An Intelligent Theory of Cost for Partial Metric Spaces

Steve Matthews¹ and Michael Bukatin²

 University of Warwick, Coventry, UK, Steve.Matthews@warwick.ac.uk,
Nokia Corporation, Boston Massachusetts, USA bukatin@cs.brandeis.edu

Abstract. Partial metric spaces generalise metric spaces, allowing non zero self distance. This is needed to model computable partial information, but falls short in an important respect. The present cost of computing information, such as processor time or memory used, is rarely expressible in domain theory, but contemporary theories of algorithms incorporate precise control over cost of computing resources. Complexity theory in Computer Science has dramatically advanced through an intelligent understanding of algorithms over discrete totally defined data structures such as directed graphs, without using partially defined information. So we have an unfortunate longstanding separation of partial metric spaces for modelling partially defined computable information from the complexity theory of algorithms for costing totally defined computable information. To bridge that separation we seek an intelligent theory of cost for partial metric spaces. As examples we consider the cost of computing a double negation $\neg \neg p$ in two-valued propositional logic, the cost of computing negation as failure in logic programming, and a cost model for the hiaton time delay.

Keywords: AGI, partial metric spaces, discrete mathematics

1 Introduction

Today it may be taken for granted that a computing system should be adaptive and intelligent. Certainly the behaviour of a hand held device running a computer game or interactive internet site is adaptive and, as it exists to serve us humans, is designed to be as intelligent as is possible. Some forty years ago programming language design was categorised into what now appear narrow forms: axiomatic (a system of logic), operational (defined by a machine model), or denotational (each program denoted by a point in some mathematical domain). Through a groundbreaking axiomatic model such as Robin Milner's Calculus of Communicating Systems (CCS) or Dana Scott's denotational theory of domains we have made great progress in specifying some behaviours, but sadly not enough to handle the adaptive and intelligent features required for today's systems. So, what went wrong? What seems to have emerged is a dominant operational view

of programming language design and a grudging acceptance of the pragmatic compromise of object orientation which in effect extends a language definition with each new object introduced. To say that as a result logic and mathematics have no place in Computer Science would be ridiculous. And so, this paper asks how we can reinvigorate progress from axiomatic and denotational models for today's adaptive and intelligent systems.

1.1 Scott's Domain Theory

For the purposes of this paper we need to first appreciate the key concepts of Dana Scott's theory of domains. A domain is in the first instance a chain complete partially ordered set $(X, \sqsubseteq \subseteq X \times X)$ with a least element \bot . A computable function is in the sense of Scott at the very least monotonic. That is, if $f: X \to X \& x \sqsubseteq y$ then $f(x) \sqsubseteq f(y)$. Any non trivial programming language has one or more iterative constructs built in, such as a while-do loop or recursion. Scott uses Alfred Tarski's least fixed point theorem [?] to define the meaning of iteration for a computable function f as follows.

$$f(\bigsqcup_{n\geq 0} f^n(\bot)) = \bigsqcup_{n\geq 0} f^n(\bot)$$

Scott's domain theory is topological. A Scott topology $(X, \tau \subseteq 2^X)$ is related to the partial ordering of a domain by, if $O \in \tau$, $x \sqsubseteq y$, & $x \in O$ then $y \in O$. A Scott topology (X,τ) is weakly separable in the sense of T_0 . That is, if $y \not\sqsubseteq x$ then there exists $O \in \tau$ such that $y \in O$ & $x \not\in O$. As a denotational model for programming language design of the 1960s Scott's groundbreaking work resolved great issues of the day such as how to model iteration and recursion in programming language design. Tarski's theorem and Scott topology as fixed entities are perfectly fine for modelling a program which remains unchanged during its execution, but demonstrably inadequate for today's adaptive and intelligent computing systems. How come?

The situation in programming language design is just that of AI as understood by the AGI community. "The original goal of the AI field was the construction of 'thinking machines' — that is, computer systems with human-like general intelligence. Due to the difficulty of this task, for the last few decades the majority of AI researchers have focused on what has been called 'narrow AI' — the production of AI systems displaying intelligence regarding specific, highly constrained tasks. In recent years, however, more and more researchers have recognized the necessity — and feasibility — of returning to the original goals of the field. Increasingly, there is a call for a transition back to confronting the more difficult issues of 'human level intelligence' and more broadly 'artificial general intelligence (AGI)" (agi-conference.org). This paper considers two narrow design aspects of programming language design, how they were innovative, and how they now call for intelligent integration with the theory of algorithmic complexity in order to progress.

1.2 Non Zero Self Distance

An interesting lesson from Scott's domain theory is that it was necessarily innovative. The nature of computability theory when studied denotationally necessitates weak properties such as partial orderings, T_0 -separability, and least fixed points, instead of the usual strong properties such as T_2 -separability (that is, if $x \neq y$ then there exists $O, O' \in \tau$ such that $x \in O, y \in O' \& O \cap O' = \phi$). Dana Scott was innovative in developing a highly nontrivial theory for non T_2 topological spaces and applying them to the new science of programming language design.

In a metric space $(X, d: X \times X \to [0, \infty))$ as introduced by Maurice René Fréchet [?,?] strong separability T_2 results from the axiom x=y iff d(x,y)=0 for all $x,y\in X$, thus resulting in self distance d(x,x)=0 for each x. This equivalence gives rise to the trivial partial ordering $x\sqsubseteq y$ iff x=y for each metric space, which is hardly surprising for any system of mathematics not constrained to be computable let alone adaptive or intelligent.

The discipline of mathematics has traditionally taken zero self distance for granted because, before computer science, there was little reason to consider the computability of a metric distance d(x,y). More precisely, mathematics has understandably assumed that each metric distance is a totally defined structure. To assert that d(x,x)=0 for each $x\in X$ is in computational terms a useful means to specify that x is totally computed. Yet, at first sight, there appears to be no non trivial overlap between domain theory and metric spaces.

Definition 1. A contraction is a function $f: X \to X$ over a metric space (X,d) for which there exists $0 \le c < 1$ such that $d(f(x),f(y)) \le c \times d(x,y)$ for all $x,y \in X$.

Banach's contraction theorem states that each contraction over a complete metric space has a unique fixed point in that metric space \cite{Gamma} . While a computable function f has a least fixed point in the sense of Tarski/Scott, f may or may not be a contraction. Similarly, a contraction may or may not be a computable function. Wadge studied a small class of functions that are both computable and a contraction \cite{Gamma} , thus demonstrating a significant overlap of domain theory and metric spaces. This approach was soon generalised to introduce a larger class of functions.

Definition 2. A partial metric space [?,?] is a pair $(X,p:X\times X\to [0,\infty))$ such that,

$$\begin{array}{l} p(x,x) \leq p(x,y) \\ p(x,x) = p(x,y) = p(y,y) \ \Rightarrow \ x = y \\ p(x,y) = p(y,x) \\ p(x,z) \ \leq \ p(x,y) + p(y,z) - p(y,y) \end{array}$$

Thus a metric space is precisely a partial metric space for which each self distance p(x,x)=0. For a partial ordering in the sense of Scott let $x \sqsubseteq y$ iff p(x,x)=p(x,y). A partial metric space is, simply speaking, just a generalised form of metric space in which self distance can be ≥ 0 . It is worth noting here that

had not the first author once taken for his doctoral study (1985) an intelligent analysis of metric space theory to resolve a problem in recursive programming the idea of non zero self distance would not have subsequently emerged in his work as that of partial metric space. Just as Dana Scott worked more generally and effectively with non T_2 separable topological spaces so it proved necessary in Computer Science to work with non zero self distance in metric spaces. It can be shown that the usual topology and T_2 -separability of metric spaces generalises to a topology and T_0 -separability in partial metric spaces. Similarly, Banach's contraction fixed point theorem of metric spaces generalises to the cycle contraction theorem (Theorem 5.1, [?]) in partial metric spaces. There is thus a strong overlap between Scott topology & least fixed points in domain theory and topology & unique fixed points of contractions in metric space theory. However, for reasons explained above, neither exemplary approach of Fréchet/Banach nor Tarski/Scott could simply subsume the other by means of some ingenious theorem. Their heirs are called upon to intelligently integrate the research of great mentors. This research could well be unified in a more abstract setting such as category theory to express the greatest common denominator, but at the high price of losing the accumulated experience of each approach. It seems that partial metric spaces in Computer Science are going nowhere. How come?

It is the essential rationale of this paper that intelligent analysis in contemporary mathematics and computer science is needed to reinvigorate denotational models of computing with the exciting innovations of AGI. To exemplify such intelligent analysis we consider the contrast of how the mathematics of domain theory has developed as a costless form while research into algorithms and their complexity is justly thriving upon interest in adaptive and intelligent systems. A key ontological hypothesis of this paper is that cost is core, and that while domain theory was necessarily cost free in its formative years, today it has to be and can be properly costed. There is no expectation of a remarkable theorem and proof for this ontological hypothesis, but there are interesting examples to demonstrate how the notion of cost can evolve intelligently to help broaden programming language design as a contribution to AGI.

2 Examples of Cost

Each of the following examples has interesting features providing evolving insight into how notions of *cost* in algorithms have been related to denotational models of computing. In the adaptive intelligent spirit of the *complexity of algorithms* we seek to adapt the partial metric notion of non zero self distance to apply to each of these examples.

2.1 The Cost of Negation

Let $\{F,T\}$ be the usual set of truth values True and False of two valued truth logic. Let $(\{F,T\},d)$ be the metric space such that d(F,T)=1. Now add \bot as a third truth value. Let $(\{\bot,F,T\},p)$ be the partial metric space such that

 $p(\bot,\bot)=p(\bot,F)=p(\bot,T)=2$, p(F,T)=1 and p(F,F)=p(T,T)=0. Then, just as required in the sense of Tarski/Scott $\bot \sqsubseteq F$ and $\bot \sqsubseteq T$. As usual we can define negation $\neg F=T, \neg T=F$ and $\neg\bot=\bot$ to be a monotonic function. This implies $\neg\neg A=A$ as expected for each proposition A in three valued truth logic. However, this partial metric space is *cost free* as it does not keep account of the cost of computing functions such as negation. Although it may be argued that logics of truth have always necessarily been cost free, it is nonetheless a luxury no longer affordable in today's computing world of adaptive and intelligent systems.

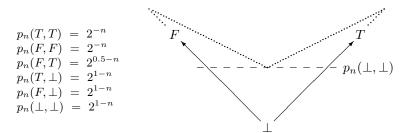
For convenience in our first example of cost let us assume applying negation to a proposition increases cost by one unit. For each $n \geq 0$ let $(\{\bot, F, T\}, p_n)$ be the partial metric space such that $p_n(\bot, \bot) = p_n(\bot, F) = p_n(\bot, T) = 2^{1-n}$, $p_n(F,T) = 2^{-n}$ and $p_n(F,F) = p_n(T,T) = 0$. Note that this particular definition for each $(\{\bot, F, T\}, p_n)$ is chosen to model the notion that if we ever manage to compute F or T it will be in some finite time n. Suppose now that given a proposition A we associate a partial metric space $(\{\bot, F, T\}, p_n)$ to keep account of its present cost. Let us speak of the costed proposition (A, p_n) to be the proposition A having an associated present cost of $(\{\bot, F, T\}, p_n)$. Then for a given costed proposition (A, p_n) we derive in one computational step the costed proposition $(\neg A, p_{n+1})$. And so, the truth $\neg A = A$ in three valued propositional truth valued logic is preserved in a computation while in general the costed proposition (A, p_n) becomes later the costed proposition $(\neg \neg A, p_{n+2})$. Any intelligent implementation of this rule would regard (F, p_n) & (F, p_n) as special cases of a costed proposition which if ever reached for some n should terminate the computation.

2.2 The Cost of Negation as Failure

To commence our second example of cost we note the following personal connection from Dana Scott to Robert Kowalski, well known for his contributions to logic programming. "I went to Stanford to study for a PhD in Mathematics, but my real interest was Logic. I was still looking to find the truth, and I was sure that Logic would be the key to finding it. My best course was axiomatic set theory with Dana Scott. He gave us lots of theorems to prove as homework. At first my marks were not very impressive. But Dana Scott marked the coursework himself, and wrote lots of comments. My marks improved significantly as a result", from Robert Kowalski: A Short Story of My Life and Work, April 2002.

In a non computing cost free world of logic we can afford the luxury of the property $\forall x.A(x)$ iff $\not\exists x.\neg A(x)$. However, in logic programming where the universe of all possible x values could necessitate an exhaustive search over an infinite domain this property is not computable in general. Negation as Failure (NAF) is a non-monotonic inference rule in logic programming used to assign a truth value to a formula $\neg A$ from the failure to derive the truth of A. Keith Clark, a student of Kowalski's, says of NAF, "Although it is in general not complete, its chief advantage is the efficiency of its implementation. Using it the deductive retrieval of information can be regarded as a computation" [?]. Correct! NAF cannot be complete in general, and hence neither is automated theorem proving. Thus logic programmers necessarily integrate automated logical inference with

their own creative human intelligence. Hence our second example of cost relates to AGI. Now we can integrate Scott's domain theory, partial metric spaces, and NAF as follows. Let $\{(\{\bot, F, T\}, p_n)|n \ge 0\}$ be the set of partial metric spaces such that,



p is monotonic in the sense that each p_n is a partial metric having the usual domain theory ordering $\bot \sqsubset F$, $\bot \sqsubset T$, and for all $x,y, 0 \le n < m$, $p_n(x,y) > p_m(x,y)$.

Note that in our first example of cost each self distance $p_n(F,F)$ and $p_n(T,T)$ is defined to be 0 as we were assuming truth to be either totally computed as the value F or T in finite time or remaining \bot indefinitely. In contrast, our second example of cost defines $p_n(F,F)$ and $p_n(T,T)$ to be 2^{-n} expressing the fact that n is the maximum cost allowed at run time in searching for a truth value before failure is to be assumed. NAF is a realistic intelligent algorithm to determine the truth of a formula as best we can through exerting reasonable cost. NAF is monotonic for as long as we are prepared to bear the cost of monotonic reasoning, after which we must rely upon our human intelligence to choose a truth value as best we can. Also note how the definition of $(\{\bot, F, T\}, p)$ evolves from a cost free single partial metric space, to a costed set of partial metric spaces in our first example, and on to further development in the second example.

2.3 Failure Takes Time

In 1979-80 the first author of this paper took an introductory course at London's Imperial College in logic programming from Robert Kowalski himself. This was his first exposure to the *Negation as Failure* inference rule. Many years later in 2011 this author's former PhD supervisor W.W. Wadge proposed the apt slogan *Failure Takes Time*. In 1977 Ashcroft & Wadge [?] introduced a declarative programming language called *Lucid* in which each input (resp. output) is a finite or infinite sequence of data values termed a *history*. In domain theory,

$$\langle \rangle \sqsubset \langle a \rangle \sqsubset \langle a, b \rangle \sqsubset \langle a, b, c \rangle \sqsubset \langle a, b, c, d \rangle \sqsubset \dots$$

where the totally defined inputs are precisely the infinite sequences, and the partially defined inputs are precisely the finite sequences. In partial metric terms $p(x,y)=2^{-n}$ where n is the largest integer (or ∞ if x=y is an infinite sequence) such that for each $0 \le i < n$ $x_i=y_i$. Then p is a partial metric inducing the above ordering. An unavoidable implication is that each data value in a sequence

is presumed to take the same amount of *time* to input (resp. output). Suppose now that *time* is to be our notion of cost for Lucid. Wadge & Ashcroft [?] recognised full well that defining a notion of cost synonymous with the data content of an input (resp. output) sequence is unrealistic in any non trivial programming language, and so presented their insightful vision of a *pause* in the execution of a program. For example, the following Lucid-like sequence seeks to introduce a special pause value termed a *hiaton* denoted * to domain theory.

$$\langle *, 2, 3, *, 5, *, 7, *, *, *, 11, \dots \rangle$$

But * is neither a well defined null data value (such as is the number 0) nor is say $\langle *,2 \rangle$ a partial value comparable to $\langle 2 \rangle$ in the partial ordering of domain theory. And so, what is a hiaton? Frustratingly the temporal intuition of a hiaton appears to be sound, but is not expressible in either domain theory or as a single partial metric space. The term hiaton is revealing, being as it is a combination of hiatus (pause) and daton (a Lucid data value). The vision of hiatons appears to infer that a pause can be fully known in order to be integrated with and presented as a well defined data value in a history. Our first two examples of cost regard cost not as a form of data but as a sophisticated interpretation of data in contemporary computing systems requiring (among other things) adaptivity and intelligence. Let us now generalise the notion of partial metric space for Lucid sequences to incorporate pauses as envisioned by Wadge & Ashcroft in a form that is consistent with domain theory. The following table is an example history of how pauses can be modelled in Lucid sequences using evolving partial metric self distances.

Time	Data	Hiatons	Partial Metric Self Distances
0	$\langle \rangle$	$\langle \rangle$	$2^{-1} + (2^{-0} - 2^{-1}) \times 2^{-0}$
1	$\sqsubseteq \langle 1 \rangle$	$\langle 1 \rangle$	$2^{-2} + (2^{-1} - 2^{-2}) \times 2^{-0}$
2	$\sqsubseteq \langle 1 \rangle$	$\langle 1, * \rangle$	$2^{-2} + (2^{-1} - 2^{-2}) \times 2^{-1}$
3	$\sqsubseteq \langle 1, 2 \rangle$	$\langle 1, *, 2 \rangle$	$2^{-3} + (2^{-2} - 2^{-3}) \times 2^{-0}$
4	$\sqsubseteq \langle 1, 2 \rangle$	$\langle 1, *, 2, * \rangle$	$2^{-3} + (2^{-2} - 2^{-3}) \times 2^{-1}$
5	$\sqsubseteq \langle 1, 2, 3 \rangle$	$\langle 1, *, 2, *, 3 \rangle$	$2^{-4} + (2^{-3} - 2^{-4}) \times 2^{-0}$
6	$\sqsubseteq \langle 1, 2, 3 \rangle$	$\langle 1, *, 2, *, 3, * \rangle$	$2^{-4} + (2^{-3} - 2^{-4}) \times 2^{-1}$
7	$\sqsubseteq \langle 1, 2, 3 \rangle$	$\langle 1, *, 2, *, 3, *, * \rangle$	$2^{-4} + (2^{-3} - 2^{-4}) \times 2^{-2}$

We now have an integrated notion of *history* for Lucid's data sequences and hiatons. Thus failure does indeed take both time and intelligence. Wadge's notion of *hiaton* is shown to be ahead of its time, and an intelligent integration of Fréchet/Banach and Tarski/Scott who came before.

3 Discrete Partial Metric Spaces

In Computer Science the term *discrete mathematics* is taken to mean the contemporary mathematics of information structures that are fundamentally finite.

Discrete mathematics is thus reasoning about structures such as finite graphs which are of interest in the understanding of algorithms or real-world computing applications. In contrast metric spaces and general topology are inherently continuous forms of mathematics where in general a point could be the limit of an infinite sequence of ever arbitrarily closer approximations. Wadge uses the interesting term Infinitesimal Logic (search online for Bill Wadge's Blog) in his work to introduce continuous mathematics to logic. As described in this paper the authors' work happened to turn out the other way round, which might be termed Logical Infinitesimals in contrast to and respect for Wadge. What is clear in both approaches is the need for and feasibility of a more intelligent integration of established research into logic, continuous mathematics, and algorithms. Cost is a key part of that integration.

Definition 3. A discrete partial metric space is a set of related partial metric spaces in which evolving self distances can be associated to represent computational costs defined by an intelligent form of discrete mathematics.

For example, a discrete partial metric space appropriate for representing Lucid with hiatons could begin with a partial metric space $(X, p: X \times X \to \{a_n^m \mid 0 \le n, m\})$ such that,

$$\forall n \geq 0 : a_n^0 > a_n^1 > a_n^2 > \dots > a_{n+1}^0$$

Our usual choice for this is such that,

$$\forall n \geq 0, m > 0 . a_n^m = a_{n+1}^0 + (a_n^0 - a_{n+1}^0) \times 2^{-m}$$

A hiaton may be understood intuitively as presented by Wadge & Ashcroft wherever the meaning is clear, or more precisely as a discrete partial metric space where the history is in part the history of *time* (as defined using evolving non zero self distance) in a sequence's computation.

Discrete mathematics, as understood in Computer Science, has developed firm roots driven by real-world applications. Discrete mathematics has proved itself to have great potential for modelling adaptive and intelligent systems. In contrast axiomatic and denotational models of computation have paid a very high price indeed for insisting that today's real-world must abide by their rigid pre-computing philosophies. Is it then just of mere academic interest to consider how, if at all, continuous and discrete forms of mathematics relate? Our examples of cost demonstrate that there are interesting ways to integrate continuous and discrete mathematics, as opposed to the prevailing view in Computer Science that continuous models of computations are mostly irrelevant to programming practice and only discrete mathematics has a future. Furthermore our research has highlighted the merits of working with logic and mathematics in contemporary computer science. Now we have an analogous but even more ambitious task of finding ways to have humans and machines think together as one new intelligent form, rather than trying to be the ultimate largest automatic theorem prover.

References

- Ashcroft, E.A., Wadge, W.W.: Lucid, a Nonprocedural Language with Iteration. Comm. ACM 20, 519–526 (1977)
- 2. Bukatin, M., Kopperman, R., Matthews, S., Pajoohesh, H.: Partial Metric Spaces. Amer. Math. Monthly, 116, 708–718 (2009)
- 3. Clark, K.L.: Negation as Failure. In: Gallaire H., Minker J. (eds.) Logic and Data Bases, pp. 113–141. Plenum Press, New York (1978)
- 4. Fréchet, M.: Sur Quelques Points du Calcul Fonctionnel. Rend. Circ. Mat. Palermo 22, 1–74 (1906)
- 5. Matthews, S.G.: Partial Metric Topology. In: Andima, S. et al (eds.) General Topology and its Applications, Proc. 8th Summer Conf. Queen's College (1992), Annals of the New York Academy of Science, vol. 728, 183–197 (1994)
- Matthews, S.G.: An Extensional Treatment of Lazy Data Flow Deadlock. Theor. Comp. Sci. 151, 195–205 (1995)
- 7. Sutherland, W.A.: Introduction to Metric and Topological Spaces. Clarendon Press, Oxford (1975)
- 8. Tarski, A.: A Lattice-theoretical Fixpoint Theorem and Its Applications. Pacific J. Math. and Appl. 5, 285–309 (1955)
- Wadge, W.W.: An Extensional Treatment of Dataflow Deadlock. Theor. Comp. Sci. 13, 3–15 (1981)
- Wadge, W.W., Ashcroft, E.A.: Lucid, the Dataflow Programming Language. Academic Press, London (1985)