A Basis Set of Operators for Space-Time Computations

Jacob Beal
BBN Technologies
Cambridge, MA, USA, 02138
Email: jakebeal@bbn.com

Abstract—Although many different models of spatial computation have been proposed, no unifying theory of computation over continuous space-time has yet been developed. Lack of such a theory has made it difficult to compare spatial computing models and impossible to determine their completeness. This paper takes a step toward the goal of a unifying model by identifying a mathematical basis set of operators from which any finitely-approximable causal computation can be constructed. The utility of this basis set of operators is then further demonstrated by using it to analyze the universality of the Proto spatial computing programming language.

I. INTRODUCTION

Many different computational models and programming languages have been proposed for specifying computation on spatial computers. Some models lean strongly on continuous abstractions, such as Proto[?], which describes computation in terms of dataflow field operators and information flow over manifolds, MGS[?], which operates on k-dimensional mathematical complexes, or Regiment[?], which operates on data streams collected from space-time regions. Others are discrete, as in the viral tuple-passing of TOTA[?] or in the distributed logical programming models of LDP[?] and MELD[?], or even regular, as in Yamins’ work on local computability[?].

At present, it is often extremely difficult to connect work done with or on such models and languages to one another. Each model tends to have its own unique set of space-time representation and operations, and as yet there is no unifying theory of computation over continuous space-time.

At first glimpse, it might be surprising that this is a problem. After all, the theory of computation is well developed for discrete computational devices, and spatial computers are typically implemented using a network of discrete devices (despite some possible exceptions, such as chemical reaction-diffusion computers[?] or optical image computation[?]). As we shall see, however, the continuous abstractions that are often present in spatial computing models mean that the model in which programs are actually specified is super-Turing.

In some relatively simple cases, it is possible to map the computation to a tractable case of existing continuous theory, such as analytic functions or dynamical systems. In general, however, properties of continuous abstraction computations can at present only be established indirectly, by considering how the computation is approximated on some realizable system (Figure 1). Much preferable would be to be able to analyze the computation directly, as we can do for those special cases, then test that an implementing platform gives a sufficiently faithful approximation.

This paper takes a step toward the goal of a unifying model by identifying a mathematical basis set of operators from which any finitely-approximable causal computation can be constructed. The utility of this basis set of operators is then further demonstrated by using it to analyze the universality of the Proto spatial computing programming language.

A. Related Work

Continuous models of computation have been studied alongside discrete models of computation throughout the history of computation, albeit at a much lesser intensity. For a thorough survey of the history of continuous computational models, see [?]. Most work on these models, however, has been in the context of real-valued systems evolving over continuous time, such as Shannon’s Differential Analyzer[?], Hopfield networks[?], and timed automata[?], or in discrete operations on real numbers, such as the BSS machine[?].

Recently, as the application of computational ideas has
broadened into fields like biology and the physical sciences, and as von-Neumann-model digital computation has begun to approach its limits, interest in continuous space models of computation has grown. Although computational models are plentiful—those referenced in the introduction are only a few of many—little theoretical grounding has been developed for the highly discontinuous and non-linear computations demanded in many applications. The main work that has been developed to date is ordinary differential equation (ODE) models of computation (e.g. [7, 8]) and an optical computation model proposed by Naughton[2].

II. Insufficiency of Turing Universality

First, let us understand why the conventional theory of computing is insufficient. Consider a spatial computing model that makes use of continuous abstractions, such as Proto[2] (continuous space and time) or OSL[2] (continuous space only). In languages such as these, the computation is specified agnostic of the system on which it will eventually be executed. In practice, however, this continuous specification is typically transformed into an approximate implementation by means of a set of operations on discrete devices. For example, a Proto program might be approximately implemented via message passing on a radio network, finite state machines on a cellular automata, or chemical signals emitted by engineered bacteria.

We might thus attempt to avoid the need for any new theory by analyzing computations only in the discrete implementation. The continuous specification would then become irrelevant, and we would need only to apply distributed computing theory, stochastic chemical models, or whatever other tool is appropriate for the medium of implementation. This constitutes an indirect analysis (Figure 1), and is the current means of proving the properties of continuous space-time computations.

There are critical shortcomings of such indirect proofs, however. First and foremost, indirect proofs may not be transferable between different discrete implementations, meaning that the same algorithm must be proved over again for each different implementation. As a corollary, it should be clear that an indirect proof does not actually establish a property for a continuous computation. Although intuition and proofs for multiple implementations may lead us to conclude that the property does hold, it has not been established with the mathematical rigor we normally demand of our proofs.

Moreover, continuous descriptions are often significantly simpler than discrete ones (an observation even more likely to hold in the case of computations that a person has chosen to describe using a continuous abstraction). This tendency is further amplified by the fact that the discrete implementation is not the native form, but a product of automated transformation, and therefore likely to be much more complicated.

Analysis done directly on a continuous computation, however, can conclusively establish a property. We need then only ask whether a particular implementation is a close enough approximation of the continuous space-time abstraction for acceptable approximation bounds to be established. For an example of such a proof and approximation bound, see the proof of self-stabilization time for the CRF-Gradient algorithm in [7].

Given that we wish to analyze computation over continuous space-time, it is clear that a unifying theory of computation over continuous space-time is desirable. The challenge is that continuous space-time computation is often theoretically super-Turing, given the uncountable number of points that may be involved in the computation. Conventional computational theory, which deals only with countable sequences of operations (even though these may involve uncountable sets of values or execution times) is thus not applicable without some adjustment.

Note, however, they we are explicitly not claiming super-Turing capabilities of spatial computers. What we are claiming is that a many useful abstractions for specifying spatial computations can have theoretically super-Turing capabilities. In general, we cannot perform useful computations on a real spatial computer unless we avoid taking advantage of those super-Turing capabilities. At the end of the day, in the real world, the computation will likely be executed on a collection of interacting Turing-equivalent machines, and any attempt to execute a super-Turing computation will fail (and may even be theoretically impossible, depending on questions like whether the universe is fundamentally discrete or continuous).

We therefore embrace a super-Turing model for its benefits in describing spatial computations. Having adopted a super-Turing model, however, questions about computational properties such as universality are not so straightforward to resolve. We must, in fact, clearly define computation on continuous space-time before we can resolve the question of what a universal basis set of operators might be.

III. Defining Space-Time Computation

Let us consider a computation occurring over some time across an unchanging region of continuous space. We may formalize the volume on which the computation occurs as the cross-product of a manifold $M$ (the space) and a real interval $T \subset (-\infty, \infty)$ (the time). In order to define a formal model of computation across such a space, we will begin with the limit model of the amorphous medium, which treats every point as a computational device, then define computation over an amorphous medium in terms of state trajectories.

A. Amorphous Medium

The amorphous medium abstraction[7] is derived from the observation that in many spatial computing applications, we are interested not in the particular devices that make up our network, but rather in the space through which they are distributed. The point of a sensor network, for example, is generally the environmental values that it senses. If more sensors are available, the area of interest can be inspected at a higher resolution, but the essential task remains the same.

The amorphous medium abstraction takes this to its logical extreme: an amorphous medium is a compact Riemannian
We shall not take this approach. The problem is that it requires a comparison between two sets of operations, to see if one is stronger than the other. We do not yet have our “Turing machine” for spatial computers, so we do not a priori know if any set of operations is “universal enough.” Put another way: we do not know enough about what should be computable on a spatial computer to test whether a given set of operations is “universal enough.”

Instead, we shall define a computation $C$ in terms of its trajectory of computed results, not concerning ourselves with how exactly these results are to be achieved. This approach is borrowed from variational mechanics, and in particular the computational presentation in [7]. In mechanics, the state trajectory representation allows a system to be considered as a whole and to be analyzed without committing to a particular set of coordinates. It will serve a similar purpose for us, allowing computations to be defined and examined without committing to a particular basis set of operators or means of realizing them.

Formally, let us define the computed state at time $t \in T$ as a function:

$$S_t : M \rightarrow V$$

where $V$ is any value (for simplicity, we shall not consider types but represent all values using arbitrary-length tuples of real numbers). The initial state of the system, $S_0$, is given rather than computed, and computation takes place with respect to some external environmental state $E$, which we shall define similarly as

$$E : M \times T \rightarrow V$$

This environmental state includes the outcome of random choices and any other sources of non-determinism used by the computation but not controllable by it—controllable state, such as actuators, will be taken to be part of $S_t$.

A computation $C$ is thus a function mapping from all conditions of execution to their trajectories of computed state:

$$C : M \times T \times E \times S_0 \rightarrow S_T$$

1 Remember, the jury is still out on whether it is physically possible to compute anything that is not computable by a Turing machine: it is simply that Turing universality covers every means of computing that has ever been implemented, and so has become accepted as clearly being “universal enough”
where \( S_T \) is a state trajectory: the collection of \( S_t \) for all \( t \in T \).

Order relationships between computations can be defined in terms of the state that they produce:

**Definition (Implements).** A computation \( C' \) implements computation \( C \) if there is a restriction of \( S'_T \) that is equal to \( S_T \) almost everywhere, and if for any non-equal point \( p \), there is a sequence of points \( p_i \) converging on \( p \) such that

\[
\lim_{i \to \infty} S'_T(p_i) = \lim_{i \to \infty} S_T(p_i).
\]

In other words, one computation \( C' \) implements another \( C \) if they produce equivalent results (discarding intermediate state used by \( C' \) in the computation). The “almost everywhere” means that there may be points that are not the same, but the measure of the non-converging set is zero; combined with the sequence condition, this allows discontinuous state functions to differ in how they assign boundary points to regions. Using this definition, two computations are equivalent if each implements the other; we shall, however, mostly be interested only in implementation and not in equivalence.

We thus have a definition of computation that is not dependent on any particular choice of a basis set of operators. To give a better intuition of this definition, some examples of computations, evaluated for particular combinations of space, time, environment and initial state, are illustrated in Figure ??.

Note that because we have not yet committed to a basis set of operators, there is nothing in this definition that requires a computation to be practically realizable or that models the cost of the computation.

**C. Universality**

A basis set of operators is a collection of computations and functions on computations, which can be composed to cover some portion of the set of possible computations. Universality for a basis set of operators is therefore defined as follows:

**Definition (Space-Time Universal).** A basis set of operators \( B \) is space-time universal if, for any computation \( C \) that can be specified by some basis set of operators (we need not know what operators or how it is specified), it is possible to implement an equivalent computation \( C' \) using operators in \( B \).

Unlike Turing equivalence, this is not fundamentally a constructive proof. Thus, even though we will know that it is possible to implement a computation using our basis operators, we may not know just how to do so or how to relate it to some other computational model. I do not regard this as a serious disadvantage, however: few programmers ever begin by asking how to achieve their goal using a Turing machine.

**D. Causal and Finitely-Approximable Computations**

As discussed in Section II, truly universal computation over space-time is unlikely to be of practical interest, as the vast majority of possible computations cannot be physically implemented. Instead, we shall limit our investigation to those computations that are practical to consider implementing: computations that are both causal and finitely-approximable.

Let us define a causal computation as one where the value computed at each point of space-time depends only on information that has possibly reached it. In other words, the computation cannot use information from the future, nor from events too far away to have communicated their information across space.

Formally, we shall define this as:

**Definition (Causal Computation).** A computation is causal if, for every point \( (m, t) \), the value of \( C(m, t, E, S_0) \) depends only on a restriction of \( M \times T \) to the set of points \( (m', t') \) with \( t' \leq t \) and non-positive interval \( d(m, m')^2 = c^2(t - t')^2 \).

This notion of intervals and causality is borrowed from relativity. For example, a computation that measures time since an environmental event last occurred within 10 meters of each device is causal, while a computation that measures the time until the next unpredictable environmental event occurs within 10 meters is not causal.

Similarly, let us define a finitely-approximable computation as one that is capable of being approximated well by a discrete implementation. Formally, we shall define this in terms of a limit as the density of approximating discrete devices increases:

**Definition (\( \epsilon \)-approximation).** Consider a finite set \( A_\epsilon \) of points in \( M \times T \), chosen such that no point in \( M \times T \) is more than distance \( \epsilon \) from a point in \( A_\epsilon \). The \( \epsilon \)-approximation of a computation \( C \) with regards to the set \( A_\epsilon \) is a function \( C_\epsilon \) that is equal to \( C \) computed with \( S_0 \) equal to \( S_0 \) at the nearest point in \( A_\epsilon \) and \( E' \) equal to \( E \) at the nearest point in \( A_\epsilon \) (choosing arbitrarily for equidistant points).

**Definition (Finitely-Approximable Computation).** A computation \( C \) is finitely-approximable if, for every countable sequence of \( \epsilon_i \)-approximations \( C_{\epsilon_i} \) of \( C \) with \( \epsilon_i < \epsilon_{i-1} \), the value of \( C_{\epsilon_i} \) converges to an implementation of \( C \).

For example, a computation that measures how many square meters of area where temperature is above 320K is finitely-approximable, as is a computation that tests whether the area

---

Fig. 3. Examples of causal and finitely-approximable space-time computations, illustrated on a 1D space: elapsed time since an environmental cue (a, darker is higher value), distance from environment-designated source points (b, darker is higher value), distance from environment-designated source points (c, darker is 'undefined'), whether an environmental cue is known to be present anywhere (c, red = yes, blue = no).
is in the range \([4 - \delta, 4 + \delta]\) square meters, where \(\delta\) is some small number. A computation that tests whether the area is equal to precisely 4 square meters, however, is not finitely-approximable: if the continuous area truly is 4 square meters, then arbitrarily small differences in approximation can switch the answer between “true” and “false.” Note also that this definition allows both continuous and discontinuous functions to be finitely approximated.

An interesting corollary observation: some analog of a Nyquist rate is likely to be useful in analyzing finitely-approximable computations. We know that a discrete implementation of a finitely-approximable computation converges to its continuous specification as the density of devices increases. Going in the other direction, as density decreases many computations reach a point where the approximation relation breaks down. For example, a computation that produces an alternating pattern where each stripe is 2 meters across cannot operate correctly when the density is significantly less than one device every two meters. Being able to identify a cusp density, below which approximation experiences qualitative degradation, appears both plausible and likely to be useful, though we shall not explore this idea further in this paper.

IV. A BASIS SET OF SPACE-TIME COMPUTATION OPERATORS

We can now propose a basis set of space-time operators on the amorphous medium model:

- \(P\) is any Turing-universal set of point-wise operators, that is, operators on state at individual points \((m, t)\). These include constants, branches, arithmetic operations, sensors, and actuators.\(^4\) Given an input function \(f : M \times T \to V\), a one-argument point-wise operator \(p \in P\) produces a function mapping: \((m, t) \to p(f(m, t))\). Multiple-argument point-wise operators are defined similarly, but with multiple inputs, and sensors also draw on the environmental state \(E\).

- \(n_v\) collects the vectors to neighbors in local coordinates, producing a function whose value at each point is: \(n) \to (n \in N(m) \to (\phi(n) - \phi(m)))\), where \(\phi\) is a chart in the manifold’s atlas that contains both \(n\) and \(m\) in its domain.

- \(g\) collects the metric tensor \(g_m\) at each point, producing a function whose value at each point is: \((m, t) \to g_m\) with respect to some system of local coordinates.

- \(n_v\) collects state from neighbors. Given an input function \(f : M \times T \to V\), applying \(n_v\) produces a function whose value at each point is a map from neighbors to their \(f\) values: \((m, t) \to (n \in N(m) \to f(n, t - d(m, n) / c))\).

- \(n_r\) restricts neighborhood-valued fields. Given two input functions, one Boolean valued \(f : M \times T \to \{0, 1\}\) and the other neighborhood-valued \(g : M \times T \to (N(m) \to V)\), applying \(n_r\) restricts the domain of the neighborhood functions to those points where \(f(m) = 1\), producing: \((m, t) \to (n \in N(m) | f(n) = 1) \to g(m)(n)\).

\(^4\)Computation on reals is super-Turing, but approximate computation of arbitrary precision (e.g. floating point arithmetic) is not.

- \(n_m\) computes minimum over neighborhood-valued fields. Given an input function \(f : M \times T \to (N(m) \to \mathbb{R})\), applying \(n_m\) produces a function that maps each point \((m, t) \to \infimum(f(m, t)(N(m)))\).

Intuitively, these operators may be thought of as belonging to three groups. The \(P\) operators implement computing on individual devices; they are not further specified since this is just ordinary computing and can be implemented in innumerable well-established ways. The \(g\) and \(n_v\) operators give access to the local structure of the manifold. Finally, the \(n_v\), \(n_r\), and \(n_m\) operators collect and process causally accessible state.

**Theorem.** Any finitely-approximable causal computation \(C\) can be implemented using the basis set of operators \(\{g, n_v, n_r, n_m\} \cup P\).

**Proof:** For space reasons, we will only sketch the proof. Consider any finitely-approximable causal computation \(C\).

Because \(C\) is causal, its value at \((m, t)\) is completely determined by some combination of properties of the manifold \(M\) and values at points \((m', t')\) that are accessible via chains of \(n_v\) functions.

Using \(n_v\), \(n_m\), \(n_d\), and \(n_r\), it is possible to collect at \((m, t)\) an arbitrary-precision sampling of a value from any past time-like region. For each state sample to be taken, we use \(n_v\) to collect \(s\), a neighborhood-valued function of the state of interest. Constructing an indicator field for the sample using operations from \(P\), we apply \(n_r\) to discard \(s\) everywhere but within \(\epsilon\) of the desired sample point. Applying \(n_m\) to the resulting field gives every point along the light cone in the

Fig. 4. Values can be sampled from past time-like regions by chaining together neighbor operators. Given a value gathered by \(n_v\), the \(n_r\) operation clips the operators to only the region where the sample should be drawn from (grey box). The \(n_m\) operation then finds the minimum value on light-cones intersecting the area (magenta line), and this value can be conveyed along light cones to the point of computation (blue lines) using another chain of \(n_m - n_v\) operations.
neighborhood a value arbitrarily close to the value of some point within $\epsilon$ of the desired sampling point—we don’t get exactly the sample we intended, but we do get one from a location within an arbitrary epsilon. By combining $n_m$ and $n_v$, the sample value can be chained from neighborhood to neighborhood until it reaches $(m, t)$. Since $P$ is Turing-universal, we can use recursion to repeat this sampling process for other points in the desired past time-like region, collecting a sampling with a distance of no more than $\epsilon$ between points.

Because $M$ is Riemannian, it is also differentiable. This, combined with the definition of a Riemannian metric, ensures that any property of the manifold can be approximated from a finite sampling of points on $M$, and that these approximations converge to the true value as the density of the sampling increases. Because the structure of the manifold may be recovered completely from $g$ and $n_d$, and because $P$ is Turing-universal, it is possible to compute an $\epsilon$-approximation of any property of the manifold at $(m, t)$, subject to causal restrictions, by sampling values of $g$ and displacements sampled from $n_d$.

Since $C$ is finitely approximable and $P$ is Turing-universal, it is possible to compute an $\epsilon$-approximation of $C$ using $\epsilon$-approximations of the geometric properties of $M$ and sampling of state from the past light-cone.

Finally, we can use recursion to define the computation of a sequence of $\epsilon$-approximations that converge to the value of $C$ almost everywhere. We thus have a computation that implements $C$.

A. Application to Proto

Proto[?] is a spatial computing language that describes computation in terms of functional dataflow computation on fields. Any reader familiar with Proto may have already noticed the similarity between Proto’s operators and the operators of the proposed basis set:

- The point-wise universal operators $P$ can be implemented by Proto’s universal set of constant, arithmetic, tuple, function, and branch operators, plus platform-specific sensors and actuators.
- $n_g$ is equivalent to Proto’s $\text{nbr-vec}$.
- $n_v$ is equivalent to Proto’s $\text{nbr}$.
- $n_c$ can be implemented by Proto’s $\text{if}$, applied to fields of neighborhood values.
- $n_m$ is equivalent to Proto’s $\text{min-hood}$.

The only operator that is missing from Proto is $g$. This tells us that Proto can (in theory) implement any finitely-approximable causal computation that does not depend on manifold properties that can only be recovered with the aid of the metric tensor $g_m$. For example, Proto can easily implement all of the computations shown in Figure 2?. It is not immediately clear, however, whether Proto can compute the divergence of a vector field.

V. Contributions

In this paper, we have taken several steps toward establishing a unifying theory of computation over continuous space-time: we have established why such a theory is needed, and how it relates to discrete implementations of spatial computations, we have established a formal definition of continuous space-time computation, and we have identified a basis set of space-time operators that can be used to implement any finitely-approximable causal computation and are useful for analyzing the capabilities of spatial computing models and programming languages.

These steps open up a host of important questions to be addressed, including:

- What is an appropriate measure of space-time computation cost, and what finitely-approximable operators are best for minimizing cost?
- Can we identify a useful analog to Nyquist rate to use for analyzing approximate implementation?
- Can we establish bounds on approximability of functions?
- What families of continuous-space proofs can be automatically translated into discrete network proofs?
- How can this theory be extended to computation on manifolds that change over time?
- What is a good way to extend Proto to cover computations that require $g$?
- How powerful are other spatial computing models?

Finally, as the theory of computation over continuous space-time continues to develop, we expect that it will become easier to analyze and compare proposed models of computation, increasing the unity of the field and the transferability of results from model to model.

REFERENCES


