DYNAMICAL SYSTEMS AND COMPUTABILITY THEORY

Forest Kobayashi and Matthew LeMay
Department of Mathematics, Harvey Mudd College, Claremont, CA, 91711

Abstract

Abstract We describe two models of computation, DFAs and Turing machines, and how they can be represented as dynamical systems. We also describe the computability hierarchy of languages and relate this to these models of computation. We discuss what it means for an arbitrary symbolic dynamical system to have computational universality (loosely defined as "the computational power of an arbitrary Turing machine"), and motivate in detail one particular definition.

1. Introduction

We'll begin by introducing some of the basic vocabulary used in formal language theory. We then discuss the layers of the *Chomsky Hierarchy*, and how the associated models of computation for each layer can be viewed as dynamical systems.

1.1. Strings and Alphabets

First, we give the definitions for *alphabets* and *strings*. These form the backdrop for all of our discussions of computability theory.

Definition 1.1 (Alphabet)**.** An *alphabet* is a finite set, often denoted by Σ. The elements of Σ are called *characters*. $\diamond$

We make no assumptions about the characters in Σ; they will *exclusively* serve as formal symbols for use in strings.

Definition 1.2 (Strings)**.** Let Σ be an alphabet. We define an operation $\cdot$ on Σ as follows: Let $\sigma_1, \sigma_2 \in \Sigma$ be arbitrary (not necessarily distinct). Then define a *new* symbol $\sigma_1\sigma_2$, and let

$$\sigma_1 \cdot \sigma_2 = \sigma_1\sigma_2.$$

Also define a special symbol $\epsilon \notin \Sigma$ with the property that for all $\sigma \in \Sigma$,

$$\epsilon\sigma = \sigma = \sigma\epsilon.$$

Define $\Sigma^\star$ to be the collection of all formal symbols generated by $\Sigma \cup \{\epsilon\}$ under the operation $\cdot$ described above. Explicitly, if $\Sigma = \{\sigma_1, \ldots, \sigma_n\}$, then we have

$$\Sigma^\star = \left\{\epsilon,\ [\sigma_1],\ \ldots,\ [\sigma_n],\ [\sigma_1\sigma_1],\ \ldots,\ [\sigma_i\sigma_j],\ \ldots\right\}$$
$$= \bigcup_{k=0}^{\infty} \left\{[\sigma_{i_1}\sigma_{i_2}\cdots\sigma_{i_k}] \mid \sigma_{i_j} \in \Sigma\right\}$$

Note: we are using the tortoise shell brackets $[]$ just to help visually separate the elements. Also, when $k = 0$, we take the convention that $[] = \epsilon$. In any case, we call

1) $\Sigma^\star$ the *Kleene star* of Σ,

2) The $\cdot$ operation the *concatenation operation*, and

3) The elements of $(\Sigma^\star, \cdot)$ *strings*. $\diamond$

Remark. This is the same as defining $(\Sigma^\star, \cdot)$ to be the *free monoid* over Σ. This isn't a terribly enriching perspective, we just think the word "monoid" sounds funny so we wanted to say it.

Definition 1.3 (Length of a string)**.** Let Σ be an alphabet, and let $\sigma \in \Sigma^\star$. Suppose σ is of the form

$$\sigma = \sigma_{i_1}\sigma_{i_2}\cdots\sigma_{i_k}$$

where $\sigma_{i_j} \in \Sigma$ for each $j = 1, \ldots, k$. Then we define the *length* of σ (denoted $|\sigma|$) to be

$$|\sigma| = k.$$ $\diamond$

Remark. Note, since $\epsilon \notin \Sigma$, our assumption that each of the $\sigma_{i_j} \in \Sigma$ makes the length function well-defined.

Definition 1.4 (Language)**.** Let Σ be an alphabet, and let $L \subseteq \Sigma^\star$. Then we call L a *language* over Σ. $\diamond$

On their own, alphabets and strings are not very interesting. Hence most of our questions center on *languages*. One might note that in our definition for a language, we made no requirements on L other than $L \subseteq \Sigma^\star$. Hence, L could be something extremely simple, such as

$$L_{\text{easy}} = \left\{\sigma \in \Sigma \;\middle|\; |\sigma| \equiv_2 0\right\},$$

or something fiendishly complex, like[1]

$$L_{\text{hard}} = \left\{\sigma \;\middle|\; \sigma \text{ is a valid proof of Hartman–Grobman.}\right\}$$

We think of L_{easy} as having a much simpler "structure" when compared to L_{hard}. This is a byproduct of the differences in the predicates we've use to define our sets. In the first, the rule is very simple to state, and very simple to verify. We could imagine a program that determines membership in L_{easy} simply by scanning left-to-right and flipping a parity bit until reaching the end of the string. Then, it would simply checks the final value of the bit to determine whether the length of the string was even or odd. See Fig. 1 for an illustration.

---

[1] Assume that we have some agreed-upon encoding scheme by which we can interpret strings in $\Sigma^\star$ as proofs.
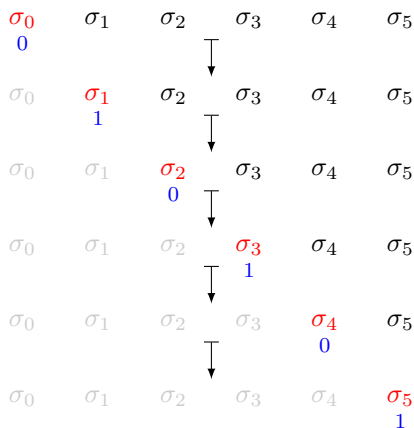
Figure 1: An illustration of the process. In each step, gray represents characters that've already been read, red symbols represent the current character, and blue symbol indicate the value of the parity bit. The final parity is odd, so we reject.

We note that a computer built to execute this algorithm could be very simple: it would only need enough memory to store the program and keep track of the parity bit, and this would be sufficient to determine membership in L_{easy} for arbitrary input.

By contrast, L_{hard} seems a lot harder. Since the input can be made arbitrarily long[2] while still remaining valid,[3] we need to be able to keep track of claims that are made arbitrarily far apart in the input string. E.g., suppose we're handed something like the following:

- Axioms: $A_0, A_1, \ldots, A_n$.

- Desired result: Z.

- Proof:
    - $1 + 1 = 2$
    - $1 + 2 = 3$
    - (A bunch of ⌊other statements that are true but irrelevant⌋ interspersed randomly with helpful ones, until finally)
    - $\ldots \implies Z$.

A priori, a verifier program has no knowledge of which claims will be relevant later to the proof. Hence, it needs to store all of the ones it encounters — even silly things like $1 + 1 = 2$. This makes it impossible to create an algorithm that looks exclusively at local data to verify whether a given string $\sigma \in L_{\mathrm{hard}}$. Thus, we would need a more sophisticated computer to execute a verifier for L_{hard} than we did for L_{easy}.

This difference is made formal by the *Chomsky Hierarchy*, which gives an ordering to languages based on the complexity of the "computers" needed to identify their

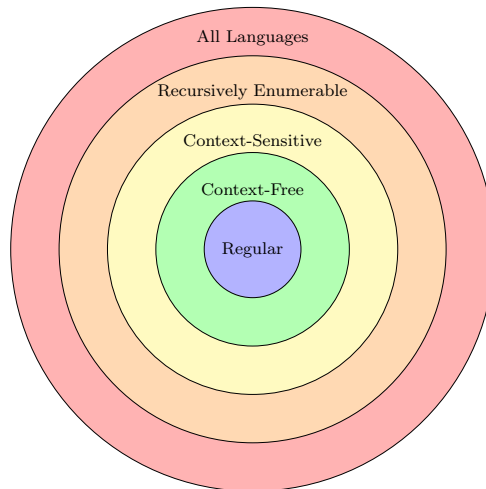strings.[4] The situation is summarized by the following graphic:

Figure 2: The Chomsky Hierarchy

L_{easy} is a *regular language*, which corresponds to one of the simplest possible models of computation (a *Deterministic Finite Automaton*). By contrast, L_{hard} requires a *Turing Machine*, one of the most powerful models of computation. All of these can be unified into a single conceptual framework by viewing them as *dynamical systems*. The reader might consider how as we begin with our formal definitions. But first, a quote from Prof. Ran:

> "There's this beautiful onion... and if you cut it open, it will make you cry."
>
> *-Prof. Ran, on the Chomsky Hierarchy*

We will strive to ensure that the reader experiences only tears of joy.

1.2. DETERMINISTIC FINITE AUTOMATA

Deterministic finite automata correspond to the innermost ring of Fig. 2, the *regular languages*. These can be described formally in a number of ways; we'll choose the following:

Definition 1.5 (Regular Language)**.** Let Σ be an alphabet. We'll define the set of all *regular languages over* Σ (denoted $\mathscr{L}_{\mathrm{reg}}(\Sigma)$) by an iterative process:

1) First, define $\varnothing$ and $\{\epsilon\}$ to be elements of $\mathscr{L}_{\mathrm{reg}}(\Sigma)$.

2) Next: For all $\sigma \in \Sigma$, define $\{\sigma\}$ to be an element of $\mathscr{L}_{\mathrm{reg}}(\Sigma)$.

3) Now, for all $L_0, L_1 \in \mathscr{L}_{\mathrm{reg}}(\Sigma)$, define
    - $L_0 \cup L_1$,

---

[2]See: this document
[3]See: maybe a different document
[4]Technically "computer" should be replaced with "computational model," but the distinction isn't important for us today

- $L_0 \cdot L_1 = \{\ell_0 \ell_1 \mid \ell_0 \in L_0, \ell_1 \in L_1\}$, and
- $L_0^\star,\ L_1^\star$

to be elements of $\mathscr{L}_{\mathrm{reg}}(\Sigma)$.

4) Finally, if L is yielded by a finite sequence of the rules above, then $L \in \mathscr{L}_{\mathrm{reg}}(\Sigma)$. $\diamond$

The idea with regular languages is that they are only slightly more complicated than Σ itself. Indeed, one might note the similarity between the axioms above and those for Definition 1.2 (strings). We'll give a few examples of regular languages before we introduce the associated model of computation. To that end, we'll first introduce a shorthand that will reappear many times throughout this paper.

Definition 1.6 (Exponent notation)**.** Let Σ be an alphabet. Then for all $\sigma \in \Sigma$, we interpret the notation σ^n by

$$\sigma^n = \underbrace{\sigma \cdot \sigma \cdots \sigma}_{n \text{ times}} \qquad \diamond$$

In light of the remark about m o n o i d s made earlier, this notation is actually quite reasonable.

Example 1.1. Let $\Sigma = \{0, 1\}$. Then L defined by

$$L = \{0^n 1^m \mid n, m \in \mathbb{N}\}$$

is a regular language. This follows from the fact that we can write L as $L = \{0\}^\star \cdot \{1\}^\star$.

Example 1.2. Let Σ be an arbitrary alphabet. Then all finite sets $L \subseteq \Sigma$ are regular. This follows from the fact that for any $\sigma \in L$, if σ decomposes as

$$\sigma = \sigma_{i_1} \sigma_{i_2} \cdots \sigma_{i_k},$$

Then we can state this equivalently as

$$\{\sigma\} = \{\sigma_{i_1}\} \cdot \{\sigma_{i_2}\} \cdots \{\sigma_{i_k}\},$$

from which it follows that $\{\sigma\}$ is regular. Finally, closure under finite union gives us that

$$L = \bigcup_{\sigma \in L} \{\sigma\}$$

is a regular language.

Example 1.3. Let $\Sigma = \{0, 1\}$. Then L given by $L = \{(01)^n \mid n \in \mathbb{N}\}$ is regular.

As stated earlier, *regular languages* are tied to *Deterministic Finite Automata*. These will lead to a more dynamical systems-esque view of regular languages.

Definition 1.7 (Deterministic Finite Automata)**.** A *deterministic finite automata* is a 5-tuple

$$M = (Q, F, \Sigma, \delta, q_0)$$

such that the following hold:

1) Q, Σ, F are finite sets. We choose the following naming conventions:
   i) Q is called the *set of states* for M,
   ii) F is called the set of *accepting* states for M, and
   iii) Σ is called the *input alphabet* for M;
2) δ is a function $\delta : Q \times \Sigma \to Q$ (we call δ the *transition function*), and
3) $q_0 \in Q$ is a distinguished element that we call the *start state*. $\diamond$

We imagine M performing a computation as follows: Suppose we're given an input string $\sigma = \sigma_{i_1} \sigma_{i_2} \ldots \sigma_{i_k}$. M initializes in the *start state* q_0, and reads the first character. M then changes state to $q_{j_1} = \delta(q_0, \sigma_{i_1})$, and reads the next character σ_{i_2}. M then transitions to state $q_{j_2} = \delta(q_{j_1}, \sigma_{i_2})$, and so on. Once M has exhausted the input, it halts computation. If it halts in an *accepting state* (i.e. $q_{\mathrm{final}} \in F$), then we say M accepts the input; else, we say M rejects it.

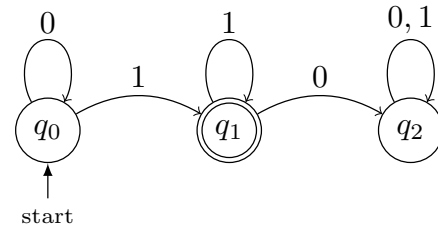It's common to visualize DFAs through diagrams like the following:

Figure 3: A DFA for Example 1.1.

The convention is that circles represent states of Q; the arrows represent the transitions given by $\delta(q, \sigma)$ (where σ is the label above the arrow), and double-circled states represent elements of F.

One can show that for every regular language L, there exists a DFA M_L such that M_L accepts a string σ iff $\sigma \in L$. We will not give a proof today since our interests are mainly in describing automata by dynamical systems. We now proceed in building this up formally.

1.3. Putting a Topology on our Strings

We want to convert our DFAs into dynamical systems. Recall, we think of an abstract dynamical system in terms of the following definition:

Definition 1.8 (Dynamical System)**.** Let M^m be an m-manifold, and let $(T, +)$ be a monoid. Let $\Phi : T \times M^m \to M^m$ such that for all $x \in M^m$,

1) $\Phi(0, x) = x$, and

2) For all $t, s \in T$, $\Phi(t + s, x) = \Phi(s, \Phi(t, x))$.

Then we call (M^m, T, Φ) a *dynamical system*. In particular, we call M^m the *phase space*, $t \in T$ the *evolution parameter*, and Φ the *evolution map*.   ◇

To make our DFAs conform to this definition, we need to find sensible things to call M^m, T, and Φ. Let's think about what we expect each of these to look like. The choice of evolution parameter is easy enough; we can just think of T as $\mathbb{N}$ with $t = 0$ corresponding to the initialization step and each $t > 0$ as corresponding to the nth step of computation. Then, Φ will correspond to executing some step of the computation.

The phase space is a bit trickier. For now, we'll set aside the concerns of what it means to be an m-manifold and just focus on thinking of what M^m might look like as a set.

Recall that we want M^m to represent all possible *states* of our system in question. In the 5-tuple formalism (Definition 1.7), we defined *state* by elements of a finite set Q; however, this is insufficient information to fully quantify the behavior of our DFA at any given point of the computation — recall that δ depends not only on the "state" q_i, but also on the current character being read from the input. Hence we need our dynamical version of "state" to include both of these pieces of information.

We might propose something like $Q \times \Sigma$ to be our state space. But there's another problem here. Say we're given two strings s_0, s_1:

$$s_0 = \sigma_{i_1} \sigma_{i_2} \cdots \sigma_{i_k}$$
$$s_1 = \sigma_{j_1} \sigma_{j_2} \cdots \sigma_{j_\ell}.$$

Suppose now that $\sigma_{i_1} = \sigma_{j_1}$. We'll call this shared symbol σ_1. Observe that feeding either s_0 or s_1 into some DFA M yields identical behavior in the first step:

$$\delta(q_0, \sigma_{i_1}) = \delta(q_0, \sigma_1) = \delta(q_0, \sigma_{j_1})$$

<center>all equal</center>

Hence in both cases, the machine enters some new state q'. But now, suppose $\sigma_{i_2} \neq \sigma_{j_2}$, and further that $\delta(q', \sigma_{i_2}) \neq \delta(q', \sigma_{j_2})$. Then M will actually behave differently on the two strings. Hence we see that taking our states to be elements of $Q \times \Sigma$ also fails to give us a full characterization of subsequent behavior.

What we really want is a way to encode the *entire* unread portion of the string into a state variable. We do exactly that with *configuartions* (note, this term is nonstandard). We will first define *unrestricted configurations*, which we'll use later in our definition of Turing machines[5] We'll then define *DFA configurations* as a particular restricted subset of these unrestricted configurations.

Definition 1.9 (Unrestricted Configuration)**.** Given a state set Q and input alphabet Σ, define the space of *unrestricted configurations over Q, Σ* to be

$$X_{\text{unrest}} = Q \times \left(\prod_{i \in \mathbb{Z}} \Sigma \cup \{\epsilon\} \right),$$

where the $\prod$ is understood as a cartesian product.   ◇

Definition 1.10 (DFA Configuration)**.** In an abuse of notation, let $\epsilon_{-\infty}^0 = \prod_{i=-\infty}^0$ (where the "multiplication" operation in $\prod$ is understood to be concatenation). Similarly, let $\epsilon_{n+1}^\infty = \prod_{i=+1}^\infty \epsilon$. Then we define $X_{\text{DFA}} \subseteq X$ by

$$X_{\text{DFA}} = \left\{ (q, \sigma) \in X_{\text{unrest}} \mid \sigma = \left(\epsilon_{-\infty}^0 \cdot [\sigma_1 \sigma_2 \cdots \sigma_n] \cdot \epsilon_{n+1}^\infty \right) \right\},$$

where $n \in \mathbb{N}$ is understood to be arbitrary, and all of the $\sigma_1, \ldots, \sigma_n \neq \epsilon$.   ◇

Recalling that ϵ is the empty string, we see that X_{DFA} is comprised of pairs of states q and finite strings σ over Σ such that each string is indexed starting from 0. We will not need the negative indices until we discus Turing machines.

In any case, this "configuration space" turns out to be the correct choice of phase space. Woohoo! Now, all that remains is to address the m-manifold condition. To that end we will first convert our *alphabets* and their *strings* into sets with topological structure.

If the reader is not familiar with topology, we've included a few definitions (topology, homeomorphism, base, and manifold) in the appendix. However, we should warn that the material below is rather technical, and might even border on overwhelming. We have chosen to include it because we feel it's helpful to see how the results are all built up. In particular, we think it lends credence to the claim that the metric we'll choose on $(\Sigma \cup \{\epsilon\})^{\mathbb{Z}}$ is indeed the natural one. But still, the reader is free to skim.

We will begin by endowing our building block sets $\Sigma \cup \{\epsilon\}$ with a topology, and then describing how to preserve this structure sensibly when we glue infinitely-many of them together.

Definition 1.11 (Discrete Topology)**.** Let X be an arbitrary set. Define the *discrete* topology $\mathcal{T}_{\text{discrete}}$ on X by letting every $U \subseteq X$ be an open set.   ◇

The intuition for the *discrete topology* is that it can be realized by a metric defined by

$$d(x_i, x_j) = \begin{cases} c_{ij} & x_i \neq x_j \\ 0 & x_i = x_j \end{cases}$$

where each $c_{ij} > 0$. In the following, we will always think of $\Sigma \cup \{\epsilon\}$ as being endowed with the discrete topology.

---

[5]We'll repeat the definition there when we need it, so don't worry about memorizing this one.

Proposition 1.1. *Consider* $\Sigma \cup \{\epsilon\}$ *with the discrete topology. Then the set of all singletons* $\mathcal{B} = \{\{\varepsilon\}, \{\sigma_1\}, \{\sigma_2\}, \dots, \{\sigma_n\}\}$ *forms a* base *for* $\Sigma \cup \{\varepsilon\}$. ◇

Definition 1.12 (Product Topology)**.** For all $i \in I$, let $(X_i, \mathcal{T}_i)$ be a topological space. Let $X = \prod_{i \in I} X_i$. We define the *product topology* $\mathcal{T}_{\text{prod}}$ on X as follows: For all $i \in I$, let $\pi_i : X \to X_i$ be the canonical projection map. Then for all open $U_i \subseteq X_i$, let

$$\pi_i^{-1}(U_i) \in \mathcal{T}_{\text{prod}}.$$

We call these $\pi_i^{-1}(U_i)$ *open cylinders*. Add all of the sets generated by taking arbitrary unions or finite intersections of the open cylinders to $\mathcal{T}_{\text{prod}}$. Then we call $\mathcal{T}_{\text{prod}}$ the *product* topology. ◇

See Fig. 4 for an example of an open cylinder in $\mathbb{R}^3 = \mathbb{R} \times \mathbb{R}^2$.

Note, the definition above abstracts on the idea that in *finite* product spaces, the projection maps $\pi_i : X \to X_i$ are continuous, and hence preserve open sets under inverse images. This turns out to yield a more natural topology on the product space than we'd have if we'd defined a topology by "products of open sets are open in the product set."
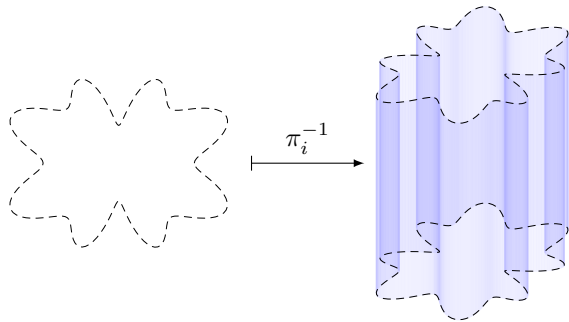

Figure 4: Example partial view of a cylinder set pulled from $\mathbb{R}^2$ into $\mathbb{R}^3$

Product topologies can be hard to think about, so we'll try to compartmentalize the results we want into little bite-sized steps. We begin with a proposition that's essentially definitional, but might be helpful as a stepping stone in building comfort. The idea is that for a product of discrete spaces, singleton cylinders give us essentially all we need to characterize openness.

Proposition 1.2. *For all $k \in \mathbb{Z}$, let $(X_k, \mathcal{T}_k)$ be discrete topological spaces. Define $X = \prod_{k \in \mathbb{Z}} X_k$, and endow X with the product topology. Fix a particular $i \in \mathbb{Z}$, and let $U_i \in \mathcal{T}_i$. Then for all $\sigma_j \in U_i$, $\pi_i^{-1}(\{\sigma_j\})$ is open in $\mathcal{T}_{\text{prod}}$, and $\pi_i^{-1}(U_i)$ can be written as*

$$\pi_i^{-1}(U_i) = \bigcup_{\sigma_j \in U_i} \pi_i^{-1}(\{\sigma_j\}). \qquad ◇$$

Proof. Note that since X_i has the discrete topology, for all $\sigma_j \in U_j$, we have $\{\sigma_j\} \in \mathcal{T}_j$. Thus $\pi_i^{-1}(\{\sigma_j\})$ is an open cylinder, and hence open (by definition of $\mathcal{T}_{\text{prod}}$). The union part follows immediately by definition; we write it out explicitly. The important part is highlighted in blue. We have

$$\pi_i^{-1}(U_i) = \left(\prod_{k=-\infty}^{i-1} X_k \right) \times U_i \times \left(\prod_{k=i+1}^{\infty} X_k \right)$$

$$= \left(\prod_{k=-\infty}^{i-1} X_k \right) \times \bigcup_{\sigma_j \in U_i} \{\sigma_j\} \times \left(\prod_{k=i+1}^{\infty} X_k \right)$$

$$= \bigcup_{\sigma_j \in U_i} \left(\prod_{k=-\infty}^{i-1} X_k \right) \times \{\sigma_j\} \times \left(\prod_{k=i+1}^{\infty} X_k \right)$$

$$= \bigcup_{\sigma_j \in U_i} \pi_i^{-1}(\{\sigma_j\}),$$

as desired. Since i, U_i were arbitrarily chosen, the claim follows. $\square$

Now, we observe that it's not generally true that the product of discrete spaces is discrete under the product topology. Indeed, we actually only get discrete-like behavior when we look at subsets of a particular form. We'll prove this over a series of claims in the below. First, we prove a general result for product spaces. The notation here begins to get quite unwieldy, apologies for any difficulties it creates.

Proposition 1.3. *For all $k \in \mathbb{Z}$, let $(X_k, \mathcal{T}_k)$ be topological spaces. Let $X = \prod_{k \in \mathbb{Z}} X_k$. Then $U \subseteq X$ is open iff U is of the form $\prod_{k \in \mathbb{Z}} U_k$, where only finitely-many of the $U_k \neq X_k$.* ◇

Proof. Since U is open, then by definition of $\mathcal{T}_{\text{prod}}$, U can be expressed as an arbitrary union of ⌈intersections of open cylinders⌉ (in the following, let $V_{j_\ell} \subseteq X_{j_\ell}$):

$$U = \bigcup_{i \in I} \bigcap_{j_\ell = j_1}^{j_{k_i}} \pi_{j_\ell}^{-1}(V_{j_\ell}).$$

The extra ℓ subscript on the j_ℓ index is just meant to encode that for different i's, we could be picking different families of U_j's to use in the intersection.[6] For each $i \in I$, let

$$j_{i,\max} = \max_{j_\ell = j_1, \dots, j_{k_i}} \{j_\ell\}.$$

And similarly for $j_{i,\min}$. Then observe that for all i, for all $k > j_{i,\max}$ or $k < j_{i,\min}$, we have

$$\pi_k \left(\bigcap_{j_\ell = j_1}^{j_{k_i}} \pi_{j_\ell}^{-1}(V_{j_\ell}) \right) = X_k.$$

---

[6] E.g., for $i = 1$, we might pick $j \in \{1, 3, 5, \dots, 21\}$, whereas for $i = 2$, we might have $j \in \{2, 3, 5, 7, 13, \dots, 23\}$, or something.

Hence, it's only possible for $V_k \neq X_k$ for $j_{i,\min} \leq n \leq j_{i,\max}$, which represents finitely-many indices.

Finally, observe that since the U_k are defined by unioning over each index, this gives us that only finitely-many of the $U_k \neq X_k$. □

Proposition 1.4. *With all variables quantified as above, if each of the X_k's have the discrete topology, then one can replace "open" with "clopen."* ◇

Before proving this, we should first remind ourselves of how set complement behaves over cartesian products. Given sets A, B, and subsets $S \subseteq A$, $T \subseteq B$, then we have

$$(S \times T)^c = (S^c \times T^c) \cup (S^c \times T) \cup (S \times T^c),$$

and **NOT**

$$(S \times T)^c = S^c \times T^c. \quad \text{✗ no!}$$

In particular, when $T = B$, we have

$$(S \times B)^c = S^c \times B.$$

Proof. In light of the note above, it suffices to show that if each of the X_k's are given the discrete topology, then the complement of an open cylinder $\pi_i^{-1}(U_i)$ is an open cylinder. This follows directly from applying Proposition 1.2 to U_i^c. □

We have one final proposition, which the reader should think of in terms of the context of X_{unrest}. Like all claims involving the product topology, the statement looks technical, but it's saying something very simple: Namely, open balls in X are collections of sequences that share a common portion around the 0th character.

Note: For the sake of making things fit on one line, we'll use the notation

$$X_{-\infty}^{k_1} = \prod_{i=-\infty}^{k_1} X_i \quad \text{and} \quad X_{k_2+1}^{\infty} = \prod_{i=k_2+1}^{\infty} X_i$$

in the following. We'll also interpret the somewhat-incomprehensible notation

$$\bigcup_{x_i \in X_i} \left(\prod_{i=k_1}^{k_2} \{x_i\} \right)$$

quite liberally as "all possible strings of characters that could appear between index k_1 and k_2 inclusive."

Proposition 1.5. *For all $k \in \mathbb{Z}$, let $(X_k, \mathcal{T}_k)$ be discrete topological spaces. Define $X = \prod_{k \in \mathbb{Z}} X_k$. Then the collection $\mathcal{B}$ given by*

$$\mathcal{B} = \bigcup_{\substack{k_1, k_2 \in \mathbb{Z} \\ k_1 \leq 0 \leq k_2}} \bigcup_{x_i \in X_i} X_{-\infty}^{k_1-1} \times \left(\prod_{i=k_1}^{k_2} \{x_i\} \right) \times X_{k_2+1}^{\infty}$$

defines a topological base for $\mathcal{T}_{\text{prod}}$. ◇

Again, since we've had to make multiple notational compromises here, we'll take the opportunity to clarify what a prototypical element of $\mathcal{B}$ looks like. It is a set. The set's elements are *all* bi-directional sequences that are identical in the blue portion of $\cdots \sigma_{k_1-1} \sigma_{k_1} \cdots \sigma_{k_2} \sigma_{k_2+1} \cdots$, where $k_1 \leq 0 \leq k_2$.

The unions we use to define $\mathcal{B}$ range over all possible choices of endpoints $k_1 \leq 0 \leq k_2$, and (for fixed k_1, k_2) over all possible values for the blue portion itself.

Proof. We are using the definition of a base given in Definition 3.4. First, note that

$$X = \bigcup_{B \in \mathcal{B}} B,$$

hence $\mathcal{B}$ satisfies the covering axiom for a base. We want to show it satisfies the second. Let $B_1, B_2 \in \mathcal{B}$ be arbitrarily chosen. Write these as

$$B_1 = X_{-\infty}^{\ell_1-1} \times \{x_{1,\ell_1}\} \times \cdots \times \{x_{1,\ell_2}\} \times X_{\ell_2+1}^{\infty}$$
$$B_2 = X_{-\infty}^{m_1-1} \times \{x_{1,m_1}\} \times \cdots \times \{x_{1,m_2}\} \times X_{m_2+1}^{\infty}$$

We have two cases.

1) Suppose there exists $k \in \{\max\{\ell_1, m_1\}, \ldots, \min\{\ell_2, m_2\}\}$ such that $x_{1,k} \neq x_{2,k}$. Then $B_1 \cap B_2 = \varnothing$. Thus in this case, the "intersection stuff" axiom for a base is satisfied vacuously. ✔

2) Suppose that no such k exists. Let $k_1 = \min\{\ell_1, m_1\}$ and $k_2 = \max\{\ell_2, m_2\}$ (note, this is the opposite of the situation we described above). Then

$$B_1 \cap B_2 = X_{-\infty}^{k_1-1} \times \{x_{k_1}\} \times \cdots \times \{x_{k_2}\} \times X_{k_2+1}^{\infty},$$

which is an element of $\mathcal{B}$, hence the "intersection stuff" axiom is satisfied here as well. ✔

In either case, we see that the second axiom for a basis ("intersection stuff") is satisfied.

Finally, we want to show it satisfies the third axiom (generation of by union). By Proposition 1.2, it suffices to show that the $\mathcal{B}$ generate arbitrary singleton cylinders. Let $i \in \mathbb{Z}$ be arbitrary, and let $x_i \subseteq X_i$ be arbitrary. Note, if $i = 0$, then we have

$$X_{-\infty}^{i-1} \times \{x_i\} \times X_{i+1}^{\infty} = X_{-\infty}^{-1} \times \{x_0\} \times X_1^{\infty}$$
$$= \pi_0^{-1}(\{x_0\}),$$

hence the statement holds.

Now, suppose that $i > 0$; the $i < 0$ case follows almost identically. Note that, employing our abuse of notation[7] again, then we have

$$\pi_i^{-1}(\{x_i\}) = \bigcup_{x_k \in X_k} X_{-\infty}^{i-1} \times \left(\prod_{k=0}^{i-1} \{x_k\} \right) \times \{x_i\} \times X_{i+1}^{\infty},$$

---

[7] I.e., $x_k \in X_k$ is quantified before k itself, and taken to mean that we allow the singletons in the product to range over all possible values over the union

hence $\mathcal{B}$ generates arbitrary singleton cylinders. It follows that $\mathcal{B}$ generates all open sets of $\mathcal{T}_{\text{prod}}$. $\qquad\square$

Proposition 1.6. *For each $k \in \mathbb{Z}$, let $(X_k, \mathcal{T}_k)$ be a discrete topological space. Then $X = \prod_{k \in \mathbb{Z}} X_k$ is metrizable, with metric given by*

$$d((x_i)_{i \in \mathbb{Z}}, (y_i)_{i \in \mathbb{Z}}) = \begin{cases} 0 & x_i = y_i \\ 2^{-|k|} & k \text{ first ind. s.t. } x_i \neq x_j \end{cases}$$

$\diamondsuit$

Proof. First, we verify this is a metric. Let $x, y, z \in X$ be arbitrarily chosen. Then

- By definition, $d(x,y) \geq 0$;
- By definition, $d(x,y) = 0 \iff x = y$; and
- By definition, $d(x,y) = d(y,x)$.

It remains to show the triangle inequality. Let $n_{(x,z)}$ be the first index at which x and z differ, and similarly for $n_{(x,y)}$, $n_{(y,z)}$.

Claim: At least two of the n's have to be equal. Further, if exactly two are equal, then they are smaller than the third.

Proof of Claim: Suppose, to obtain a contradiction, that none of the n's are equal. Then they are strictly ordered. WLOG, suppose $n_{(x,z)} < n_{(x,y)} < n_{(y,z)}$. Then by definition of $n_{(x,z)}$, we have

$$x_{n_{(x,z)}} \neq z_{n_{(x,z)}},$$

but note, by definition of $n_{(y,z)}$, y, z cannot differ before $n_{(y,z)}$. Hence, $n_{(x,z)} < n_{(y,z)}$ implies

$$y_{n_{(x,z)}} = z_{n_{(x,z)}}$$

thus $y_{n_{(x,z)}} \neq x_{n_{(x,z)}}$ as well. But this is a contradiction, $n_{(x,z)} < n_{(x,y)}$! It follows that at least two of the n's have to be equal.

To show that [in the case where exactly two are equal they are smaller than the third], note that the same contradiction proof applies for $n_{(x,z)} < n_{(x,y)} = n_{(y,z)}$. Anyways, we now finally establish the triangle inequality. We have two subcases.

1) Suppose that all three of the n's are equal. Denote their common value by N. Then

$$2^{-N} < 2^{-N} + 2^{-N},$$

as desired.

2) Else, suppose that exactly two of the n's are equal. Denote the value of the smaller two by N_{small}, and that of the bigger one by N_{big}. Observe that both

$$2^{-N_{\text{small}}} < 2^{-N_{\text{small}}} + 2^{-N_{\text{big}}}$$

and

$$2^{-N_{\text{big}}} < 2^{-N_{\text{small}}} + 2^{-N_{\text{small}}},$$

so no matter which n is which, we get the triangle inequality.

Hence, we see that the proposed metric *is* indeed a metric.

Lastly: To see that this metric generates the product topology, note that taking sufficiently small radii, the open balls in this metric yield the exact same base as in Proposition 1.5. Hence, we see the product topology is metrizable with the given metric. $\qquad\square$

Remark. Note that if $\Sigma = \{0, 1\}$, this is the *p*-adic metric on $\{0,1\}^{\mathbb{Z}}$.

OK! *Finally* we can summarize all of this in terms of what it means for our actual set of interest, X_{unrest}. The upshot is as follows.

- The elements of X_{unrest} are pairs (q, σ) where σ is a bi-directional infinite sequence of characters from $\Sigma \cup \{\epsilon\}$.
- *Open balls* in X_{unrest} are sets of sequences that share a segment around the 0th character.
- The open balls defined above are actually *clopen*.
- *Open sets* in X_{unrest} are arbitrary unions and/or finite intersections of the open balls above.
- We can think of the *union* operation for our open sets as upping the number of valid options for some index in our sequence, while the *intersection* operation restricts our choices.

1.4. Converting to a Dynamical System

Alright! Now, we can get to turning our DFAs into dynamical systems. Let Q be our state set, and let Σ be our alphabet. As always we take ϵ to be the empty string. Recall, we define X_{DFA}

$$X_{\text{DFA}} = \left\{ (q, \sigma) \in X_{\text{unrest}} \mid \sigma = \left(\epsilon^0_{-\infty} \cdot \sigma_1 \sigma_2 \cdots \sigma_n \cdot \epsilon^\infty_{n+1} \right) \right\},$$

and interpret our string σ as a countable cartesian product of singletons (characters). Endow X_{DFA} with the subspace topology from X_{unrest}. Then we have the following proposition:

Proposition 1.7. X_{DFA} *is a 0-manifold.* $\qquad\diamondsuit$

Note, 0-manifolds are exactly discrete topological spaces.

Proof. Let $x \in X_{\text{DFA}}$ be arbitrary. Write it as

$$x = \epsilon^0_{-\infty} \cdot [\sigma_1 \sigma_2 \cdots \sigma_n] \cdot \epsilon^\infty_{n+1}.$$

Then note that taking $\varepsilon = 2^{-(n+1)}$, we see

$$B_\varepsilon(x) = \{ x' \in X_{\text{dfa}} \mid x'_1 \cdots x'_{n+1} = \sigma_1 \cdots \sigma_n \epsilon \}$$
$$= \{x\},$$

Since by definition, as soon as the ϵ's start for an element of X_{DFA}, they keep on crankin' out until the entropic cows come home to the heat-death of the universe.   □

Neato!!!!!!!!!!!!!!! Now we can define the whole thing:

Definition 1.13 (DFA Dynamical System)**.** Let $M = (Q, F, \Sigma, \delta, q_0)$. Define X_{DFA} as usual, with the subspace topology from X_{unrest}. We'll omit the $\epsilon^0_{-\infty}$ and ϵ^∞_{n+1} prefixes/suffixes for the sake of notational brevity.

Let $T = \mathbb{N}$. Then define $\Phi : T \times X_{\text{DFA}} \to X_{\text{DFA}}$ by

$$\Phi(1, (q, [\sigma_1 \sigma_2 \cdots \sigma_n])) = (\delta(q, \sigma_1), [\sigma_2 \cdots \sigma_n]),$$

and

$$\Phi(n, (q, \sigma)) = \underbrace{\Phi(1, \Phi(1, \Phi(\cdots \Phi(1, (q, \sigma)))))}_{n \text{ times}}$$

Then this gives us a dynamical systems encoding for DFAs.   ◇

Remark. If we had defined X_{DFA} instead by

$$X_{\text{DFA}} = \left\{ (q, \sigma) \in X_{\text{unrest}} \;\middle|\; \sigma = \epsilon^{k_1 - 1}_{-\infty} \cdot \sigma_{k_1} \cdots \sigma_{k_2} \cdot \epsilon^\infty_{k_2 + 1} \right\}$$

then we could make the evolution map Φ invertible, and in fact a homeomorphism. We chose not to do so, since being agnostic about prior state is more in keeping with our intuition about DFAs.

The big-picture takeaways are as follows:

- Any valid initial condition for $t = 0$ must have $q = q_0$.

- The *fixed points* of $(X_{\text{DFA}}, T, \Phi)$ occur exactly when $x = (q, \sigma)$ with

$$\sigma = \epsilon^\infty_{-\infty},$$

which reflects the intuition that the DFA performs no computation on the empty string. That is, "halting states of the DFA == fixed points of the Dynamical System."

The constructions for the next two layers (*context-free* and *context-sensitive*) are somewhat painful to encode, so we'll omit them and jump straight to the dynamical systems for *recursively enumerable* languages: Turing machines.

1.5. TURING MACHINES

Like a DFA, a *Turing machine* (often TM for short) is an abstract model of computation that allows us to encode an algorithmic process as collections of sets with well-defined maps. Turing machines are named for *Alan Turing*, a brilliant mathematician who was horribly persecuted because of his sexuality.

There are many "equivalent" definitions for Turing machines;[8] we'll give a slightly-modified version of the canonical definition.

Remark. For those in the know: As far as we can tell, in order to turn TM's into dynamical systems, it's necessary that we include *two* "blank" characters. The first, ϵ, represents the *empty string*, and will denote the state for an uninitialized tape cell. Our TM will not be allowed to write ϵ characters to the tape. In fact, upon seeing one, our TM will be required to immediately replace it with a ␣, which we call the *blank character*. This seems to be the only way to get our 0-manifold condition.

Definition 1.14 (Turing Machine)**.** A *Turing machine* is an 8-tuple

$$M = (Q, F, \Gamma, \Sigma, \delta, \epsilon, \text{␣}, q_0)$$

such that the following conditions hold:

1) Q, F, Γ, Σ and F are finite sets, with the following naming conventions:
   i) Q is called the set of *states* for M,
   ii) $F = \{\checkmark, \text{✗}\}$ is called the set of *final states* (read "accept" and "reject" respectively),
   iii) Γ is called the *work alphabet*, and
   iv) Σ is called the *input alphabet*;
2) ϵ (called the *empty string*) is neither an element of Σ nor of Γ;
3) ␣ $\in \Gamma$ is a distinguished element called the *blank character*, and we have ␣ $\notin \Sigma$;
4) $q_0 \in Q$ is a distinguished element called the *start state* or *initial state*;
5) We have $F \subseteq Q$ and $\Sigma \subseteq \Gamma \setminus \{\text{␣}\}$; and
6) δ (called the *transition function*) has the following properties:
   i) δ is a partial function $\delta : (Q \setminus F) \times \Gamma \cup \{\epsilon\} \nrightarrow Q \times \Gamma \times \{L, R\}$,[9] where L, R are understood as "shift" directions, and finally
   ii) For all $q \in Q$, we have $\delta(q, \epsilon) = (q', \text{␣}, \text{Dir})$.   ◇

We think of Turing machines as operating on an infinite *work tape*, where the cells of the tape each represent

---

[8]Here, "equivalent" means that the models of computation they define are equally powerful. That is, given two distinct definitions D_0, D_1 for Turing machines, the behavior of any machine defined with D_0 can be simulated using a machine defined by D_1 and vice versa.

[9]Note the inclusion of the word *partial* — in general we do not require δ to be defined on all of $(Q \setminus F) \times \Gamma$.

a symbol from the alphabet $\Gamma \cup \{\epsilon\}$.[10] One can think of this tape as representing the "memory" of the TM.

The tape is initialized with a finite word $\sigma = \sigma_1 \sigma_2 \cdots \sigma_n$ where the $\sigma_i \in \Sigma$ (this represents the input to our program), while the rest of the tape is filled with ϵ characters (we'll draw these lowered). The TM initializes in the *start state*, with the *tape head* (drawn with an $\uparrow$ in the below) on σ_1. We think of the tape head as "pointing" to whatever character the TM is reading currently.
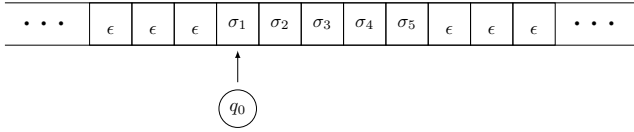

Figure 5: Turing machine initialized with the input string $\sigma_1 \sigma_2 \sigma_3 \sigma_4 \sigma_5$.

The "program" is then executed by the δ function. For instance, suppose $\delta(q_0, \sigma_1) = (q_i, \gamma_i, R)$. Then we interpret this as the Turing machine writing a γ_i on the current tape, updating its state to q_1, and moving one space to the right on the tape.
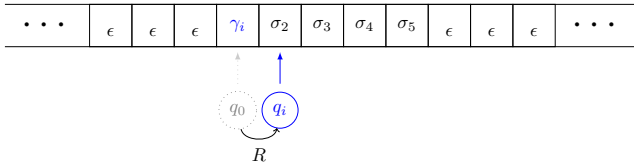

Figure 6: Turing machine after performing a single step. Differences shown in blue.

At each step of the computation, the Turing machine performs a similar process: Read the tape, rewrite and move as dictated by δ, and then repeat. Note, if the Turing machine ever encounters an ϵ, it immediately replaces it with a ␣. If the TM reaches the state ✓, then it halts and the input is said to be "accepted." If it reaches ✗, the state is "rejected." In the case that the machine never halts, we treat this as a form of rejection.

We'll now move to encoding this idea in a dynamical system. Because of the groundwork we established in our rumpus through DFAs and the product topology, this will follow fairly naturally. There's only one major recalibration we have to make, which is that it's more natural to think about the tape sliding around underneath the tape head as opposed to thinking about the tape head as moving over a static tape.

First, we define X_{unrest}:

$$X_{\text{unrest}} = Q \times \left(\prod_{i \in \mathbb{Z}} \Gamma \cup \{\epsilon\} \right).$$

Note: the ⌊correspondence with the X_{unrest} definition given in discussing DFAs⌋ can be established by taking $\Gamma = \Sigma$.

As before, we now restrict X_{unrest} to configurations for which all but finitely many characters on the tape are ϵ. Let X_{TM} be given by

$$Q \times \left(\{\epsilon_{-\infty}^{\infty}\} \cup \bigcup_{\substack{k_1, k_2 \in \mathbb{Z} \\ k_1 \leq 0 \leq k_2}} \left\{ \epsilon_{-\infty}^{k_1-1} \cdot \gamma_{k_1} \cdot \gamma_{k_1+1} \cdots \gamma_{k_2} \cdot \epsilon_{k_2+1}^{\infty} \right\} \right),$$

where $k_1 \leq 0 \leq k_2$, and $\gamma_{k_1}, \gamma_{k_1+1}, \ldots, \gamma_{k_2} \in \Gamma$ are freely chosen. The $k_1 \leq 0 \leq k_2$ condition is what will encode our idea that the tape-head stays "fixed" at 0.

We now define the set of valid initial conditions: Let X_{TM_0} be given by

$$\{q_0\} \times \left(\{\epsilon_{-\infty}^{\infty}\} \cup \bigcup_{k \in \mathbb{Z}^{\geq 0}} \left\{ \epsilon_{-\infty}^{-1} \cdot \sigma_0 \cdot \sigma_1 \cdots \sigma_k \cdot \epsilon_{k+1}^{\infty} \right\} \right),$$

where each $\sigma_i \in \Sigma$. As usual, we endow both X_{TM} and X_{TM_0} with the subspace topology from X_{unrest}.

Proposition 1.8 (0-manifold-ness). *Under the subspace topology from X_{unrest}, both X_{TM_0} and X_{TM} are 0-manifolds (discrete spaces).* ◇

Proof. It suffices to prove the claim for X_{TM}, since $X_{\text{TM}_0} \subseteq X_{\text{TM}}$ and the subspace of a discrete space is discrete. To that end: Let $(q, \gamma) \in X_{\text{TM}}$ be arbitrary. Suppose γ is of the form

$$\gamma = \epsilon_{-\infty}^{k_1-1} \cdot \gamma_{k_1} \cdot \gamma_{k_1+1} \cdots \gamma_{k_2} \cdot \epsilon_{k_2}^{\infty}.$$

Let $n = \max\{|k_1|, |k_2|\}$, and let $\varepsilon = 2^{-(n+1)}$. Note that our choice of ε restricts us to strings γ' of the form

$$\gamma' = \jmath_0 \cdot \lceil \epsilon_{-(n+1)}^{k_1-1} \cdot \gamma_{k_1} \cdot \gamma_{k_1+1} \cdots \gamma_{k_2} \cdot \epsilon_{k_2}^{n+1} \rceil \cdot \jmath_1$$

where $\jmath_0, \jmath_1$ are arbitrary strings. By definition of X_{TM}, the only possible string of this form is γ itself, hence we have

$$B_\varepsilon(\gamma) = \{\gamma\},$$

hence the set of admissible tape states is discrete. Since we endow Q with the discrete topology, and the finite product of two discrete spaces is discrete, it follows that X_{TM} is discrete, as desired! □

---

[10]Beyond requiring ␣ $\in \Gamma$, ␣ $\notin \Sigma$, we don't actually need Γ, Σ to be distinct. Doing so is mainly a matter of convenience. All we really need out of Γ is the ability to put down a *delimiter* character that helps us distinguish between separate blocks in our tape. This is important in being able to use the tape as a form of "memory" for the machine. Note, even if Σ is a unary alphabet like $\Sigma = \{0\}$, we can use the blank to create *logical* characters that function as if they're part of an expanded glyph set (we'll denote these by char$_L$, L for "logical"). E.g., we could have ⌊␣$_L$ = ␣␣⌋, ⌊$0_L = 0$␣⌋, ⌊$1_L = 00$␣⌋, ⌊$2_L = 000$␣⌋2, and so on. All of these parsing rules can be treated in state.

We have our phase space, now for the rest. As usual, let $T = \mathbb{N}$, and interpret 1 step in T as performing 1 step of the computation. We now construct the evolution map Φ.

First: In the edge case where the current character under the tape head (i.e. at the 0-index) is the ϵ, we define

$$\Phi(1, (q, s_{\text{left}} \cdot \epsilon \cdot s_{\text{right}})) = (q, s_{\text{left}} \cdot {}_{\llcorner\lrcorner} \cdot s_{\text{right}}).$$

Note, $\cdot {}_{\llcorner\lrcorner} \cdot$ is not *just* a slick new emoticon, it's also meant to denote "concatenate a blank with the things on either side."

Now the other case: Let $(q, s) \in X_{\text{TM}}$ where

$$s = \epsilon_{-\infty}^{k_1 - 1} \cdot \gamma_{k_1} \cdot \gamma_{k_1+1} \cdots \gamma_{k_2} \cdot \epsilon_{k_2+1}^{\infty},$$

and $k_1 \leq 0 \leq k_2$. Also let $(q', \gamma_0', \text{Dir}) = \delta(q, \gamma_0)$. Let

$$s' = \epsilon_{-\infty}^{k_1-1} \cdot \gamma_{k_1} \cdot \gamma_{k_1+1} \cdots \gamma_{-1} \cdot \gamma_0' \cdot \gamma_1 \cdots \gamma_{k_2} \cdot \epsilon_{k_2+1}^{\infty}$$

let s_{left}' be a copy of s' shifted to the left, and s_{right}' be a copy shifted to the right. In both cases, if shifting lands the 0-index on an ϵ, replace it with a $_{\llcorner\lrcorner}$. Then define

$$\Phi(1, (q, s)) = \begin{cases} (q', s_{\text{left}}') & \text{if Dir} = \text{Left, and} \\ (q', s_{\text{right}}') & \text{if Dir} = \text{Right.} \end{cases}$$

This gives us a full dynamical encoding of Turing machines! Here are the big takeaways:

- "Fixed points" correspond to either accepting or rejecting states.

- Including *two* blank characters is necessary to guarantee that the region of the tape we've visited at any given point in the computation always acts like a single continuous block. This is essential in guaranteeing that we can find an $\varepsilon > 0$ such that $B_\varepsilon(\gamma) = \{\gamma\}$. Else, we could create problems with TMs that perform algorithms like the following:

  1) Write a character at the current position.
  2) Delete it.
  3) Move one cell to the right.
  4) Repeat.

  Note, as this TM runs, it generates configurations that become arbitrarily close to $\epsilon_{-\infty}^{\infty}$, which breaks the 0-manifold condition.

- In general, Φ is not invertible. For instance, we can imagine a machine that replaces any input with a string of $_{\llcorner\lrcorner}$'s whose length is the next-largest power power of 2, and then halts. Then for any input of length $2^n + 1$ to 2^{n+1}, the machine halts in an identical configuration.

There are many questions we wanted to ask about this dynamical encoding that we never had the opportunity to investigate due to time constraints. For instance, do randomized approximation algorithms correspond to

TM's with attracting fixed points? How can we interpret numerical instability in terms of chaos? To what extent can we understand the difficulty of the halting problem in terms of detecting aperiodic orbits? So on and so forth. In the meantime though, we did take a deep dive into computational universality, which will comprise the remainder of the paper.

2. Computational Universality

A Universal Turing Machine is a Turing machine which can simulate any arbitrary Turing machine. It takes as its input a Turing Machine code $\langle M \rangle$ (for turing machine M we'll denote the code of M by $\langle M \rangle$), and an input word w, and then simulates what M would do on input w. Intuitively, any modern computer is a (finite memory) Universal Turing Machine: it can store arbitrary computer programs as data, and then run them on other data it stores or with user input.

In a similar vein, a system is *computationally universal* (or just *universal*) if it has the ability to simulate an arbitrary Turing machine. This is a loose definition because it is not immediately clear what it means for a dynamical system to simulate a Turing machine. There are many possible ways to create a more precise definition; we will follow the method of Delvenne et al. [2]. Before we do, though, we note a few reasonable expectations about what simulating a Turing machine should mean:

- If a dynamical system can simulate an arbitrary Turing machine, we should be able to take any combination of Turing machine and input, and convert it into some question about the system. The question should be "equivalent" to the question of whether the Turing machine would accept its input, in the sense that it should have a "yes" answer if the Turing machine accepts, and a "no" answer if the Turing machine rejects or never halts.

- The Universal Turing Machine is a specific Turing Machine that exists, and therefore it is a dynamical system on the space of its possible configurations. We would definitely expect this dynamical system to be considered universal.

- There are a number of other computational models which have ability to simulate Turing Machines, and which can also be considered dynamical systems. One good example is cellular automata, which are like Turing machine tapes where every cell has its own state and updates itself based on its neighbors' states. If a given instance of a computation model is capable of simulating arbitrary Turing machines, we would expect its associated dynamical system to be universal.

We will follow the lead of Delvenne et al. [2] in defining what it means for a dynamical system to be universal. Like them, we will consider only *symbolic* dynamical

systems, that is, dynamical systems on spaces consisting of infinite words made out of symbols drawn from a finite alphabet. We will need to do some spadework before we're ready to give the full definition.

2.1. Spadework Part I - R.E.-completeness

Our ultimate goal is, given the code for a Turing machine M, and the input w, to ask a question about a particular dynamical system which has a "yes" answer exactly when M accepts w. In the terms used in the field of computability, we want to be able to reduce an arbitrary instance of the question "Will Turing machine M accept the input w?" to a question about the dynamical system.

Fortunately, much is already known about the question "Will Turing machine M accept the input w?". In particular, it is equivalent to the question "Will Turing machine M halt on the input w?" in the sense that if we had a way to answer either of the questions, we could easily answer the other as well. This second question is called the Halting Problem, and computer scientists say that it is "complete for the class of recursively enumerable problems."

So, depending on your background, you may be asking two questions: "What is the class of recursively enumerable problems?" and "What does it mean to be complete for it?" A problem is called recursively enumerable if it can be solved by a Turing machine that may not ever halt, that is, if the question has a "yes" answer, the Turing machine will accept in finite time, but if the question has a "no" answer, the machine may run indefinitely. They are called "recursively enumerable" because it would be possible for a Turing machine (a computation model capable of *recursion*) to *enumerate* every input that would have a "yes" answer, in such a way that every such input would eventually be listed in finite time. (The machine could do this by listing all inputs lexicographically, and spending half its time simulating the computation on the first input, a quarter of its time on the second input, and so on, printing out any input when it is accepted.)

A given problem, call it P, is complete for a class of problems, C, if two conditions hold. First, P must be an element of C. Second, all problems in C must be reducible to P, meaning that if $P' \in C$, we can convert any instance of P' to an instance of P that will have the same answer, without doing an unreasonable amount of work in the conversion (for our purposes here, any finite computation time is reasonable). The Halting Problem is known to be complete for the recursively enumerable problems, or r.e.-complete.

The question "Will Turing machine M accept input w?" (call it Q) is also r.e.-complete. Recall that our goal was to find a question QDS about a dynamical system, which we could reduce Q to. One way to express this constraint is that QDS should be r.e.-complete: then both QDS and Q could be reduced to each other,

so they would be equivalent questions, which is what we are looking for. In fact, we will see that the r.e.-completeness of QDS is exactly what we will require.

2.2. Spadework Part II - Effective Symbolic Systems

A symbolic set can be thought of as a set of words made from a finite alphabet A. We could express such a set as $(A \cup \{B\})^{\mathbb{N}}$, meaning the set of one-sided infinite words with characters which are either drawn from A or are the "blank" symbol B. Finite words would then end be expressed as infinite words ending with an infinite tail of Bs. But we could encode each element of $A \cup \{B\}$ with a binary sequence (with all encodings the same length), so any symbolic set can be encoded as a subset of the set $\{0, 1\}^{\mathbb{N}}$ of one-sided infinite binary words.

We give $\{0, 1\}^{\mathbb{N}}$ the discrete topology discussed above. Under this topology, the set of all sequences beginning with a finite binary word w is a both a closed and open set (a "clopen" set) under this metric. Call sets like this, generated by a common prefix, cylinders. It can be shown that the clopen sets of $\{0, 1\}^{\mathbb{N}}$ are precisely the finite unions of cylinders. What is special about this is that we can associate with each clopen subset of $\{0, 1\}^{\mathbb{N}}$ a unique finite set of finite words, which are the generating prefixes for the cylinders that make up that set. So we can express every clopen set of $\{0, 1\}^{\mathbb{N}}$ in a finite way, which means the we can list off all the clopen sets in some lexicographic order (in other words, the set is countable). We'll be exploiting this fact later.

Now we're ready to define the main dynamical systems we'll be working with. If $X \subseteq \{0, 1\}^{\mathbb{N}}$ is a symbolic set and $f : X \to X$ is a continuous map on X, then (X, f) is an *effective symbolic system* if:

(i) X is closed.

(ii) Checking whether some clopen set $Y \subseteq \{0, 1\}^{\mathbb{N}}$ has a non-zero intersection with X is decidable by a Turing machine in finite time.

(iii) The inverse map f^{-1} on clopen sets of X can be computed in finite time.

These requirements may seem a bit arbitrary, but they are made for a good reason. We will shortly be performing some operations on the clopen sets of X, and it turns out that, since X is closed, these are exactly the intersections of X with the clopen sets of $\{0, 1\}^{\mathbb{N}}$, so we know this is a countable set. Requirement (ii) helps ensure that a Turing machine could list out these clopen sets without ever getting stuck checking if a particular clopen set of $\{0, 1\}^{\mathbb{N}}$ has a nonempty intersection with X. And requirement (iii) foreshadows operations we will be doing on clopen sets.

2.3. Spadework Part III - Temporal Logic

Recall that we are looking for a way to "ask questions" about dynamical systems, whose answers will tell us

something about the behavior of Turing machines. In order to be able to construct such questions, we will use subsets of state space to encode logical statements. Since we want to be able to make statements about the *eventual* behavior of the system, we will use a logical framework called temporal logic. Temporal logic includes all of the standard logical operators:

- $\top$: always true

- $\bot$: always false

- $\vee$: or

- $\neg$: not

- $\wedge$: and (which is actually redundant because it is constructible from $\vee$ and $\neg$)

It adds two additional operators to the list:

- $\circ$: next, a unary operator, with $\circ\phi$ meaning that ϕ will be true after one time step.

- $\mathcal{U}$: until, a binary operator, with $\phi\mathcal{U}\psi$ meaning that ψ will be true within finite time, and for all time between the present and one step before the time when ψ is true (inclusive), ϕ will be true

The developers of temporal logic, Arthur Prior and Hans Kamp, have described a set of rules for incorporating these operators into classical logic in a way that is consistent with our interpretation of them [2]. For our purposes, suffice it to say that it works.

How do we propose to use temporal logic to construct statements about effective symbolic systems? Well, as we have suggested above, we are interested in operating on the clopen sets of symbolic systems, and here is where that will come into play. Let's say we have some effective dynamical system (X, f). We will use the fact that the clopen subsets of X are countable, and let $P_0, P_1, P_2, ...$ be a listing of all the clopen subsets of X (including $\emptyset$ and X in some positions).

Then let $\mathcal{P}_0, \mathcal{P}_1, \mathcal{P}_2, ...$ be a set of propositional symbols, which are the logical "units" that along with the operators above are the building blocks of temporal logic formulas. The listing will contain $\top$ and $\bot$ in some positions. We're going to define an "interpretation" operator $|\cdot|$ that takes logical formulas to subsets of X. The intuition behind it will be that if ϕ is a logical formula that expresses something about the "current configuration of the system," $|\phi|$ is the set of possible system configurations (points of X) for which that statement is true. Formally, the operator will behave like this:

- If ϕ is just the symbol $\mathcal{P}_n$, $|\phi| = P_n$, and we stipulate that the orderings should align such that $|\top| = X$ and $|\bot| = \emptyset$. This means that the symbol $\mathcal{P}_n$ represents the statement "The current system configuration is in the set P_n," which is of course always true if $P_n = X$ and never true if $P_n = \emptyset$.

- $|\phi_1 \vee \phi_2| = |\phi_1| \cup |\phi_2|$, because for either ϕ_1 or ϕ_2 to be true, the system can be in any state in which

either is true.

- $|\neg\phi| = X \setminus |\phi|$, because the set of states for which ϕ is not true is the complement of the set of states for which it is.

- $|\circ\phi| = f^{-1}(|\phi|)$, which means that $\circ\phi$ represents the statement, ϕ will be true after one application of f, meaning that each application of f corresponds to one "time step" within the temporal logic system. Note that we know that this is computable, by requirement (iii) of effective symbolic systems.

- $|\phi_1 \mathcal{U} \phi_2| = \bigcup_{n\in\mathbb{N}} A_n$ with $A_0 = |\phi_2|$ and the other sets defined by the recurrence relation $A_{n+1} = f^{-1}(A_n) \cap |\phi_1|$. Here A_n can be interpreted as, the set of system states for which ϕ_2 will be true on the nth time step from the current state, and ϕ_1 will be true on the $0, ...n-1$ time steps. The union of all of these represents all states for which "ϕ_1 is true until ϕ_2 is eventually true."

So, in short, we have used temporal logic to construct a set of formula that express statements about the state of the system. We say a given formula is satisfiable if its interpretation is nonempty, meaning that there is some nonempty set of X that satisfies it.

2.4. Universality

Our spadework concluded, we're ready to say what it means for an effective dynamical system to be universal. We will use Delvenne et. al's precise definition here:

Definition 2.1 (universality). "An effective dynamical system is *universal* if there is a recursive family of temporal formulae such that knowing whether a given formula of the family is satisfiable is an r.e.-complete problem."   ◇

A "recursive" family of temporal formulae is a set of temporal formula for which membership in the set can be decided by a turing machine in finite time; this is essentially a regularity condition which prevents the family from being outlandishly defined.

To get a feel for what this definition means, and ensure that it is reasonable, let's start by confirming that the Universal Turing Machine dynamical system is, as we expected from the start, universal.

A configuration of the Universal Turing Machine, as with all Turing machines, is defined by finite control state and its tape contents, both of which can be encoded in a binary strings. So the configuration space of a UTM is a subset X of $\{0, 1\}^{\mathbb{N}}$, and the code for the UTM is some specific transition function on this space, f. Let's assume we are dealing with a a UTM that takes as input $\langle M \rangle, w, x$ with some special encoding of the delimiter ",". The machine first checks that its input is of this form, and halts immediately if it is not, and otherwise ignores x and simulates M on w. (We put the

x there so that we can have an arbitrary string on the end of our input, and the set of starting configurations given a specific choice of M and w will be a clopen set.)

It can be easily shown that this is an effective dynamical system. To show it is universal, we're looking for recursive family of formula whose satisfiability is r.e.-complete. Consider all the possible pairs of turing machine encodings, $\langle M \rangle$, and inputs w. Given such a pair, let P_m represent the clopen set of all UTM configurations in the starting state and tape contents beginning with "$\langle M \rangle, w,$". Let P_h represent the clopen set consisting of all halting configurations. (This will be clopen if we encode configurations in such a way that the finite control state is listed first, since the halt state will then be the common prefix). Then the formula corresponding to M and w is

$$\mathcal{P}_m \wedge \left(\top \mathcal{U} P_h \right).$$

This formula's interpretation is the set of configurations which are in P_m (they are starting configurations of the UTM which encode M as the TM to simulate and w as the input), and which will eventually halt. Since the UTM just does whatever M would do on w, if the formula above is satisfiable, M halts on w, and if it is not satisfiable, M does not halt on w. So if we could answer the satisfiability question for all formulas of the form above, we would have a way to solve the halting problem. This means that the satisiability problem for this family of formulae is r.e.-complete, so the dynamical system is universal under the definition!

It may seem strange that we introduced the useless input x into our dynamical system. This was not in fact necessary to make the dynamical system universal, but we did it to make explanation easier. The reason this made our lives easier our formulas encode statements about clopen sets, but we wanted to talk about a particular initial configuration, so we made the machine clearly treat all the elements in the clopen set as the same input. A reasonable question would be, why make our definition involve clopen sets at all, rather than encode single configurations with formulas? In fact it is possible to do rewrite the definition without clopen sets, using only single points, but if we did so, then the shift map on $\{0,1\}^{\mathbb{N}}$ would be considered universal. This is because we could encode an infinite computation process in an infinite binary string, and then ask whether some number of shifts brings us to the halting configuration. The fundamental issue is that encoding with points allows points to store an infinite amount of information, whereas in real systems configurations can only store finite information. Because we are interested in the map itself performing computation rather than the chosen points, we require clopen sets instead.

Similar arguments can be used to show that cellular automata and other models of computation capable of simulating Turing machines, when made into dynamical systems, satisfy our universality definition. What is interesting, though, is that this definition opens up the possibility that dynamical systems which are not immediately associated with Turing machines, might be universal. In fact, Delvenne et al. show that there does exist a system which is both chaotic and universal. They also show that certain properties of dynamical systems, like equicontinuity of the map, imply that a dynamical system cannot be universal.

Since many dynamical systems are conjugate to symbolic ones, this opens up an interesting area of research: the question of, what physical systems can be considered universal? For example, actual computers in the real world are physical systems that are capable of computation (although because they have bounded memory, they are technically equivalent to DFAs and not Turing machines). It is possible there are other physical systems with computational ability that is less obvious.

3. Appendix

Here, we include some of the background definitions the reader might not be acquainted with. First, we define a *topology*.

Definition 3.1 (Topology)**.** Let X be an arbitrary set. Let $\mathcal{T}$ be a collection of $U \subseteq X$ such that

1) (Containment of trivial elements): $\varnothing, X \in \mathcal{T}$,

2) (Closure under arbitrary unions): For arbitrary collections $\{U_i\}_{i \in I} \subseteq \mathcal{T}$, we have

$$\left(\bigcup_{i \in I} U_i \right) \in \mathcal{T},$$

and

3) (Closure under finite intersection): For finite collections $\{U_i\}_{i=1}^n$, we have

$$\left(\bigcap_{i=1}^n U_i \right) \in \mathcal{T}. \qquad \diamondsuit$$

Then we call $\mathcal{T}$ a *topology* on X.

The definition of a *topology* is meant to axiomatize the properties of *open sets* that we encounter in analysis. Indeed, one might note that in any metric space X, (1) both $\varnothing$ and X are open sets, (2) open sets are closed under arbitrary union, and (3) open sets are closed under finite intersection.

However, one should note that this is a *proper* generalization, in the sense that there exist topologies which cannot arise from a metric. For instance, consider the following:

Example 3.1 (Double-headed snake)**.** Let $X = \{0_A, 0_B\} \cup (0,1]$. Here, the labels A and B are just to distinguish the two "copies" of 0 that we've made. Define a topology $\mathcal{T}$ on X as follows:

1) Let $U \subseteq (0, 1]$. Then if U is open in $[0, 1]$ viewed as a metric space, we have $U \in \mathcal{T}$.

2) Similarly: Given such a U, we have $\{0_A\} \cup U$, $\{0_B\} \cup U \in \mathcal{T}$.

This effectively gives us a version of $[0, 1]$ where we have two copies of 0. One can verify that the topology axioms are satisfied

So, why isn't this topology metrizable? In a nutshell, the problem is that 0_A, 0_B act as if they are distance 0 apart from each other, which breaks the "$d(x, y) = 0 \iff x = y$" axiom for metric spaces.[11]

This underscores that topology is generally a more abstract way of looking at properties of space. There are a number of situations in which this abstraction is desired. First, it helps us to focus on more "abstract" properties of space that are invariant under stretching and deforming (but not gluing or tearing) — e.g., in a topological context, the two shapes shown in Fig. 7 are the "same."
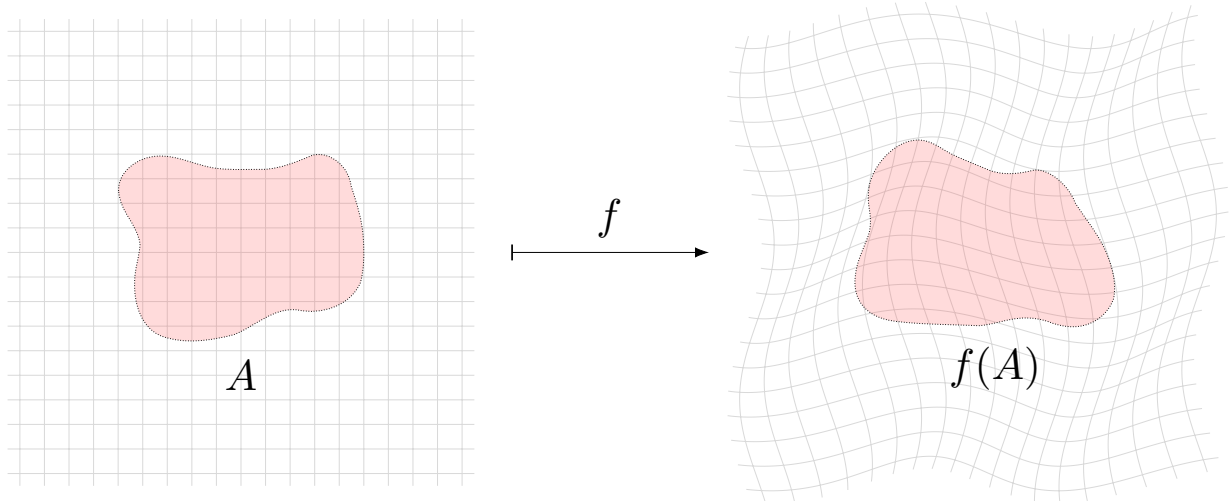

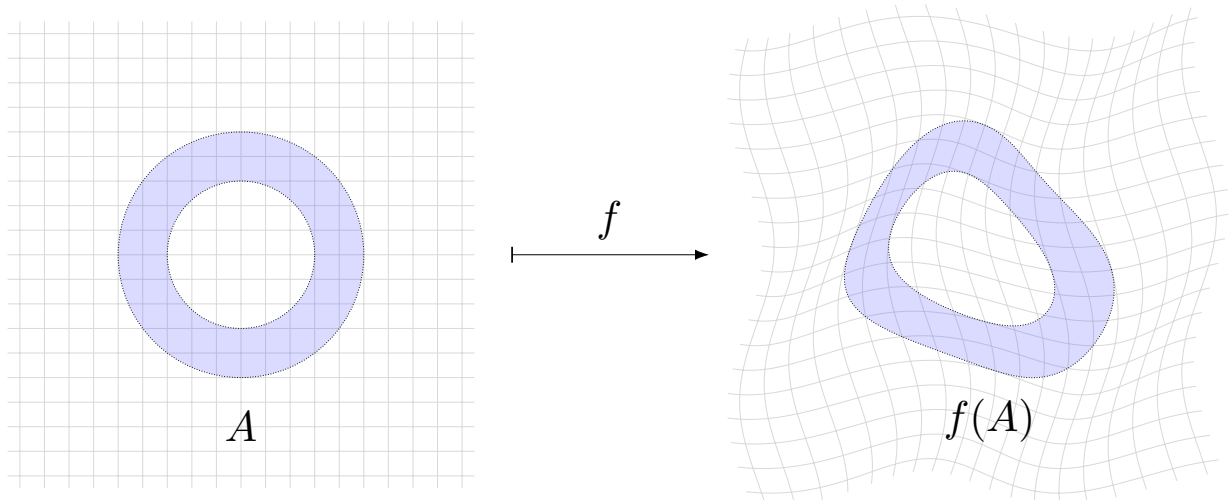Figure 7: Two shapes that are topologically equivalent

Figure 8: Two shapes that are topologically equivalent to each other, but not to those in Fig. 7

---

[11] A more rigorous argument can be constructed by noting that $\forall \varepsilon > 0$, $\{0_A\} \cup (0, \varepsilon)$ and $\{0_B\} \cup (0, \varepsilon)$ are open and then doing something like $0_A = \lim_{\varepsilon \to 0} \varepsilon = 0_B$, a contradiction.

Often, these are the kinds of properties we're interested in when we try to find ways of thinking of random sets X that we've just scooped out of a bush on the side of the street in terms of "spaces." Understanding exactly what we mean by invariance under "stretching and deforming" is the subject of the next section.

3.1. Continuity

In abstract algebra, we define *homomorphisms* to be functions that preserve the "structure" of algebraic objects like groups, rings, and so on. In linear algebra, *linear transformations* similarly preserve the structure of vector spaces. *Isomorphisms* in group theory provide a sense of "perfect, reversible matching," just as *invertible linear transformations* ("vector space isomorphisms") do in linear algebra.

These are examples of *morphisms*, a broader term for structure-preserving maps that sees a lot of tasteless name-dropping by people like us who got a little too excited about category theory once upon a time. In defining an abstract topology, we have created a structure. We now define the maps that preserve that structure. The answer is continuous functions, and the inspiration comes from thinking about continuous functions in metric spaces.

Let (E, d), (E', d') be metric spaces, and let $f : E \to E'$ be continuous. Then recall the following properties of f:

- For all compact sets $V \subseteq E$, $f(V)$ is compact in E',
- For all connected sets $C \subseteq E$, $f(C)$ is connected in E', and
- For all open sets $U \subseteq E'$, the inverse image $f^{-1}(U)$ is open in E. In fact, this is an equivalent way to characterize continuity.

For all these reasons (especially the final one), it turns out that continuous functions provide the natural definition of morphism for our topological spaces. We can loosely interpret continuous functions as preserving the notion of *closeness* of points, which is essentially the fundamental "structure" of a metric space. Again, this is analogous to the definition of a group homomorphism

$$\varphi(g + h) = \varphi(g) + \varphi(h),$$

or a linear transformation

$$T(c_1 u + c_2 v)) = c_1 T(u) + c_2 T(v).$$

In particular, writing out the definition of continuity, we see that it has the same basic structure of "property in the domain is preserved in the codomain:"

$$d(x_0, x_1) < \delta \implies d(f(x_0), f(x_1)) < \varepsilon.$$

The details get slippery if you poke around too closely, but the intuition is there. Anyways: Since not all topologies are metrizable, we choose a definition that relies only on topological structure, but which coincides with continuity in the case of metric spaces.

Definition 3.2 (Continuous function)**.** Let $(X, \mathcal{T}_X)$, $(Y, \mathcal{T}_Y)$ be topological spaces. Let $f : X \to Y$. Then we say f is *continuous* iff for all $U \in \mathcal{T}_Y$,

$$f^{-1}(U) \in \mathcal{T}_X. \qquad\qquad \diamond$$

This is a bit strange, since the structure-preservation occurs in taking the inverse image, not the image. We won't go into the weeds of trying to interpret this, and will instead just talk about the concept that *does* port well: topological isomorphism, almost universally referred to as *homeomorphism*.

Definition 3.3 (Homeomorphism)**.** Let $(X, \mathcal{T}_X)$, $(Y, \mathcal{T}_Y)$ be topological spaces. Let $f : X \to Y$ be continuous and bijective. Further suppose f^{-1} is continuous. Then we call f a *homeomorphism*. $\diamond$

Homeomorphisms provide the rigorous underpinning for thinking about "smooth deformations" on a space.

Analogously to bases in linear algebra and generators in group theory, we have the notion of a "base" in topology.

Definition 3.4 (Base)**.** Let $(X, \mathcal{T})$ be a topological space. Let $\mathcal{B} \subseteq \mathcal{T}$ such that

1) (Covering): We have

$$\bigcup_{B \in \mathcal{B}} B = X,$$

    and

2) (Intersection stuff): For all $B_1, B_2 \in \mathcal{B}$ and for all $x \in B_1 \cap B_2$, there exists $B_3 \in \mathcal{B}$ such that $x \in B_3$ and

$$B_3 \subseteq B_1 \cap B_2.$$

3) (Generation by union): For all $U \in \mathcal{T}$, there exists $\mathcal{B}' \subseteq \mathcal{B}$ such that

$$\bigcup_{B \in \mathcal{B}'} B = U.$$

Then we call $\mathcal{B}$ a *base* or *basis* for the topology $\mathcal{T}$. $\diamond$

These properties should be reminiscent of open balls in $\mathbb{R}^n$. Note that open balls (1) cover $\mathbb{R}^n$, and (2) satisfy the "intersection stuff" property — namely, although the intersection of two open balls B_1, B_2 is generally not another open ball, for each $x \in B_1 \cap B_2$, we can find a *third* open ball B_3 such that $x \in B_3 \subseteq B_1 \cap B_2$. A base is meant to extend this idea to general topologies.

Lastly, we introduce the idea of a manifold. Recall that in single-variable calculus, we can often understand properties of functions by thinking about them as being *locally linear*. In multivariable calculus, we think about surfaces in terms of *local planarity*. This is the fundamental idea behind a manifold, only here we'll loosen our definition to encompass objects that might be a little spikier (e.g., $|x|$).

Definition 3.5 (Manifold)**.** Let $(X, \mathcal{T})$ be a topological space. Suppose there exists $n \in \mathbb{Z}^{\geq 0}$ such that for all $x \in X$, there exists an open set $U_x \subseteq X$ (with $x \in U_x$) and a homeomorphism $f_x : U_x \to \mathbb{R}^n$. Then we call X a n-manifold. $\diamondsuit$

This definition exactly encodes the idea that an n-manifold is something that can be "locally" into $\mathbb{R}^n$. Note that this *is* different from a global condition: E.g., a *torus* (the surface of a donut) is a 2-manifold, but cannot be globally deformed into a plane without performing some cutting or gluing operations.

REFERENCES

[1] V. Benci, C. Bonanno, S. Galatolo, G. Menconi, and M. Virgilio. Dynamical systems and computable information. *Discrete & Continuous Dynamical Systems - B*, 4(4):935, 2004.

[2] J.-C. Delvenne, P. Kurka, and V. Blondel. Computational Universality in Symbolic Dynamical Systems. *Margenstern M. (eds) Machines, Computations, and Universality*, 2005.

[3] P. Koiran, M. Cosnard, and M. Garzon. Computability with low-dimensional dynamical systems. *Theoretical Computer Science*, 132(1):113–128, Sep 1994.

[4] P. Kůrka. On topological dynamics of Turing machines. *Theoretical Computer Science*, 174(1):203–216, Mar 1997.

[5] C. Moore. Unpredictability and undecidability in dynamical systems. *Phys. Rev. Lett.*, 64(20):2354–2357, May 1990.

[6] H. T. Siegelmann and S. Fishman. Analog computation with dynamical systems. *Physica D*, 120(1):214–235, Sep 1998.

[7] J. V. Tucker and J. I. Zucker. Computable total functions on metric algebras, universal algebraic specifications and dynamical systems. *Journal of Logic and Algebraic Programming*, 62(1):71–108, Jan 2005.