute certainty: move right, move left the head, read/write 1 or 0 on the tape ..., according to a program written in a finite number of well-formed lines, possibly encoded by 0's and 1's. This is the action of the leibnizian man in the act of deducing, the human computor, the definitive Calculemus: no further reduction is possible to more elementary steps. And yet the Machine is fully expressive: Curry, Herbrand, Gödel computable functions, but Church's ([Church32]) and Kleene's as well can all be programmed over Turing Machines (TM), [Kleene36]. At once, an "absolute" was proposed, in Gödel's words, and Computer Science was born. The absolute, logical Machine.

Invented for the purposes of formal Logic, TM's engendered the early generations of computers, the sequential machines that dominated the scene for very long. The issue of space, and of its geometric intelligibility, had been programmatically excluded from the foundational project, all centered on Logic since Frege (see [Tappenden95], [Longo02] and their references for more on this): the result is that the expressive power of TM's, a formal machine for sequential deductions, does not depend on its being in space, e.g. it is independent from

the Cartesian dimension of the device (TM's, in any finite dimension, compute the same class of functions). We will hint how this affects the geometric understanding of computations, briefly presented below (its so called "Denotational Semantics") as well as the novel challenges posed by today's modern computing, that of *concurrent* systems. These systems happen to be *distributed in space*, as networks of computers. As a matter of fact, classical (and intuitionistic) Logic and their formal machines are no longer sufficient to deal with the new mathematical challenges posed by actual computing over concurrent systems. TM's are logical machines, not physical ones, as no relevant physical process is independent from the dimension of space (many depend only on it). And Mathematics, Geometry, proves it.

1.2 Dimensions

Computations take place along time, a linear time scanned by the clock of a Turing Machine (TM). This absolute, Newtonian understanding of time will be in the background of the approach to computations mostly followed here. But what about the dimensions of space? There are two (equivalent) ways to understand its role in Computability Theory. First, some simple theorems prove that one linear tape, one head or finitely dimensional tapes and heads TM's have all the same expressive power, modulo some (low) complexity issue. That is, they compute the same class of functions, the Partial Recursive Functions (PR), as functions from ω to ω , the integer numbers. Second, once given PR, how to define computable functions over ω^n, for any n? Easy: a many arguments function is **computable** iff it is so *componentwise*. Equivalently, in order to define computability from ω^n to ω^m, encode ω^n onto ω, in some effective and bijective way, and use PR. These two facts are a symptom of the irrelevance of physical dimension in classical computability.

As a consequence, all categories for computations below may be easily shown to be Cartesian, that is, to be closed under finite products. But more than this, functions of any finite number of parameters are all treated, computationally, in the same way. The componentwise nature of computability will be better understood by the Cartesian Closure of some of these categories, a stronger property (see below). These categories are an effective frame for dealing with functions of higher types (functions of functions ...) and on abstract structures. This is needed, in a mathematical treatment of computations, as programs may compute with (encoded) programs as input and, more generally, with abstract types of data. That is, a TM may take (the program of) another TM as input: mathematically this yields a functional acting on a function.

Of course, componentwise computability of PR (and the irrelevance of dimensions) presents a first mathematical difficulty, when dealing with them and their function space by analytic tools: continuity is a topological notion and global continuity (w. r. to several parameters) is not entailed by componentwise continuity. Moreover, Cartesian dimension is a topological invariant. That is, if one embeds discrete structures into topological continua as space manifolds (in Riemann's sense), then the simple isomorphisms above between arbitrary finite dimensional discrete spaces, between ω^n and ω, say, cannot

be extended to continuous isomorphisms, even not to isomorphic embeddings. This is so, at least, for traditional approaches to the continuum, which refer to physical continua and yield a physically plausible (and classical) notion of neighborhood, i.e. where different points in space may be separated by the topology (or the topology is at least T_2, i.e. Hausdorff).

Thus, the invention of suitable, weakly separated topologies was required, which serve the "embedding purposes" and is fully motivated by the theorems in section 2.2. The Mathematics is very interesting and has had many relevant applications, in particular to the Theory of Programming, but the loss of the sense of space is not less relevant. Interestingly enough, this came out to the limelight from within Computer Science itself, as it will be hinted below.

This lecture is not meant to be a complete survey of "Geometry in Computing", but it mainly focuses on a (rather sucessful) topological approach. For this purpose, we will first survey the general frame for computability in some mathematical spaces and try to stress the need for topological tools for their sound treatment (sections 2, 3). Then, two specific applications of topological methods will be given. They are mathematical applications, and properly "analytic"; that is, a purely combinatorial statement will be proved by an embedding of discrete structures into topological continua (sections 4, 5).

The limits of this approach will be hinted as well, in reference to the novel needs for Logic and Computing (see the Intermezzo). The point is that these topologies are directly (and provably) related to classical computability, that is to the logico-mathematical theory born in the '30's, while both Logic and Theoretical Computing need today to reconsider the mathematical understanding of space as an integral part of their contribution to knowledge and applications. Turing Computability should be gradually considered as classical (Laplace's) Rational Mechanics in Mathematical Physics: a major start, but entirely superseded by the Mathematics (the Geometry) of Dynamical Systems, Relativity and Quantum Mechanics. Yet, the main ideas, born in between 1930 and 2000, are worth recalling, as they may yield some fundamental mathematical concepts. In Rational Mechanics, say, laplacians or hamiltonians, as geodetics and energy conservation properties, passed over to XX century Mathematical Physics, as key conceptual and mathematical invariants (with all due changes, of course). As for now, the notion of sequential computation, Type, Polymorphism, sub-typing, some limit constructions and their topological and categorical understanding seem to remain basic tools in Programs' Theory and Design, in spite of the dramatic changes Computer Science is undergoing. Future History will better tell.

2 Scott Topologies and Computable Functionals

It is fair to say that Myhill-Shepherdson Theorem [MyhillSheph55], in Recursion Theory, is one of the earliest results which provided a "topological", though implicit, characterization of a purely recursion theoretic notion of functional (or function with function argument). The topological content of this

relevant theorem was made explicit and generalized by Dana Scott in the early '70's, for an analysis of typed and type-free computations (see below). Since then, the Scott topologies and their numerous variants have had a major role, both in higher-types Recursion Theory (where Ershov later rediscovered the same structures, within an original frame) and, more importantly, in the Denotational Semantics of Programming Languages.

Higher Type Recursion and Denotational Semantics have different but related motivations. First, the intuitionistic analysis of propositions naturally yields higher type computations: as it will be better explained below, alternating quantifiers are understood as (effective) functions ("for all x, there exists y ... " yields a computable function from x to y - this is the sense of the "constructive existence" of y), then further alternations of quantifiers yield functions of functions ... (see section 4). Second, Programming Languages need types, as they work on various "types of data", in particular of any finite order or level, as programs may take numbers as inputs but programs as well or even programs acting on programs .... In either case, in Logic and in Programming, a crucial mathematical issue is posed: in Mathematics, when working with functions, as sets of input/output values, one has to deal with infinite objects. And the royal way out towards infinity, in constructive systems, is provided by approximation. But approximation means *order*, and *topology*.

2.1 Continuous Domains

Recall that computations, since Turing's early results, are essentially partial (the halting problem is undecidable or there exists partial computable functions that cannot be extended to total ones). There is now a natural way to order partial functions as input/output devices: they can be partially ordered by graph inclusion (that is, "smaller" means "less defined"). This is the underlying intuition that motivates the constructions below.

Consider a partial order $D = (D, \leq)$. For the purposes of an analysis of approximation, we need to generalize the notion of "chain" and "limits" in total orders to partial ones. A subset A of D is **directed** if for x and y in A, there exists z in A such that $x, y \leq z$ (a chain is particular directed set, of course). The limits will be provided by the suprema of directed sets, $\sup A$, whenever they exist.

Definition 2.1 *Let x and y be elements of D, then x is* **way below** *y (notation: $x \ll y$) if: for any directed set A in D, if $y \leq \sup A$, then $\exists a \in A\ x \leq a$.*

Clearly, the relation $\ll$ is finer than $<$, and one has: $x \ll y$ and $x' \leq x$ and $y \leq y'$ imply $x' \ll y'$. Notation: $\uparrow a = \{b \in D \mid a \ll b\}$; $\downarrow a = \{b \in D \mid b \ll a\}$.

Definition 2.2 *D is a* **Continuous Domain** *if, for each $a \in D$, the set $\downarrow a$ is directed and $a = \sup \downarrow a$.*

Consider a continuous domain D. Then $\emptyset \cup \{\uparrow a \mid a \in D\}$ trivially forms a basis for a topology (the order topology, i.e. $x \leq y$ iff for all open U,

$x \in U \Rightarrow y \in U$). Call this topology, the **Scott topology**. A subset D_0 is a **basis**, if, for each $a \in D$, the set $\downarrow a \cap D_0$ is directed and $a = \sup \downarrow a \cap D_0$. Clearly, also $\emptyset \cup \{a \mid a \in D_0\}$ is a basis for the Scott topology. If D has a countable base, then D is **ω-continuous**.

Note that the notion of approximation (or *neighborhood*) is given "from below" and that Scott open sets are upward closed: moving up, along the partial order, gives more and more defined (better approximated) elements ([Scott70], [Scott72]).

Example 2.3 *Let $P\omega$ be the powerset of ω , the set of natural numbers, and P the set of partial (number theoretic) functions. $P\omega$ and P are ω-continuous domains by set inclusion, where $a \ll b$ for some b iff a is finite iff $a \ll a$ (finite, in P, means "function with a finite graph", or equivalently "with a finite domain" - of definition; approximation is graph inclusion - thus, the total functions are the maximal elements).*

In general, call **algebraic** any element a in a continuous domain, that satisfies the $a \ll a$ relation. This is a key general notion of "finiteness" that will be extended to functions and functionals in higher types: it yields the useful properties of "finite elements" in more general settings, where there are no (properly) finite elements. As a preliminary exercise, consider two continuous domains A, B and the collection $CD[A, B]$ of continuous functions (w.r.t. the Scott topology); give to $CD[A, B]$ the structure of a continuous domain by taking the pointwise convergence topology on it. (When $A = B = P$, finite elements in $CD[P, P]$ are non-obvious but useful functionals - for example, they can be effectively enumerated, see below).

However, many interesting domains have no algebraic elements. Here is one that will be used later on, for further applications of Domain theory.

Example 2.4 *(The* **upperspace** *of $[0, 1]$). Consider the collection $I([0, 1])$ of the compact intervals in the real interval $[0, 1]$. Let $D = (I([0, 1]), \supseteq)$, that is $\leq$ is the* reverse *inclusion. Then:*

- *$(I([0, 1]), \supseteq)$ is an ω-continuous domain (base: the intervals with rational end points).*
- *$[a, b] \ll [c, d]$* iff *$[c, d] \supseteq]a, b[$ (note!!)*
- *the singleton intervals are the largest elements and $[0, 1]$ is the least one.*

The intuition should be clear: in $I([0, 1])$, the real points are approximated from below by decreasing the size of the intervals containing it. No element of this domain is algebraic. Note also that a base for the Scott topology is given by $\{[a, b] \mid [a, b] \subseteq O\}$, when O ranges over the open sets (of the real topology over $[0, 1]$). Clearly, the induced topology from $I[0, 1]$ over $[0, 1]$, the set of maximal elements, is the real topology. Thus, approximation from below gives the reals as fully specified elements (the total ones, in a sense), with their familiar topology.

2.2 Myhill-Shepherdson Theorem

Consider now an ω-continuous domain D. Then its basis D_0 can be enumerated by a bijection $e : \omega \to D_0$, say. The domain is called **effective** if $e(n) \leq e(m)$ is a decidable predicate in n and m. This is a key property as for effectiveness, as it clearly applies to our leading examples above, in particular $P\omega$ and P, and it allows to define the collection of computable elements in an effective domain: just take the limit (the sup) of all effectively enumerable directed set in D_0. That is, consider the directed subsets A which possess a recursively enumerable (r.e.) set of indices, i.e. $\{n \,|\, e(n) \in A\}$ is r.e.. Their suprema, if they exist, will be the generalized computable functions and functionals in (higher) types or in abstract structures. In $P\omega$ and P, these sets give exactly the r.e. sets ($RE \subseteq P\omega$) and the partial recursive functions ($PR \subseteq P$). In $(I([0,1]), \supseteq)$ they are the intervals with computable reals as end points (a real is computable, when it is "effectively generated", as a sequence). Thus, if one starts by considering TM's and their computable functions (PR or RE), we are now seeing them as embedded into suitable continua (P or $P\omega$), endowed with an order topology (note that both $(P, \subseteq)$ and $(P\omega, \subseteq)$ contain chains with the order structure of the reals).

Is this construction of a category of domains, really "natural"? At least a theorem with two important corollaries proves it, by establishing a nice correspondence between topologies and computations. In order to state them, observe first that it is easy to enumerate the limits or sups of r.e. directed sets of the base, if they exist: just use the indices of the r.e. set, which enumerate the directed set below it. Each of these limits, of course, will have infinitely many indices, at least as many as the indices of enumerating r.e. set. In an effective ω-continuous domain D, call D_c the set of the computable elements in D, and $w : \omega \to D_c$ this (canonical) enumeration of D_c. If all limits of the r.e. directed sets of the base exist in D, call then **effectively complete** such an effective domain. Then one has:

Theorem 2.5 *In an effectively complete ω-continuous domain D, consider a canonical enumeration $w : \omega \to D_c$ and $B \subseteq D_c$. Then, if $\{n \,|\, w(n) \in B\}$ is recursively enumerable, one has that B is an open set (in the Scott topology). Conversely, for any basic open set $B \subseteq D_c$, the set $\{n \,|\, w(n) \in B\}$ is recursively enumerable.*

The first implication is a non-obvious fact, with several consequences (the converse is easy). Observe first that there is no apparent reason by which a recursion theoretic notion, the recursive enumerability of a set of indices, should lead to topological openness. Look at it in $P\omega$ or P, where it was first proved by Rice and Shapiro (see Theorem 14.XIV in [Rogers67]) by a non-trivial argument: its key lemma, later called of "effective discontinuity", has a relevant logical-intuitionistic meaning, and it may be also reconstructed from the proof of Myhill-Sheperdson Theorem, see corollary 2.7 below (the generalisation to domains given here is easy). The theorem above says that if you take a collection of r.e. sets or of partial recursive functions, then their semi-decidability gives a certain order or topological structure to the set of

sets or of functions, considered in extenso (as set-theoretic relations, e.g. as elements of $P\omega$ or P). An immediate consequence is that a property of r.e. sets or partial recursive functions is decidable iff it is trivial. More generally:

Corollary 2.6 *In an effectively complete ω-continuous domain D, with $w : \omega \to D_c$ and $B \subseteq D_c$, $\{n \mid w(n) \in B\}$ is recursive iff B is empty or coincides with D.*

Proof. Observe that an effectively complete ω-continuous domain has necessarily a least element, the sup of the empty set (which may be effectively enumerated by the ... the empty set of numbers). Assume then that $\{n \mid w(n) \in B\}$ is recursive, i.e. that it is r.e. with its complement, then the set and its complement are both Scott open, by the theorem. One of them must contain the least element and thus, coincide with D. The converse is obvious. □

The relevance of the Corollary should be clear, even in the basic case of $P\omega$ or P, where the result was originally given in [Rice53]. In summary, no non-trivial property of programs is decidable (Corollary 2.6), and the semi-decidable properties are "very few" or "topologically very structured" (Theorem 2.5). In particular, we cannot decide nor semi-decide whether a program is correct, that is if it computes a given function or a function in the desired set. Large part of Theoretical Computer Science has been dedicated, in the last 30 years, to the analysis of program properties and to the invention of methods for partial correctness, exactly because these properties are not decidable (and very rarely partially decidable). Type Theory provides some of these methods and we will discuss it as well as the topological structures that it inspired.

The approach by Scott domains easily gives a notion of computable functional or function in all finite (higher) type. Call now SD the category of (possibly, effectively complete ω-continuous) Scott domains and continuous maps as morphisms. Observe then that any finite Cartesian product of these domains is still a domain (in each of the intended categories: continuous, ω-continuous, effectively complete ω-continuous ...). That is, they are all Cartesian Categories. Consider further two (effectively complete ω-continuous) domains D and D', then one may show that the set of continuous functions from D to D' yields an effectively complete ω-continuous domain, $SD[D, D']$. The construction of $SD[D, D']$ is (relatively) easy: partially order this set of functions pointwise, observe that the Scott topology is the pointwise convergence topology and give all the required enumerations.

But something more happens: products and function spaces nicely commute. That is, there is a natural isomorphism, in the sense of Category Theory, between the functors $SD[D \times \ -, C]$ and $SD[D, C^-]$ (where C^- is the contravariant exponent functor). In technical terms, these categories are Cartesian Closed. The natural isomorphisms guaranty that the higher type sets, e.g. $SD[D', C]$, are internally represented by the exponents; that is, $C^{D'}$ is an object of the category that "corresponds" to the set of function $SD[D', C]$. Thus, in each higher type one has, *internally*, continuous (and computable) elements, continuous (and computable) functions on them and so on so forth. Moreover,

a function of several variables may be seen as a continuous function of one variable at the time (or, componentwise continuity implies global continuity, as $SD[D, C^{D'}]$ and $SD[D \times D', C]$ are isomorphic).

However, does this approach yield a sound notion of higher type computation? In order to prove this, consider a more elementary or purely computational approach against which we can check the one above; an approach with no reference to topologies and approximations. Given two enumerated sets A and A', i.e. two countable sets and surjective maps (enumerations) $w : \omega \to A$ and $w' : \omega \to A'$, there is an obvious notion of"effective transformation" $F : A \to A'$. Just define F to be an **effective transformation**, if there exists a total recursive function f such that $F(w(n)) = w'(f(n))$. That is, F may be effectively computed over the indices. This category EN of enumerated sets and effective transformations is Cartesian but, and this is the rub, it is not Cartesian Closed. In short, there is no natural way to turn into an enumerated set the function space of two enumerated sets, or, in general, this construction does not commute with products, in the sense given above by a natural isomorphism.

The solution follows from the construction of Scott Domains. Consider the category CSD whose objects are the computable parts of effectively complete ω-continuous domains. That is, the computable elements D_c , D'_c in Scott domains D and D', enumerated by $w : \omega \to D_c$ and $w' : \omega \to D'_c$, respectively. By the construction above, this is Cartesian Closed, if one takes as morphisms the continuous and computable functions (as elements of the higher types). Then one has:

Corollary 2.7 *CSD is a full sub Cartesian Closed Category of EN.*

Proof. The only delicate point is in the proof that the embedding is full. But, by Theorem 2.5, the effective transformations on enumerated sets are continuous (the inverse image of an r.e. set by a total recursive function is r.e.). □

This fact shows that a purely recursion theoretic, or combinatorial, definition of higher type function (by using computable functions over the "indices", i.e. within EN) can be extended to all finite higher types. The definition actually commutes with products and yields the required uniformity in one or several parameters (componentwise computability and continuity), categorically called Cartesian Closure. This is obtained by embedding partial (recursive) functions into suitable topological spaces and by using continuity of functions and pointwise convergence topologies on their function spaces. Corollary 2.7 is a (broadly) generalized version of Myhill-Shepherdson theorem.

It should be clear that we are not following history here. As a matter of fact, Rice theorem was proved first, as a consequence of the halting problem ([Rice53]), then Myhill-Shepherdson and Rice-Shapiro theorems followed, but just over $P\omega$ and P, with no explicit reference to topology (see [Rogers67]). Note finally that a function on sets or functions (a type-two functional) transforms or "reduces" a set or a function to another: Myhill-Shepherdson functionals, say, define "enumeration reducibilty", while further "reducibility notions"

are given in purely recursion-theoretic terms (Turing reducibility, or reducibility by oracles, truth-table reducibility ..., see [Rogers67]). Yet, as proved in [BarendregtLo82]), by (small) changes in the (induced) topology on $P\omega$, one may characterize these notions of "recursion-theoretic reducibility". Once more, a "geometric" perspective unifies notions and results. And, more importantly, Scott Domains gave the general mathematical frame and extended the results for type-two recursive functionals to the so-called Denotational Semantics of typed and type-free Functional Programming Languages, see section 3.

2.3 Total Functionals, Filter Spaces and Real Numbers

As already mentioned, the notion of "partial map" is fundamental in Recursion Theory, in view of the early results on undecidability. Attempts to work directly with total maps lead to complex syntactic approaches. The reason is first due to the impossibility to enumerate effectively the total recursive functions; thus only some subclasses (e.g. the primitive recursive functions) bear some interesting recursion theoretic analysis. As for higher types, Geometry, by an analysis of convergence, spells out the reasons for the mathematical difficulties. In [Hyland77] an elegant approach is proposed by Filter or Limit Spaces, à la Kuratowski. A Filter Space is a set with a local "notion of convergence" (for each point, a filter of filters of subsets); the category Fil has Filter Spaces, as objects, and, as morphisms, all maps transforming converging filters (towards a point) into converging filters (towards its image). Any Topological Space is a Filter Space with, at each point, the filter of neighborhoods as least filter (of convergence); this yields an adjunction from Top into Fil. Yet, not any collection (filter) of converging filters has a least element; that is, there are interesting objects of Fil, which are not in Top. For example, the type two *total* recursive functionals, formally defined by Kleene by 9 extremely complex axioms, [Kleene59]. By Hyland's "limit approach", some feasible (and elegant) Mathematics begun also on hereditarely total maps (see [Normann80]). More recent consequences use this very weak notion of convergence and the following approach to computability over metric spaces.

In example 2.4 above, the reals, with the Euclidean metric, were embedded, as maximal elements, into a Scott Domain (the so-called "upperspace"). This is a paradigmatic treatment of classic structures within a suitable frame for computability and opened the way to some relevant effective analysis of continua. Computable reals, say, are exactly the maximal computable elements of the upperspace, as an effectively complete ω-continuous domain. The work by Edalat and others (see [Edalat95A], [Edalat95B], [Escardo96]) allowed to embed several structures for Effective Analysis, including some Geometry of Dynamical Systems, into Scott's categories, where closure properties under higher type constructions and existence of effective limits have been used to revisit several known results, in a constructive frame (effective versions and/or improvement of Banach fixed point theorem, Hutchinson's theorem, existence of Markov operators, see [Edalat97B] for a survey). A general idea, for example, is to look at spaces of open balls in a metric space as a Continuous Domain, partially

ordered under reverse inclusion, as in the upperspace over $[0,1]$ (example 2.4); then Ruelle's notions of "attractivity" and "invariance" can be described as convergence and fixed points and dealt with in a constructive way, also in higher types (see below and [Edalat97A]).

A blend of the two approaches above has been recently proposed in [DeJaeger02]. The filter structure on reals and, hereditarely, on their functions and functionals, gives a sound notion of effectiveness in higher types (functionals and measures over the reals) by the combined use of weak forms of convergence (filters) and strong algebraic-geometric properties (convexity and the vectorial structure of the reals and their continuous functions). But this work belongs already to the new century.

3 Types

As already mentioned, Computability Theory was born by the encounter of formalist and intuitionistic views on foundations. In modern terms, this may be understood by the "Types as Propositions" analogy (also known as the Curry-Howard isomorphism below). In [Church32] a simple formal language for computing was defined, the type-free λ-calculus. The purpose, as for Herbrand, Gödel, Turing and the others in the '30's, was the understanding of formal deductions as effective/finitistic procedures. Church's elegant theory of computable functions only relies on two notions: given the formal definition of a function by a term, bind (abstract) a variable by a λ; apply a term to another term. Briefly, $\lambda x.f(x)$ is an (explicit) function of x and when applied to a, say, it gives $f(a)$.

More formally: variables are terms; if M is a term, then $\lambda x.M$ is a terms; if M and N are terms, then MN is a term. Warning: also xx is a term! This immediately produces a fixed-point operator: $Y = \lambda y.(\lambda x.y(xx))(\lambda x.y(xx))$. As a consequence, Curry gave a simple antinomy within the calculus, which originally included an negation operation, Neg. As an informal paraphrasis of it, note that $YM = M(YM)$ for each term M, thus $Y(Neg) = Neg(Y(Neg))$ contradicts any reasonable property for negation or complement. It was simple to give a solution to the antinomy (yet another obtained by a diagonal construction): avoid complementation in the calculus, as this remains very expressive. As a matter of fact, by a simple encoding of numbers as terms, type-free λ-calculus computes all (Turing) computable functions: it actually served as a tool for proving the equivalence results between the various formalisms, in '35-'36 (the Y operator above yields recursion). The lack of negation may be then understood as the fact that recursively enumerable sets are not closed under complement, an easy consequence of Turing's undecidability of the Halting Problem.

Yet, there is another relevant way out from antinomies: give types to variables, and thus to terms as suggested by [Church41]. Actually, one can introduce and eliminate types and define terms at once, by two rules:

$$\frac{\begin{array}{c}[x:A]\\ \vdots\\ b:B\end{array}}{(\lambda x:A.b):A\to B}\ (\to I) \qquad \frac{a:A \quad c:A\to B}{ca:B}\ (\to E)$$

The reader will recognize in $(\to E)$ the classic 'modus ponens': if A and A imply B, then B. While the rule $(\to I)$ transfers into the language of formulas the metalinguistic deduction: that is "from A infer B" is transformed into an implication, $A \to B$. By this very construction, the types of terms are also propositions of a simple implicative calculus. Just add products "$\times$" as conjunction and you have Intuitionistic Propositional Calculus, with the obvious rules that characterize the Cartesian projections. This was Curry-Howard's remark in the '60's that gave full logical meaning to Church's typed version of λ-calculus. The idea is that terms living in types are *proofs* of these types, seen as propositions: $(\lambda x : A.b) : A \to B$ is a proof of $A \to B$ in the precise sense that it takes a proof of A and transforms it into a proof of B, as expressed by $(\to E)$.

Note now that, by definition, several argument functions in λ-calculus are given componentwise. If M (possibly) contains x and y, the function explicitly depending on them as variables is obtained by successive binding by λ: $\lambda x.\lambda y.M$. Thus, as for TM's or any other calculus for recursive functions, computability is defined componentwise; actually, in this calculus, the types $A \times B \to C$ and $A \to (B \to C)$ are provably isomorphic (there exists invertible terms in either direction). Clearly, this isomorphism has a precise logical meaning: as propositions, it is easy to prove that $A \times B \to C$ and $A \to (B \to C)$ are equivalent. This is a basic example of the connection between computations (as terms) and proofs, thus between types and propositions; their isomorphism is constructively given by the terms proving their equivalence as propositions.

And here come again the Cartesian Closed Categories (CCC's), as this isomorphism is the natural isomorphism, which defines them (see above). Thus, Logic and computations force this topologically un-natural feature: global continuity is equivalent to componentwise continuity, a mathematical translation (or semantics) of the computational and logical isomorphism above. Yet, the construction of Scott Domains gave a simple answer to the problem, in topological terms. And the construction was far from arbitrary: functions, as infinite objects, are naturally partially ordered by graph inclusion (and this yields the pointwise convergence topology). Finite functions (functions of finite graphs) provide the finitary (algebraic) approximating elements, hereditarily extended to higher types by the categorical construction; theorem 2.5 and its corollaries above further justify it. Clearly the price to be paid is that "topological approximation" is defined only from below: these topologies are order topologies, and not Hausdorff.

Besides this general motivation for continuity (approximation), there is a further motivation for Scott topologies, at least up to these days. The type-free λ-calculus is expressive enough to compute all partial recursive functions: this is easily show by implementing recursion within the calculus, by Curry's fixed-point operator $Y = \lambda y.(\lambda x.y(xx))(\lambda x.y(xx))$. Recall that, in this calculus,

functions may be inputs as well (any term can be applied to any other term: there no difference of type). Formally, the domain of terms, D, is isomorphic to the domain of functions on terms, $D \to D$. But how to find a solution D to the equation (isomorphism) $X = X \to X$?

By Cantor's cardinality theorem, this is far from obvious, if we want D to be different from the singleton set (e.g. to contain all natural numbers and computable functions). Well, in Scott Domains, by an inverse limit construction, one can find a solution to this equation, as isomorphism between a domain and its function space (the continuous functions or morphisms in SD), called the D_∞ model in [Scott72] (see [Barendregt84], [AbramskyJung92] for surveys and references). A simpler, but weaker, solution, on $P\omega$ actually, is given in [Scott76] (for general category-theoretic and set-theoretic analyses of continuous domains and their equations, see [SmythPlo82] and [Longo83], respectively). It is worth noticing that, so far, while there are plenty of interesting Cartesian Closed Categories (thus, models of the *typed* λ-calculus), no model has been given of the type-free calculus, without some use of continuity and topological notions.

Clearly, the very simple order topology of these models has an immediate advantage. Recursive functions are defined by ... recursion; that is, a function f is usually given by a formal expression, a program, a "context of definition" F and an equation $f = F(f)$. Thus, partially ordered spaces, which are (effectively) complete, easily yield a solution f_0 of the equation: once interpreted F by a continuous, thus monotone map F, take the least upper bound of the chain $\bot = F(\bot) = F(F(\bot)) = \dots$, where $\bot$ is the least element of the partial order. The (continuous and computable) functional Fix, defined by $Fix(F) = \sup_n F^n(\bot)$, provides a simple mathematical interpretation of the formal operator Y above. Thus, a solution of the formal equation $f = F(f)$ is given in these topological spaces as $f_0 = Fix(F)$

As for the type equation $X = X \to X$, or more generally $X = \Sigma(X)$, where Σ is a type constructor or a recursive definition of data types, one may try to use a similar techniques, at the level of the category, by interpreting Σ as an (endo-)functor. However, the key case above - and most interesting ones - use functors that may be covariant in one argument (the second one), but contravariant in the other: thus, no obvious extension of the monotonicity argument can be given. Scott's '72 paper, by an original inverse limit construction, started the Denotational Semantics of Programming Languages, as the mathematical investigation of recursive definitions of functions and, more importantly, of data types: this crucial practice of programming found by this a proper mathematical frame, while the first (topological) model of type-free λ-calculus was given, i.e. a solution of the equation $X = X \to X$.

In conclusion, types of (typed) λ-terms are propositions; their mathematical meaning is that of objects in categories with some closure properties, that relate products and function spaces. Some of them contain non-trivial solutions of the equation $X = X \to X$. These correspondences (types as propositions as objects of CCC's) greatly enriched Proof Theory as well, in particular since the work of Girard and Martin-Löf, in the '70's. On the side of applications, type-free and typed λ-calculi have been the paradigms for Functional Languages,

one of the main programming styles, for more than 30 years. The main dialects of LISP, born in the '60's, directly derived from the type-free calculus; Edinburgh ML, an extension of the typed calculus of the mid '70's, started a major flow of programming languages. By the notion of *polymorphism*, due in Proof Theory to Girard's second order λ-calculus, novel and rigorous reusability tools moved into programming (see [Girard89], [Castagna97] for surveys). Moreover, their mathematical semantics provided connections to further non-trivial constructions, which also originated in Geometry: Toposes.

In short, polymorphism may be understood as the possibility, for a function (a term) to live in many types. The identity function, for example, belongs to $X \to X$, for any X; or, one should be able to write just one program computing it, in all types. The idea is that types may include "variable types". Then, both types and terms may depend on types, as variables: e.g., $(\lambda X.\lambda x : X.x) : (\forall X.X \to X)$ is the polymophic identity function, or the program for this function may be "reused" over all types. More generally, if A is a type, also $\forall X.A$ is a type, for X ranging over types (formally: $(\forall X : Type.A) : Type$). Thus one may define a type by a quantification over the very collection of types that is being defined, see [Girard89]. This challenging impredicative feature is the reason for the expressivity of the calculus (Girard's system computes all recursive functions which are provably total in II order Arithmetic, and thus it proves the consistency of - II order - Arithmetic), as well of its success, due to the idea of "reusing" as polymorphisms or variable types, with no restrictions (no stratified universes, for example).

Girard's second order calculus (with type variables) may be interpreted over toposes (where variation is described as "fibration"), with a non-obvious closure property, which may be informally expressed as follows. The intended category must be closed under fibred or indexed products *over itself*. That is, it must contain, as an object, the result of a product indexed over the entire collection of objects ("small completeness"). The first construction of this kind was proposed by Moggi, in 1985, and developed by many since then ([LongoMoggi91], [Hyland87], [Pitts87]; see [AspertiLongo91] for an approach via "internal categories" and more references). Moggi's idea may be accommodated into several realisability toposes, a natural generalization of computability in categorical/ intuitionistic frameworks, which contains various versions of Scott Domains. Their logic is essentially intuitionistic and they allow strong closure properties exactly because they yield "very few morphisms" (only the "constructive ones"), see [Birkedal02] for surveys and recent advances.

Realizabilty toposes do not necessarily relate to continuity notions. However, their origin is due to Grothendiek's idea of topos, which extended the gluing technique of Riemann, for differentiable functions on manifolds, to more general structures, see [MacLaneMoer92]. Lawvere made the connection to Logic, [Lawvere76]. The main theorem in [LongoMoggi91] uses Lawvere's understanding of (second order) universal quantification as right adjoint to the diagonal functor (note that existential quantification is understood as the left adjoint: a very elegant "geometric" understanding of logical quantification, by a symmetry).

Intermezzo: Logic, Concurrent Processes and Geometry

Logic and Sequentiality

As mentioned in the Introduction, the logico-formal origin of Computability Theory is reflected into the topological structures that were first proposed to understand it (as already hinted, since [Frege84], foundations only relied on "logic judgements", with the exclusion of "geometric judgements", as hypothesis grounded on key regularities of space, in the sense of [Riemann54], [Weyl27]; see [Longo02] for a discussion). The componentwise analysis of continuity, as Cartesian Closure of the intended categories, the weakness of the topology (T_0 separation) ... are all symptoms of this "lack of attention" to physical space (and its cartesian dimension) which is typical of this theory. The sequentiality of (classical and intuitionistic) logical deduction is at the core of it. TM's are conceived as a human computor sequentially writing/erasing/moving 0's and 1's. And theorems prove that λ-calculus is a sequential system (see [BerryCurien82], [AmadioCurien98]). Also the results mentioned in subsection 2.2 have an essentially sequential nature. A close analysis of the proof of the "effective discontinuity lemma" (see the comments after theorem 2.5) shows that its proof relies on the sequential generation of an r.e. set: a bifurcation towards a superset or a finite subset is decided, with no other alternatives, along its effective production; and this seems crucial to the construction. As a further remark, recall that the type-free λ-calculus may be considered as the cleanest functional theory of computations, as it represents all partial recursive functions with the simplest syntactic structure. Observe then that all its models force the strong isomorphic embedding that we considered as a paradigm of "being unrelated to physical space" (as dimension is a topological invariant): in any Cartesian Closed Category, a solution D of the equation $X = X \to X$ is such that $D \times D$ can be isomorphically embedded into D (a simple consequence of the syntax or of the categorical characterization in [LongoMoggi90]).

Yet, the deep links between Computability, Cartesian Closure and weak topologies, as concrete models for types and computations, have had lots of remarkable fall-out. Functional programming, since the design of (dialects of) LISP (such as Scheme, and their models) and even more as for Edinburgh ML, till the more recent versions of polymorphic programming languages, have been invented and enriched on the grounds of this mathematical understanding. As already hinted, some of the related ideas passed into Object Oriented progamming: for example, subtyping helped to formally describe inheritance (the features of programs may be inherited and reused, a further form of polymorphism, see [Castagna97]). But even Logic has been influenced by these structures. On the grounds of early work by Berry, in [Girard86] it was remarked that one can give models for typed and type-free computations by considering a restricted class of continuous functions as morphisms, the stable ones. In the categories of Berry-Girard Coherent Domains, the key property of "finiteness", interpreted by the algebraic elements of Scott Domains, is nicely preserved, as proper finiteness also in higher types (there are only finitely many

elements below a finite - algebraic - one; see [AmadioCurien98] for an account). And this was the beginning of new ideas in Proof Theory: Girard's Linear Logic was conceived by observing that stable maps include some interesting *linear* ones, [Girard87]. In this case, then, the mathematical structure preceded and suggested the logic formalization: the linear maps interpret a novel, very fruitful, notion of implication.

Girard's work went further on, in this revision of classical and intuitionistic systems. Linear Logic originated an original analysis of proofs, in terms of their geometric structures: proof-nets describe rules and proofs also on the grounds of *symmetries* and *connectivity*. The logic machinery of the founding fathers is thus enriched by some fundamental geometric judgements that bring space back into Foundations, through the window of deductions. A further insight on the relevance of spatial organization in Logic is given in [Girard01].

Concurrency and Geometry

It is clear that Turing machines may run in parallel, and that one machine can simulate any finite number of them. This is parallelism and its use largely enhanced complexity, but did not require any change in paradigm. The situation, in actual computing, changed dramatically since distributed systems and networks of computers came to the limelight. In these systems, possibly asynchronous processors concur in performing a task. That is, they exchange messages that may modify each individual computation, in a largely unpredictable way, in particular in view of the lack of an absolute, Newtonian clock. Distribution in space is crucial, as well as a richer understanding of physical time. Their mathematics is changing accordingly. A wide effort is carried on in order to provide a suitable mathematical frame for Concurrency (see [Aceto02] for recent approaches and surveys). Categories play a major role, see [WinskelNiel95], [Joyal96] among others, but we are far from a unified understanding of a very complex phenomenon, where computations are no longer compositional (the result of truly concurrent processes cannot be analyzed by composing the independent analyses of individual processes; this radically departs from Turing's approach[1]).

Even access to stored data is becoming a major mathematical challenge: competing independent processors, in a distributed systems, may modify a database while accessing to it and, by this, influence the other computations. A recent geometric approach to this relevant and difficult problem uses non-trivial tools from Homotopy Theory. The idea is that the allowed computations may be seen as homotopic paths in a continuous phases space (see [Goubault00]). Fault tolerance, typically, is analyzed in terms of small continuous deformations along oriented paths. Space is finally back and true geometry follows. The point is that Computer Science is moving from the discipline engendered from

[1]The number-theoretic functions computed by these systems are the usual ones, of course. The point is that, in Concurrency, what really matters is the data flow, not the input-output relation. Church Thesis, the thesis for the old absolutes of the 30's, for Computing (and Mind (!!)), may be soundly considered the "tar-pit" of modern (mathematical) computing, see [Aceto02].

pure Logic, an heritage of the spaceless foundations à la Frege and Hilbert, focusing only on sequential languages for deductions, Integer Numbers and their Arithmetic, to an analysis of actual computations by concurrent agents, distributed in space and computing along a dynamical understanding of time (time as scanned by irreversible and asynchronous bifurcations). The founding fathers of Logic, one century ago, opened the way, by their mathematical inventions, to the early logico-formal machines; but these machines are changing nowdays. And it is time to move towards new ideas for the "next machine", possibly grounded on physical space and its mathematics. A good source of inspiration may be another relevant machine, rather unrelated to the digital ones, the animals' neural system (which includes ours, of course). This machine originated from *action in space*, along the evolution of life, and its elaboration of information seems more to be a dynamics of forms, of a geometric nature (from the tridimensional folding of proteins, as neuroreceptors, to the structure of neural nets - see the project "Géometrie et Cognition" in Longo's web page; but this is yet another possible program for the new century).

4 The Hereditary Partial Effective Functionals

In this section and in the following one, two "finitary" statements are given, which have been proved by an explicit use of the continuous structures we previously introduced. It is fair to say that "analytic" methods have been used, for elements of domains are described by finite approximations, similarly to approximations by finite expansions of Fourier series in complex function theory. Of course, we do not yet deal with the width and the difficulties of number theory: the Theory of Numbers is the oldest and most basic topic in Mathematics, since Pythagora's school. Computability Theory is surely not as old as that[2].

More precisely, this section presents an elementary (arithmetic) notion of higher type computation. The basic properties of the type structure are proved by an essential use of continuous domains for denotational semantics and generalized computability. In section 5 below, a (basic) fact for the solution of recursive equations between programs is discussed. This is done for our core functional language, λ-calculus. The proof uses a non-axiomatizable ("infinitary") consistent extension of λ-calculus and a continuous topological model. Unfortunately, some technical definitions are unavoidable.

Gödel and Kreisel's motivation for classical higher type recursion theory refers to the analysis of the meaning in constructive mathematics of Peano or Heyting arithmetic (see Kreisel's paper in [Heyting59]). Recall first that $\forall x \exists y\ A(x,y)$ (constructively) yields a function f from x to y, i.e. it corresponds to $\exists f \forall x\ A(x, f(y))$. More generally, following Kreisel's "no counterexample" interpretation,

[2]The two theorems in sections 4 and 5 have been proved by [LongoMoggi84] and [BergstraKlo80], respecively. In [Longo84] they were already presented in a unified perspective.

from $\exists x \forall y \exists z \forall w\ A(x,y,z,w)$
write $\exists F \exists G \forall f \forall g\ A(F(f,g), f(F(f,g)), G(f,g), g(F(f,g).G(f,g)))$,
where F, G, f, g are function variables in the due type (type 1, the type of number theoretic functions, as for f and g; (pure) type 2, as for the functionals F and G, since they are functions of functions). Then x and y are effectively recovered from f and g, if F and G are (higher type) total *computable* functionals, and so on so forth for further alternations of quantifiers. Thus the need to define this sort of computable maps, in any finite higher type.

Recall now that the partial recursive functions (PR) can be *effectively* numbered (notation: $PR = \{\phi_i\}_{i\in\omega}$); thus, also the universal function $u(i,x) = \phi_i(x)$ is in PR. By using this idea, in [LongoMoggi84] a direct elementary approach was proposed to higher type partial functionals (the HPEF below).

The difference w.r. to Scott-Ershov approach (or Hyland's, for total functionals) is that the later first introduces the notion of continuity, possibly in abstract categorical settings, then defines the continuous and computable objects in any type, by using also PR. The HPEF instead are presented by an inductive elementary construction, only based on PR. Interestingly enough, the key structural property of the HPEF, which may be expressed in purely arithmetical terms, could only be proved by using continuous structures. This may be considered a further motivation to the invention of the topological category of domains.

4.1 The problem

For sets A, B, write $f : A \to B$ if f is a total map from A to B. By setting $\omega^\perp = \omega \cup \{\perp\}$, we may write $f : \omega \to \omega^\perp$ for any partial number-theoretic function ($\{\perp\}$ means "divergence"). As for higher types, integer (or, respectively, pure) types are defined by $n+1 : n \to n$ (or, respectively, $n+1 : n \to 0$), for $n \in \omega$.

Let $<,>$ be any effective coding of pairs in ω; that is $<,>$ codes a pair of numbers by a single number. Set $C^{(0)} = \omega$ and $C^{(1)} = PR$. In order to define $C^{(n)}$, the key idea is to extend to a map from $C^{(n-1)}$ to $C^{(n)}$ the notion of gödel-numbering, which is welknown as a map from $C^{(0)}$ to $C^{(1)}$

We will define then $C^{(n)}$, for all $n > 1$, as the set of **hereditary partial effective functionals** (HPEF) of integer type n, by induction on n, by using a set $C^{(n.5)}$ of maps from $C^{(n-1)}$ to $C^{(n)}$ (functions of "intermediate" type). The maps in $C^{(n)}$ will go from $C^{(n-1)}$ to $C^{(n-1)}$. The coding $<,>$ is extended to higher types in one of the several possible ways. In other words, the extented $<,>$ codes $C^{(n)} \times C^{(n)}$ by $C^{(n)}$, for any n, as it will be shown below.
Notation:

$$\lambda xy.g(x,y) \text{ is the map } < x, y > \vdash g(x,y).$$

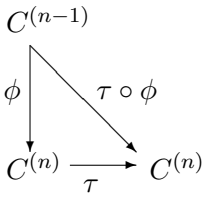
Definition 4.1 *(i)* $C^{(n.5)} = \{\phi : C^{(n-1)} \to C^{(n)} \mid \lambda xy.\phi(x)(y) \in C^{(n)}\}$;
ii) $C^{(n+1)} = \{\tau : C^{(n)} \to C^{(n)} \mid \forall \phi \in C^{(n.5)} \tau \circ \phi \in C^{(n.5)}\}$.

In order to see that this definition makes sense, let's check the types.

Assume that $C^{(n-1)}$ and $C^{(n)}$ are given. Consider first $\phi : C^{(n-1)} \to C^{(n)}$ and set $\psi(< x, y >) = \phi(x)(y)$, for $x, y \in C^{(n-1)}$ (check the typing for exercise:

e.g., observe that $\phi(x) : C^{(n-1)} \to C^{(n-1)}$). Clearly $\psi : C^{(n-1)} \to C^{(n-1)}$. Then, by (i) in the definition above, $\phi \in C^{(n.5)}$ iff $\psi \in C^{(n)}$.

The following diagram should help in understanding how $C^{(n+1)}$, the "next" higher type, is defined in (ii), by using $C^{(n.5)}$, which has been defined in (i) from the given $C^{(n-1)}$ and $C^{(n)}$:



That is, $\tau \in C^{(n+1)}$ iff $\forall \phi \in C^{(n.5)}\ \ \tau \circ \phi \in C^{(n.5)}$, as in (ii) in the definition.

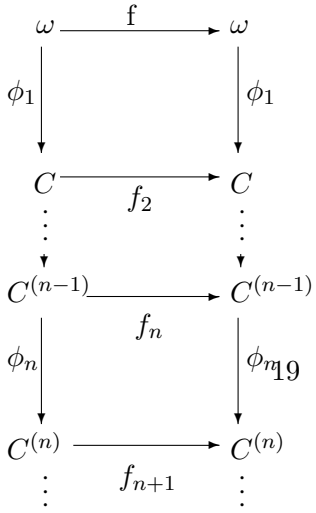
Note that the pure type functionals $\{PC^{(n)}\}_{n\in\omega}$, i.e. for $n+1 : n \to 0$), may be also defined, by the technique above: just substitute $PC^{(n)}$ in the second instance of $C^{(n.5)}$ in the definition of $PC^{(n+1)}$ corresponding to definition 4.1 (ii) (see [LongoMoggi84]; 3.12)). In either cases, one only needs to know $C^{(0)} = \omega$ and $C^{(1)} = \mathrm{PR}$, to start with. Apparently, though, it may seem that the full strength of Set Theory is needed in order to define the sets $C^{(n.5)}$ and $C^{(n+1)}$ above. As a matter of fact, a consequence of the results mentioned below is that, for all $n, C^{(n)}$ and $C^{(n.5)}$ are countable and can be effectively numbered. By this we do not need to look, say, at the set of *all* functions from $C^{(n-1)}$ to $C^{(n)}$ in order to define $C^{(n.5)}$, but a countable subset would do. A posteriori, this is all expressible in Peano Arithmetic.

We want to know, now, what the $C^{(n)}$'s are and how they behave, for $n > 1$. It is easy to see that all acceptable Gödel-numberings of $C^{(1)} = \mathrm{PR}$ live in $C^{(1.5)}$. It would be nice, say, if one could extend the notion of effective numbering to maps in $C^{(n.5)}$, for $n > 1$. In view of the s-m-n (iteration) theorem (see [Rogers67]), this corresponds to proving the property (P) below. This property says that each $C^{(n.5)}$ contains a function ϕ_n to which any other function, in the same type, can be reduced (by a function $f_n \in C^{(n)}$). Function ϕ_n palys the role of "generalized gödel-numbering".

$$(P) \qquad \forall n \geq 0\ \exists \phi_n \in C^{(n.5)}\ \forall \phi \in C^{(n.5)} \exists f_n \in C^{(n)} \phi = \phi_n \circ f_n$$

Recall now that for all $f_{n+1} \in C^{(n+1)}$ one has $f_{n+1} \circ \phi_n \in C^{(n.5)}$. Then by (P), for some $f_n \in C^{(n)}$, $f_{n+1} \circ \phi_n = \phi_n \circ f_n$.

By this, HPEF may be visualized in the integer types by the following diagram:



In order (P) to hold, the ϕ_n's have to be surjective as well and, by this, preserve the cardinality of ω at higher types. Again, (P) is elementary and has a combinatory flavour.

4.2 The method

In this section we give the basic hints for the main result concerning the HPEF, with the only aim to display the use of continuous topological structures in the proofs. In [LongoMoggi84], (P) has been proved by an heavy induction load. Note first that, by looking only at definition 4.1, the HPEF do not need to be ... well defined. As a matter of fact, $<,>: \omega^2 \leftrightarrow \omega$ is not trivially inherited at higher types. A first sight, one would just set $< f, g > (n) = < f(n), g(n) >$ as a definition for $<,>: C \times C \leftrightarrow C$ and so on. This is clearly wrong, for divergence of f or g cause essential problems. Set then $< f, g > (n) =$ **if** n is even **then** $f(\frac{n}{2})$ **else** $g(\frac{n-1}{2})$.

This is fine for $C^{(1)}$, but it doesn't automatically prove that
$< \tau, \sigma > (f) = < \tau(f), \sigma(f) >$
defines an (effective) coding of $C^{(n)} \times C^{(n)}$ into $C^{(n)}$, for $n \geq 2$. The proof of this goes inductively together with the proof of (P) and of two more facts, which need a few definitions.

Recall that a numbered set is a pair (D, e) where $e : \omega \to D$ is onto. $f : D \to D'$ is a **morphism** (of the numbered sets $(D, e), (D', e')$) if $\exists f' \in R$ $f \circ e = e' \circ f'$ where R are the recursive functions. Define then $d : \omega \to D$ is an **acceptable numbering** of D if $\exists f, g \in R$ $e = d \circ g$ and $d = e \circ f$ (or, d and e are recursively equivalent). Now, one can enumerate D' from D: that is, $\phi : D \to D'$ is a **relative numbering** (of (D', e') w.r.t. (D, e)) if $\phi \circ e$ is an acceptable numbering of (D', e'). Finally, $\phi : D \to D'$ is a **principal numbering** (of (D', e') w.r.t. (D, e)) if ϕ is a morphism and $\forall \psi \in Mor(D, D')$ $\exists \theta \in Mor(D, D)$ $\psi = \phi \circ \theta$. Clearly acceptable, relative and principal numberings are onto maps and generalize Gödel-numberings.

Consider now $E = Cont(\omega, \omega^{\perp})$, where ω is given the discrete topology. Clearly, E is the effective domain of the partial functions $E_c = PR$. Set $E^{(n+1)} = Cont(E^{(n)}, E^{(n)})$. We can then state the main theorem in [LongoMoggi84].

Theorem 4.2 *For all $n \geq 1$ one has:*
(1) $C^{(n)} = E_c^{(n)}$
(2) $C^{(n)} \times C^{(n)} \cong C^{(n)}$ via $<,>$
(3) $C^{(n.5)} = Cont(E^{(n-1)}, E^{(n)})_c$
(4) $\exists \phi \in C(n.5)$ principal relative numbering.

Properties (1) - (4) are proved by combined induction. (1) and (3) give the continuity of the HPEF. More precisely, (2) at type n+1 is obtained from (1) + (2) at type n. by first showing that, in an effective domain, any effective coding of pairs of algebraic elements by algebraic elements is inherited to a corresponding effective coding at the higher type (the function space). (1) + (3) + (4) give (1) at the higher type. (1) + (2), again, prove (3) and so on, with various combinations, to prove the rest.

Note, now, that once proved that each $C^{(n)}$ is a countable set, with the natural numbering of constructive domains, (P) above amounts to say that each $C^{(n.5)}$ contains a principal numbering. In fact, by generalized Myhill-Shepherdson theorem (see Corollary 2.7), the continuous and computable functions over the computable elements of effective domains coincide with their morphisms, as numbered sets.

There may be direct arguments for proving the arithmetic statement (P), i.e. with no use of the continuous structure. In [LongoMoggi84], though, it seemed essential to use finite approximants (algebraic elements) and their effectiveness properties to deal with the functions in the $C^{(n)}$'s, which are infinite objects.

Remark. Topological or limit structures are implicit in the definition of the Kleene-Kreisel countable functionals, when given by the "associates" (see [Normann80]). This is not so for the HPEF, where in the definition, there is no flavour of continuity nor approximation. Though, as soon as we just want to check if the definition makes sense in any type, finite approximants, that is the algebraic elements of suitable domains, come in. Of course, Myhill-Shepherdson theorem and its generalization is the key fact to understand this; once more, that theorem provides the main justification for Scott's continuity and its deep connections to computability.

5 Invertible terms in λ-calculus

Terms of λ-calculus are defined by variables, formal application and the binding operator λ. The computation (reduction) rules are axiomatized as follows:

$$(\beta) \quad (\lambda x.M)N \geq [N/x]M \quad , \quad \text{for } N \text{ free for } x \text{ in } M, \text{ as usual;}$$

$$(\eta) \quad \lambda x.Mx \geq M \quad , \quad \text{for } x \text{ not free in } M.$$

The reduction predicate "$\geq$" is reflexive, transitive and substitutive. "$=$" is the least equivalence relation such that $M \geq M' \Rightarrow M = N$; "$=$" also is substitutive. A term is in **(β-)normal form** ($\beta\eta$-normal form) if no subterm is the LHS of axiom (β)((β) or (η)). A term M **possesses a ($\beta\eta$-) normal form** if, for some M' in ($\beta\eta$)normal form, one has $\lambda\beta(\eta) \vdash M = M'$. Finally, M is **solvable** (or has head normal form), if, for some $P_1, ..., P_q, \vec{x}$ and y, $\lambda\beta \vdash M = \lambda x_1 ... x_n.yP_1...P_q$, M is **unsolvable**, otherwise. The paradigmatic role of λ-calculus and Combinatory logic, as functional programming languages, is well known since long, see [Backus78].

5.1 The problem

Combinatory Logic and λ-calculus, as type free applicative languages, have a set of terms which forms a monoid with $I \equiv \lambda x.x$ as identity and composition "$\circ$" as application, where $M \circ N = BMN$ for $B = \lambda xyz.x(yz)$. Church proved that the closed terms of $\lambda\beta\eta$-calculus form a recursively presented monoid, with an unsolvable word problem. Note, now, that equations among terms give

a standard methods for defining terms, as well as for programs. Equations, though, as well known from algebra, can be more easily solved when working in a group. Consider, for example, the following definition of the (unknown) term $X : M \circ X = X \circ P$. If M, say, possesses an inverse, M^{-1} , an easy application of Curry's fixed point combinator gives X from $X = M^{-1} \circ X \circ P$, since we obtain an ordinary recursive definition of X. It is then an important question, raised by [CurryFeys58], under what conditions a term in the monoid has an inverse.

Definition 5.1 *Let T be (an equational extension of) λ-calculus. M is T-***invertible** *if $\exists N\ T \vdash M \circ N = I$ and $T \vdash N \circ M = I$. (In this case we that M, N are T-***inverse***).*

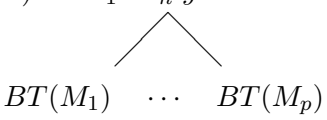
[Dezani76] gives a characterization of invertible terms in $\lambda\beta\eta$-calculus, among those which *possess a normal form* (for $\lambda\beta$-calculus, the group coincides with I). This was done by an (implicit) use of the notion below of tree for terms.

For the purposes of the main theorem of this section, a fundamental definition is required: a term of λ-calculus may be seen as a tree, by taking as root the "head" of the term (e.g. $\lambda x.y$ in $\lambda x.yM$), hereditarely.

Definition 5.2 *(Informal) The* **Böhm tree** *of M is given by:*

$BT(M) = \Omega$ *if M has no head normal form (i.e. M is unsovable)*

$BT(M) = \lambda x_1 ... x_n.y \qquad if\ M =_\beta \lambda x_1 ... x_n.yM_1 ... M_p$



The definition is informal, since Böhm-trees may be infinite (see [Barendregt84] for a precise definition). In particular, $BT(M)$ finite and Ω-free iff M has a normal form. Thus, there are plenty of interesting terms with infinite Böhm-trees. Take, say the fixed point combinator $Y = \lambda y.(\lambda x.y(xx))(\lambda x.y(xx))$ or the solution of $X = \lambda z.zX$, and a lot more. Since early work by Wadsworth and Hyland, Böhm-trees have been a basic tool in the comparison of operational and denotational semantics of λ-calculus (see [Barendregt84], [Longo83]). Roughly, this is because all terms, as all programs, are finite Böhm-trees "display" their computational (operational) behaviour, which may be infinite (see [Longo84] for a discussion).

Böhm-trees may be partially ordered by setting Ω, the undefined tree, as the least one and, then, by setting $BT(M) \subseteq BT(N)$ if $BT(N)$ is obtained from $BT(M)$ by replacing Ω in some leaves of $BT(M)$ by some Böhm-tree. Dezani's result (see theorem 5.4, below) uses a difficult combinatory technique, where the finiteness of the Böhm-trees considered has an essential role. The question remained of characterizing all invertible terms, not necessarily with normal form; equations like the example above are perfectly sound also if $BT(M)$ is infinite, for M could still possess a head normal-form.

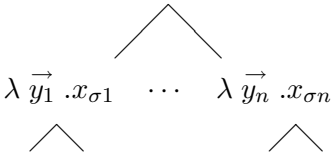
[BergstraKlo80] fully answered the general questions, by using results in [Dezani76][3]. Our present interest in the methods used in [BergstraKlo80] relies on the essential application made of infinitary, continuous structures to solve this typically combinatorial problem. What's nice is that the characterization in [BergstraKlo80] confirms the result in [Dezani76], since invertible terms turn out to live only among normal forms. To see this, though, one needs to look also at terms with an infinite computational behaviour.

5.2 The method

Let $\sigma_1, ..., \sigma_n$ be a permutation of $1, ..., n$ and $\vec{z}$ be a finite string of variables.

Definition 5.3 *(i) The* **hereditary permutators** *(HP) is the set of λ-terms defined by:*

$$M \in HP \quad if \quad BT(M) = \lambda z x_1 ... x_n . z$$



and each subtree is the tree of a term in HP.

(ii) The **finite hereditary permutators** *(FHP) are the terms in HP with a finite, Ω-free Böhm-tree.*

Theorem 5.4 *Assume that M has a normal-form. Then M is $\beta\eta$-invertible iff $M \in FHP$.*

Proof. [Dezani76] (see also [BergstraKlo80]). □

Theorem 5.5 *M is $\beta\eta$-invertible iff $M \in FHP$.*

Proof. [BergstraKlo80] (see also [Barendregt84]). □

It is clear that in order to prove theorem 5.5 from theorem 5.4 one only has to show

(O) M is $\beta\eta$-invertible $\Rightarrow$ M has a normal form,

since any M in FHP automatically has a normal form.

Bergstra and Klop give two arguments for (O), by embedding λ-terms into different continuous models, D_∞ and $P\omega$ (see section 3). Observe first that the valid equalities between λ-terms in $P\omega$ are characterized by the following theorem.

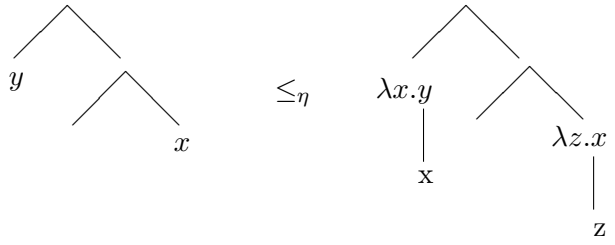
Theorem 5.6 *(Hyland) $P\omega \models M = N$ iff $BT(M) = BT(N)$.*

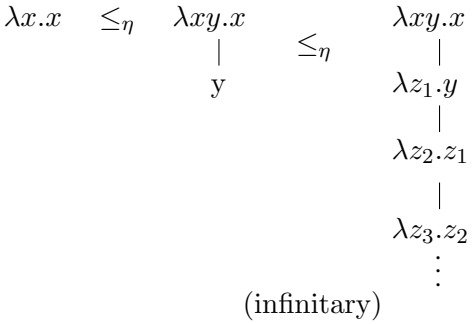
Proof. (see [Barendregt84]). □

[3] A categorical application of invertible terms is given in [BruceEtAl92], as these terms are used to caracterize the isomorphisms which hold in all Cartesian Closed Categories.

We also need to introduce an interesting extension of $\lambda\beta\eta$-calculus, which may be defined in terms of Böhm-trees. Note first that, once the (possibly infinite) operational behaviour of a λ-term has been displayed by its Böhm-tree, one may also consider infinitary reduction rules (or expansion rules, their inverse operations). In particular, one may η-expand (possibly infinitely many) nodes of a tree. Call "$=_\eta$" such an equality obtained by (possibly infinitely many) η-expansions between trees.

Examples





It is easy to see that the last tree is $BT(J)$, for $J = \lambda xy.x(Jy)$.

Definition 5.7 $H^\star = \{M = N \mid M,\ N$ *are* λ*-terms and* $BT(M) =_\eta BT(N)\ \}$

$H^\star$ is a consistent extension of $\lambda\beta\eta$-calculus. It is actually the largest consistent extension of the theory H, where all unsolvable terms are equated ([Barendregt84]). Thus $H^\star$ is the unique Hilbert-Post completion of H.

Theorem 5.8 *If* M *is* $\beta\eta$*-invertible, then* $M \in HP$.

Proof. (Sketch) Assume that M, N are $\beta\eta$-inverse. Then M, N are also $H^\star$-inverse, since $\lambda\beta\eta \subseteq H^\star$. Write now $M \in_k HP$ if M is in HP up to level k of $BT(M)$ (the root is level 0 of a tree).

If we prove

$$1) \qquad \forall k\ (M\ H^\star\text{-invertible} \Rightarrow M \in_k HP),$$

we are done, by the inductive definition of HP.

1) above holds for $k = 1$, by (a lot of) combinatory work on $BT(M)$, up to the first two levels (see [Dezani76], [BergstraKlo80] or [Barendregt84]).

Assume now (1) for $k > 1$. Then, by some further non-obvious work, prove that, if M, N $H^\star$-inverse, then $N_i, M_{\sigma i}$ are $H^\star$-inverse.

Thus, by induction hypothesis, $N_i, M_{\sigma i} \in_k HP$ and, clearly, $M, N \in_{k+1} HP$. □

The key idea is that, by the infinitary character of the provable equalities, $H^\star$ allows to compare terms with infinite Böhm-trees. Roughly, it is like looking at full decimal expansions of rational numbers, without restricting the attention to numbers with finite expansions (normal forms). The proof, though, works, at finite levels, exactly as for the finitary result and it is then analytically extended to the infinite expansions.

Church original work on λ-conversion used a more restrictive notion of λ-term. Namely, λI-terms are defined similarly to λ-terms, provided that $\lambda x.M$ is a λI-term iff x occurs free in M. It is easy to see that a λI-term M has a normal form iff all of its subterms have a normal form. By looking at the definition of HP, one may work out the following fact.

Theorem 5.9 *If $M \in HP$, then $BT(M)$ is Ω-free and there exists a λI-term M' such that $BT(M') = BT(M)$.*

Let's now follow stepwise Bergstra and Klop's argument for

(O) If M is $\beta\eta$-invertible, then M has a normal form. Assume that M, N are $\beta\eta$-inverse and that M has no normal form. Then

(1) There exist λI-terms M', N' s.t. $BT(M') = BT(M)$ and $BT(N') = BT(N)$, by theorem 5.9.

(2) $BT(M')$ is infinite, by assumption and theorem 5.8, and, hence M' has no normal form.

(3) $M' \circ N'$ is a λI-term, for $M' \circ N' \equiv BMN$ where $B \equiv \lambda xyz.x(yz)$, and, hence, it has no normal form, by (2).

(4) $P\omega \models M' = M$ and $P\omega \vdash N' = N$. by (1) and theorem 5.6.

(5) $P\omega \models M' \circ N' = M \circ N$. by (4).

(6) $BT(M' \circ N') = BT(M \circ N)$, by (5) and theorem 5.6, again.

(7) $BT(M \circ N)$ is infinite, by (3) and (6), and, hence , $M \circ N$ has no normal form.

The conclusion in (7) is clearly impossible, for $\lambda\beta\eta \models M \circ N = I$.

Comment 5.10 *The role of $P\omega$, as continuous topological model should be clear. It essentially allows to go from (1) to (6), by using theorem 5.6. Since BT-equal terms are the same object in $P\omega$ by theorem 5.6, one has (4) and, hence, (5) and (6) trivially follow.*

It is like looking at a property of natural numbers by studying them as elements of a more structured continuum they live in, the real or complex field, say. Where does continuity come in, in this case? The proof of theorem 5.6 heavily relies on continuity properties of $P\omega$. To see this more closely, one may consult [Barendregt84] or look at results which distillate the notion of "approximable application" in [Longo83]. This notion relates, in a general setting, the interpretation of formal application to the limit structure of models. Can one directly go from (1) to (6) by working over Böhm-trees with some smart combinatory technique? This is done in [Barendregt84], but the continuity argument is important as well. As a matter of fact one has to take the *completion* of the set of Böhm-trees, partially ordered as after definition 5.2, and heavily work with approximation and Scott continuity over this domain.

A final remark on cardinality. It is possible to take a countable model instead of $P\omega$, in the argument above, since $RE \subseteq P\omega$, the set of recursively enumerable sets, is an equationally equivalent sub-model of $P\omega$. That is, by the same notions of application and abstraction, RE yields a model which satisfies the same equalities as $P\omega$. More generally, for any infinite cardinal α, there is an applicative structure of cardinal α which yields lots of different λ-models, including one equationally equivalent to $P\omega$ (see [Longo83]). The use of RE, though, instead of $P\omega$ would be like looking at rational numbers (or complex numbers with rational coordinates), for the analysis of the behaviour of continuous functions or predicates over the real (or complex) numbers. Those countable subsets are dense and they perfectly determine continuous functions or predicates on the latter sets. Similarly, RE is dense in $P\omega$ w.r.t. the Scott topology (and contains all algebraic elements).

It should be clear that this informal survey is far away from being exhaustive as far as the use of "analytic" methods in Computer Science is concerned. We hope, though, that the examples mentioned provide some sufficiently clear analyses of combinatorial facts within continuous/topological frameworks even in Theory of Computations.

References

[AbramskyJung92] Abramsky S., Jung A., "Domain Theory", in Abramsky S. et al. (eds.) **Handbook of logic in computer science**, Vol. I and 2, Oxford: Clarendon Press, 1992.

[Aceto02] Aceto L., Bjorn V., Longo G. (eds.) **The difference between Turing Computability and Concurrent Systems**, special issue **Mathematical Structures in Computer Science**, Cambridge U.P., to appear, 2002.

[AmadioCurien98] Amadio R., Curien P.-L., **Domains and lambda-calculi**, Cambridge Univ. Press, 1998.

[AspertiLongo91] Asperti A., Longo G., **Categories, Types and Structures**, M.I.T. Press, 1991.

[Backus78] Backus J., Can Programming be liberated from the Von Neumann style? A functional style and its Algebra of Programs, **Communications of ACM**, 21,8, 1978.

[Barendregt84] , Barendregt H.P. **The lambda calculus: its syntax and semantics**, North-Holland, Amsterdam (revised edition), 1984.

[BarendregtLo82] Barendregt H.P., Longo G. "Recursion theoretic operators and morphisms of numbered sets", **Fundamenta Mathematicae**, (119):49–62,1982.

[BergstraKlo80] Bergstra, J. , Klop, J.W. "Invertible terms in the lambda-calculus", **Theor. Comp. Sci.** ll, 19-37, 1980.

[BerryCurien82] Berry G., Curien P.L. "Sequential Algorithms on Concrete Data Structures", **Theor. Comp. Sci.** 20, (265-321), 1982.

[Birkedal02] Birkedal L., J. van Oosten, G. Rosolini, D.S. Scott (eds.) **Realizability Semantics and Applications**, special issue **Mathematical Structures in Computer Science**, vol. 12, n. 3, 2002.

[BruceEtAl92] Bruce K., Di Cosmo R. , Longo G. "Provable isomorphisms of types", **Mathematical Structures in Computer Science**, 2(2):231–247, 1992.

[CurryFeys58] Curry, H. B. , Feys, R., **Combinatory Logic**. North-Holland, Amsterdam, 1958.

[Castagna97] Castagna G. , **Object-Oriented Programming: A Unified Foundation**, Birkäuser, 1997.

[Church32] Church A. "A set of partulates for the Foundation of Logic" **Annals of Math.** XXXIII (348-349) and XXXIV (839-864), 1932.

[Church41] Church A. **The Calculi of Lambda Conversion**, Princeton Univ. Press, reprinted 1963 by University Microfilms Inc., Ann Arbor, Michigan, U.S.A, 1941.

[Curry30] Curry H.B., "Grundlagen der kombinatorischen Logik", **Amer. J. of Math.**, 52, 1930.

[DeJaeger02] De Jaeger F., "An approach to Effective Functionals on Real Numbers via Filter Spaces", proceedings of **Summer Conference Series in Topology and Applications**, 2002.

[Dezani76] Dezani, M. "Characterization of normal forms possessing inverse in the $\lambda\beta\eta$-calculus", **Theor. Comp. Sci.** 2, 323-337, 1976.

[Edalat95A] Edalat A., "Dynamical systems, Measures and Fractals via Domain Theory", **Information and Computation** 120(1), (1995), 32-48.

[Edalat95B] Edalat A., "Domain Theory and Integration" **Theoretical Computer Science** 151, (1995), 163-193.

[EdalatHeck95] Edalat A. , Heckmann R., "A Computational Model for Metric Spaces", **Theoretical Computer Science** 193, (1998) 53-73.

[Edalat97A] Edalat A., "When Scott is Weak on the Top", **Mathematical Structures in Computer Science** 7, (1997), 401-417.

[Edalat97B] Edalat A., "Domains for Computation in Mathematics, Physics and Exact Real Arithmetic", **Bulletin of Symbolic Logic**, 3(4), (1997), 401-452.

[Escardo96] Escardo M., "PCF extended with real numbers", **Theoretical Computer Science**, vol. 162, 79-115, 1996.

[Ershov76] Ershov, Yu. L. "Model C of partial continuous functionals" In **Logic Coll.** 76, eds. R. Gandy, M. Hyland. North-Holland, Amsterdam, 1976.

[Frege84] Frege G. **The Foundations of Arithmetic**, 1884 (Engl. transl. Evanston, 1980.)

[Gierz80] Gierz, G. et al. **A compendium of Continuous lattices**, Springer-Verlag, Berlin, 1980.

[Girard86] Girard J.Y. "The system F of variable types, fifteen years later" **Theoretical Comp. Sci.**, 45 (159-192), 1986.

[Girard87] Girard J.-Y. "Linear Logic", **Theoretical Comp. Sci.**, 50 (1-102), 1987.

[Girard89] Girard J.Y., Lafont Y., Taylor P. **Proofs and Types**, Cambridge U. Press, 1989.

[Girard01] Girard J.Y., "Locus Solum", **Mathematical Structures in Computer Science**, vol.11, n.3, 2001.

[Gödel31] Gödel K., "Uber formal unentscheidbare Satze der Principia mathematica und verwandter Systeme I", **Monastshefte f. Mathematik und Physik** 38, 1931.

[Goldfarb87] Goldfarb H., **Jacques Herbrand: logical writings**, 1987.

[Goubault00] Goubault E. (ed.) **Geometry in Concurrency**, special issue, **Mathematical Structures in Computer Science**, Cambridge U.P., vol.10, n.4, 2000.

[Heyting59] Heyting, A. **Constructivity in Mathematics**. North-Holland, 1959.

[Hilbert99] Hilbert D., **Les fondements de la géométrie**, 1899 (trad. fran., Dunod, 1971).

[Hyland77] Hyland M. "Filter Space and Continuous Functionals", **Annals of Mathematical Logic** 16, (101-143), 1977.

[Hyland82] Hyland M. “The effective Topos," in **The Brouwer Symposium**, (Troelstra, Van Dalen eds.) North-Holland, 1982.

[Hyland87] Hyland M. “A small complete category" Lecture delivered at the Conference Church’s Thesis after 50 years, Zeiss (NL), June 1986 (Ann. Pure Appl. Logic, 1987).

[HylandPitts87] Hyland M., Pitts A. “The Theory of Constructions: categorical semantics and topos theoretic models" **Categories in Comp. Sci. and Logic**, Boulder (AMS notes), 1987.

[Kleene36] Kleene S. C. “Lambda definability and recursiveness," **Duke Math. J.**, 2, (pp. 340-353), 1936.

[Kleene59] Kleene S.C. “Recursive functionals and quantifiers of finite type I", **TRANS AMS**, vol. 91, 1959 (and II, vol. 95, 1963).

[Joyal96] Joyal A., Nielsen M., Winskel G. “Bisimulation from Open Maps", **Information and Computation** 127(2): 164-185, 1996.

[Lawvere76] Lawvere F.W. “Variable quantities and variable structures in topoi", in **Algebra Topology and Category Theory: a collection of papers in honor of Samuel Eilenberg**, A. Heller and M. Tierney (eds.), Academic Press, (101-131), 1976.

[Longo83] Longo, G. “Set-theoretical models of λ-Calculus: theories, expansions, isomorphisms", **Ann. Pure Applied Logic**, 24, 153-188, 1983.

[Longo84] Longo, G. “Continuous structures and Analytic Methods in Computer Science", in **Ninth Colloquium on Trees in Algebra and Programming** (CAAP 84), pages 1–22 (Courcelle, editor). Cambridge University Press, 1984.

[Longo02] Longo G. “Space and Time in the foundation of Mathematics, or some challenges in the interaction with other sciences”, invited lecture at the First AMS/SMF meeting, Lyon, July 2001 (to appear).

[LongoMoggi84] Longo G. , Moggi E. “The hereditary partial recursive functionals and recursion theory in higher types", **Journal of Symbolic Logic**, 49(4):1319 – 1332, 1984.

[LongoMoggi90] Longo G., Moggi E. “A category-theoretic characterization of functional completeness", **Theor. Comp. Sci.** vol. 70, 2, (pp. 193-211), 1990.

[LongoMoggi91] Longo G., Moggi E. “Constructive Natural Deduction and its omega-Set Interpretation", **Mathematical Structures in Computer Science**,vol.1, n.2, 1991.

[MacLaneMoer92] Mac Lane S., Moerdijk I. **Sheaves in Geometry and Logic**. Springer Verlag, Berlin, 1992.

[MyhillSheph55] Myhill J., Shepherdson J. C., "Effective operations on partial recursive functions", **Zeit. f. Math. Logik und Grund. der Math.**,vol. 1, 1955.

[Moschovakis64] Moschovakis Y., "Recursive metric spaces", **Fundam. Mathematicae**, LV, 1964.

[Nivat79] Nivat M., "Infinite words, infinite trees, infinite computations", **Math. Cent. Tracts** 109, 1-52, 1979.

[Normann80] Normann, D. **Recursion on the countable functionals**. LNM 811, Springer-Verlag, Berlin, 1980.

[Pitts87] Pitts A. "Polymorphism is Set Theoretic, constructively" **Symposium on Category Theory and Comp. Sci.**, SLNCS 283 (Pitt et al. eds), Edinburgh, 1987.

[Rogers67] Rogers H., **Theory of Recursive Functions and Effective Computability**, 1967.

[Rice53] Rice H. "Classes of recursively enumerable sets and their decision problem", **Trans. AMS**, vol. 74, 1953.

[Riemann54] Riemann B. "On the hypothesis which lie at the basis of geometry", 1854 (English transl. by W. Clifford, **Nature**, 1873).

[Scott70] Scott, D. S. "Outline of a Mathematical Theory of computation", **4th Ann. Princeton conf. on Info. Sci. and Syst.**, 1970.

[Scott72] Scott, D.S. "Continuous lattices", In **Toposes, Algebraic Geometry and Logic**. LNM 274, Springer-Verlag, Berlin, 1972.

[Scott76] Scott, D.S. "Data types as lattices", **SIAM J. of Comput.**, 5, 522-587, 1976.

[SmythPlo82] Smyth M., Plotkin G., "The category-theoretic solution of recursive domain equations" **SIAM Journal of Computing** 11, 1982.

[Tappenden95] Tappenden J. "Geometry and generality in Frege's philosophy of Arithmetic" **Synthese**, n. 3, vol. 102, March 1995.

[Turing36] Turing A. "On computable numbers with an application to the entscheidungsproblem", **PROC. London Math. soc.** 42, 1936.

[Weyl27] Weyl H. **Philosophy of Mathematics and of Natural Sciences**, 1927 (Engl. transl., Princeton University Press, Princeton, New Jersey, 1949).

[WinskelNiel95] Winskel G., Nielsen M. "Models of Concurrency", **Handbook of Logic and the Foundations of Computer Science**, vol IV, Abramsky et al. editors, Oxford University Press, 1995.