

Physical Systems as Constructive Logics

Peter Hines*

York University, York YO10 5DD,
North Yorkshire, U.K.
`peter.hines@cs.york.ac.uk`

Abstract. This paper is an investigation of S. Wolfram’s *Principle of Computational Equivalence* – that (discrete) systems in the natural world should be thought of as performing computations. We take a logical approach, and demonstrate that under almost trivial (physically reasonable) assumptions, discrete evolving physical systems give a class of logical models. Moreover, these models are of *intuitionistic*, or *constructive* logics – that is, exactly those logics with a natural computational interpretation under the Curry-Howard ‘proofs as programs’ isomorphism.

1 Introduction

One of the more notable claims of [11] is that *physical systems* should naturally be thought of as *computational systems*. Although this claim is often backed up with reference to models of computation based on either cellular automata (as in [11]) or Turing machines [3], we address this claim from a *logical* perspective.

We consider a very broad class of physical systems evolving (in discrete steps) over time, and study them as though they are physical computers. We consider different descriptions of how they evolve over time, and introduce a partial ordering on this set of *machine evolutions*. Under assumptions about termination, this gives a familiar class of logical models – Heyting algebras. These are related to the *constructive* or *computational* logic known as intuitionistic logic, and play the same rôle for intuitionistic logic that Boolean algebras play for classical propositional logic.

2 Discrete physical systems

Our intuition of a discrete physical system is the following: we assume a set of *configurations* together with a rule \mathcal{R} that describes how configurations change over time. The only assumptions we require are that configurations change in discrete steps (so we do not need to worry about Zeno’s paradox [9]), and there are no ‘hidden variables’ — the current configuration unambiguously determines successive configurations.

This situation is easily formalised :

* This research is part of QIP IRC www.qipirc.org (GR/S82176/01)

Definition 1. Abstract Computing Machines

An ACM, or **Abstract Computing Machine**, $\mathcal{M} = (X, \succrightarrow)$ is specified by a set X of **configurations** where every configuration $x \in X$ has at most one **next configuration** y , written $x \succrightarrow y$. This is of course equivalent to the existence of a partial function $Next : X \rightarrow X$. The interpretation is that if an ACM is in configuration x , the next configuration (assuming this exists) it takes on is $y = Next(x)$.

This definition, so far, is almost laughably simple. We merely have a (partial) function acting on a set, defining a binary relation via the usual representation of partial functions as relations. The interest arises in considering this partial function to be *iterated* — the intuition is that it describes a physical system with a simple rule for discrete evolution over time. The *Next* function is (for example) the von Neumann architecture for a computer, the evolution rule for a cellular automata or Turing machine, the (discrete analogue of a) Hamiltonian for a physical system, or similar.

Fig. 1. An Abstract Computing Machine

Interpretation A question of interpretation arises in that we allow for partiality in the \succrightarrow relation. Strictly, a physical system in a certain configuration will always have a next configuration under the application of a physically reasonable rule. We interpret a ‘terminal configuration’ (i.e. where $Next(x)$ is undefined) in one of two (related) ways:

- *Physical partiality* The next configuration is simply not accessible to us: the system evolves over time until it is outside of our domain of reference. The next configuration is beyond the scope of the observer — out of sight, perhaps.

– A *Halting Scheme* Partiality may also arise via the imposition of a **halting scheme**. Two possible examples are as follows:

- Given an ACM (X, \succ) , we may specify a subset (or subspace, or whatever) of Halting configurations $H \subseteq X$, and restrict the next configuration relation to the complement of H . This gives a new relation \succ_H , specified by

$$x \succ_H y \Leftrightarrow x \succ y \text{ and } x \notin H$$

- Alternatively, we may restrict the next configuration relation to situations where the configuration *changes* — ruling out ‘halting configurations’ where $x \succ x \succ x \succ \dots$

Examples The definition of an Abstract Computing Machine is intentionally framed in as broad a manner as possible, to cover any discrete evolving physical system. We observe that those mentioned in discussions on the Principle of Computational Equivalence [11] — that is, digital computers, cellular automata, (discrete models of) weather systems, Turing machines, &c. — are covered by this definition.

We now analyse these systems using tools from theoretical computer science and logic, and demonstrate that even this simple definition gives rise to an interesting theory.

Definition 2. The ‘Leads To’ relation

Let $\mathcal{M} = (X, \succ)$ be an ACM. We define the binary relation \leadsto on the set X of configurations to be the transitive closure of the \succ relation. Hence, \leadsto is the smallest binary relation satisfying :

- $x \succ y$ implies that $x \leadsto y$
- $a \leadsto b$ and $b \leadsto c$ implies that $a \leadsto c$

We refer to this as the **leads to** or **subsequentness** relation.

Interpretation The intuition of the ‘leads to’ relation is simple: a configuration describes the entire state of a computing machine, and $x \leadsto y$ simply means that if the machine is in configuration x it will, under whatever physical or logical rule \mathcal{R} governs its evolution, be in configuration y at some later point.

We observe that the ‘leads to’ relation \leadsto contains strictly less information about the evolution of an ACM than the ‘next’ relation, \succ .

Proposition 1. Let $\mathcal{M}_1 = (X, \succ_1)$ and $\mathcal{M}_2 = (X, \succ_2)$ be two distinct ACMs with the same configuration set, $X = \{a, b, c\}$, and with ‘next’ relations \succ_1 and \succ_2 specified by

$$a \succ_1 b \succ_1 c \succ_1 a$$

and

$$a \succ_2 c \succ_2 b \succ_2 a$$

These two inequivalent ACMs give rise to the same subsequentness relation (the universal relation), where $x \leadsto x'$ for all $x, x' \in X$.

Proof. The proof of this is almost immediate, given the definition of \rightsquigarrow as the transitive closure of \rightarrow . Of more interest is the observation that the subsequentness relation loses information about *causal ordering* — we cannot tell whether a is followed by b then c , or vice versa. \square

Interpretations Clearly, the subsequentness relation is important, computationally. However, the loss of information about the causal ordering is a non-trivial problem. There are three possibilities:

1. It is a simple observation that when two ACMs have the same subsequentness relation \rightsquigarrow on a configuration set X , they may often be identified up to a permutation of X . However, we take a category-theorist’s point of view that *isomorphism* should not be confused with *strict identity*, and note that the quotient induced by such a permutation will often identify all configurations — leading to trivial systems.
2. As a fairly ‘blunt instrument’ approach, we could introduce a *global clock* as an essential part of the structure of the ACM, together with the assumption that a single step $x \rightarrow y$ takes exactly one clock cycle. This would involve replacing the configuration space X with $X \times \mathbb{N}$ or $X \times \mathbb{Z}$, and replacing the transition $x \rightarrow y$ with the countable family of transitions $(x, n) \rightarrow (y, n+1)$. This will allow for the recovery of the causal ordering, albeit at the expense of a desperately expanded configuration set.
3. We could restrict the computational paths considered to those that *do not repeat configurations*. This prevents the sort of situation described in Proposition 1 from occurring, and allows causal orderings to be deduced from the subsequentness relation.

In what follows, we broadly follow 3. above, and restrict the computational paths considered. However, this is better done at the level of functions on the configuration set, rather than at the level of the configuration set itself. This also fits in with the category-theoretic approach that structure-preserving maps on a mathematical object are the correct level of discourse, rather than the elements of the mathematical object itself.

2.1 Evolutions and Semantics of Abstract Computing Machines

Our intuition of an Abstract Computing Machine is that of a set X of configurations together with some rule \mathcal{R} that describes — in a deterministic manner — how one configuration evolves into another.

In both physical and computational systems, we are often interested in studying systems at different levels of abstraction. Consider a *physical* computer, based on the *von Neumann architecture*, executing a *Java* program, using a *Java Virtual Machine*. We have a very different view of this according to whether we describe it at the level of machine language, interpreted byte code, high-level program code, or simply as a ‘black box’ that takes inputs to outputs.

In order to axiomatise this, we study the collection of partial functions on configuration sets that respect the ‘leads to’ relation, and introduce a partial ordering that corresponds to ‘different levels of abstraction’:

Definition 3. Evolutions, Machine Semantics, cycle-freeness

Let $\mathcal{M} = (X, \rightsquigarrow)$ be an abstract computing machine. We define an **evolution** of \mathcal{M} to be a partial function $\eta : X \rightarrow X$ satisfying the following condition:

$$\eta(x) = y \Rightarrow x \rightsquigarrow y \quad (1)$$

Of course, machine evolutions are far from unique — note that this definition even allows for the nowhere-defined partial function $0_X : X \rightarrow X$ as an evolution of \mathcal{M} . For an ACM \mathcal{M} , we define the **Machine Semantics** of \mathcal{M} to be the set of all evolutions, which we denote $[\mathcal{M}]$.

When an evolution η of \mathcal{M} satisfies the condition $\eta^K(x) \neq x$ for all $x \in X$ and $K \in \mathbb{N}^+$, we say that η is a **cycle-free evolution**. We denote the set of cycle-free evolutions by $[[\mathcal{M}]]$, and refer to this as the **cycle-free semantics** for \mathcal{M} . Note that nilpotent evolutions (i.e. evolutions $\eta \in [\mathcal{M}]$ where there exists some non-zero integer N such that $\eta^N = 0_X$) are always cycle-free; for finite configuration sets, the converse also holds. When the Next partial function is cycle-free it is clear that all machine evolutions are cycle-free. In this case, we say that \mathcal{M} is a **cycle-free ACM**.

Interpretation Informally, an evolution is a partial function where, given a machine \mathcal{M} in the configuration x , the machine will at some later point be in configuration $\eta(x)$ (provided $\eta(x)$ is defined). The term ‘machine semantics’ comes from the position that the meaning, or structure, of an ACM is best studied in terms of its set of evolutions.

We introduce a natural way of comparing cycle-free machine evolutions that has both a nice physical or computational interpretation, and well-behaved mathematical properties. The restriction to cycle-free evolutions is important, both in order to preserve the causal structure (as in Proposition 1), and in order to allow for the usual mathematical tools to be applied to the theory of machine semantics. Intuitively, we are restricting ourselves to considering computations that cannot ‘get stuck in an infinite loop’.

Definition 4. The primitiveness relation

Let $\mathcal{M} = (X, \rightsquigarrow)$ be an abstract computing machine, and let η, μ be cycle-free evolutions of \mathcal{M} , so $\eta, \mu \in [[\mathcal{M}]]$. We say that η is **more primitive than** μ , written $\mu \leq \eta$ exactly when, for all $\mu(c) = d$, there exists some integer $k \in \mathbb{N} = \{1, 2, \dots\}$ such that $\mu(c) = d = \eta^k(c)$. (We emphasise that in this definition, k is not a fixed integer; it may vary as a function of both c, d and η, μ).

Interpretation The motivation for the primitiveness relation is, as stated above, that we wish to “study ACMs at different levels of generality”. The primitiveness relation captures the informal idea that the description of a computer at the level of machine code is ‘more primitive’ than a description in terms of a compiled or interpreted language such as *Java*, which in turn is more fundamental than a description as a ‘black box’ that merely takes input to outputs.

Fig. 2. The primitiveness partial order

3 The mathematical setting for the primitiveness relation

We demonstrate below (Theorem 1) that the correct mathematical setting for the primitiveness relation is in the theory of partial orders and (for cycle-free machines) lattices. The basics of this theory, as in [12] are as follows:

Definition 5. *Partial orders, Lattices, &c.*

A **partial order** on a set P is a relation \leq satisfying

1. Reflexivity $a \leq a$ for all $a \in P$
2. Antisymmetry $a \leq b$ and $b \leq a$ implies $a = b$, for all $a, b \in P$
3. Transitivity $a \leq b$ and $b \leq c$ implies $a \leq c$, for all $a, b, c \in P$

A binary relation that is only required to satisfy 1. and 3. is called a **pre-order**, instead of a partial order. Every set with a pre-order determines a partially ordered set by taking the minimal (order-preserving) equivalence classes determined by the congruence $x \sim y$ iff $x \leq y$ and $y \leq x$.

Given a pair of elements of a partially ordered set, $a, b \in P$, an **upper bound** of a and b is an element z satisfying $a \leq z$ and $b \leq z$. The **least upper bound** or **join** (when it exists) of a and b , denoted $a \vee b$, is the unique upper bound satisfying $a \vee b \leq z$, for all upper bounds z of a and b . **Lower bounds**, and the **greatest lower bound** or **meet** (denoted $a \wedge b$), are defined dually.

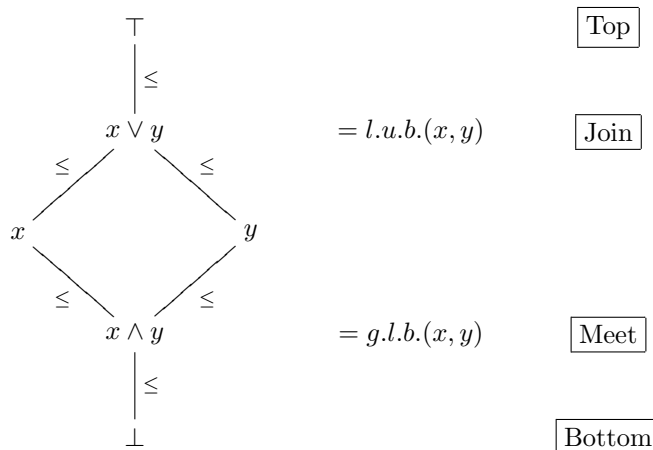
A **top element** for a partially ordered set P is a unique element $\top \in P$ satisfying, $p \leq \top$, for all $p \in P$. Similarly, a **bottom element** $\perp \in P$ is a unique element satisfying $\perp \leq p$ for all $p \in P$.

A **lattice** is defined to be a partially ordered set where all pairs of elements (and hence by induction, all finite sets of elements) have both least upper bounds

and greatest lower bounds. A lattice is called **upper-complete** when all sets of elements (i.e. not just finite sets) have a least upper bound. Similarly it is **lower-complete** when all sets of elements have a greatest lower bound.

By definition, all finite lattices are both upper-complete and lower-complete. Also, every upper-complete lattice has a top element given by $\top = \bigwedge L$ and every lower-complete lattice has a bottom element given by $\perp = \bigvee L$.

The term ‘lattice’ comes from the graphical representation for finite lattices as a Hasse Diagram, such as the following simple example:



Theorem 1. Let $\mathcal{M} = (X, \rhd)$ be an ACM, with machine semantics $[\mathcal{M}]$, and cycle-free semantics $\llbracket \mathcal{M} \rrbracket \subseteq [\mathcal{M}]$. Then $(\llbracket \mathcal{M} \rrbracket, \leq)$ is a partial order, with $0_X \in \llbracket \mathcal{M} \rrbracket$ as bottom element.

Proof.

To show that $(\llbracket \mathcal{M} \rrbracket, \leq)$ is a partial order, we need to demonstrate *reflexivity*, *anti-symmetry* and *associativity*:

- *Reflexivity* This is immediate from the definition : for all $\eta(c) = d$, trivially, we have that $\eta^K(c) = d$, where $K = 1$, and so $\eta \leq \eta$.
- *Anti-symmetry* Given $\eta \leq \mu$ and $\mu \leq \eta$, then for all $\eta(c) = d$, there exists $K > 0$ such that $\mu^K(c) = d$. Therefore, $\text{dom}(\eta) \subseteq \text{dom}(\mu)$. Similarly, $\text{dom}(\mu) \subseteq \text{dom}(\eta)$ and so $\text{dom}(\mu) = \text{dom}(\eta)$.

We now we prove by contradiction that $\eta = \mu$. Given $d = \eta(c) = \mu^K(c)$, assume that $K \neq 1$. We then define

$$\mu(c) = e_1, \mu^2(c) = e_2, \dots, \eta^k(c) = e_k = d$$

As $\mu \leq \eta$, there exists $L_1, \dots, L_K > 0$ such that

$$e_1 = \eta^{L_1}(c), e_2 = \eta^{L_2}(e_1), \dots, e_K = d = \eta^{L_K}(e_{K-1})$$

This gives

$$\eta(c) = \eta^{L_k + L_{k-1} + \dots + L_1}(c) \quad \text{where} \quad \sum_{j=1}^K L_j > 1$$

a contradiction of the cycle-freeness of η . Therefore $K = 1$, and so $\eta(c) = \mu(c)$ for all $c \in \text{dom}(\eta) = \text{dom}(\mu)$.

- *Transitivity* Given $\eta \leq \mu$ and $\mu \leq \zeta$, then, for all $\eta(c) = d$, there exists $K \in \mathbb{N}^+$ such that $\eta(c) = d = \mu^K(c)$. Similarly, for all $\mu(c) = e$, there exists $L \in \mathbb{N}^+$ such that $\zeta^L(c) = e$. Therefore, for all $\eta(c) = d$, we deduce that there exists some finite series of positive integers $\{L_1, L_2, \dots, L_K\}$ such that $\zeta^{L_1+L_2+\dots+L_K}(c) = d$. Therefore $\eta \leq \zeta$, and our result follows.

It is also trivial from the definition that the nowhere-defined partial function 0_X satisfies $0_X \leq \eta$ for all $\eta \in \llbracket \mathcal{M} \rrbracket$, and so is a bottom element for this partial ordering. \square

4 The order theory of cycle-free abstract machines

As an indication that we may find interesting computational structures within the theory of evolutions and the primitiveness relation, we now analyse the cycle-free case in logical, or computational, terms. This is clearly the ‘simplest possible’ case — we briefly discuss the general case in Section 5. The intention in presenting this case is to show that we may derive non-trivial theories even from the simplest case of ACMs.

Interpretations We are now restricting ourselves to the theory of cycle-free ACMs. The definition of a cycle-free ACM is that the $Next : X \rightarrow X$ partial function is cycle-free (and hence, trivially, all machine evolutions are cycle-free). In the finite case, this is exactly equivalent to the condition that $Next : X \rightarrow X$ is *nilpotent*. The intuition of this is not that we are considering computer programs without loops or control structure — rather, (in the finite case) we are considering programs with guaranteed *termination*, or *halting*. Although this is often difficult or impossible to prove, we refer to [5] for an decidedly non-trivial example of a computational system based on provably nilpotent maps.

Theorem 2. *Let $\mathcal{M} = (X, \rightsquigarrow)$ be a cycle-free ACM – i.e. $Next(x) \neq x$, for all $x \in X$, and all evolutions are cycle-free, so $[\mathcal{M}] = \llbracket \mathcal{M} \rrbracket$. Then*

- (i) *The partial order $([\mathcal{M}], \leq)$ has a top element.*
- (ii) *$([\mathcal{M}], \leq)$ is closed under finite meets.*
- (iii) *$([\mathcal{M}], \leq)$ is closed under arbitrary joins.*

Proof.

(i) The top element is quite simply the $Next : X \rightarrow X$ partial function. By definition of machine evolutions, $\eta(c) = d$ implies that $Next^N(c) = d$ for some $N > 0$. Also, as \mathcal{M} is cycle-free, there does not exist any machine evolution $\zeta \in [\mathcal{M}]$ such that $\zeta^K(c) = Next(c)$ for $K > 1$, so $Next$ is the top element of this partial order. For the remainder of this proof, we use lattice-theoretic notation, and write $\top(c)$ for $Next(c)$.

(ii) Having established the existence of a top element, we explicitly exhibit the join of a set of elements $\{\eta_i\}_{i \in I}$, again in terms of the top element. Given a

configuration $x \in X$, then:

- $(\bigvee_{i \in I} \eta_i)(x)$ is undefined exactly when $\eta_i(x)$ is undefined for all $i \in I$.
- Consider the subset $\{\eta_j\}_{j \in J \subseteq I}$ of evolutions where $\eta_j(x)$ is defined. For all η_j , there exists some minimal n_j such that $\eta_j(x) = \top^{n_j}(x)$. We then define an integer n in terms of the greatest common divisor, $n = \gcd(\{n_j\}_{j \in J})$ and take $(\bigvee_{i \in I} \eta_i)(x) = \top^n(x)$.

It is then immediate by the definition of the primitiveness ordering and the top element that this is the least upper bound of $\{\eta_i\}_{i \in I}$.

(iii) We explicitly exhibit the meet of two elements η, μ , again in terms of the top element:

- Given $x \in X$, then $(\eta \wedge \mu)(x)$ is *undefined* when either $\eta(x)$ is undefined or $\mu(x)$ is undefined, or both.
- Assuming that both $\eta(x)$ and $\mu(x)$ exist, then by (i) above, there exist least integers p, q such that $\top^p(x) = \eta(x)$, and $\top^q = \mu(x)$ (note that p and q are not fixed, but vary with x). The meet of η and μ is defined in terms of the least common multiple of p , and q , by $(\eta \wedge \mu)(x) = \top^r(x)$ where $r = \text{lcm}(p, q)$.

It is again immediate, by the definition of the primitiveness ordering, that this is the greatest lower bound of η and μ . \square

4.1 Cycle-free abstract machines describe constructive logics

We now observe that there is a close connection between partially ordered sets (with additional ‘closure’ properties), and both theoretical computer science and logic. We refer to [1, 6] for good expositions, and restrict the following exposition to one example that is relevant to the theory of abstract computing machines — that of Heyting algebras.

Heyting algebras play the same rôle for intuitionistic logic that Boolean algebras play for classical logic. Unfortunately, there is no space here to present an overview of intuitionistic logic, so we refer to [4] for a good introduction. Very broadly, intuitionistic logic is a restriction of classical logic that exactly captures the constructive fragment of logic (the original intention was that a proof of the existence of a mathematical object is exactly a construction of that object). Because of this constructive aspect, it is perhaps most familiar as the logic used in the PROLOG programming language [7] — it is also the logic required for the “Proofs - as Programs” correspondence given by the Curry-Howard isomorphism [10] between (intuitionistic) logics and typed lambda calculi.

Definition 6. Heyting algebras, (relative) pseudocomplements

A **Heyting algebra** is a lattice L with (distinct) top and bottom elements, where for every pair of elements g, f the set $\{h : f \wedge h \leq g\} \subseteq L$ is bounded above. The

(unique) upper bound of this set is called the **relative pseudocomplement** of f with respect to g , and denoted $f \Rightarrow g$. Clearly, the relative pseudocomplement of f with respect to g is used to model (intuitionistic) implication. There is also a similar concept for (intuitionistic) negation:

The **pseudocomplement** of $f \in L$, denoted $\neg f$ is defined in terms of the bottom element of the lattice, as $\neg f = f \Rightarrow \perp$.

The precise connection between intuitionistic logic and Heyting algebras is then the following:

Definition 7. Logical equivalence, the Lindenbaum-Tarski algebra

Consider a logical theory \mathcal{T} , consisting of formulæ, connectives, and rules of implication. Formulæ p, q are called **logically equivalent**, when p can be deduced from q and q can be deduced from p . We denote this equivalence relation by $p \sim q$.

The **Lindenbaum-Tarski algebra** of \mathcal{T} has as elements equivalence classes of formulæ under this equivalence relation. The operations of the Lindenbaum-Tarski algebra are those inherited from the logical theory \mathcal{T} (such as conjunction, disjunction, negation, &c.), provided they are well-defined under this quotient (— which is a reasonable assumption for logical theories).

Although this is a very general concept, for intuitionistic logics there is a good characterisation of the corresponding Lindenbaum-Tarski algebras:

Proposition 2. Heyting algebras exactly the Lindenbaum-Tarski algebras of intuitionistic logics.

Proof. This is a standard result of mathematical logic — we refer to [2] for a good exposition. \square

We now demonstrate that machine semantics for Abstract Computing Machines are Heyting algebras, and hence by Proposition 2, are intuitionistic logics, with logically equivalent terms identified.

Proposition 3. For an arbitrary cycle-free ACM \mathcal{M} , the primitiveness partial order gives a Heyting Algebra structure to the machine semantics.

Proof. We have seen that the machine semantics for an ACM has (distinct) top and bottom elements, is closed under finite meets and arbitrary joins. The existence of the relative pseudocomplement follows almost trivially from the closure under arbitrary joins, by

$$f \Rightarrow g = \bigvee \{h : f \wedge h \leq g\} \quad (2)$$

\square

Corollary 1. Let \mathcal{M} be a cycle-free abstract computing machine. Then the machine semantics of \mathcal{M} is a model of an intuitionistic logic, under the logical equivalence relation.

5 Commentary

We have only scratched the surface of the theory of Abstract Computing Machines, and machine semantics. Notably, all substantial results have been about the very special case of cycle-free machines. The following comments are more speculative, but hopefully provide more intuition:

Mathematically: For an ACM \mathcal{M} , the lattice of machine evolutions $[\mathcal{M}]$ is closed under composition, and hence is a *semigroup*. However, in general, the set of cycle-free evolutions is *not* closed under composition. Even for cycle-free machines, the composition is not order-preserving (since $a \leq b$ does *not* imply that $ax \leq bx$, so $[\mathcal{M}]$ is not – for example – a quantale). The interaction of the composition and the partial ordering is a non-trivial subject and presumably related to the domain-theoretic notion of ‘computation by the calculation of least fixed points’. We also observe that the order-theory of arbitrary ACMs remains to be studied. Although it is relatively easy to show that we may recover domain-theoretic notions such as directed-completeness, compactness, &c. (we refer to [1] for a good exposition of these notions) the full theory has not been given.

Physically: Although the definition of Abstract Computing Machines was framed very widely, there were nevertheless certain underlying assumptions. We may think of the partial functions (i.e. the machine evolutions) as describing possible *observations* of an evolving system. This is the underlying assumption that observing a system is not a physical change to it; similarly, the use of partiality is an implicit acceptance that either the halting scheme used is physically reasonable, or our mathematical tools are appropriate to analyse a system where we do not have access to all possible configurations. These assumptions (and many others) will need to be considered in more detail if we try to analyse quantum-mechanical systems in a similar way.

Computationally It is interesting to observe that the cycle-free machines (i.e. those that guarantee not to enter closed loops – unconditional termination in the finite case) are exactly those that provide models of intuitionistic logic (also very closely related to unconditional termination via Robinson’s unification Algorithm [7] and various decidability results [4]). In the general case, the set of cycle-free evolutions is not a lattice, because it has no top element (the *Next* : $X \rightarrow X$ partial function is *not* cycle-free). Preliminary studies suggest that the correct setting is domain theory, and computational interpretations may be found in areas such as models of functional programming or untyped lambda calculus [1, 8].

The Principle of Computational Equivalence? No claim has been regarding Abstract Computing Machines and computational universality. However, there are a number of approaches to computationally universal systems via order theory; undoubtedly the most famous of these is Dana Scott’s semantics for the

pure untyped lambda calculus [8]. As for interpretations, or whether this paper supports Wolfram's principle, the computationally significant structure in our theory arises from comparing distinct ways of observing (subproperties of) an evolving physical system. One possible conclusion is that if we are free to look at (subsections of) a physical system in any way we wish, we can quite easily see significant computational structures. However, the actual computational structure arises from our perceptions of evolving systems, rather than being intrinsic to the systems themselves.

References

1. Abramsky, S., Jung, A. : Domain Theory, in *S. Abramsky, D. M. Gabbay, T. S. E. Maibaum, editors, Handbook of Logic in Computer Science. III* Oxford University Press (1994)
2. Borceux, F. : Handbook of Categorical Algebra 3, In *Encyclopedia of Mathematics and its Applications, Vol. 53*, Cambridge University Press (1994)
3. Bunimovich, L. : Many Dimensional Lorentz Cellular Automata and Turing Machines, *Int Jour of Bifurcation and Chaos* 6 (1996) 1127-1135
4. Dummett, M. : *Elements of Intuitionism*, Oxford University Press (2000)
5. Girard, J.-Y. : Geometry of Interaction 1, *Proceedings Logic Colloquium '88*, North-Holland (1989) 221-260
6. Johnstone, P. : The point of pointless topology, *Bulletin American Mathematical Society*, 8(1) (1983) 41-53
7. Robinson, J. : A machine-oriented logic based on the resolution principle, *Communications of the ACM*, 5 (1965) 23-41
8. Scott, D. : Outline of a mathematical theory of computation, In *4th Annual Princeton Conference on Information Sciences and Systems* (1970) 169176
9. Silagadze, Z. : Zeno meets modern Science, *Acta Phys.Polon. B36* (2005) 2886-2930
10. Urzyczyn, P., Sorensen, M. : *Lectures on the Curry-Howard Isomorphism* Elsevier, (2006)
11. Wolfram, S. : *A New Kind of Science*, Wolfram Media (2002)
12. Vickers, S. : *Topology via Logic*, Cambridge Tracts in Theoretical Computer Science 5 (1998)