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.

example, a data domain may model some kind of data structure built from storage cells containing primitive values, and it might be important to make precise the sense in which a computation producing or consuming data operates incrementally. The development of concrete domains was originally motivated by problems such as these, arising notably in the study of dataflow networks and sequentiality.

The foundations for a theory of dataflow networks were laid in 1974 by Kahn, in the seminal paper [26]. This now represents a classic example of the use of elementary domain-theoretic concepts in modelling parallel programs. Kahn described a simple language for parallel programming, based on an abstract dataflow model of deterministic parallel computation: a network of autonomous, sequential, functional processes, connected by communication lines that represent unbounded queues. The notion of sequential process was rather intuitive: at all times a process is either computing or waiting for information on a single input line. In Kahn's model, each communication line is capable of carrying data of some given type D, and a (possibly infinite) history of the traffic along a communication line is represented as an element of a "sequence domain" denoted D^{ω}; each process computes a continuous function from input histories (or "streams") to output histories. A network is then described by a set of (mutually recursive) functional equations, one for each communication line, and the behavior of the network is obtained in a natural way by taking the least fixed point. The coincidence of this least fixed point with the operational behavior of networks is now known as "Kahn's principle". Although Kahn did not provide a formal operational semantics for networks, this principle seems very appealing and intuitive. Indeed, Kahn's principle has been adopted as a critical test in judging the correctness of subsequent semantic treatments of dataflow networks, including generalizations to permit nondeterministic processes [22, 33, 1, 29].

In joint work with David MacQueen, Kahn later provided an operational model for networks based on demand-driven, coroutine-like execution [27]: processes "consume" input data in response to requests to "produce" output. This operational view clearly suggests the need for a domain-theoretic account of incremental computation. Kahn also realized that queues and the corresponding sequence domains ought to be a special case of a more general kind of data structure and domains.

Problems concerning the domain-theoretic characterization of sequentiality also emerged from foundational work on the denotational and operational semantics of λ-calculi. Dana Scott [35] introduced LCF, a logic for computable functions based on a simply typed combinatory logic with arithmetic and boolean primitives and recursion. Scott gave a model in which datatypes are modelled as domains and phrases of functional type denote continuous functions. He suggested investigating the relation between his semantics and syntactic reduction rules. He also asked whether the presence of intuitively parallel (i.e. nonsequential) functions in the semantics, such as Kleene's "parallel or", was desirable. Plotkin took up Scott's suggestion and gave an operational semantics to a λ-calculus variant of Scott's combinatory logic, christened PCF. The operational semantics brings out the essentially sequential character of the language; this was made precise by Berry's sequentiality theorem described below [3].

Plotkin [34] proved an adequacy theorem showing that the reduction rules for PCF programs accorded with their denotations – as imagined by Scott. Plotkin further showed that as a result of the presence of the parallel functions, the Scott model fails to be fully abstract. That is, there are terms with different denotations in the model that are nonetheless operationally indistinguishable, in the sense that one can be substituted for the other in any program context without changing the result of evaluating the program. Early semantic formulations of sequentiality, aiming to construct models that contain no parallel functions, were of limited utility [43, 30, 33]. Milner's model [30], while fully abstract, is "syntactic" in nature, and left open the problem of finding a semantic characterization of sequential functions. The need thus arose for a domain-theoretic notion of sequentiality refining the notion of continuity.

At about the same time, Gérard Berry [2] introduced the notion of stability used in [6, 9] to study canonical forms and optimal computations for recursive programs. Berry [3] showed that the λ-calculus is syntactically stable and sequential [3], and introduced a model using stable functions between dI-domains, employing the stable order rather than the pointwise order typical in Scott's theory [4]. The stable order ties in nicely with properties of program computations. Stable functions and the stable ordering have many pleasing algebraic properties; for instance, the category of dI-domains and stable functions is cartesian-closed. A large body of work concerning or using stability has been developed. For instance, Girard [23] gave a model for polymorphism using qualitative domains and stable functions, and Coquand et al. [15] generalized this to obtain a model for polymorphism using dI-domains and stable functions[1]. Taylor [41] concerns the algebraic theory of stable domains, and Droste [21] investigates some of the properties of stable domains. While the notion of stability refines continuity in an appealing way, Berry's class of stable functions still included some intuitively nonsequential functions. It did not seem possible to give an adequate definition of sequential functions using either Scott domains or dI-domains, and the problem of finding a satisfactory semantic notion of sequentiality remained.

In this paper Kahn and Plotkin proposed the first general domain-theoretic framework in which a semantic account of incremental computation and sequential functions could be given. Starting from the idea that domains should be (at least) ω-algebraic, coherent, complete partial orders, they progressively introduced extra assumptions designed to characterize a class of "concrete domains" that can plausibly serve as models of data and can be equipped naturally with a notion of incremental computation. They also introduced "information matrices", essentially abstract descriptions of data structures, built from "cells", for which there is a natural notion of incremental computation by filling cells according to accessibility constraints.

[1] It is also possible to model polymorphism using Scott domains and continuous functions [15].

Kahn and Plotkin proved a representation theorem making precise the intuition that "concrete domains are the domain-theoretic counterparts to information matrices". Winskel [44] gave an improved proof for the representation theorem for concrete domains, and this formed the basis for the proof by Curien in [18]. The name "information matrix" has been superceded by the term "concrete data structure" (introduced by Berry)[2].

Kahn and Plotkin showed that concrete domains are closed under cartesian product, separated sum, upper section, and a form of grafting. They also showed that concrete domains are closed under certain restricted forms of inverse limit, thus justifying the solution of recursive domain equations. Kahn and Plotkin identify an important subclass consisting of the distributive concrete domains (corresponding to distributive concrete data structures). Every distributive concrete domain is also a dI-domain, and every Kahn–Plotkin sequential function between distributive concrete domains is also stable. The converse fails, however, since there are stable functions that fail to be Kahn–Plotkin sequential.

Nielsen et al. [31] later constructed a domain-theoretic framework for modelling Petri nets, based on "event domains", with a concrete representation using "event structures". Winskel [44] developed an extensive theory of event structures and event domains. Event structures provide a more general framework than (distributive) concrete domains; certain event structures (characterized by a "stability" property) give a representation for dI-domains [46]. A good survey of event structures is provided in [45], and Droste [19, 20] reports some more recent work on the properties of event domains and concrete domains.

In a separate (unpublished) manuscript Kahn and Plotkin showed that concrete domains indeed form the basis for a general theory of sequential functions; the details of this theory were summarized by Berry and Curien [7, 18]. Berry's syntactic sequentiality theorem [5, 4, 18] provides an interesting link with the work of Corrado Böhm: the set of PCF terms and the set of their Böhm trees form concrete domains, and the function that maps each PCF term to its Böhm tree is sequential. This gives a domain-theoretic way to express the sequentiality of the usual β-reduction evaluation strategy for λ-expressions.

Since the sequential functions are closed under composition and recursion, one can use the Kahn–Plotkin framework to give a denotational semantics to the deterministic dataflow networks considered in [26], enabling a precise formulation of the "folk theorem" that Kahn networks (with sequential primitives) compute sequential functions (see, for instance, [32]) and demonstrating that concrete domains do indeed provide an appropriate generalization of the sequence domains used in [26]. However, the category of concrete domains and sequential functions is not

[2] According to Gilles Kahn, he and Plotkin considered a "concrete" domain to be a domain that could be associated naturally with a data structure, the use of an adjective being justified because arbitrary domains do not appear to have this property; on the other hand, data structures were already concrete enough.

cartesian-closed, since the set of sequential functions between two concrete domains may fail to form a concrete domain [18]. The Kahn–Plotkin notion of sequential function makes sense at first-order types, but this limitation prevents its use at higher order. The problem remained of constructing a sequential model for PCF.

In a major development, building on Kahn and Plotkin's work, Berry and Curien [7, 18] introduced a theory of sequential algorithms between concrete data structures. A sequential algorithm can be viewed as a Kahn–Plotkin sequential function together with a (sequential) computation strategy for that function. The operational behavior of sequential algorithms, described in [18], is demand-driven and coroutine-like, thus emphasizing the connection with the earlier work of Kahn and MacQueen [27]. The Berry–Curien category of concrete domains and sequential algorithms is cartesian-closed, and thus provides an intensional sequential model of PCF; the Kahn–Plotkin category can be recovered as an extensional quotient [18]. Reference [10] introduced a cartesian-closed category of generalized concrete data structures and continuous functions, and used this as the basis for a category whose morphisms can be viewed as parallel algorithms, generalizing the Berry–Curien algorithms in a natural way to permit parallel computation strategies.

There have been further developments generalizing the Kahn–Plotkin framework, aiming for cartesian closure and full abstraction, including [12, 11]. Working with qualitative domains equipped with a coherence structure (QDCs), Bucciarelli and Ehrhard [12] showed that the Kahn–Plotkin notion of sequentiality can be captured (at first-order types) as a kind of preservation property ("strong stability") generalizing Berry's notion of stability. The strongly stable functions between two QDCs, ordered stably, form a QDC, so that the notion of strong stability extends to higher-order types. They thus obtained a cartesian-closed category of QDCs and strongly stable functions. The work reported in [11] concerns a framework of sequential functions between "indexed domains", domains equipped with a parametrized notion of incremental computation; the indexed domains are closed under sequential function space, and this is true for both the stable order and the pointwise order; the model obtained there is fully abstract for a subset of PCF obtained by imposing a simple syntactic constraint on the use of application. In both cases [12, 11] the new definition of sequential functions coincides with the Kahn–Plotkin definition when restricted to concrete domains and first-order types, showing the robustness of the original work.

The Berry–Curien algorithms model is fully abstract with respect to an intensional form of program behavior (in which one can observe computation strategy) [7, 8]. Recent results [14, 17] have shown that an extended version of the sequential algorithms model (incorporating error values) yields an extensional model, fully abstract for an extension of PCF that includes certain control facilities. Indeed, the original sequential algorithms model is fully abstract for an extension of PCF including a "catch" control primitive, without requiring the inclusion of errors [13]. Despite these results the original full abstraction problem for PCF remains open. It is likely that any solution to this long-standing problem will build ultimately on the pioneering work of Kahn and Plotkin.

Acknowledgment

I would like to thank Samson Abramsky, Gérard Berry, Pierre-Louis Curien, Gilles Kahn, Gordon Plotkin and Glynn Winskel for helping to clarify the history behind this work and for suggesting various improvements to the text. Additional thanks to Samson Abramsky for encouraging Gilles Kahn to translate the original technical report.

References

[1] S. Abramsky, A generalized Kahn principle for abstract asynchronous networks, in: *Mathematical Foundations of Programming Semantics, Proc. 5th Internat. Conf.*, Lecture Notes in Computer Science, Vol. 442 (Springer, Berlin, 1989).

[2] G. Berry, Bottom-up computations of recursive programs, *RAIRO Inform. Théor. Appl.* **10** (1976) 47–82.

[3] G. Berry, Séquentialité de l'évaluation formelle des lambda-expressions, in: B. Robinet, ed., *Program Transformations, Proc. 3rd Internat. Colloq. on Programming* (Dunod, Paris, 1978).

[4] G. Berry, Stable models of typed λ-calculi, in: *Proc. 5th Colloq. on Automata, languages and programming*, Lecture Notes in Computer Science, Vol. 62 (Springer, Berlin, 1978) 72–89.

[5] G. Berry, Modèles complètement adéquats et stables des lambda-calculs typés, Thèse de Doctorat d'Etat, Université Paris VIII, 1979.

[6] G. Berry and B. Courcelle, Program equivalence and canonical forms in stable discrete interpretations, in: R. Milner and S. Michaelson, eds., *Proc. 3rd Internat. Conf. on Automata, Languages, and Programming* (Edinburgh Univ. Press, 1976).

[7] G. Berry and P.-L. Curien, Sequential algorithms on concrete data structures, *Theoret. Comput. Sci.* **20** (1982) 265–322.

[8] G. Berry, P.-L. Curien and J.-J. Lévy, Full abstraction for sequential languages: the state of the art, in: M. Nivat and J. Reynolds, eds., *Algebraic Methods in Semantics* (Cambridge Univ. Press, Cambridge, 1985).

[9] G. Berry and J.-J. Lévy, Minimal and optimal computations of recursive programs, *J. Assoc. Comput. Mach.* **26** (1979) 148–175.

[10] S. Brookes and S. Geva, Continuous functions and parallel algorithms on concrete data structures, in: *Proc. 7th Internat. Conf. on Mathematical Foundations of Programming Semantics*, Lecture Notes in Computer Science, Vol. 598 (Springer, Berlin, 1991).

[11] S. Brookes and S. Geva, Sequential functions on indexed domains and full abstraction for a sub-language of PCF, in: *Proc. 9th Internat. Conf. on Mathematical Foundations of Programming Semantics*, Lecture Notes in Computer Science, to appear.

[12] A. Bucciarelli and T. Ehrhard, Sequentiality and strong stability, in: *Proc. 6th Annu. IEEE Symp. on Logic in Computer Science*, 1991.

[13] R. Cartwright, P.-L. Curien and M. Felleisen, Fully abstract models of observably sequential languages, *Inform. and Comput.*, to appear.

[14] R. Cartwright and M. Felleisen, Observable sequentiality and full abstraction, in: *Proc. 19th Ann. ACM Symp. on Principles of Programming Languages*, 1992.

[15] Th. Coquand, C.A. Gunter and G. Winskel, DI-domains as a model of polymorphism, Technical Report 107, Cambridge University Computer Laboratory, 1986.

[16] Th. Coquand, C.A. Gunter and G. Winskel, Domain-theoretic models of polymorphism, *Inform. and Comput.* **81** (1989) 123–167.

[17] P.-L. Curien, Observable algorithms on concrete data structures, in: *Proc. 7th Annu. IEEE Symp. on Logic in Computer Science*, 1992.

[18] P.-L. Curien, *Categorical Combinators, Sequential Algorithms and Functional Programming*, Research Notes in Theoretical Computer Science (Pitman, London, 2nd ed., 1986); expanded and updated, published by Birkhäuser, Boston, 1993.

[19] M. Droste, Event structures and domains, *Theoret. Comput. Sci.* **68** (1989) 37–48.

[20] M. Droste, Recursive domain equations for concrete data structures, *Inform. and Comput.* **82** (1989) 65–80.

[21] M. Droste, On stable domains, *Theoret. Comput. Sci.* **111** (1993) 89–102.

[22] A. Faustini, The equivalence of a denotational and an operational semantics for pure dataflow. Ph.D. Thesis, University of Warwick, 1982.

[23] J.-Y. Girard, The system *F* of variable types, fifteen years later, *Theoret. Comput. Sci.* **45** (1986) 159–192.

[24] C. Gunter, *Semantics of Programming Languages* (MIT Press, Cambridge, MA, 1992).

[25] C.A. Gunter and D.S. Scott, Semantic domains, in: J. van Leeuwen, ed., *Handbook of Theoretical Computer Science, Vol. B: Formal Models and Semantics* (Elsevier, Amsterdam, 1990).

[26] G. Kahn, The semantics of a simple language for parallel processing, in: Information Processing 74 (North-Holland, Amsterdam, 1974).

[27] G. Kahn and D.B. MacQueen, Coroutines and networks of parallel processes, in: *Information Processing 77* (North-Holland, Amsterdam, 1977) 993–998.

[28] G. Kahn and G.D. Plotkin, Domaines Concrets, Rapport 336, IRIA-LABORIA, 1978.

[29] N.A. Lynch and E.W. Stark, A proof of the Kahn principle for input/output automata, *Inform. and Comput.* **82** (1989) 81–92.

[30] R. Milner, Fully abstract models of typed lambda-calculi, *Theoret. Comput. Sci.* **4** (1977) 1–22.

[31] M. Nielsen, G. Plotkin and G. Winskel, Petri nets, event structures and domains, *Theoret. Comput. Sci.* **13** (1981) 85–108.

[32] P. Panangaden, V. Shanbhogue and E.W. Stark, Stability and sequentiality in dataflow networks, Technical Report TR 89-1055, Cornell University, Department of Computer Science, 1989.

[33] D. Park, The "fairness" problem and non-deterministic computing networks, in: *Foundations of Computer Science IV, Part* 2, Vol. 159 (Mathematisch Centrum, Amsterdam, 1982).

[34] G.D. Plotkin, LCF considered as a programming language, *Theoret. Comput. Sci.* **5** (1977) 223–256.

[35] V. Yu. Sazonov, Sequentially and parallelly computable functionals, in: *Proc. Symp. on Lambda-Calculus and Computer Science Theory*, Lecture Notes in Computer Science, Vol. 37 (Springer, Berlin, 1975).

[36] D. Schmidt, *Denotational Semantics: A Methodology for Language Development* (Allyn & Bacon, Newton, MA, 1986).

[37] D.S. Scott, A type-theoretic alternative to CUCH, ISWIM, OWHY, unpublished manuscript, 1969.

[38] D.S. Scott, Outline of a mathematical theory of computation, in: *Proc. 4th Annu. Princeton Conf. on Information Sciences and Systems*, 1979; expanded version appeared as Technical Monograph PRG-2. Oxford University Programming Research Group, 1970.

[39] D.S. Scott and C. Strachey, Towards a mathematical semantics for computer languages, in: *Proc. Symp. on Computers and Automata*, Microwave Research Institute Symposia Series, Vol. 21 (Polytechnic Univ. of Brooklyn, 1971).

[40] D.S. Scott, Domains for denotational semantics, in: *Proc. ICALP '82*. Lecture Notes in Computer Science, Vol. 140 (Springer, Berlin, 1982).

[41] J. Stoy, *Denotational Semantics: The Scott–Strachey Approach to Programming Language Theory* (MIT Press, Cambridge, MA, 1977).

[42] C. Strachey, Towards a formal semantics, in: *Proc. IFIP TC2 Working Group on Formal Language Description Languages for Computer Programming* (North-Holland, Amsterdam, 1966).

[43] P. Taylor, An algebraic approach to stable domains, *J. Pure Appl. Algebra* **64** (1990) 171–203.

[44] R.D. Tennent, *Semantics of Programming Languages* (Prentice-Hall, Englewood Cliffs, NJ, 1991).

[45] J. Vuillemin, *Proof techniques for recursive programs*, Ph.D. Thesis, Stanford University, 1973.

[46] G. Winskel, *Events in Computation*, Ph.D. Thesis, Edinburgh University, 1981.

[47] G. Winskel, Event structures, in: W. Brauer, W. Reisig and G. Rozenberg, eds., *Petri Nets: Applications and Relationships to Other Models of Concurrency*, Lecture Notes in Computer Science, Vol. 255 (Springer, Berlin, 1987).

[48] G. Winskel, An introduction to event structures, in: *Lecture Notes for the REX Summer School on Temporal Logic (1988)*, Lecture Notes in Computer Science, Vol. 354 (Springer, Berlin, 1989).