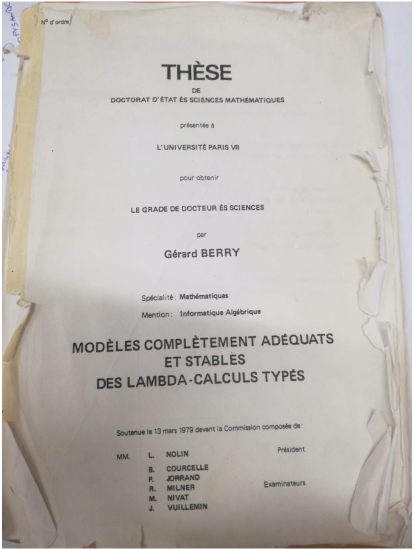
# The Philology of Strong Stability

## a semi-serious tribute to Thomas Ehrhard and some other founding father

Antonio Bucciarelli, IRIF, Université Paris Cité

September 29, 2022

# In the beginning was a mysterious, ancient code...


N° d'ordre

THÈSE

DE

DOCTORAT D'ÉTAT ÈS SCIENCES MATHÉMATIQUES

présentée à

L'UNIVERSITÉ PARIS VII

pour obtenir

LE GRADE DE DOCTEUR ÈS SCIENCES

par

Gérard BERRY

Spécialité : Mathématiques

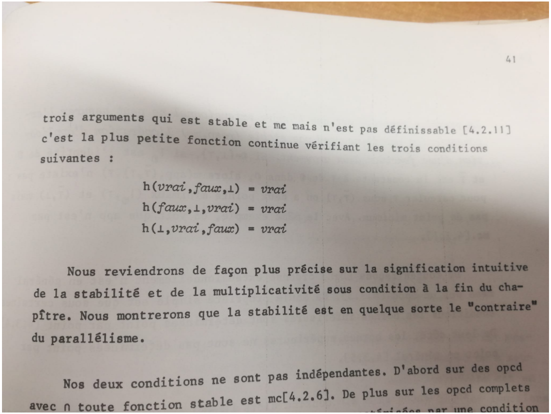
Mention : Informatique Algébrique

MODÈLES COMPLÈTEMENT ADÉQUATS
ET STABLES
DES LAMBDA-CALCULS TYPÉS

Soutenue le 13 mars 1979 devant la Commission composée de :

| | | | |
|---|---|---|---|
| MM. | L. | NOLIN | Président |
| | B. | COURCELLE | Examinateurs |
| | P. | JORRAND | |
| | R. | MILNER | |
| | M. | NIVAT | |
| | J. | VUILLEMIN | |

# …containing invaluable treasures…





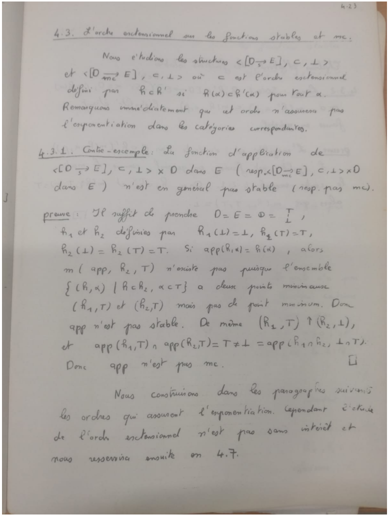
trois arguments qui est stable et mc mais n'est pas définissable [4.2.11] c'est la plus petite fonction continue vérifiant les trois conditions suivantes :

$$h(\mathit{vrai},\mathit{faux},\bot) = \mathit{vrai}$$
$$h(\mathit{faux},\bot,\mathit{vrai}) = \mathit{vrai}$$
$$h(\bot,\mathit{vrai},\mathit{faux}) = \mathit{vrai}$$

Nous reviendrons de façon plus précise sur la signification intuitive de la stabilité et de la multiplicativité sous condition à la fin du chapître. Nous montrerons que la stabilité est en quelque sorte le "contraire" du parallélisme.

Nos deux conditions ne sont pas indépendantes. D'abord sur des opcd avec ∩ toute fonction stable est mc[4.2.6]. De plus sur les opcd complets

...but decipherable only by very few initiated (the Gustave's braves, members of a sect located on the western outskirts of Paris)...



4.23

4.3. L'ordre extensionnel sur les fonctions stables et mc.

Nous étudions les structures $<[D \xrightarrow[s]{} E], \subset, \perp>$ et $<[D \xrightarrow[mc]{} E], \subset, \perp>$ où $\subset$ est l'ordre extensionnel défini par $f \subset f'$ si $f(\alpha) \subset f'(\alpha)$ pour tout $\alpha$. Remarquons immédiatement que cet ordre n'assurera pas l'exponentiation dans les catégories correspondantes.

4.3.1. Contre-exemple : La fonction d'application de $<[D \xrightarrow[s]{} E], \subset, \perp> \times D$ dans $E$ (resp. $<[D \xrightarrow[mc]{} E], \subset, \perp> \times D$ dans $E$) n'est en général pas stable (resp. pas mc).

preuve : Il suffit de prendre $D = E = \mathbb{O} = \begin{smallmatrix} \top \\ \vdots \\ \perp \end{smallmatrix}$, $f_1$ et $f_2$ définies par $f_1(\perp) = \perp$, $f_1(\top) = \top$, $f_2(\perp) = f_2(\top) = \top$. Si $app(f, \alpha) = f(\alpha)$, alors $m(app, f_2, \top)$ n'existe pas puisque l'ensemble $\{(f, \alpha) \mid f \subset f_2, \alpha \subset \top\}$ a deux points minimaux $(f_1, \top)$ et $(f_2, \top)$ mais pas de point minimum. Donc app n'est pas stable. De même $(f_1, \top) \uparrow (f_2, \perp)$, et $app(f_1, \top) \cap app(f_2, \top) = \top \neq \perp = app(f_1 \cap f_2, \perp \cap \top)$.
Donc app n'est pas mc. □

Nous construirons dans les paragraphes suivants les ordres qui assurent l'exponentiation. Cependant l'étude de l'ordre extensionnel n'est pas sans intérêt et nous reservira ensuite en 4.7.

# ...but having pierced the code, and climbed some other stormy peaks...

Annals of Pure and Applied Logic 37 (1988) 129–177 129
North-Holland

## NORMAL FUNCTORS, POWER SERIES AND λ-CALCULUS

Jean-Yves GIRARD
*Université de Paris VII, UER de Mathématiques, Tour 45-55, 2 Place Jussieu, 75221 Paris, France*

Communicated by D. van Dalen
Received 8 July 1985

### Introduction

This paper presents a new approach to the general subject known as 'continuous functionals' which includes computer-science oriented topics such as models of various λ-calculi.

Let us describe shortly the main features of this approach: a *type* is a collection of atomic 'states'; if $A$ is a type, an *object of type A* is any superposition of atomic states. So an object of type $A$ can be viewed as a sum $\sum_{a \in A} m_a \cdot \boldsymbol{a}$, where the 'vectors' $\boldsymbol{a}$ represent the atomic states, and the $m_a$'s are cardinal numbers (in good cases integers) counting the multiplicities of each atomic state. So this differs from current constructions in two ways:

(i) because we count multiplicities (instead of just considering whether a state is present),

(ii) because no inner compatibility between the states is required: all combinations are accepted.

Let us say shortly that for the usual types of data, this makes a reasonable sense: certain combinations represent the usual outputs of computations, and general combinations are just formal superpositions of these usual outputs: such superpositions are made natural to consider if we think of parallel programs (then orders of multiplicity count the number of ways of getting the answer: typically in the case of the parallel 'or', if we have an atomic state $T$ for 'true', then $\boldsymbol{T}$ or $\boldsymbol{T} = 2 \cdot \boldsymbol{T}$, because there are two ways to compute the value . . .), but also of random programs, where we can consider the superposition of all possible outputs depending on the random choices made during the execution. Section 4 which is quite poor in results, gives some idea how to interpret recursive functions in that framework.

The origin of the work was in the need for some link between say functionals of finite type and $\Pi^1_2$-logic. The results of [5, Ch. XII] are certainly of interest in this connection (see for instance the remarkable work of Päppinghaus [7]) but

0168-0072/88/$3.50 © 1988, Elsevier Science Publishers B.V. (North-Holland)

Theoretical Computer Science 50 (1987) 1–102 1
North-Holland

## LINEAR LOGIC*

Jean-Yves GIRARD
*Équipe de Logique Mathématique, UA 753 du CNRS, UER de Mathématiques, Université de Paris VII, 75251 Paris, France*

Communicated by M. Nivat
Received October 1986

*A la mémoire de Jean van Heijenoort*

**Abstract.** The familiar connective of negation is broken into two operations: linear negation which is the purely negative part of negation and the modality "of course" which has the meaning of a reaffirmation. Following this basic discovery, a completely new approach to the whole area between constructive logics and programmation is initiated.

### Contents



* Because of its length and novelty this paper has not been subjected to the normal process of refereeing. The editor is prepared to share with the author any criticism that eventually will be expressed concerning this work.

0304-3975/87/$3.50 © 1987, Elsevier Science Publishers B.V. (North-Holland)

Theoretical Computer Science 121 (1993) 179–186 179
Elsevier

## Historical introduction to "Concrete domains" by G. Kahn and G.D. Plotkin

Stephen Brookes
*School of Computer Science, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213-3890, USA*

The original version of this paper appeared as a technical report 15 years ago [28], based on the work carried out at Edinburgh University in the Fall of 1975. The ideas and concepts introduced in this work have been widely referenced and have become the basis for a significant body of foundational research on semantics of programming languages. The purpose of this introduction is to sketch the background for this research and to mention briefly some of this later work, thus placing the paper in context.

Christopher Strachey's [40] work on the formal description of programming language concepts raised problems concerning the nature and existence of semantic models for λ-calculi. Dana Scott developed a theory of domains and continuous functions that established adequate mathematical foundations for Strachey's semantic descriptions and formed the basis for the denotational approach to semantics, advocated in [37] and later applied to a wide variety of programming languages [39, 34, 42, 24]. Semantic domains were typically taken to be complete lattices [38] or more general kinds of complete partial order [38, 25], the precise combination of order-theoretic assumptions being chosen to suit the application and often justified by appealing to intuitive arguments concerning computability. A good survey of domain theory is provided in [24].

However, these semantic treatments typically make no intrinsic distinction between domains used to represent data and domains used to represent procedures or functions. For certain purposes it seems appropriate to make such distinctions: for

*Correspondence to:* S. Brookes, School of Computer Science, Carnegie-Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213-3890, USA. Email: brookes@cs.cmu.edu.



Theoretical Computer Science 20 (1982) 265–321 265
North-Holland Publishing Company

## SEQUENTIAL ALGORITHMS ON CONCRETE DATA STRUCTURES

G. BERRY and P.L. CURIEN
*Centre de Mathématiques Appliquées, Ecole des Mines—Sophia-Antipolis, 06560 Valbonne, France*

Communicated by M. Nivat
Received January 1980
Revised September 1981

**Abstract.** We provide a sequential denotational semantics for sequential programming languages, based on a new notion of sequential algorithm on the Kahn-Plotkin concrete data structures. Intuitively an algorithm may be seen either as a concrete object – a "program" in a simple output-driven language – or as an abstract object – the pair of a sequential function and of a computation strategy for it. The concrete and abstract presentations are equivalent, as shown by a representation theorem. The algorithms form a cartesian closed category with straightforward solutions to recursive domain equations. Hence they may replace functions in the denotational semantics of any sequential language. An applicative programming language based on sequential algorithms is presented in a companion paper.

### 1. Introduction

The term "sequential programming language" is often used when referring to languages such as ALGOL, PASCAL, LISP etc. . . The quoted phenomenon is the existence of a "natural" order in the evaluation of program pieces, which is usually implicit in the definition of their operational behaviour (some attempts for a formal definition are presented in [3, 12]). Our purpose is to present a "sequential" denotational semantics for the sequential programming languages, with application to completeness and optimality results, to data structure definition and to coroutine systems. The present paper is devoted to a theoretical study of the basic objects of our semantics: sequential algorithms between Kahn–Plotkin's concrete data structures [14]. In another paper [7] we build a functional programming language based on algorithms, which is a generalization of both Milner's applicative language ML [11] and Kahn–MacQueen's coroutine system [13]. Many intuitions and examples may be found in [7].

The classical denotational semantics of Scott–Strachey [29, 19] is usually presented in the category of complete partial orders and continuous functions, where no reference is made to sequentiality. Plotkin [24] and Milner [21] have shown that this leads to an intrinsic incompleteness of the semantics w.r.t. program equivalence: two procedures which differ in the denotational semantics may very well be opera-

0304-3975/82/0000-0000/$02.75 © 1982 North-Holland

## Sequentiality and Strong Stability

Antonio Bucciarelli
Dipartimento di Informatica
Università di Pisa, Italy
and
L.I.E.N.S.
Ecole Normale Supérieure, France
email: buccia@dmi.ens.fr

Thomas Ehrhard
L.I.T.P., Institut Blaise Pascal
Université Paris VII, France
email: ehrhard@litp.ibp.fr

**Abstract**

We show that Kahn-Plotkin sequentiality can be expressed by a preservation property similar to stability and that this kind of generalized stability can be extended to higher order, giving rise to a new model of PCF.

### Introduction

One of the main goals of denotational semantics is to find as adequate models as possible of purely functional programming languages like PCF [9], [3]. This consists in finding cartesian closed categories of domains where there are "as few points as possible", aiming at a situation where any finite point is definable by a term of PCF. In that situation, the model is fully abstract (see [8]). In this research, most people have followed the rule of not making any reference to the syntax of PCF in the constructions of domains. By the nature of the problem, that rule seems very sound, and we follow it.

The problem of full abstraction is still open, but several significative steps have been performed in the direction of its solution.

Plotkin [9] remarked that the presence of functions like "parallel or" in the Scott models prevents them from being fully abstract. Kahn, Plotkin ([6]) and Milner [8] tried to solve this problem by adding different "sequentiality" constraints on morphisms which turned out to be equivalent (at least for first order functions). However, they could not define any cartesian closed category of sequential functions.

These works gave rise to two main developments. The first one consisted in weakening the notion of sequentiality in a property extendable to higher orders, and this is what Berry [1] did introducing stable semantics. The second one, carried out by Berry and Curien (see [2]) was to stick firmly to the Kahn-Plotkin's notion of sequentiality, but the price to pay was the impossibility of keeping functions as morphisms; they were obliged to switch to sequential algorithms.

The model presented here is more in the spirit of the former development firstly because the morphisms of our category are functions and secondly because they are characterized by a preservation property similar to stability, this is why we call them "strongly stable". Indeed, in this model, the domains are endowed with a certain notion of "coherence" which is a predicate on the set of their finite subsets. A function has then to preserve this coherence and to commute with the glbs of coherent set. The interesting phenomenon is that Kahn-Plotkin sequentiality may be expressed in these terms. The main result is then the construction of a model where all morphisms are functions and, at ground types, these functions are sequential. The trouble with Kahn-Plotkin sequentiality is that it seems not to make sense at higher order in a model where morphisms are functions. This is why the question whether functionals of this model are sequential or not is probably unrelevant. However it would be interesting to find a connection with the category of sequential algorithms.

### 1 Qualitative domains and coherence spaces

We recall here some definitions and results about stable semantics as presented by Girard in [5].

**Definition 1** *A qualitative domain (QD) is a pair $(|X|, X)$ where $|X|$ is a set and $X$ is a subset of $\mathcal{P}(|X|)$ such that*

- $\emptyset \in X$ .
- $\forall a \in |X| \quad \{a\} \in X$ .
- $\forall x \in X \; \forall y \subseteq x \quad y \in X$ .
- *For any $\subseteq$-directed subset $D$ of $X$ we have $\bigcup D \in X$.*

*Furthermore, $|X|$ will always be countable.*
*We say that $X$ is a coherence space if*

$$x \in X \quad \text{iff} \quad \forall a, b \in x \; \{a, b\} \in X$$

138

CH3025-4/91/0000/0138$01.00 © 1991 IEEE

# ......but having pierced the code, and climbed some other stormy peaks...

Annals of Pure and Applied Logic 37 (1988) 129–177 — North-Holland — 129

**NORMAL FUNCTORS, POWER SERIES AND λ-CALCULUS**

Jean-Yves GIRARD

Université de Paris VII, UER de Mathématiques, Tour 45-55, 2 Place Jussieu, 75221 Paris, France

Theoretical Computer Science 50 (1987) 1–102 — North-Holland — 1

**LINEAR LOGIC***

Jean-Yves GIRARD

Équipe de Logique Mathématique, UA 753 du CNRS, UER de Mathématiques, Université de Paris VII, 75251 Paris, France

*A la mémoire de Jean van Heijenoort*

Theoretical Computer Science 121 (1993) 179–186 — Elsevier — 179

**Historical introduction to "Concrete domains" by G. Kahn and G.D. Plotkin**

Stephen Brookes

School of Computer Science, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213-3890, USA

Theoretical Computer Science 20 (1982) 265–321 — North-Holland Publishing Company — 265

**SEQUENTIAL ALGORITHMS ON CONCRETE DATA STRUCTURES**

G. BERRY and P.L. CURIEN

Centre de Mathématiques Appliquées, Ecole des Mines—Sophia-Antipolis, 06560 Valbonne, France

**Sequentiality and Strong Stability**

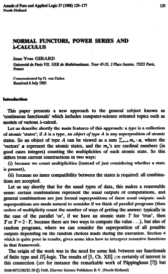
Antonio Bucciarelli, Dipartimento di Informatica, Università di Pisa, Italy and L.I.E.N.S., Ecole Normale Supérieure, France

Thomas Ehrhard, L.I.T.P., Institut Blaise Pascal, Université Paris VII, France

CH3025-4/91/0000/0138$01.00 © 1991 IEEE

**...uhmm, I praise myself...**
**the fact is that strong stability was defined here**

# ......but having pierced the code, and climbed some other stormy peaks...



Annals of Pure and Applied Logic 37 (1988) 129–177
North-Holland

129

**NORMAL FUNCTORS, POWER SERIES AND λ-CALCULUS**

Jean-Yves GIRARD
*Université de Paris VII, UER de Mathématiques, Tour 45-55, 2 Place Jussieu, 75221 Paris, France*

Communicated by D. van Dalen
Received 8 July 1985

**Introduction**

This paper presents a new approach to the general subject known as 'continuous functionals' which includes computer-science oriented topics such as models of various λ-calculi.

Let us describe shortly the main features of this approach: a *type* is a collection of atomic 'states'; if *A* is a type, an *object of type A* is any superposition of atomic states. So an object of type *A* can be viewed as a sum $\sum_{a \in A} m_a \cdot a$, where the 'vectors' *a* represent the atomic states, and the $m_a$'s are cardinal numbers (in good cases integers) counting the multiplicities of each atomic state. So this differs from current constructions in two ways:

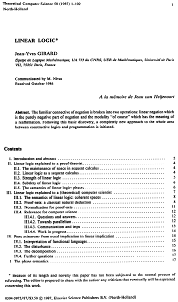
(i) because we count multiplicities (instead of just considering whether a state is present),

(ii) because no inner compatibility between the states is required: all combinations are accepted.

Let us say shortly that for the usual types of data, this makes a reasonable sense: certain combinations represent the usual outputs of computations, and general combinations are just formal superpositions of these usual outputs. Such superpositions are made natural to consider if we think of parallel programs (then orders of multiplicity count the number of ways of getting the answer: typically in the case of the parallel 'or', if we have an atomic state *T* for 'true', then *T* or *T* = 2 · *T*, because there are two ways to compute the value ...), but also of random programs, where we can consider the superposition of all possible outputs depending on the random choices made during the execution. Section 4 which is quite poor in results, gives some idea how to interpret recursive functions in that framework.

The origin of the work was in the need for some link between say functionals of finite type and $\Pi^1_2$-logic. The results of [5, Ch. XII] are certainly of interest in this connection (see for instance the remarkable work of Päppinghaus [7]) but

0168-0072/88/$3.50 © 1988, Elsevier Science Publishers B.V. (North-Holland)



Theoretical Computer Science 50 (1987) 1–102
North-Holland

1

**LINEAR LOGIC***

Jean-Yves GIRARD
*Équipe de Logique Mathématique, UA 753 du CNRS, UER de Mathématiques, Université de Paris VII, 75251 Paris, France*

Communicated by M. Nivat
Received October 1986

*A la mémoire de Jean van Heijenoort*

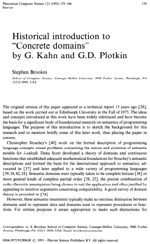
**Abstract.** The familiar connective of negation is broken into two operations: linear negation which is the purely negative part of negation and the modality "of course" which has the meaning of a reaffirmation. Following this basic discovery, a completely new approach to the whole area between constructive logics and programming is initiated.

**Contents**

| | |
|---|---|
| I. Introduction and abstract | 2 |
| II. Linear logic explained to a proof-theorist | 4 |
| II.1. The maintenance of space in sequent calculus | 4 |
| II.2. Linear logic as a sequent calculus | 4 |
| II.3. Strength of linear logic | 5 |
| II.4. Subtlety of linear logic | 6 |
| II.5. The semantics of linear logic: phases | 6 |
| III. Linear logic explained to a (theoretical) computer scientist | 7 |
| III.1. The semantics of linear logic: coherent spaces | 8 |
| III.2. Proof-nets: a classical natural deduction | 8 |
| III.3. Normalization for proof-nets | 11 |
| III.4. Relevance for computer science | 12 |
| III.4.1. Questions and answers | 12 |
| III.4.2. Towards parallelism | 12 |
| III.4.3. Communication and tips | 13 |
| III.4.4. Work in progress | 14 |
| IV. Pons asinorum: from usual implication to linear implication | 14 |
| IV.1. Interpretation of functional languages | 15 |
| IV.2. The disturbance | 15 |
| IV.3. The decomposition | 16 |
| IV.4. Further questions | 17 |
| 1. The phase semantics | 17 |

* Because of its length and novelty this paper has not been subjected to the normal process of refereeing. The editor is prepared to share with the author any criticism that eventually will be expressed concerning this work.

0304-3975/87/$3.50 © 1987, Elsevier Science Publishers B.V. (North-Holland)



Theoretical Computer Science 121 (1993) 179–186
Elsevier

179

Historical introduction to "Concrete domains" by G. Kahn and G.D. Plotkin

Stephen Brookes
*School of Computer Science, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213-3890, USA*

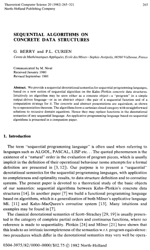
The original version of this paper appeared as a technical report 15 years ago [28], based on the work carried out at Edinburgh University in the Fall of 1975. The ideas and concepts introduced in this work have been widely referenced and have become the basis for a significant body of foundational research on semantics of programming languages. The purpose of this introduction is to sketch the background for this research and to mention briefly some of this later work, thus placing the paper in context.

Christopher Strachey's [40] work on the formal description of programming language concepts raised problems concerning the nature and existence of semantic models for λ-calculi. Dana Scott developed a theory of domains and continuous functions that established adequate mathematical foundations for Strachey's semantic descriptions and formed the basis for the denotational approach to semantics, advocated in [37] and later applied to a wide variety of programming languages [39, 34, 42, 24]. Semantic domains were typically taken to be complete lattices [38] or more general kinds of complete partial order [38, 25], the precise combination of order-theoretic assumptions being chosen to suit the application and often justified by appealing to intuitive arguments concerning computability. A good survey of domain theory is provided in [24].

However, these semantic treatments typically make no intrinsic distinction between domains used to represent data and domains used to represent procedures or functions. For certain purposes it seems appropriate to make such distinctions: for

*Correspondence to:* S. Brookes, School of Computer Science, Carnegie-Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213-3890, USA. Email: brookes@cs.cmu.edu.

0304-3975/93/$06.00 © 1993—Elsevier Science Publishers B.V. All rights reserved



Theoretical Computer Science 20 (1982) 265–321
North-Holland Publishing Company

265

SEQUENTIAL ALGORITHMS ON CONCRETE DATA STRUCTURES

G. BERRY and P.L. CURIEN
*Centre de Mathématiques Appliquées, Ecole des Mines—Sophia-Antipolis, 06560 Valbonne, France*

Communicated by M. Nivat
Received January 1980
Revised September 1981

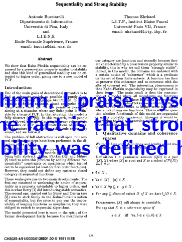
**Abstract.** We provide a sequential denotational semantics for sequential programming languages, based on a new notion of sequential algorithm on the Kahn-Plotkin concrete data structures. Intuitively an algorithm may be seen either as a concrete object – a "program" in a simple output-driven language – or as an abstract object – the pair of a sequential function and of a computation strategy for it. The concrete and abstract presentations are equivalent, as shown by a representation theorem. The algorithms form a cartesian closed category with straightforward solutions to recursive domain equations. Hence they may replace functions in the denotational semantics of any sequential language. An applicative programming language based on sequential algorithms is presented in a companion paper.

**1. Introduction**

The term "sequential programming language" is often used when referring to languages such as ALGOL, PASCAL, LISP etc. . . The quoted phenomenon is the existence of a "natural" order in the evaluation of program pieces, which is usually implicit in the definition of their operational behaviour (some attempts for a formal definition are presented in [3, 12]). Our purpose is to present a "sequential" denotational semantics for the sequential programming languages, with application to completeness and optimality results, to data structure definition and to coroutine systems. The present paper is devoted to a theoretical study of the basic objects of our semantics: sequential algorithms between Kahn-Plotkin's concrete data structures [14]. In another paper [7] we build a functional programming language based on algorithms, which is a generalization of both Milner's applicative language ML [11] and Kahn-MacQueen's coroutine system [13]. Many intuitions and examples may be found in [7].

The classical denotational semantics of Scott-Strachey [29, 19] is usually presented in the category of complete partial orders and continuous functions, where no reference is made to sequentiality. Plotkin [24] and Milner [21] have shown that this leads to an intrinsic incompleteness of the semantics w.r.t. program equivalence: two procedures which differ in the denotational semantics may very well be opera-

0304-3975/82/0000-0000/$02.75 © 1982 North-Holland



**Sequentiality and Strong Stability**

Antonio Bucciarelli
Dipartimento di Informatica
Università di Pisa, Italy
and
L.I.E.N.S.
Ecole Normale Supérieure, France
email: buccia@dmi.ens.fr

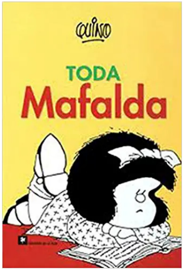
Thomas Ehrhard
L.I.T.P., Institut Blaise Pascal
Université Paris VII, France
email: ehrhard@litp.ibp.fr

**Abstract**

We show that Kahn-Plotkin sequentiality can be expressed by a preservation property similar to stability and that this kind of generalized stability can be extended to higher order, giving rise to a new model of PCF.

CH3025-4/91/0000/0138$01.00 © 1991 IEEE

**...uhmm, I praise myself...**
**the fact is that strong**
**stability was defined here**



...the questing semantician discovers the first pearl of Thomas Ehrhard:

*Math. Struct. in Comp. Science* (1993), *vol.* 3, *pp.* 365–385 

# Hypercoherences: a strongly stable model of linear logic

THOMAS EHRHARD

*L.I.T.P. I.B.P., Couloir 55-56, premier étage, Université Paris VII, 2 Place Jussieu, 75251 Paris CEDEX 05, France,*
*and*
*I.G.M., Université de Marne-la-Vallée, 2 Allée Jean Renoir, 93160 Noisy-le-Grand, France,*
*e-mail: ehrhard@litp.ibp.fr*

*Received 3 March 1992; revised 13 October 1992*

We present a new model of classical linear logic based on the notion of *strong stability* that was introduced recently in a work about sequentiality written jointly with Antonio Bucciarelli.

## 1. Introduction

In the denotational semantics of purely functional languages (such as PCF (Plotkin 1977; Berry *et al.* 1985)), types are interpreted as objects and programs as morphisms in a cartesian closed category (CCC for short). Usually, the objects of this category are at least Scott domains, and the morphisms are at least continuous functions. One carefully avoids making any reference to the syntax of the language in defining such a model; the goal of semantics is to express precisely, and in a "purely abstract way", some interesting properties of the language.

One of these properties is "continuity". It corresponds to the basic fact that any computation that terminates can use only a finite amount of data. The corresponding semantics of PCF is the continuous one, where objects are Scott domains, and morphisms are continuous functions.

But the continuous semantics does not capture an important property of computations in PCF, namely "determinism". It is much harder to model abstractly the idea of determinism. Vuillemin and Milner produced the first (equivalent) definitions of sequentiality. Kahn and Plotkin (1978) generalized this notion of sequentiality. More precisely, they defined a category of "concrete domains" (represented by "concrete data structures") and of sequential functions.

We shall begin with an intuitive description of what sequentiality is, in the framework of concrete data structures (CDS's). A CDS $D$, very roughly, is a Scott domain equipped with a notion of "places" or "cells". An element of $D$ is a partial piece of data, where some cells are filled, and others are not. A cell can be filled, in general, by different values. (Think of the cartesian product of two ground types: there are two cells corresponding to the two places one can fill in a couple.) In a CDS, an element $x$ is less than an element



https://doi.org/10.1017/S0960129500000281 Published online by Cambridge University Press

# Stability and sequentiality: one very same pattern

| | stability | sequentiality |
|---|---|---|
| "Circumstantiated" | dI-domains and stable functions | Concrete domains and sequential functions<br>Concrete data structures and sequential algorithms |
| Algebraic | Qualitative domains and cm functions | QD with coherence and strongly stable functions |
| Linear | Coherence spaces | Hypercoherences |

Thank you all, and cheers, Thomas!