Continuity in Semantic Theories of Programming

HISTORY AND PHILOSOPHY OF LOGIC, **00** (Month 200x), 1–19

Continuity in semantic theories of programming

FELICE CARDONE*

* Dipartimento di Informatica. Università di Torino,
Corso Svizzera 185, I-10149, Torino (Italy),
e-mail: felice.cardone@unito.it

Received 00 Month 200x; final version received 00 Month 200x

A Gabriele Lolli,
in occasione del suo settantesimo compleanno

Continuity is perhaps the most familiar characterization of the finitary character of the operations performed in computation. We sketch the historical and conceptual development of this notion by interpreting it as a unifying theme across three main varieties of semantical theories of programming: denotational, axiomatic and event-based. Our exploration spans the development of this notion from its origins in recursion theory to the forms it takes in the context of the more recent event-based analyses of sequential and concurrent computations, touching upon the relations of continuity with non-determinism.

1. Introduction

While various forms of hypercomputationalism are challenging one of the traditional cornerstones of computability theory, namely the finitary nature of the operations performed while calculating, we explore in this paper the historical development and the conceptual basis of what is perhaps the most familiar characterization of this finitary character: continuity.

The connections between classical computability and the topological notion of continuity are a central issue in recursion theory and found formal expression in a body of results from the second half of the 1950s, most notably the Rice-Shapiro and Myhill-Shepherdson theorems, see *Rogers 1967*. We shall not dwell here on these well-known milestones of recursion theory.[1] Rather, taking Kleene's First Recursion Theorem of *Kleene 1952* as a keynote, our investigation will be carried out mainly from the point of view of semantic theories of programming languages, and the overall structure of our story will match approximately the three main approaches to the semantics of programming languages: denotational, axiomatic and operational.

Continuity started playing its prominent role in the foundational work of Dana Scott on domains for denotational semantics at the end of the 1960s, see for example *Scott 1970*. On the one hand continuity was the essential feature of the abstract theory of computable higher-type functions proposed in *Scott 1969b*. On the other hand, continuous functions have elegant fixed-points properties that make them suitable for interpreting systems of recursive definitions. Furthermore, it turned out that the order-theoretic machinery developed in order to define continuity also allowed to solve the recursive domain equations needed in Strachey's approach to the denotational semantics of programming languages, leading to Scott's celebrated construction of models for the (untyped) λ-calculus in *1969a*. After introducing a small amount of examples and technical terminology pertaining to Scott domains, we shall discuss the informal arguments for continuity as a counterpart of computability, and outline the interpretation of (Scott) open subsets of a domain as computable properties.

[1] See *Longo 2005*. We also recommend *Longley 2001* for a survey of the many different notions of computable functional and their topological interpretations (see especially §2.4 therein).

History and Philosophy of Logic ISSN: 0144-5340 print/ISSN 1464-5149 online
http://www.informaworld.com DOI: 10.1080/0144534YYxxxxxxxx

The notion of continuity is formulated in the abstract language of partial order which is the mathematical context of Scott's domain theory. However, its scope is by no means limited to the denotational interpretation of programming constructs; indeed it translates directly into general theories of programming. Working on the language of guarded commands from an axiomatic perspective, *Dijkstra 1976* has shown that continuity is closely associated with bounded non-determinism. This enables another reading of continuity, as a means of ruling out computational supertasks like making infinitely many decisions in a finite amount of time, leading to a further manifestation of continuity in models of computation based on events. These originate from early approaches to the phenomenon of concurrency in computational systems, most notably *Petri 1963*, where computation is viewed as a spatially distributed, concurrent activity consisting in the occurrence of (discrete) events connected by a relation of causality. In this context continuity shows up in the form of finiteness restrictions imposed on the causal ordering among events.

In this paper we follow the historical development of continuity by expanding to some extent the above outline: we hope to show that this notion can be used as a conceptual key to many foundational issues in programming theory and their relations to abstract notions of computation.

2. The heritage of Kleene's first recursion theorem

2.1. *Kleene's fixed-point theorem*

From a strictly historical point of view, the first appearance of continuity, albeit implicit, can be found in Kleene's proof of his First Recursion Theorem in *Kleene 1952* (§66, pp. 348–50). After showing how recursive functions can be described by means of systems of equations (*ibid.*, §54) and investigating some basic properties of functionals seen as a means of expressing uniform definition of functions (§§47, 63), Kleene states the theorem in the following form:

> THEOREM XXVI. For any $n \geq 0$, let $\mathsf{F}(\zeta; x_1, \ldots, x_n)$ be a partial recursive functional, in which the function variable ζ ranges over partial functions of n variables. Then the equation
>
> $$\zeta(x_1, \ldots, x_n) \simeq \mathsf{F}(\zeta; x_1, \ldots, x_n) \tag{1}$$
>
> has a solution φ for ζ such that any solution φ' for ζ is an extension of φ, and this solution φ is partial recursive [...]

Restricting, for simplicity, to unary functions, we have a partially ordered set $(\mathrm{N} \rightharpoonup \mathrm{N})$ of *partial* functions, where ψ extends φ if and only if, for every $x \in \mathrm{N}$ such that $\varphi(x)$ is defined and equal to k, $\psi(x)$ is also defined and equal to k. The empty partial function is the least element in this partial ordering.

There are three main steps in the construction of the least solution to the equation (1) and they can be described to a large extent on the basis of the order structure of $(\mathrm{N} \rightharpoonup \mathrm{N})$.

First, we have the sequence $\varphi_0, \varphi_1, \varphi_2, \ldots$ of partial functions, where φ_0 is the completely undefined function, and $\varphi_{n+1}(x) \simeq \mathsf{F}(\varphi_n; x)$. It is proved in *Kleene 1952* (§64, Theorem XXI(a)), that F preserves the extension ordering (is *monotonic*), therefore (a straightforward inductive argument shows that) every term in this sequence extends all previous terms.

Second, we construct the "limit function":

$$\varphi(x) =_{\text{def}} \begin{cases} \varphi_s(x) & \text{if } \varphi_s(x) \text{ is defined for some } s \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Then $\varphi = \bigcup_{i \in \mathrm{N}} \varphi_i$ is the least function that extends all the terms in the sequence.

Finally, we prove that $\varphi(x) \simeq \mathsf{F}(\varphi; x)$. It is here that a property of the functional F depending on its finitary definition turns out to be necessary: this is where continuity comes to light. In fact, it is not necessarily true in general that the iteration of the functional F starting from

the completely undefined function yields a solution of (1) within ω steps. What is needed in addition is a proof that, when $\mathsf{F}(\varphi; x)$ is defined, then $\mathsf{F}(\varphi_s; x)$ is already defined for some index s. This exploits a compacteness argument justified by the definition of $\mathsf{F}(\varphi; x)$ by means of a finite system of equations. Because then, if $\mathsf{F}(\varphi; x) \simeq k$ we know that the proof of this fact uses a finite set of equations that correspond to pairs in the graphs of finite functions $\varphi_{s_1}, \ldots, \varphi_{s_j}$. By taking $s \geq \max\{s_1, \ldots, s_j\}$ we can prove that $\mathsf{F}(\varphi_s; x) \simeq k$, hence $\varphi_{s+1} \simeq \mathsf{F}(\varphi_s; x) \simeq k$, and finally $\varphi(x) \simeq k$.

2.2. *Domains for denotational semantics, and data types as lattices*

The importance of Kleene's First Recursion Theorem for the analysis of recursion in programming languages modeled via the λ-calculus was recognized at an early stage by James H. Morris in his MIT thesis, *Morris 1968*, Chapter III, who highlighted the use of the fixed-point combinator of the λ-calculus in solving functional equations. Morris also showed that what is obtained by this procedure is the least solution under (an analog of) the extension order on λ-terms defined in operational terms.[2]

However, it was Dana Scott who coded definitively the relevant notion of continuity in the language of abstract partially ordered sets and topology, in work carried out starting from the end of the 1960s on the mathematical foundations of the denotational style of semantics of programming languages developed by him in collaboration with Christopher Strachey at Oxford.[3] Scott himself recognized the influence of work in recursion theory on his ideas (especially *Lacombe 1955*, *Nerode 1959* and *Platek 1966*, see *Scott 1970*, §7) and developed a theory of computable functions (and functionals) over data types (like the data type of lists, for example) described as partially ordered structures, generically called *domains*, making their first appearance in *1969c*. The partial order over a domain represents the degree of definedness of its elements and is assumed to satisfy natural completeness and countability requirements providing a suitable notion of *limit* of increasing sequences of informations:

> Suppose $x, y \in D$ are two elements of the data type, [...] y, say, may be a *better* version of what x is trying to approximate. In fact, let us write the relationship $x \sqsubseteq y$ to mean intuitively that y is *consistent with* x and is (possibly) *more accurate than* x [...] thus $x \sqsubseteq y$ means that x and y want to approximate the same entity, but y gives *more* information about it. This means we have to allow "incomplete" entities, like x, containing only "partial" information (*Scott 1970*, pp. 170–71).

As a simple example, a domain associated with natural numbers might be one in which there is a totally undefined element $\perp$ dominated by pairwise incomparable elements $0, 1, 2, \ldots$. This represents situations where information about numerical values has an an all-or-nothing character. A different domain for natural numbers is one where the element $\perp$ containing no information can become more defined in two different ways: either as 0, whose information content cannot be further increased, or as a partial number $s(\perp)$ about which we only know that it is positive. The latter information can be further refined by yielding a maximal element 1 or another partial number $s(s(\perp))$ whose information content represents its being larger than 1, and so on following the same pattern.[4]

Given a generic data type $\langle D, \sqsubseteq \rangle$, a subset X of D is *directed* if every finite subset $u \subseteq X$ has an upper bound in X, i.e., an element $z \in X$ such that $x \sqsubseteq z$ for all $x \in u$. For example,

[2]The definition of this partial (pre)order on λ-terms, and the associated equivalence, was based on the behavior of terms when plugged into contexts. This contextual approach will have a huge influence in later research on the operational semantics of programming languages and process calculi for concurrent computation.

[3]For more details on the work of Scott and a sketch of its influence on later developments, especially in the model-theory of the λ-calculus, see *Cardone and Hindley 2009*, §9.1.

[4]This interpretation of information content for elements of domains is justified by the representation theory for domains as *information systems* developed later, in *Scott 1982*, where an element is identified with a consistent propositional theory. In the first example, beside a proposition Δ true of all elements, there are infinitely many pairwise inconsistent propositions of the form $\dot{n}$, for each $n \in \mathrm{N}$, where $\dot{n}$ is true of the natural number n only. In the second example, we also have propositions of the form $\breve{n}$ for each $n \in \mathrm{N}$, where $\breve{n}$ is true of all natural numbers m greater than n. A natural entailment relation between propositions in this case is generated by all sequents of the forms $\dot{n} \vdash \breve{m}$ and $\breve{n} \vdash \breve{m}$ whenever $m < n$.

every chain and the set of finite subsets of a set are directed. One completeness property that a data type should have is closure under least upper bounds of directed subsets. Let us call *complete partial order* (*cpo*) a partially ordered set $\langle D, \sqsubseteq \rangle$ with a least element $\perp_D$, such that every directed subset X of D has a least upper bound (a "limit") $\bigsqcup X$. If $\langle D, \sqsubseteq \rangle$ is a cpo, an element $d \in D$ is *finite* (or *compact*, or *algebraic*) if, for every directed $X \subseteq D$

$$d \sqsubseteq \bigsqcup X \Rightarrow \exists x \in X . d \sqsubseteq x.$$

For any $d \in D$, we denote by $\mathcal{A}(d)$ the set of finite elements $e \in D$ such that $e \sqsubseteq d$. $\mathcal{A}(D)$ is the set of finite elements of D.[5] A cpo $\langle D, \sqsubseteq \rangle$ is *algebraic* if, for all $d \in D$, the set $\mathcal{A}(d)$ is directed and $d = \bigsqcup \mathcal{A}(d)$. If $\langle D, \sqsubseteq \rangle$ is algebraic, then its *basis* is $\mathcal{A}(D)$.

Continuity is usually defined in terms of preservation of certain limits; continuity in the sense of Scott is defined as preservation of least upper bounds of directed sets:

Definition 2.1: If $\langle D, \sqsubseteq_D \rangle$ and $\langle E, \sqsubseteq_E \rangle$ are cpo's and $f : D \to E$ is a monotonic function (i.e., one for which $f(x) \sqsubseteq_E f(y)$ whenever $x \sqsubseteq_D y$) then f is *continuous* if

$$f(\bigsqcup X) = \bigsqcup f(X)$$

for all directed subsets X of D.

> If a function is *computable* in some intuitive sense, then getting out a "finite" amount of information about one of its values ought to require putting in only a "finite" amount of information about the argument (*Scott 1970*, p. 172).

Formally, we must have

$$e \in \mathcal{A}(f(x)) \Rightarrow \exists d \in \mathcal{A}(x) . e \sqsubseteq_E f(d).$$

We have the following important characterization:[6]

Proposition 2.2: *For algebraic cpo's $\langle D, \sqsubseteq_D \rangle, \langle E, \sqsubseteq_E \rangle$, and monotonic $f : D \to E$, the following conditions are equivalent:*

(1) For all $e \in \mathcal{A}(f(x))$ there is $d \in \mathcal{A}(x)$ such that $e \sqsubseteq_E f(d)$;
(2) f is continuous.

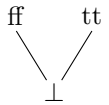
Continuity is closely related to computability also through its alternative definition, whereby a function $f : D \to E$ between topological spaces is continuous precisely when $f^{-1}(U) =_{\text{def}} \{d \in D \mid f(d) \in U\}$ is an open subset of D, for every open $U \subseteq E$. In the case of domains, this notion of continuity is coextensive with its definition as preservation of limits when we endow each domain D with the *Scott topology*, whose open sets are the $U \subseteq D$ that satisfy:

(1) $d \in U$ and $d \sqsubseteq d'$ impy $d' \in U$,
(2) if $\bigsqcup X \in U$ for some directed $X \subseteq D$, then $d \in U$ for some $d \in X$.

This is a generalization of the weak topology of *Nerode 1959*, defined on domains of the form $\mathcal{P}(A)$, for a set A. The computational meaning of Scott open sets emerges when we consider the logical interpretation of set-theoretical operations on open sets and look at the topology of a domain as a *theory*, guided by the analogy between open sets and semi-decidable properties first described in *Smyth 1983*:

> We think of a topological space as a "data type", with the open sets as the (*computable*) *properties* defined on the type. Taking a *predicate* on a space X to be a continuous map from X into the

[5] Observe that in the cpo $\mathcal{P}(X)$ of subsets of an arbitrary set X, $u \in \mathcal{P}(X)$ is finite if and only if it has finite cardinality.
[6] The proof can be found, for instance, in *Amadio and Curien 1998*, Proposition 1.1.16.

Boolean cpo B = 

, we have (trivially) that a subset S of X is open iff S is $p^{-1}(\mathrm{tt})$ for some predicate p [...] intuitively, the idea of a computable property p is simply this: we have a uniform procedure that, given (a code for) an element x, tells us within a finite time that $p(x)$ holds, whenever that is true (*Smyth 1983*, p. 664).

The importance of continuous functions for the semantics of recursion is a consequence of the following well-known and fundamental result, which formalizes in an abstract setting the essential part of Kleene's First Recursion Theorem:

Theorem 2.3: *Let $f : D \to D$ be a continuous function and $d \in D$ be such that $d \sqsubseteq f(d)$. Then $\bigsqcup_{n\in\mathrm{N}} f^{(n)}(d)$ is the least $x \sqsupseteq d$ such that $f(x) = x$.*

By induction on $n \in \mathrm{N}$ we prove that $d \sqsubseteq f^{(n)}(d)$. Clearly the sequence $\{d, f(d), f^2(d), \ldots\}$ is increasing and has a least upper bound which is a fixed point of f, and the least one dominating d. As a corollary, the least fixed point of a continuous $f : D \to D$ is

$$\mathsf{fix}(f) =_{\mathrm{def}} \bigsqcup_{n\in\mathrm{N}} f^{(n)}(\bot).$$

As an example of the use of fixed points of continuous function(al)s in the semantics of programming languages, consider the instruction **while** b **do** C in imperative programming, which should have the same interpretation as **if** b **then begin** C; **while** b **do** C **end**. Assuming that states form a domain and the interpretation of instruction C is given by a continuous state transformation $\mathcal{C}[\![C]\!]$, we need to solve the equation

$$\mathcal{C}[\![\textbf{while } b \textbf{ do } C]\!] = \mathcal{C}[\![\textbf{if } b \textbf{ then begin } C; \textbf{while } b \textbf{ do } C \textbf{ end}]\!].$$

The right hand side of this equation has the form $F(\mathcal{C}[\![\textbf{while } b \textbf{ do } C]\!])$ for a continuous functional F built by the techniques of denotational semantics, therefore we need in fact to solve a fixed-point equation

$$\mathcal{C}[\![\textbf{while } b \textbf{ do } C]\!] = F(\mathcal{C}[\![\textbf{while } b \textbf{ do } C]\!]),$$

which can be done by a straightforward application of Theorem 2.3 as, e.g., in *Stoy 1977* (p. 205).

The construction of the solution to the functional equation (1) in the First Recursion Theorem, whose graph contains exactly the pairs that have a proof from the system of equations associated with the functional F, is the prototype of a different connection between continuity and finite generation processes through inductive definitions, presented abstractly in terms of rules in *Aczel 1977*.

Definition 2.4: A (finitary) *rule* over a set A is a set Φ of pairs of the form $\langle X, a\rangle$, where $a \in A$ and $X \subseteq_{\mathrm{fin}} A$.

If $Y \subseteq A$, we say that Y is Φ-*closed* if $a \in Y$ whenever $\langle X, a\rangle \in \Phi$ for some $X \subseteq Y$. We can associate to every rule Φ over A a function $\Gamma_\Phi : \mathcal{P}(A) \to \mathcal{P}(A)$ by setting:

$$\Gamma_\Phi(Y) =_{\mathrm{def}} \{a \in A \mid \exists X \subseteq Y\ \langle X, a\rangle \in \Phi\}.$$

Then $Y \subseteq A$ is Φ-closed precisely when $\Gamma_\Phi(Y) \subseteq Y$; now, the set T_Φ *defined inductively* by Φ is the smallest Φ-closed subset of A, namely the set

$$T_\Phi =_{\mathrm{def}} \bigcap\{U \subseteq A \mid \Gamma_\Phi(U) \subseteq U\}.$$

What is especially interesting from the present point of view is that there is a bijective correspondence between finitary rules Φ and continuous operators $\Gamma_\Phi : \mathcal{P}(A) \to \mathcal{P}(A)$: the argument showing this is another instance of the leitmotiv associated with continuity since the proof of the First Recursion Theorem. In order to prove that Γ_Φ is continuous, using Proposition 2.2(1), we assume that $\beta \subseteq_{\text{fin}} \Gamma_\Phi(U)$: then $\beta = \{a_1, \ldots, a_n\}$, and there are $\langle X_1, a_1\rangle, \ldots, \langle X_n, a_n\rangle \in \Phi$ such that $X_i \subseteq U$. Then $\alpha = \bigcup_{i=1}^n X_i$ is still a finite subset of U and $\beta \subseteq \Gamma_\Phi(\alpha)$.

Finally, the set inductively generated by a rule can be characterized equivalently as the least fixed point of the continuous function Γ_Φ; in fact we can prove

Proposition 2.5: *If $\Gamma : \mathcal{P}(A) \to \mathcal{P}(A)$ is continuous, then*

$$\bigcup_{n\in\mathrm{N}} \Gamma^{(n)}(\emptyset) = \bigcap\{U \subseteq A \mid \Gamma(U) \subseteq U\}$$

3. Continuity and bounded nondeterminism

Alternative to the denotational style of semantics advocated by Strachey and Scott, whereby programming constructs are interpreted by means of suitable continuous functions over domains, was the axiomatic description of programs developed by Dijkstra, whose first systematic exposition is *Dijkstra 1976*.

The main feature of Dijkstra's approach is the association, to each program S of his guarded command language, of a function mapping each *post-condition* R describing a set of final states to its *weakest pre-condition* $\mathrm{wp}(S, R)$ satisfied by all and only the states such that the activation of S in each of these initial states will certainly result in a properly terminating happening leaving the system in a final state satisfying R. Such a function is the *predicate transformer* associated with the post-condition R (*Dijkstra 1976*, p. 16). Typical examples of statements in the guarded command language are the *alternative construct*

$$\mathrm{IF} =_{\text{def}} \textbf{if } G_1 \to S_1 \,[\!]\, \cdots \,[\!]\, G_n \to S_n \textbf{ fi}$$

(where each G_i is a *guard* expressing a condition whose truth enables the execution of the associated statement S_i) whose associated predicate transformer applied to a post-condition R gives

$$(\exists i \in \{1, \ldots, n\}\ G_i) \wedge (\forall i \in \{1, \ldots, n\}(G_i \Rightarrow \mathrm{wp}(S_i, R)))$$

and the *repetitive construct*

$$\mathrm{DO} =_{\text{def}} \textbf{do } G_1 \to S_1 \,[\!]\, \cdots \,[\!]\, G_n \to S_n \textbf{ do}$$

whose weakest pre-condition is, for any post-condition R,

$$\exists r \geq 0\ H_r(R) \tag{2}$$

where the sequence H_r is defined recursively by the clauses:

$$H_0(R) = R \wedge \neg\exists i \in \{1, \ldots, n\}\ G_i$$
$$H_{k+1}(R) = \mathrm{wp}(\mathrm{IF}, H_k(R)) \vee H_0(R)$$

Another important feature of Dijkstra's language is non-determinism, that arises when the guards G_i are not mutually exclusive. A program exhibits *unbounded* non-determinism when: (a) it is guaranteed to terminate, and (b) the set of final states arising from its execution is infinite.

Unbounded non-determinism can be identified with the presence of a command for random assignment of a natural number to a variable x, written $x :=?$ and characterized by the following predicate transformer:

(i) $\mathrm{wp}(x :=?, 0 \leq x) = \mathit{true}$,
(ii) $\mathrm{wp}(x :=?, x \leq s) = \mathit{false}$, for all $s \geq 0$.

One might suggest implementing random assignment by the statement

$$\begin{array}{lll} S: & \textbf{do} & go\_on \rightarrow x := x + 1 \\ & [\!] & go\_on \rightarrow go\_on := \mathit{false} \\ & \textbf{od} & \end{array}$$

However, the calculation of the corresponding predicate transformer yields $\mathrm{wp}(S, \mathit{true}) = \neg go\_on$. This means that termination is only guaranteed provided all the guards are false in the initial state (so all the assignments involved are skipped)

> and if we wish to stick to our interpretation of $\mathrm{wp}(S, \mathit{true})$ as the *weakest* precondition guaranteeing termination, we must reject any [implementation] in which the freedom of choosing would be exercised so "fairly" with respect to the various alternatives that each possible alternative will be selected sooner or later (*Dijkstra 1976*, p. 204)

because in those implementations the machine would be guaranteed to terminate by choosing eventually the second alternative. Therefore, the fact that unbounded non-determinism does not arise in the language of guarded commands is necessary in order to show that its axiomatic semantics in term of predicate transformers complies with the operational intuitions that underlie it. The same argument shows also that fairness and unbounded non-determinism are closely related problems.[7]

Dijkstra's proof that the non-determinism of guarded commands is bounded is part of an argument showing that his language is implementable. In fact, an implementation might be hampered by the absence of bounds on the integer values that can be assigned to variables. If programs are meant to be run on a hypothetical unbounded machine, then we may still hope that, for every fixed program S in a given initial state, the integers manipulated by S belong to a bounded interval: in this case the existence of a physical machine implementing S is not a priori impossible. In order to meet this requirement, however, non-determinism must be bounded. A first version of Dijkstra's proof, later replaced by the published version of Chapter 9 of *Dijkstra 1976*, used the operational description of the alternative and repetitive constructs:

> each guarded command list contains a fixed and finite number of alternatives. Secondly the a priori upper bound on the number of computational steps implies an upper bound on the number of times the non-deterministic choice can be made (*Dijkstra n.d.*, p. 3).

An appeal to König's lemma then allows to conclude that the execution tree of a program is finite, in particular it has finitely many leaves (i.e., final states). Therefore non-determinism is bounded. This argument was later rejected by Dijkstra's himself, for reasons discussed in *1982c*. On the one hand, the argument exploits operational aspects of guarded commands that are not part of their semantic definition:

> The fact that our program texts admit the alternative interpretation of "executable code" has played a role in our motivations, but plays no role in the definition of the semantics of our programming language [...] As long as we are interested in program correctness, it suffices to interpret the text as a code for a predicate transformer and nothing is gained by simultaneously remembering that the text can also be interpreted as executable code (*Dijkstra 1976*, pp. 202–03).

[7] Our discussion is based on Chapter 26 of *Dijkstra 1976* and on *Dijkstra n.d., 1982c,b,a*.

On the other hand, the argument does not prove a property of the language, but rather of one of its implementations. The operational approach, in itself, does not rule out the possibility of infinitely many permissible final states.

It is in the reformulation of the proof that non-determinism is bounded that continuity is essential,[8] where continuity in this context is equivalent to the property that, for any mechanism S and any infinite sequence of predicates $C_0, C_1, C_2, \ldots$ such that C_r implies C_{r+1} for all $r \geq 0$, we have

$$\mathrm{wp}(S, \exists r \geq 0\ C_r) = \exists s \geq 0\ \mathrm{wp}(S, C_s). \tag{CONT}$$

Then the argument may exploit the fact that, for every statement of the language, the corresponding predicate transformer is continuous, whereas the predicate transformer associated to random assignment is not, because otherwise

$$\begin{aligned} \mathit{true} &= \mathrm{wp}(x :=?, x \geq 0) \\ &= \mathrm{wp}(x :=?, \exists r \geq 0\ (x \leq r)) \\ &= \exists r \geq 0\ \mathrm{wp}(x :=?, x \leq r) \\ &= \exists r \geq 0\ \mathit{false} \\ &= \mathit{false} \end{aligned}$$

David Park in *1980* (§3), criticized Dijkstra's insistence on continuity, pointing out that the definition of the weakest pre-condition of a repetitive statement already implicitly exploits a continuity assumption. According to Dijkstra's definition, the statement

$$\begin{aligned} S: \quad & \mathbf{do} \quad x < 0 \to x :=? \\ & \quad [\!] \quad x > 0 \to x := x - 1 \\ & \mathbf{od} \end{aligned}$$

has the predicate $x \geq 0$ as weakest pre-condition corresponding to the post-condition *true* (*Dijkstra 1976*, p. 77), against the operational intuition according to which the statement terminates in every initial state. This is taken by him as an argument against unbounded non-determinism. Park objects that it is Dijkstra's definition (2) the responsible for the mismatch. One could define instead $\mathrm{wp}(S, \mathit{true})$ as the least fixed point of the monotonic, albeit not continuous, operator on predicates (taken as sets of states)

$$F(P) = (x = 0) \vee \mathrm{wp}(\mathbf{if}\ x < 0 \to x :=?\ [\!]\ x > 0 \to x := x - 1\ \mathbf{fi}, P),$$

by an application of Tarski's theorem on the existence of least fixed points of monotonic functions over complete lattices. Equivalently, one may extend to the transfinite the iterative construction of the least fixed point of F given in Theorem 2.3, observing that in this particular case the iteration takes $\omega + 1$ steps. Then, Park shows that, with this new definition of $\mathrm{wp}(S, \mathit{true})$, all integer values of x guarantee termination, in accordance with operational intuition.

[8] *Dijkstra 1976*, p. 78, acknowledges John C. Reynolds for pointing out the role of continuity in this context. Reynolds, in a personal communication to the author on 16 April, 2012, recalls as follows Dijkstra's original position: "I don't remember the date, but when Dijkstra developed his guarded commands logic, he gave a presentation to a WG2.3 meeting that I attended. During the presentation, he remarked that at the last minute he had decided not to impose fairness on the nondeterminism in his language. I told him I thought he had made the right decision, since if he imposed fairness, his rule for the while command would not be sound. I illustrated this with an example. Then I showed him how Scott continuity could be used to prove the while rule in the absence of fairness, but the proof failed in the presence of fairness. If I remember correctly, this came as a surprise to him, but nevertheless his intuition had been strong enough to steer him away from fairness."

An implicit answer to *Park 1980* can be read from *Dijkstra and van Gasteren 1986*, which proves that, even without assuming continuity, the characterization of the weakest pre-condition for the repetitive command as a least fixed point can be obtained without using tranfinite ordinals. This seems to integrate Park's objections with the point of view of Dijkstra.

But now the only reason for dismissing unbounded non-determinism is its physical unfeasibility:

> A mechanism of unbounded non-determinacy yet guaranteed to terminate would be able to make within a finite time a choice out of infinitely many possibilities: if such a mechanism could be formulated in our programming language, that very fact would present an insurmountable barrier to the possibility of implementation of that programming language (*Dijkstra 1976*, p. 77).

Thus, although a formal analysis of computation in operational terms has been sidestepped in Dijkstra's theory of predicate transformers, his discussion of unbounded indeterminacy leads in a natural way to look at computations in terms of steps and then to guarantee their physical feasibility by means of global finiteness contraints on their structure. This has been one of the main concerns of the approach to computation based on *events*, to which we turn now, looking for avatars of continuity in this new context.

4. Events in computation

Besides the linguistic standpoint represented, in different ways, by the work of Scott and Dijkstra, there is a parallel, in fact older tradition in the foundations of computing that focuses, instead of programming languages, on *computing systems*. The subject lies at the confluence of mathematical logic, biology, engineering and computer science.[9] Here, automata are regarded as systems of components distributed in space, whose behaviors consist of occurrences of physical events involving those components.

The constraints that automata and their behaviors must satisfy in order to be physically realizable are at the basis of the general model of computation developed by Carl Adam Petri since the beginning of the 1960s, starting with *Petri 1962*. It is here that a general notion of (computational) *event* inspired by physics, in particular relativity theory, makes its first appearance.

4.1. *Petri's analysis of concurrency*

The starting point of Petri's research is an argument against the adequacy of the traditional theory of (synchronous) automata as a model of physical computing machine that support communication by means of languages defined recursively, like most programming languages.

> The theory of automata is shown not capable of representing the actual physical flow of information in the solution of a recursive problem. The argument proceeds as follows:
>
> (1) We assume the following postulates: a) there exists an upper bound on the speed of signals; b) there exists an upper bound on the density with which information can be stored.
> (2) Automata of fixed, finite size can recognize, at best, only iteratively defined classes of input sequences. [...]
> (3) Recursively defined classes of input sequences that cannot be defined iteratively can be recognized only by automata of unbounded size.
> (4) In order for an automaton to solve a (soluble) recursive problem, the possibility must be granted that it can be extended unboundedly in whatever way it might be required.
> (5) Automata (as actual hardware) formulated in accordance with automata theory will, after a finite number of extensions, conflict with at least one of the postulates named above. Suitable

[9] A short historical account of the early results in this area, from the point of view of mathematical logic, is given in *Church 1963*.

conceptual structures for an exact theory of communication are then discussed, and a theory of communication proposed.

[...] The proposed representation differs from each of the presently known theories concerning information on at least one of the following essential points:

(1) The existence of a metric is assumed for neither space nor time nor for other physical magnitudes.
(2) Time is introduced as a strictly local relation between states.
(3) The objects of the theory are discrete, and they are combined and produced only by means of strictly finite techniques (*ibid.*, p. iii)

These ideas lead to a theory of *asynchronous* computing machines where no central clock synchronizing the operations of the different physical parts is assumed.[10] These can also be extended indefinitely while operating, something that would be prevented by the presence of a global clock signal:

if we assume that the temporal separation of the states is defined by an oscillator with a certain base frequency, then the signal transmission times possible after the extension of the machine may exceed the basic period of the oscillator; in any case, the basic period will be exceeded after some finite number of extensions. We must therefore lower the base frequency, and after some finite number of frequency reductions the metric properties of the components will have to be changed to fit the new base frequency (*Petri 1962*, p. 6).

In order to achieve this, it is not possible to rely any longer on traditional abstractions, in particular the notion of global *state* of a system is not available basically because, due to the relativistic limitations on the speed of signals, there is no objective notion of global time. Petri's reformulation of automata theory is a comprehensive "signal combinatorics" formulated as an axiomatic theory of *events* in space-time, *Petri 1967*, inspired by the axioms of Reichenbach and *Carnap 1958* (§48) from which Petri drops assumptions that have no operational meaning, for example the density of the ordering of events on a world-line as in *Carnap 1958* (T16):

we shall speak as if the discrete objects were embedded in a continuous space-time world, [but] this is by no means necessary; the ostensible continuum rests in the final analysis on the erroneous assumption that the axiom of density (see *Carnap 1958*, p. 154) has an operational sense (*Petri 1962*, p. 34).

One aspect of Petri's signal combinatorics that is especially relevant to our topic is the way he replaces the requirement of density. Carnap (*ibid.*, T35) proved that every three-dimensional cross section of space-time, formalized as a simultaneity class of events, intersects every world line, but this result depends on Dedekind continuity of the order of events, which is excluded from the combinatorial reconstruction of Petri. Instead he considers, in *Petri 1977, 1980, 1987, Petri and Smith 1987*, axioms for structures of the form (X, co), where co denotes *concurrency* and X is a set (of space-time points, or generically *elements*). Concurrency may be taken as incomparability of elements with respect to order. Concurrency is not transitive:

The first misconception is the firmly entrenched idea that all points of a process execution occur, *in reality*, at a well-defined point "in time", that is, on a linear absolute Newtonian time scale, *common to all observers* (*Petri 1987*, p. 6).

The theory of concurrency aims at being applicable to a wide range of situations where we have a relation of "indifference" which is symmetric but not transitive, like in measurement:

The second misconception also refers to observation. In the theories of observation and measurement, a basic predicate is

"x cannot be distinguished from y, by observation,
with respect to a specified order $<$"

[10] *Petri 1962* contains the blueprint of a Turing Machine built along these lines, whose cornerstone is the construction of an asynchronous, extendable push-down storage device. See also *Furtek 1973* for a description of the basic ideas.

> The specified order may refer to amounts of time, length, mass etc. Anyway, the "indifference" relation is a similarity like co (*ibid.*, p. 7)

The complement of the relation of concurrency expresses causal connection of elements. Let a *cut* be a set of pairwise concurrent elements which is maximal, in the sense that every element outside the cut is causally connected to some element in the cut. Dually, a *line* is a set of elements pairwise causally connected, and such that every element outside the line is concurrent with some element in the line. Then Petri requires the following property:

K-density: every cut and every line have one element in common.

K-density replaces Carnap's density assumption mentioned above. Its implications for the global structure of infinite sets of elements are somewhat surprising, however: while it looks as a requirement that there be enough space-time points, it entails *discreteness* properties for processes.

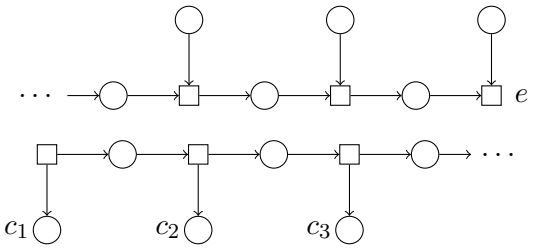
This is especially evident for concurrency structures arising form *Petri nets*, in particular *occurrence nets*. These are structures

$$N = (B, E, F)$$

where B is a set of *conditions*, E a set of *events* and $F \subseteq (B \times E) \cup (E \times B)$ is the *flow* relation, which is acyclic and such that each condition has at most one predecessor and at most one successor in the flow relation. The concurrency structure associated to an occurrence net has $X = B \cup E$ and co defined as incomparability with respect to the strict partial ordering $x < y$ that holds iff x is related to y by the transitive closure of the flow relation. We just describe the two simplest and most typical examples, from *Best 1980b*, in order to explain why K-density is indeed a discreteness property.[11] Consider the following occurrence nets, where circles and squares represent conditions and events, respectively, and arrows describe the flow relation:

(a)

(b)



In both nets we have a cut that does not intersect a line, equivalently a strict partial order with a maximal antichain that does not intersect any maximal chain, so neither of these is K-dense. Net (a) can be read as the description of a process where the event e can occur only after infinitely many events have occurred: this is an event with an infinite past. Net (b) is the description of a process where the cut $\{c_1, c_2, c_3, \ldots\}$ cannot be reached in finitely many steps: this represents the situation in which Achilles should find himself immediately before catching the Tortoise (an event not shown in the net), see *Best 1980b*. These "terminating non-terminating processes" are therefore ruled out as a consequence of global assumptions on the structure of space-time where computational events take place.

The same concerns about the exclusion of supertasks from axiomatic theories of computation emerged in the framework of another event-based model, Hewitt's theory of actors, that we now address.

[11] A complete study of discreteness properties of K-dense occurrence nets has been carried out in *Best 1980a, Best and Fernàndez 1988*.

4.2. *Message passing and asynchronous computation*

The approach to concurrency based on Petri nets spread rapidly among the researchers of the Computation Structures Group, led by Jack B. Dennis at the Massachusetts Institute of Technology. They were investigating the theory and practice of asynchronous computational systems exhibiting concurrent activity, and Petri nets were used very early as a fundamental formalism also under the influence of the work of Anatol W. Holt, head of the Information Systems Theory Project at Applied Data Research, reported in *Holt 1968, Holt and Commoner 1970*. At the same time was taking shape a notion of *object* as a basic metaphor in the development of what eventually came to be called *object-oriented programming*, see *Kay 1996*. It is in this milieu that emerged Carl Hewitt's theory of *actors*, a comprehensive attempt at modeling computations in terms of active entities distributed in space and coordinating through the exchange of messages, *Hewitt 1977*. As natural for distributed systems,[12] global states and global time are replaced also in Hewitt's approach by the use of partial orderings on events, to model various types of causal relations. Clearly, not every ordering is admissible as the description of a set of events that may actually take place during some computation; for this reason it is necessary to axiomatize the possible orderings of events. A set of laws constraining these partial orderings so as to rule out physically unfeasible computations has been arrived at by successive refinements, and is the result of joint efforts of Hewitt and his students at the Massachusetts Institute of Technology, starting from the work of Irene Grief *1975* and reaching a stable form in *Clinger 1981* after the fundamental work described in *Hewitt and Baker 1977, 1978a,b*.

Every event in the actor model of computation is the *arrival* of a *message* to a *target* actor. A message is itself an actor, so the participants in an event are the message, its target and the actors they can directly send messages to. Every event is required to have finitely many participants. One type of causality among events occurs when the receipt E_1 of a message by an actor results in the sending of another message to a second actor (not necessarily different from the sender). The arrival of that message to its target is then an event E_2 *caused* by E_1, and the two events are related in the *activation* ordering. A second type of causality takes into account the fact that events may be co-located: given an actor x, the events that take place at x are ordered in the *arrival* ordering. This ordering is linear and represents the *local* time of an actor. Causal dependence of events is described by the combined ordering $\longrightarrow$, defined as the least partial ordering of events containing the activation and arrival orderings.

The first[13] law that applies to the combined ordering rules out cycles in causal chains, i.e., the combined ordering is strict:

Law of strict causality. For no event E, $E \longrightarrow E$.

Another finiteness requirement on actor computation entails the well-foundedness of the causal ordering:

Law of finite predecession. For all events E, the set of events E' such that $E' \longrightarrow E$ is finite.

Together with the assumption that there are only countably many events, Clinger (*loc. cit.*, Theorem 1, p. 32) proves that these law are equivalent to an axiom that relates the combined ordering to a notion of *global* time:

The strong axiom of realizability. There exists a one-to-one mapping g from the events $\mathbf{E}$ into the nonnegative reals that preserves the combined ordering $\longrightarrow$ and such that $g^{-1}(I)$ is finite for every bounded interval I of R. Equivalently there exists a one-to-one mapping $g : \mathbf{E} \to \omega$ that preserves $\longrightarrow$, where ω is the set of natural numbers (*Clinger 1981*, p. 28).

A weak version of the axiom of realizability is obtained by dropping the restriction to non-

[12] "A system is distributed if the message transmission delay is not negligible compared to the time between events in a single process. [...] In a distributed system, it is sometimes impossible to say that one of two events occurred first. The relation "happened before" is therefore only a partial ordering of the events in the system", *Lamport 1978*, p. 558.

[13] Our account here merges *Hewitt and Baker 1977, 1978a,b* with the streamlined version of the laws in *Clinger (1981)*, Chapter II.

negative reals, and by taking the set Z of integers instead of the natural numbers. Observe that now the Law of finite predecession fails: this corresponds intuitively to allowing events with infinite past. Instead, one can use the

Law of discreteness. For all events E_1, E_2, the set $\{E \mid E_1 \longrightarrow E \longrightarrow E_2\}$ is finite.

Equivalently we have the

Law of Finite Chains between events in the Combined Ordering. There are no infinite chains of events between two events in the strict partial ordering $\longrightarrow$.

The insistence on such finiteness properties is, of course, motivated by the requirement that computations be physically realizable by ruling out the pathologies that, as we have seen, also puzzled Dijkstra:

> So-called Zeno machines are paradoxical machines that can do infinitely many things in a finite amount of time. An example is Huffman's Lamp, which when switched on lights for only thirty seconds before turning itself off for fifteen seconds, and then comes back on for seven and a half seconds before turning off for three and three quarters seconds, and so on. After one minute it ceases to change state. At one second into the second minute, is it on or off? Zeno machines, if they existed, could be used for many useful purposes such as providing a decision procedure for first order predicate calculus (*Clinger 1981*, p. 28).

It is precisely the axiom of realizability (in one of its forms) that allows to rule out such hyper-computational devices as Zeno machines, because then a computation embedded in real time having bounded duration can only consist of finitely many events (Clinger, loc. cit.). The non-existence of Zeno machines is entailed by the requirement that events occurring in computations form *discrete* collections. We can read the (weak) axiom of realizability as one such requirement, because it is equivalent to each of the following assumptions:

- the derived set of the image $g(\mathrm{E}) \subseteq \mathrm{R}$, under the order topology on R, is empty,
- for every bounded interval $I \subseteq \mathrm{R}$, the intersection $I \cap g(\mathrm{E})$ contains finitely many points (*Clinger 1981*, pp. 28–9).

There are two important types of messages in actor computations:

(1) *requests*, that include a message and another actor (the *continuation*) to which the reply should be sent, of the form $a \leftarrow [\mathit{request} : n, \mathit{reply\text{-}to} : a']$, and
(2) *replies* to actors, of the form $a \leftarrow [\mathit{reply} : n]$.

When a request event E happens, it may spawn an *activity* that consists of the events that are caused by E but precede the reply associated – in a natural sense – to that request. Then events E_1 and E_2 are related in the *continuation* ordering $E_1 \overset{\text{cont}}{\longrightarrow} E_2$ if $E_1 \longrightarrow E_2$ and both events are part of the same activity (so the continuation ordering is included in the combined ordering). As an example, it may be useful to look into the details of the events involved in the actor computation of $\mathit{fib}(3)$, where

$$\begin{aligned} &\mathit{fib}(1) = 1 \\ &\mathit{fib}(2) = 1 \\ &\mathit{fib}(n) = \mathit{fib}(n-1) + \mathit{fib}(n-2) \qquad\qquad \text{for } n > 2. \end{aligned} \tag{3}$$

Here *fib* and + are actors; the computation is triggered by the request of the value of $\mathit{fib}(3)$, with the corresponding reply to be sent to actor c. The receipt of this request immediately causes two further request events $\mathit{fib} \leftarrow [\mathit{request} : 2, \mathit{reply\text{-}to} : c']$ and $\mathit{fib} \leftarrow [\mathit{request} : 1, \mathit{reply\text{-}to} : c'']$, whose replies $c' \leftarrow [\mathit{reply} : 1]$ $c'' \leftarrow [\mathit{reply} : 1]$ can be immediately sent to actors c', c'', respectively. Upon reception of the replies, a further request $+ \leftarrow [\mathit{request} : [1\ 1], \mathit{reply\text{-}to} : c]$ is sent to actor + who replies to c by sending the value 2, which is the final value of the computation of $\mathit{fib}(3)$.

We can compare two approaches to recursive definitions like that of *fib*: one based on continuous functionals like in the denotational approach of Scott discussed in §2.2, and another based on events, like that we have just outlined. Hewitt and Baker do this by observing that on the graph of *fib* it is possible to single out, for each pair $(n, fib(n))$, a set of *immediate descendants* which is finite by the laws on causal orders. These are the pairs $(n-1, fib(n-1))$ and $(n-2, fib(n-2))$ that are needed to compute the value of *fib* at argument n. In the example, we have that the immediate descendants of the pair $(3, 2)$ are $(2, 1)$ and $(1, 1)$. If we write $\Phi(X)$, for any subset X of the graph of *fib*, to denote the set of pairs whose immediate descendants are in X, we can prove the following[14]

> **Theorem:** If an actor f behaves like a mathematical function then Φ is a continuous functional in the sense of Scott and graph(f) is the limit of Φ beginning with the empty graph $\emptyset$, i.e.
>
> $$\text{graph}(f) = \bigcup_{i \in \mathrm{N}} \Phi^i(\emptyset)$$
>
> where graph(f) is the set of input-output pairs of f. It immediately follows that graph(f) is the minimal fixed point of Φ since graph(f) = Φ(graph(f)).
> The above theorem makes precise the physical basis for believing that the graph of every physically realizable mathematical function is the limit of a continuous functional: the Law of Finitely Many Immediate Successors and The Law of Finite Chains between two Events in the Continuation Ordering. (*Hewitt and Baker 1978b*, p. 22)

The proof of this theorem is an application of the argument that we have already encountered in Section 2.2 while proving that $\Gamma_\Phi : \mathcal{P}(A) \to \mathcal{P}(A)$ is a continuous function for a finitary rule Φ, and is just another instance of the idea that the elements of the least fixed point of such a function are exactly those generated by a finite number of application of the rule Φ (hence have a well-founded proof).

4.3. *Events in sequential computations*

The event-based analysis of computations that lies at the basis of the actor model are remarkably close to ideas that independently arose in the French school of researchers working on the semantics of recursive program schemes along the path set by the works of Nivat and Vuillemin. Among these, Gerard Berry investigated *bottom-up* computations, where recursion is conceived as a production mechanism:

> a program is considered as defining a *recursion structure* to which several exploration algorithms can be applied. In general, a recursion structure is defined on the graph of the function being computed [...] and is characterized by the relation "x is an immediate intermediate value in the computation of y" [...] under certain conditions concerning the base functions and here satisfied, this relation is a well-founded partial order such that each point dominates finitely many points. The recursion structure can then be regarded as an infinite acyclic graph [...] If x is a point of the graph, we shall call *minimal producer* of x, abbreviated $mp(x)$, the set of immediate predecessors of x, and *domain of* x, abbreviated $dom(x)$, the set of all predecessors of x: $dom(x)$ is the set of intermediate values necessary and sufficient for the computation of x (*Berry 1977*, pp. 115–16, my translation).
>
> The production mechanism can be illustrated by the Fibonacci program [as in (3) above[15]]. Given a subset X of the graph of *fib*, we can use the program to generate new points on the graph. For instance, if X contains the two points $(4, fib(4) = 3)$ and $(5, fib(5) = 5)$, we can generate the new point $(6, fib(6) = 3 + 5 = 8)$. More precisely, the set produced by X in this example can be defined

[14]The notation has been slightly adapted. The proof uses an axiom that *Clinger 1981*, Chapter II, drops, namely the Law of Finitely Many Immediate Successors. However Clinger shows that the theorem holds even without that assumption.
[15]The example has been slightly adapted to ease the comparison with its development in the context of actor theory.

by

$$\Phi(X) = \{(n, z) \in \mathrm{N}^2 \mid \text{if } n = 1 \vee n = 2 \text{ then } z = 1 \text{ else } \exists y_1, y_2, z = y_1 + y_2 \wedge (n-1, y_1) \in X \wedge (n-2, y_2) \in X\}$$

> The purpose [...] is to formalize a translation of a recursive program into a predicate defining the production function Φ. This translation is such that the least fixpoint of Φ represents the graph of the function computed by the program (*Berry 1976*, p. 49).

A model of computation based on events is essential in order to study intensional aspects of computations, like *sequentiality* in programming languages, especially in relation to the problem of *full abstraction* for PCF, a typed λ-calculus extended with arithmetic constants and recursion, *Milner 1977, Plotkin 1977*. Basically, the problem consists in building denotational models for programming languages where the induced denotational equivalence matches the operational equivalence that identifies two terms if either can be replaced by the other in the same program context without altering the behavior.[16] This involves a certain amount of reconstruction of operational features within the models. In particular, it is necessary to define domains reflecting the way information on the result of an expression grows while the computation progresses. Therefore we need semantical structures that allow to reason about *events* that happen at possibly remote *places*. This, and the need to give a semantical definition of *sequential* continuous function, led to the theory of concrete domains of *Kahn and Plotkin 1978* and to the comprehensive theory of events in computation put forward in *Winskel 1980*, unifying to a large extent the ideas of Petri and Scott. Apparently we are now very far from the partial order on data that originally motivated Scott in his choice of the axioms for domains. Yet, we can recover a natural interpretation for these axioms and new ones also in these event-based models:

> information has to do with (occurrences of) events: namely the information that those events occurred. For example [...] suppose we have a Turing machine (TM) with an input tape and an output tape. The tape squares can be blank, or, after printing, contain a 0 or a 1. The events that can occur are the printing of a 0 or a 1 on the (next) output tape square by the TM or the inputting (by some unspecified agency) of a 0 or a 1 on the (next) input tape square. Thus the output domain is just the collection of all possible sets of events that could occur, ordered by the subset ordering; equivalently this is the set of all finite or infinite binary sequences with the subsequence ordering. For example 0111 means that 0 was printed on the first tape square, and 1 on the second, third and fourth tape squares, and thereafter there was no more output. The same (isomorphic) domain is associated with the input. The cpo is called Tapes, not unnaturally, and the reader will see that the function $f : \mathsf{Tapes} \to \mathsf{Tapes}$ computed by the TM must be continuous. [...] In terms of events a finite amount of information should just be a finite set of events. [...] It is reasonably self-evident that any physically feasible function must be monotonic and obey [the formulation of Scott continuity in Proposition 2.2(1)] as an output event could hardly depend on infinitely many input events as a machine should only be able to do a finite amount of computation before causing a given output event (*Plotkin 1978*).

We find here another instance of what has been one of the recurring themes of our account, which closes with a formal statement of the relations between what Winskel has called *Scott's Thesis*, stating that computable functions are continuous, and the discreteness properties of computable processes:

> assume a process is modelled by a partial order on events, $\mathrm{E} = (E, \leq)$. [...] We can chose to imagine some of the events of E as being events of input E_0 from some datatype, some as internal events, and others as events of output E_1 to some datatype. The datatypes may have their own causal dependencies, which contribute to the dependency of the full process, so the input datatype can carry a partial order $\mathrm{E}_0 = (E_0, \leq_0)$ and the output datatype a partial order $\mathrm{E}_1 = (E_1, \leq_1)$. [...]

[16] A survey of the quest for a solution to this problem, finally achieved in *Abramsky et al. 1994, Hyland and Ong 2000*, is given in *Ong 1995*.

There are natural domains of information associated with the two datatypes, *viz.* their domains of left-closed sets of events [$\mathcal{L}(E_0)$ and $\mathcal{L}(E_1)$]. The process induces a function between the domains. Define

$$f_{E_0,E_1} : \mathcal{L}(E_0) \to \mathcal{L}(E_1)$$

to map $x \mapsto \{e \in E_1 \mid \lceil e \rceil \cap E_0 \subseteq x\}$ [where $\lceil e \rceil$ denotes the principal order ideal generated by e]. The idea is that an event of E occurs once the necessary input events have occurred. [...] We say E *obeys Scott's thesis* iff

$$\forall \mathrm{E}_0, \mathrm{E}_1.(\mathrm{E}_0 \subseteq \mathrm{E} \& \mathrm{E}_1 \subseteq \mathrm{E} \Rightarrow f_{E_0,E_1} \text{ is continuous}).$$

[...] *The partial order E obeys Scott's thesis iff* $\forall e \in E.\{e' \in E \mid e' \leq e\}$ *is finite* (*Winskel 1987*, §1.4).

5. Concluding remarks

We have investigated the developments of continuity as a unifying theme in semantical accounts of programming concepts mainly by following a trail of quotations where the ideas and their motivations have been observed *in statu nascendi*.

There are other paths which we could have taken. For example, it would be possible to present the episodes of the story outlined here as a collection of interpretations of (the proof of) the fixed point theorem, in one of the forms we have discussed in §2.2. This choice would lead to a very formal setting to the detriment of the historical development, yet we believe that it might be an interesting experiment to carry out.

There are several ways in which the matter of this paper could be pursued. We have outlined the relations of continuity with bounded non-determinism and we have mentioned in passing how fairness is involved, but the status of fairness is controversial (see for example *Dijkstra 1988* and *Lamport and Schneider 1988*). It would be interesting to extend our investigation to the analysis of this status from a historical standpoint. In a different direction, we point out that the few remarks of Section 4.1 on the work of Carl Adam Petri touch upon a tiny part of his ideas on the relations between physics and computing. We are currently studying how these developed, in the hands of Petri and of Anatol W. Holt, into a comprehensive theory of computers broadly conceived as devices for the disciplined information flow in support to organized human activity.

Acknowledgments. The author is grateful to Prof. John Reynolds for providing information on the topics discussed in section 3. He also thanks the two anonymous referees, who suggested several improvements on an earlier version of this paper.

References

Abramsky, S., Jagadeesan, R., and Malacaria, P., 1994. Full abstraction for PCF (extended abstract), *in:* H. Hagiya and J.C. Mitchell, eds., *Theoretical Aspects of Computer Software. International Symposium TACS '94*, Berlin/Heidelberg: Springer-Verlag, *Lecture Notes in Computer Science*, vol. 789, 1–15.

Aczel, P., 1977. An introduction to inductive definitions, *in:* J. Barwise, ed., *Handbook of Mathematical Logic*, Amsterdam: North-Holland, 739–782.

Amadio, R. and Curien, P.L., 1998. *Domains and Lambda-Calculi*, *Cambridge Tracts in Theoretical Computer Science*, vol. 46, Cambridge, England: Cambridge University Press.

Berry, G., 1976. Bottom-up computation of recursive programs, *RAIRO Informatique Théorique et Applications*, **10**, 47–82.

Berry, G., 1977. Calculs ascendants du programme d'Ackermann: Analyse du programme de J. Arsac, *RAIRO Informatique Théorique et Applications*, **11**, 113–126.

Best, E., 1980a. The relative strength of K-density, *in:* W. Brauer, ed., *Net Theory and Applications*, Berlin/Heidelberg: Springer-Verlag, *Lecture Notes in Computer Science*, vol. 84, 261–276.

Best, E., 1980b. A theorem on the characteristics of nonsequential processes, *Fundamenta Informaticae*, **3** (1), 77–94.

Best, E. and Fernàndez, C., 1988. *Nonsequential Processes*, Berlin/Heidelberg: Springer-Verlag.

Cardone, F. and Hindley, J.R., 2009. Lambda-calculus and combinators in the 20th century, *in:* D.M. Gabbay and J. Woods, eds., *Handbook of the History of Logic, Volume 5: Logic from Russell to Church*, Amsterdam: North-Holland, 723–817.

Carnap, R., 1958. *Introduction to Symbolic Logic and Its Applications*, New York, NY, USA: Dover Publications.

Church, A., 1963. Logic, arithmetic, and automata, *in: Proceedings of the International Congress of Mathematicians. International Congress of Mathematicians, 15–22 August 1962, Stockholm*, Institut Mittag-Leffler, Djursholm, Sweden, 1963, 23–35.

Clinger, W., 1981. Foundations of Actor Semantics, Technical Report AI-TR-633, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, Massachusetts.

Dijkstra, E.W., 1976. *A Discipline of Programming*, Prentice-Hall.

Dijkstra, E.W., 1982a. The equivalence of bounded nondeterminacy and continuity, *in: Selected Writings on Computing: A Personal Perspective*, Berlin/Heidelberg: Springer-Verlag, 358–359.

Dijkstra, E.W., 1982b. On weak and strong termination, *in: Selected Writings on Computing: A Personal Perspective*, Berlin/Heidelberg: Springer-Verlag, 355–357.

Dijkstra, E.W., 1982c. A somewhat open letter to EAA or: why I proved the boundedness of the non-determinacy in the way I did, *in: Selected Writings on Computing: A Personal Perspective*, Berlin/Heidelberg: Springer-Verlag, 284–287.

Dijkstra, E.W., 1988. Position paper on "fairness", *SIGSOFT Software Engineering Notes*, **13** (2), 18–20, circulated privately in October 1987 as typewritten note EWD1013.

Dijkstra, E.W., n.d. On avoiding the infinite, circulated privately.

Dijkstra, E.W. and van Gasteren, A.J.M., 1986. A simple fixpoint argument without the restriction to continuity, *Acta Informatica*, **23** (1), 1–7.

Furtek, F.C., 1973. Asynchronous push-down stacks, Computation Structures Group Memo 86, Project MAC, Massachusetts Institute of Technology.

Grief, I., 1975. Semantics of communicating parallel processes, Technical Report MAC TR-154, Laboratory for Computer Science, MIT.

Hewitt, C., 1977. Viewing control structures as patterns of passing messages, *Artificial Intelligence*, **8**, 323–363.

Hewitt, C. and Baker, H., 1977. Laws for communicating parallel processes, *in:* B. Gilchrist, ed., *Information Processing 77*, Amsterdam: North-Holland, 987–992.

Hewitt, C. and Baker, H., 1978a. Actors and Continuous Functionals, *in:* E. Neuhold, ed., *Formal Description of Programming Concepts*, Amsterdam: North-Holland, 367–390.

Hewitt, C. and Baker, H., 1978b. Actors and Continuous Functionals, Technical report MIT/LCS/TR-194, Laboratory for Computer Science, Massachusetts Institute of Technology.

Holt, A., 1968. Final Report of the Information System Theory Project, technical report RADC-TR-68-305, Rome Air Development Center, Air Force Systems Command, Griffiss Air Base.

Holt, A. and Commoner, F., 1970. Events and conditions, *in: MIT Conference on Concurrent Systems and Parallel Computation*, New York, NY, USA: Association for Computing Machinery, 3–52.

Hyland, J.M.E. and Ong, C.H.L., 2000. On full abstraction for PCF: I. Models, observables and the full abstraction problem, II. Dialogue games and innocent strategies, III. A fully abstract and universal game model, *Information and Computation*, **163**, 285–408.

Kahn, G. and Plotkin, G.D., 1978. Domaines concrèts, Rapport 336, IRIA Laboria, english

transl. in *Theoretical Computer Science* 121 (Böhm Volume 1993), pp. 187–277.

Kay, A., 1996. The early history of Smalltalk, *in:* T. Bergin and K. Gibson, eds., *History of Programming Languages II*, New York: ACM Press, 511–598.

Kleene, S.C., 1952. *Introduction to Metamathematics*, New York: Van Nostrand.

Lacombe, D., 1955. Extension de la notion de fonction récursive aux fonctions d'une ou plusieurs variables réelles, I, *Comptes rendus hebdomadaires des séances de l'Académie des sciences*, **240**, 2478–2480.

Lamport, L., 1978. Time, clocks and the ordering of events in a distributed system, *Communications of the ACM*, **21** (7), 558–565.

Lamport, L. and Schneider, F., 1988. On E. W. Dijkstra's position paper on "fairness", *SIGSOFT Software Engineering Notes*, **13** (3), 18–19.

Longley, J.R., 2001. Notions of computability at higher types, I, *in: Logic Colloquium 2000*, Natick, Massachusetts, USA: A. K. Peters, *Lecture Notes in Logic*, vol. 19, 32–142.

Longo, G., 2005. Some topologies for computations, *in:* J. Kouneiher, D. Flament, P. Nabonnand, and J.J. Szczeciniarz, eds., *Géométrie au XXe siècle, 1930–2000: Histoire et horizons*, Paris: Hermann, 662–675.

Milner, R., 1977. Fully abstract models of typed λ-calculi, *Theoretical Computer Science*, **4**, 1–22.

Morris, J.H., 1968. *Lambda-calculus Models of Programming Languages*, Ph.D. thesis, Massachusetts Institute of Technology, Cambridge, Mass., U.S.A.

Nerode, A., 1959. Some Stone spaces and recursion theory, *Duke Mathematical Journal*, **26**, 397–405.

Ong, C.H.L., 1995. Correspondence between operational and denotational semantics, *in:* S. Abramsky, D. Gabbay, and T.S.E. Maibaum, eds., *Handbook of Logic in Computer Science*, England: Oxford Univ. Press, vol. 4, 269–356.

Park, D., 1980. On the semantics of fair parallelism, *in: Abstract Software Specifications. Proceedings of the Copenhagen Winter School*, Berlin/Heidelberg: Springer-Verlag, *Lecture Notes in Computer Science*, vol. 86, 167–183.

Petri, C.A., 1962. *Kommunikation mit Automaten*, Ph.D. thesis, Darmstadt Technical University, English translation as Technical Report RADC-TR-65-377, Vol. 1, Supplement 1, Griffiss Air Force Base, 1966.

Petri, C.A., 1963. Fundamentals of a theory of asynchronous information flow, *in:* C.M. Popplewell, ed., *Proc. of IFIP Congress 62*, Amsterdam: North Holland, 386–390.

Petri, C.A., 1967. Grundsätzliches zur beschreibung diskreter prozesse, *in: 3. Colloquium über Automatentheorie*, Basel: Birkhäuser Verlag, vol. 6, 121–140.

Petri, C.A., 1977. Non-sequential processes, Interner Bericht ISF-77-5, Gesellschaft für Mathematik und Datenverarbeitung, Bonn.

Petri, C.A., 1980. Concurrency, *in:* W. Brauer, ed., *Net Theory and Applications*, Berlin/Heidelberg: Springer-Verlag, *Lecture Notes in Computer Science*, vol. 84, 251–260.

Petri, C.A., 1987. Concurrency theory, *in:* W. Brauer, W. Reisig, and G. Rozenberg, eds., *Petri Nets: Central Models and Their Properties, Advances in Petri Nets 1986, Part I, Proceedings of an Advanced Course, Bad Honnef, September 1986*, Berlin/Heidelberg: Springer-Verlag, *Lecture Notes in Computer Science*, vol. 254, 4–24.

Petri, C.A. and Smith, E., 1987. Concurrency and Continuity, *in:* G. Rozenberg, ed., *Advances in Petri Nets*, Berlin/Heidelberg: Springer-Verlag, *Lecture Notes in Computer Science*, vol. 266, 273–292.

Platek, R.A., 1966. *Foundations of Recursion Theory*, Ph.D. thesis, Stanford University.

Plotkin, G., 1978. The category of complete partial orders: a tool for making meanings, *in: Proceedings of the Summer School on Foundations of Artificial Intelligence and Computer Science*, Pisa, Italy: Istituto di Scienze dell'Informazione, Università di Pisa.

Plotkin, G.D., 1977. LCF considered as a programming language, *Theoretical Computer Science*, **5**, 223–257.

Rogers, H., 1967. *Theory of Recursive Functions and Effective Computability*, New York: McGraw Hill.

Scott, D.S., 1969a. Lattice-theoretic models for the λ-calculus, incomplete typescript, 50 pp., Oxford University.

Scott, D.S., 1969b. A theory of computable functions of higher type, Informally distributed, notes for a November 1969 seminar, Oxford University.

Scott, D.S., 1969c. A type-theoretical alternative to ISWIM, CUCH, OWHY, Informally distributed, dated October 1969. Published with additions by the author in *Theoretical Computer Science* **121** (1–2) (1993), pp. 411–440.

Scott, D.S., 1970. Outline of a mathematical theory of computation, *in: Proceedings of the Fourth Annual Princeton Conference on Information Sciences and Systems*, Dept. of Electrical Engineering, Princeton University, 169–176, see also Technical Monograph PRG-2, Programming Research Group, Oxford University, England.

Scott, D.S., 1982. Domains for denotational semantics, *in:* M. Nielsen and E. Schmidt, eds., *Automata, Languages and Programming, Ninth International Colloquium*, Berlin/Heidelberg: Springer-Verlag, *Lecture Notes in Computer Science*, vol. 140, 577–613.

Smyth, M.B., 1983. Powerdomains and predicate transformers: a topological view, *in:* J. Diaz, ed., *Automata, Languages and Programming, Tenth International Colloquium*, Berlin/Heidelberg: Springer-Verlag, *Lecture Notes in Computer Science*, vol. 154, 662–675.

Stoy, J., 1977. *Denotational Semantics: The Scott-Strachey Approach to Programming Language Theory*, Cambridge, Mass., U.S.A.: M.I.T. Press.

Winskel, G., 1980. Events in computation, Ph.D. thesis, University of Edinburgh, Department of Computer Science, Edinburgh.

Winskel, G., 1987. Event structures, *in: Petri Nets: Applications and Relationships to other models of concurrency*, Berlin/Heidelberg: Springer-Verlag, *Lecture Notes in Computer Science*, vol. 255, 325–392.