Symmetry **2015**, *7*, 1289-1332; doi:10.3390/sym7031289

OPEN ACCESS

symmetry

ISSN 2073-8994

www.mdpi.com/journal/symmetry

Article

Computing with Colored Tangles

Avishy Y. Carmi and Daniel Moskovich *

Faculty of Engineering Sciences & Center for Quantum Information Science and Technology, Ben-Gurion University of the Negev, Beer-Sheva 8410501, Israel;
E-Mail: avcarmi@bgu.ac.il

* Author to whom correspondence should be addressed; E-Mail: dmoskovich@gmail.com; Tel.: +972-58-6563724.

Academic Editor: Louis H. Kauffman

Received: 30 April 2015 / Accepted: 14 July 2015 / Published: 20 July 2015

Abstract: We suggest a diagrammatic model of computation based on an axiom of distributivity. A diagram of a decorated colored tangle, similar to those that appear in low dimensional topology, plays the role of a circuit diagram. Equivalent diagrams represent bisimilar computations. We prove that our model of computation is Turing complete and with bounded resources that it can decide any language in complexity class IP, sometimes with better performance parameters than corresponding classical protocols.

Keywords: diagrammatic algebra; low dimensional topology; computation; Turing machine; interactive proof

MSC classifications: 68Q05; 03D10; 57M25

---

1. Introduction

The present research represents a step in a program whose goal is to study topological aspects of information and computation. Time is a metric notion, and so, any such topological aspects would presumably have no internal notion of time. The present paper formulates a notion of computation that is independent of time and that is natively formulated in terms of information. We formulate a diagrammatic calculus whose elements we call tangle machines. These were first defined in [1]. A key feature of tangle machines is that they come equipped with a natural notion of equivalence, which originates in the beautiful diagrammatic algebra of low dimensional topology. Tangle machines serve

in this paper as abstract flowcharts of information in computation. The computational paradigm that we propose contains Turing machines and interactive proofs (thus, it does not "lose anything"), and it also contains additional models (e.g., Section 8.1).

The world we observe around us evolves along a time axis, so a tangle machine could not be used as a blueprint for a classical computer. Time is a more nebulous concept in the quantum realm, however, and it might be that tangle machine constructions are relevant for adiabatic quantum computations or in other quantum contexts. In particular, they naturally incorporate the axiom of uniform no-cloning (Remark 5). The main relevance of our work would probably be to isolate and access natively topological aspects of classical and quantum computation. We also speculate that tangle machine computations can emerge physically via dynamical processes on a tangle machine, given a set of input colors, and that perhaps something like this actually occurs in nature. After all, natural computers are not Turing machines.

How might tangle machines manifest themselves in nature? The authors make the following speculation. Evolutionary biology provides an analogue to tangle machines in the notions of phenotype *versus* genotype [2,3]. The external characteristics of an organism, such as its appearance, physiology, morphology, as well as its behaviors are collectively known as the phenotype. The genotype, on the other hand, refers to the inherent and immutable information encoded in the genome. Two phenotypes may look entirely different, but may nevertheless share the same genotype. Could information about an organism be encoded as a tangle machine, where equivalent machines represent different phenotypes that share the same genotype? Might the process of evolution of an organism be described by a series of basic transformations akin to the Reidemeister moves exerted by the environment on the organism, which change its phenotype while preserving its genotype, along with occasional "violent" local moves on a current configuration that change its genotype? Might tangle machines describe a way in which nature processes its information primitives, that is its organisms?

There are two obvious advantages to a topological model of computation. The first is that it is very flexible by construction. Bisimilar computations (Definition 10) are represented by topologically equivalent objects, which are related in a simple way (Section 10). The second, which we do not discuss in this paper, is that we have a notion of topological invariants, which are characteristic quantities that are intrinsic to a bisimilarity class of computations.

In the Introduction, we briefly introduce tangle machines in Section 1.1 after which we state our main results in Section 1.2 and give scientific context in Section 1.3.

1.1. What Is a Tangle Machine Computation?

A tangle machine is built up of registers, each of which may hold an element of a set Q. The set Q comes equipped with a set B of binary operations representing basic computations. For $\triangleright \in B$, we read $x \triangleright y$ as "the result of running the program $\triangleright y$ on input data x". An alternative evocative image is that $x \triangleright y$ is a "fusion of information x with information y using algorithm $\triangleright$". Our binary operations satisfy the following axioms, which equip (Q, B) with what is called a quandle structure (see Section 3.2 for this and extensions):

Idempotence: $x \triangleright x = x$ for all $x \in Q$ and for all $\triangleright \in B$. Thus, x cannot concoct any new information from itself.

Reversibility: The map $\triangleright y\colon\ Q \to Q$, which maps each color $x \in Q$ to a corresponding color $x \triangleright y \in Q$, is a bijection for all $(y, \triangleright) \in (Q, B)$. In particular, if $x \triangleright y = z \triangleright y$ for some $x, y, z \in Q$ and for some $\triangleright \in B$, then $x = z$. Thus, the input x of a computation may uniquely be reconstructed from the output $x \triangleright y$ together with the program $\triangleright y$.

Distributivity: For all $x, y, z \in Q$ and for all $\triangleright, \blacktriangleright \in B$:

$$(x \triangleright y) \blacktriangleright z = (x \blacktriangleright z) \triangleright (y \blacktriangleright z) \ . \tag{1}$$

This is the main property. It says that carrying out a computation $\blacktriangleright z$ on an output $x \triangleright y$ gives the same result as carrying out that computation both on the input x and also on the state y and then combining these as $(x \blacktriangleright z) \triangleright (y \blacktriangleright z)$. In the context of information, this is a no double counting property [4].

Later in this paper, in Remark 5, we show that uniform no-cloning and no-deleting, which are fundamental properties of quantum information, follow from reversibility and distributivity for an appropriate coloring by a generalization of a quandle called a quagma. This observation argues for reversibility and distributivity as being nature's most fundamental information symmetries.

The basic unit of computation in a tangle machine is an interaction representing multiple inputs $x_1, \ldots, x_k$ independently fed into a program $\triangleright y$, as depicted in Figure 1. These are concatenated (perhaps also with wyes) to form a tangle machine; see Section 3.

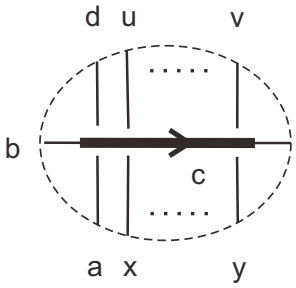


Figure 1. An interaction in a machine, where $z_i \stackrel{\text{def}}{=} x_i \triangleright y$ for $i = 1, 2, \ldots, k$.

A tangle machine computation begins with an initialization of a specified set of input registers to chosen colors in Q. A disjoint set of output registers is chosen. If the colors of the input registers uniquely determine colors for the output registers, then the colors of the output registers are the result of the computation. Otherwise, the computation cannot take place. (See Definition 6 and Figure 2). This provides a model of computation.

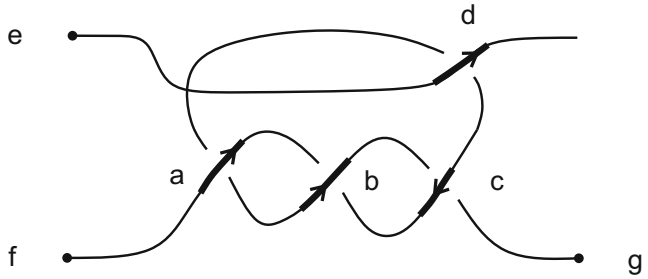


Figure 2. A sample computation. Determining colors for input registers In_1 and In_2 uniquely and instantaneously determines the color for the output register Out.

Remark 1. *More generally, we may allow the result of the computation to be the (possibly empty) set of all possible colors of output registers. This level of generality is not required in this paper.*

A tangle machine computation has the following features:

1. Whereas the alphabet of a classical Turing machine is discrete (usually just zero and one and maybe two), the alphabet Q of a tangle machine can be any set, discrete or not. Two values of Q may be chosen to represent zero and one, while the rest may represent something else, perhaps electric signals.
2. Whereas a Turing machine computation is sequential with each step depending only on the state of the read/write head and on the scanned signal, a tangle machine computation is instantaneous and is dictated by an oracle. Time plays no role in a tangle machine computation.
3. A tangle machine computation may or may not be deterministic (colors may represent random variables and not their realizations), and it may or may not be bounded (contain a bounded number of interactions). Quandles and tangle machines are flexible enough to admit several different interpretations. In this paper, we use tangle machines to realize both logic gates (deterministic, composing perhaps unbounded computations) and interactive proof computations (probabilistic, bounded size).
4. A tangle machine is flexible. There is a natural and intuitive set of local moves relating bisimilar tangle machine computations (Section 10).
5. A tangle machine representation is abstract. A tangle machine computation takes place on the level of information itself, with no reference to time. The axioms of a quandle have intrinsic interpretation in terms of the preservation and non-redundancy of information.

1.2. Results

The purpose of this note is to show the following theorems:

Theorem 1. *Any binary Boolean function can be realized by a tangle machine computation.*

This statement is not new: a previous such realization (not with tangle machines, but with colored braids) is recalled in Section 4.3.

By Turing completeness of the Boolean circuit model, we have the following:

Corollary 2. *Tangle machines (with an unbounded number of interactions) are Turing complete.*

In Section 5, we further prove the following.

Theorem 3. *Any Turing machine can be simulated by a tangle machine. Such a tangle machine is colored by a quandle (Q, B) whose set of binary operations B has cardinality $\mathcal{O}(n)$ where n is the number of states in the finite control of the underlying Turing machine.*

Tangle machines, whose notion of computation is based on an oracle, which produces output colors from input colors, can, in fact, perform super-Turing computations. See Remark 4.

Colors of registers in tangle machines evolve at interactions. If we bound the number of interactions, tangle machine computations include computations in a complexity class, which we call TangIP, which includes inside it a class that we call BraidIP. Letting χ denote the number of interactions in the machine, letting δ denote a "noise parameter" and letting c and s denote completeness and soundness correspondingly (see Section 2.3), we have the following:

Theorem 4. IP $\subseteq$ BraidIP $\{\delta, \chi\}$ *where:*

$$I(c\delta) < \chi < \frac{1}{I(1-s\delta)}, \tag{2}$$

with $I(p) \stackrel{\text{def}}{=} -p^{-1}\log p$. *The growth rate of* χ *is* $\mathcal{O}(\frac{1}{\delta})$ *as* $\delta \to 0$.

Thus, tangle machine computations with bounded interactions can decide any language in class IP, which is known to equal PSPACE [5].

Moreover, in the special setting of non-adaptive three-bit probabilistically-checkable proofs (PCP), there exists a tangle machine that achieves a better soundness parameter than the best known classical single-verifier non-adaptive three-bit PCP algorithm (Section 9.1). Yet, the soundness parameter for this tangle machine is above the conjectured lower limit. This suggests the possibility that tangle machines behave like very good single verifiers.

Finally, in Section 10, we discuss the equivalence of machines. Machine equivalence formally parallels the equivalence of tangled objects in low dimensional topology, and it gives us a formalism with which to discuss bisimulation. After justifying the definition, we introduce a notion of zero knowledge for machines, in which the proof is kept secure from untrusted verifiers at intermediate nodes.

Tangle machines are thus revealed to be a flexible model for computation.

1.3. Other Low Dimensional Topological Approaches to Computation

The first person to consider distributivity as an axiom of primary importance and to suggest diagrammatic calculi for logic (and perhaps by extension to computation) was American philosopher Charles Saunders Peirce. The following Peirce quotation was pointed out by Elhamdadi [6]: "These are other cases of the distributive principle... These formulae, which have hitherto escaped notice, are not without interest." [7]

The idea to use diagrammatic calculi from low-dimensional topology to model computation was pioneered by Louis Kauffman, who used knot and tangle diagrams to study automata [8], nonstandard set theory and lambda calculus [9,10]. There is also a diagrammatic π calculus formulation of virtual tangles [11]. The diagrammatic calculus of braids (braids are a special class of tangles) also underlies topological quantum computing; see e.g., [12,13]. Universal logic gates (Toffoli gates) have been realized using colored braids [14–16]. This has led to a proposal for circuit obfuscation, masking the true purpose of a circuit, using braid equivalence [17]. Buliga has suggested to represent computations using a calculus of colored tangles, which is different from ours [18]. In another direction, a different diagrammatic calculus, originating in higher category theory, has been used in the theory of quantum information; see e.g., [19–21].

In our approach, the tangle diagrams themselves are computers, representing a flowchart of information during a computation whose basic operations are distributive (compare [22]). This is not a diagrammatic lambda calculus or pi calculus, but rather it is a natively low dimensional topological approach to computation. In this note, we relate this approach to other approaches by showing that tangle machine computation is Turing complete and in the bounded resource setting that it can decide any language in the complexity class IP.

1.4. Contents of This Paper

We begin in Section 2 by recalling relevant models of computation, such as Turing machines, IP and PCP. Then, in Section 3, we recall the formalism of tangle machines [1]. Our definition is more general than the one used in that paper. Next, in Section 4, we show that tangle machines are Turing complete, and we show in Section 5 how tangle machines may simulate Turing machines. Restricting to a bounded resources setting, we construct networks of deformed IP verifiers in Section 6, defining a complexity class BraidIP. In Section 7, we show that IP $\subseteq$ BraidIP. Section 8 shows how to make our network computations more efficient by getting rid of the global time axis, making use of a machine we call the Hopf–Chernoff machine. Restricting further to PCP proofs, in Section 9, we show that the Hopf–Chernoff machine gives us perfect completeness and a better soundness parameter than the best-known non-adaptive three-bit PCP verifier. Finally, we define the equivalence of machines in Section 10, where we also discuss the tangle machine analogue of a zero knowledge proof.

2. Models of Computation

In this section, mainly to fix terminology and notation, we recall the notion of a Turing machine (Section 2.1), of decidable languages (Section 2.2), of an interactive proof (Section 2.3) and of a probabilistically-checkable proof (Section 2.4).

2.1. Turing Machines

The theory of computation and complexity theory are based on the notion of a Turing machine [23,24]. We recall its definition, following [25].

Definition 5. *A Turing machine is a triple* $(\Sigma, \mathcal{S}, \delta)$ *where:*

- Σ *is a finite set of symbols called the alphabet, which contains a "blank" symbol.*
- $\mathcal{S}$ *is a finite set of "machine states" with* $q_0 \in \mathcal{S}$ *and* $q_h \in \mathcal{S}$ *being, respectively, the initial and final (halting) states.*
- $\delta\colon\ \mathcal{S} \times \Sigma \longrightarrow \mathcal{S} \times \Sigma \times \epsilon$ *is a transition function.*

The set $\epsilon = \{0, 1, 2\}$ indicates the movement of a tape (left, stationary, right) or, equivalently, indicates the movement of a reading/writing (R/W) head following a writing operation. For convenience and without loss of generality, we limit the alphabet to three colors, $\Sigma = \{0, 1, 2\}$, where 2 represents the blank symbol.

A Turing machine is composed of two primary units. A finite control unit remembers the current state and determines the next state based on the current reading from a memory unit. The memory unit records symbols on a finite, possibly unbounded tape. Reading and writing operations retrieve or modify a symbol in the current position of the R/W head along the tape.

2.2. Computable Functions and Decidable Languages

Computable functions are the basic objects of study in computability theory. In the context of Turing machines, a partial function $f\colon\ \Sigma^k \to \Sigma$ is computable if there exists a Turing machine that terminate on the input x (input means tape content) with the value $f(x)$ stored on the memory tape if $f(x)$ is defined and which never terminates on input x if $f(x)$ is undefined.

A related notion is the notion of a decidable language. A set $L \subseteq \{0,1\}^*$, called a language, is said to be decided by Turing machine M if there exists a computable function $f\colon\ \Sigma^k \to \Sigma$ satisfying $f(x) = 1$ if $x \in L$ and $f(x) = 0$ if $x \notin L$ for all $x \in \{0,1\}^*$. A language is decidable if it is decided by some Turing machine.

2.3. Interactive Proof

The interactive proof model of computation involves bounded resources, by which we mean that computations are constrained to make use of only a finite number of steps, polynomial in the length $|x|$ of the word $x \in \{0,1\}^*$ [26]. Again, the goal is to determine whether $x \in L$ or $x \notin L$ for a language L. A verifier V interrogates a prover P who claims to have a proof that $x \in L$. Both the prover and the verifier are assumed to be honest, and queries are assumed to be independent. We are given two parameters, completeness c and soundness s, with $c, s \in [0,1]$. For the classical setting of **IP**, we set $c = 2/3$ and $s = 1/3$. The verifier V believes that $x \in L$ at time t with probability V_t. This belief is updated each time P responds to a query, beginning from $V_0 = 0$. We say that the statement $x \in L$ is decided at time t if:

$$\begin{aligned} &\text{(Completeness)} \quad x \in L \ \longrightarrow\ \Pr(V_t = 1) \geq c; \\ &\text{(Soundness)} \qquad x \notin L \ \longrightarrow\ \Pr(V_t = 1) \leq s. \end{aligned} \tag{3}$$

The class IP (interactive polynomial time) consists of those languages L that are decidable in time χ polynomial in $|x|$. A celebrated result in complexity theory states that IP equals PSPACE, the class of problems solvable by a Turing machine in polynomial space [5].

The class IP can be expanded to the class MIP in which the verifier has access to not one, but many provers, which can be interrogated independently [27]. It has been shown that MIP equals the large class NEXPTIME [28].

2.4. Probabilistically-Checkable Proofs

The class $\mathrm{PCP}_{c,s}(r(|x|), q(|x|))$ is a restriction of IP in which the verifier is a polynomial-time Turing machine with access to $\mathcal{O}(r(|x|))$ uniformly random bits and the ability to query only $\mathcal{O}(q(|x|))$ bits of the proof, with completeness c and soundness s [29,30]. The celebrated PCP theorem states that $\mathrm{PCP}[\mathcal{O}(\log|x|), \mathcal{O}(1)] = \mathrm{NP}$ [31]. For PCP, the prover is thought of as an oracle. We restrict to the

case of three-bit PCP verifiers, *i.e.*, to those for which $q(|x|) = 3$, and to those that are non-adaptive, *i.e.*, for which verifier queries are independent of previous responses by the oracle.

The ideal PCP verifier would have $c = 1$ with minimal soundness s. A simple and good three-bit PCP verifier with $s = \frac{3}{4} + \sigma \approx 0.75$ for arbitrarily small $\sigma > 0$ was designed by Håstad [32]. The current record is $s = \frac{20}{27} + \sigma \approx 0.741$ for an arbitrarily small $\sigma > 0$ [33]. It is conjectured that the lowest possible value of s is $\frac{5}{8} = 0.625$ [34].

3. Tangle Machines

In this section, we define tangle machines, first ignoring colors (Section 3.1), and then, after defining the sets, we color with (Section 3.2), then with colors (Section 3.3), where we also define tangle machine computations.

3.1. Without Colors

We define here a tangle machine, without colors, to be a diagram that occurs by concatenating (connecting endpoints) of a finite number of generators of the form in Figure 3. Thickened lines in interactions (Figure 3b) are called agents, and each thin line is called a patient. Only agents are directed, and their is no compatibility condition between the directions of different agents.

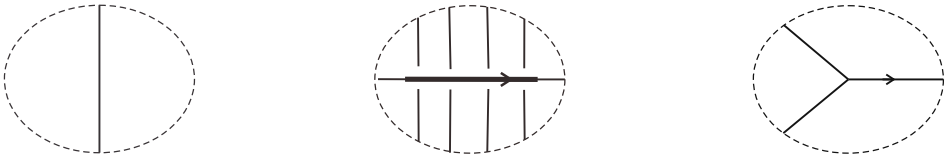


Figure 3. Generators for tangle machines are struts and interactions. Generators for trivalent tangle machines are struts, interactions and wyes. (**a**) strut; (**b**) interaction; (**c**) wye.

Arcs in a tangle machine, ending as under-arcs passing under an agent or at a wye or at a machine endpoint (thus "continuing right through agents"), are called registers. Later on, registers will contain colors.

Tangle machines have struts and interactions as generators, whereas trivalent tangle machines have struts, interactions and wyes as generators. An example of a machine constructed by concatenation is presented in Figure 4. As in the theories of virtual knots and of w-knotted objects, concatenation lines may intersect [35,36]. Furthermore, as in the theory of disoriented tangles, no compatibility condition is imposed for directions of concatenated agents [37].

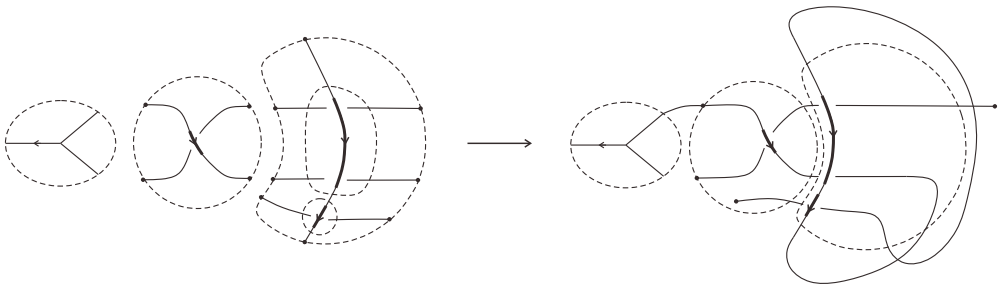


Figure 4. Concatenation of tangle machines.

3.2. Colors

Let Q be a set equipped with a family B of binary operations that satisfy the following three axioms:

Idempotence: $x \triangleright x = x$ for all $x \in Q$ and for all $\triangleright \in B$.

Reversibility: The map ${\triangleright}y\colon\ Q \to Q$, which maps each color $x \in Q$ to a corresponding color $x \triangleright y \in Q$, is a bijection for all $(y, \triangleright) \in (Q, B)$. In particular, if $x \triangleright y = z \triangleright y$ for some $x, y, z \in Q$ and for some $\triangleright \in B$, then $x = z$.

Distributivity: For all $x, y, z \in Q$ and for all $\triangleright \in B$:

$$(x \triangleright y) \triangleright z = (x \triangleright z) \triangleright (y \triangleright z) \ . \tag{4}$$

Remark 2. *Reversibility may be weakened by requiring only that* ${\triangleright}y$ *be an injection for all* y*, but not necessarily a surjection. This is indeed the case for the machines that we describe in the context of interactive proofs.*

The pair (Q, B) is called a quagma. It is called a quandle if the operations in B distribute over one another, in the sense that:

$$(x \triangleright y) \blacktriangleright z = (x \blacktriangleright z) \triangleright (y \blacktriangleright z) \ , \tag{5}$$

for all $\triangleright, \blacktriangleright \in B$. This definition is a variant of definitions found in [38–40]. If we weaken reversibility to require only that ${\triangleright}y$ be an injection for all $y \in Q$ and for all $\triangleright \in B$, the resulting structure is called a quandloid [4]. Quagmas, quandloids and quandles serve as the content of registers in tangle machines.

Example 1 (Conjugation quandle)**.** *Colors might be elements of a group* Γ*, and the operation might be conjugation:*

$$x \blacktriangleright y = y^{-1}xy \ . \tag{6}$$

The pair $(\Gamma, \{\blacktriangleright\})$ *is called a conjugation quandle. Such quandles feature in knot theory, e.g., [41].*

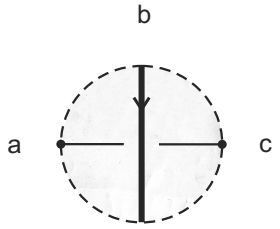
Example 2 (Linear quandle)**.** *Colors might be real numbers, and the operations might be convex combinations:*

$$x \triangleright_s y = (1 - s)x + sy \qquad s \in \mathbb{R} \setminus \{1\} \ . \tag{7}$$

The pair $\left(Q, \{\triangleright_s\}_{s \in \mathbb{R}}\right)$ *is called a linear quandle.*

3.3. With Colors

A coloring of a tangle machine M is an assignment ϱ of a binary operation in B to each agent and an assignment ρ of an element of Q to each register, such that, at each interaction, the color $z \in Q$ of the patient to the right of the agent (according to the right-hand rule) equals the color of its corresponding patient to the left of the agent $x \in Q$ right-acted on by the color of the agent $y \in Q$ via the operation of the agent $\triangleright \in B$:



(8)

Thus, an interaction "realizes" the action of the quagma or of the quandle.

To color a trivalent tangle machines, Q has to come equipped with a complete ordering. We include also an assignment of either $\max$ or $\min$ to each wye, so that the exiting register of the wye is the maximum of the two inputs if the wye is labeled $\max$ and is their minimum otherwise. We graphically denote a $\min$ label with a white circle at the branch-point of the wye and a $\max$ label by a black circle. See Figure 5.

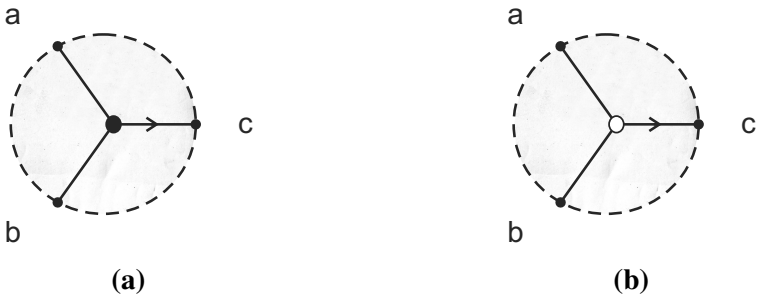


Figure 5. Coloring wyes colored by $\max$ and $\min$.

Remark 3. *The notion of the tangle machine given above is more general than in [1,4], in which all tangle machines are quandle-colored and without wyes.*

Finally, we come to the notion of a tangle machine computation.

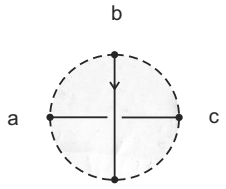
Definition 6. *A computation of a (trivalent) tangle machine M is:*

1. *A choice of a set S_{in} of input registers in M.*
2. *A choice of a set S_{out} of output registers in M with $S_{in} \cap S_{out} = \emptyset$.*
3. *A coloring ϱ of all agents in M (and an assignment of either $\max$ or $\min$ to each wye).*
4. *A coloring ρ_{in} of all registers in S_{in}.*
5. *A unique (oracle) determination of a coloring ρ_{out} of all registers in S_{out}. If ρ_{in} does not uniquely determine ρ_{out}, the computation cannot take place.*

4. Tangle Machines Are Turing Complete

Our goal in this section is to realize the universal set of gates, $\mathrm{NOT}(\neg)$ and $\mathrm{AND}(\wedge)\}$, as well as a multiplexer that duplicates the content of a register, using tangle machines. This realizes the Boolean circuit model, thus showing that tangle machines are Turing complete.

Remark 4. *A tangle machine computation, which is based on an oracle, which tells us the colors of output registers given colors of input registers, can carry out super-Turing computations. Consider the conjugation quandle of a group* Γ *with unsolvable conjugacy problem and color agents* In_1 *and* In_2 *by elements of* Γ. *The color of* Out *might not be Turing computable, but the tangle machine computation computes it.*



(9)

4.1. Quagma Approach

Choose the set of colors to be $Q \stackrel{\text{def}}{=} \mathbb{Q}^{2\times 2}$, equipped with a set B of binary operations whose elements are the following:

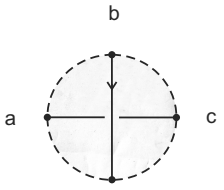
$$X \blacktriangleright Y = \begin{cases} Y^{-1}XY, & \text{if } \det(Y) \neq 0; \\ X, & \text{otherwise.} \end{cases} \tag{10}$$

$$X \triangleright_s Y = (1-s)X + sY, \quad \text{for } s = \frac{1}{2} \text{ or } s = 2. \tag{11}$$

The structure $(Q, \{\triangleright_{0.5}, \triangleright_2, \blacktriangleright\})$, henceforth referred to simply as Q, is a quagma.

To realize Boolean logic, let $A_0 \stackrel{\text{def}}{=} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$ and $A_1 \stackrel{\text{def}}{=} \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$ stand in for the digits zero and one, correspondingly. Incidentally, these happen to coincide with the Pauli spin matrices of quantum mechanics.

A NOT gate is described by a single interaction:



X	$\neg X$
A_0	A_1
A_1	A_0

(12)

where the respective truth-table is shown to the right of the diagram. Explicitly,

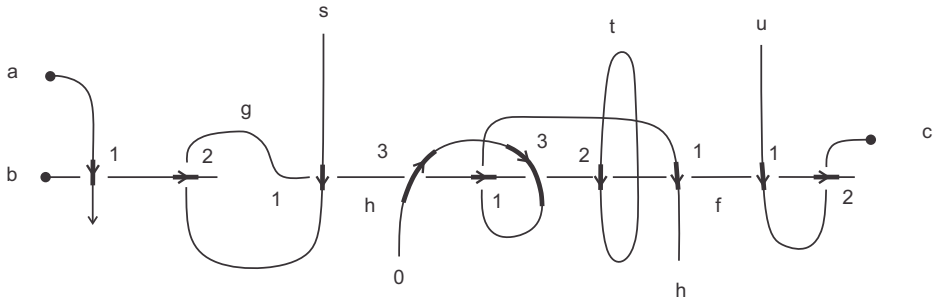
$$\neg X = X \blacktriangleright (A_0 + A_1) = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}^{-1} X \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} = \frac{1}{2}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} X \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{13}$$

The input is the register labeled X; the output is the register labeled $\neg X$; and the remaining register is always colored by the constant value $A_0 + A_1$.

Realizing an AND gate can be split into several successive computations, let X and Y be the inputs to the gate, and write $\mathbf{0}$ for $\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$. The following instructions end up with the desired operation $X \wedge Y$.

1. $\beta_1 = (A_0 + A_1) \blacktriangleright (X \triangleright_{0.5} Y) = (X+Y)^{-1}(A_0 + A_1)(X+Y)$
2. $\beta_2 = \beta_1 \triangleright_{0.5} (A_0 + A_1) = \frac{1}{2}\beta_1 + \frac{1}{2}(A_0 + A_1)$
3. $\beta_3 = (\beta_2 \triangleright A_0) \triangleright_{0.5} \beta_2 = \frac{1}{2}A_0\beta_2 A_0 + \frac{1}{2}\beta_2$
4. $X \wedge Y = A_1 \blacktriangleright (\beta_3 \triangleright_{0.5} A_1) = (\beta_3 + A_1)^{-1} A_1 (\beta_3 + A_1)$

A tangle machine that realizes these steps is given below together with the respective truth-table.



(14)

X	Y	β_1	β_2	β_3	$X \wedge Y$
A_0	A_0	$A_0 - A_1$	A_0	A_0	A_0
A_0	A_1	$A_0 + A_1$	$A_0 + A_1$	A_0	A_0
A_1	A_0	$A_0 + A_1$	$A_0 + A_1$	A_0	A_0
A_1	A_1	$A_1 - A_0$	A_1	$\mathbf{0}$	A_1

(15)

In addition to the universal set of gates, we also need to be able to duplicate the content of a register. The conventional Boolean circuit model includes junction points along wires. The tangle machine analogue for such a junction is a multiplexer, that is a machine whose output colors are duplicates of the color in one of its inputs. A multiplexer takes an input of the form $\{X, \underbrace{0, \ldots, 0}_{n-1 \text{ times}}\}$ and outputs $\underbrace{\{X, \ldots, X\}}_{n \text{ times}}$. The operation of a multiplexer is captured by the machine in Figure 6.

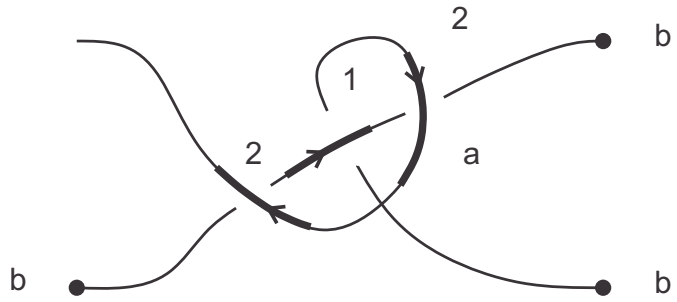


Figure 6. A multiplexer.

Remark 5. *Two fundamental properties of quantum information are no-cloning and no-deleting. The uniform version of no-cloning states that there does not exist a unitary operator C for which*

$C(A \otimes e)C^\dagger = A \otimes A$ for all states A, where e denotes the identity operator. The uniform version of no-deleting states that there does not exist a unitary operator U for which $U(A \otimes A)U^\dagger = A \otimes e$ for all states A. Both of these statements are captured by the quagma axioms. Consider the quagma $(Q, \{\triangleright_{0.5}, \blacktriangleright\})$ where Q is the set of invertible operators and $\blacktriangleright$ is any binary operation that distributes over $\triangleright_{0.5}$, e.g., conjugation. If U were a universal cloning operator with respect to $\blacktriangleright$, i.e., if $(A \otimes e) \blacktriangleright U = A \otimes A$ for all A, then for any state A, both machines in Figure 7 would carry out the same computation and, in particular, $\mathrm{Out}_1 = \mathrm{Out}'_1$. However, then, $(A \triangleright_{0.5} B) \otimes (A \triangleright_{0.5} B) = \mathrm{Out}_1 = \mathrm{Out}'_1 = (A \otimes A) \triangleright_{0.5} (B \otimes B)$, which is false in general. Thus, universal cloning violates distributivity. No-deleting follows from no-cloning, because if U were a universal deleting operator with respect to $\blacktriangleright$, then U would also be a universal cloning operator with respect its inverse operation $\blacktriangleleft$, which exists thanks to reversibility. This would violate distributivity, because $\blacktriangleleft$ also distributes over $\triangleright_{0.5}$, as can be seen by applying $\blacktriangleleft Z$ to both sides of the equation:

$$X \triangleright_{0.5} Y = \Big((X \blacktriangleleft Z) \triangleright_{0.5} (Y \blacktriangleleft Z)\Big) \blacktriangleright Z \ .$$

We parenthetically note that no-cloning and no-deleting are also captured by a different diagrammatic calculus, that of categorical quantum mechanics [42].

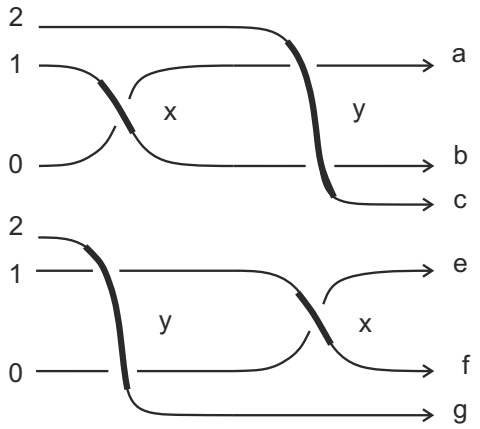


Figure 7. Universal cloning violates distributivity.

4.2. Wye Approach

Realizing a universal set of logic gates is easier if we allow wyes and may be realized with a quandle coloring.

We color our machines by elements of the quandle $Q \stackrel{\text{def}}{=} \{0, 1, 2\}$ subject to the quandle operation $x \triangleright y = 2y - x \bmod 3$ (this is a Fox three-coloring). We totally order Q by $0 < 1 < 2$. Color-code 0 as red, 1 as blue and 2 as green. For this particular quandle the direction of the agent does not matter because the operation $\triangleright$ is its own inverse. Let 0 stand in for the digit zero and 1 stand in for the digit one. A universal set of logic gates and a multiplexer can be obtained as in Figure 8, where an incoming arrow represents input and an outgoing arrow represents output.

4.3. Nonabelian Simple Group Approach

The AND gate constructed in Section 4.1 was realized as a tangle machine colored by a quagma, which is not a quandle, and the AND gate of Section 4.2 used a wye. In fact, it is possible to realize a universal set of logic gates with a conjugation quandle colored tangle machine without using wyes.

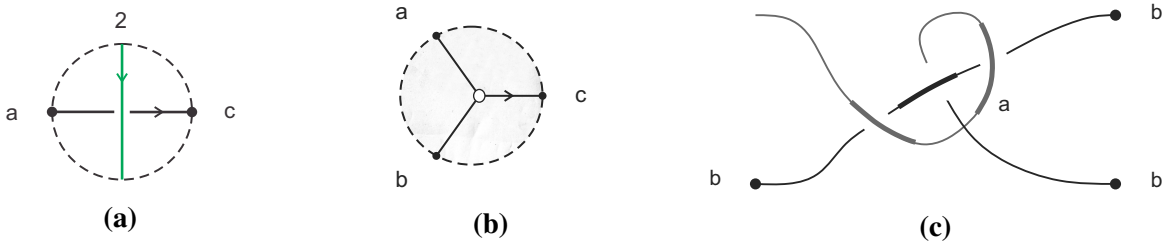


Figure 8. Logic gates for the three-color approach. Inputs are the colors on the left, and outputs are colors on the right. (**a**) NOT gate; (**b**) AND gate; (**c**) multiplexer.

The construction is based on Barrington's theorem [43,44]. Following [16], Appendix A of [17] constructs a 132 crossing 14 strand braid colored by the conjugation quandle of the finite simple group A_5, which realizes the Toffoli gate that is a universal logic gate [45]. This braid is made a tangle machine by interpreting each crossing as an interaction. Three of its registers are inputs; three are outputs; and it contains a number of ancilla, which are what we called control registers.

5. Turing Machine Simulation

In this section, we show how to simulate a Turing machine using a tangle machine.

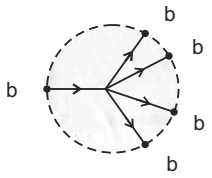
5.1. Turing Tangle Machine

In this section, we make use of the linear quandle whose underlying set of elements Q is the set of the rational numbers and whose set of operations B is:

$$x \triangleright_s y = (1-s)x + sy \qquad s \in \mathbb{Q} \setminus \{1\} \quad . \tag{16}$$

There are several sub-machines that recur in the construction of a Turing tangle machine, the tangle machine analog of a Turing machine. To simplify our diagrams, we represent these sub-machines graphically.

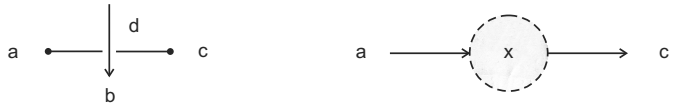
Multiplexer We indicate a multiplexer by splitting a strand.



(17)

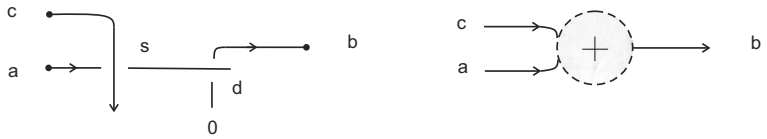
The multiplexer is realized by composing diagrams of the form in Figure 6.

Negation $\neg x \stackrel{\text{def}}{=} 1 - x$.



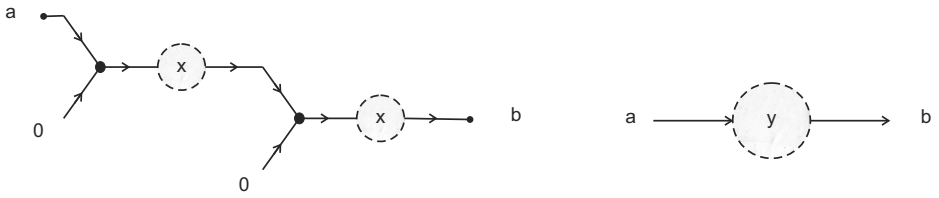
(18)

Addition



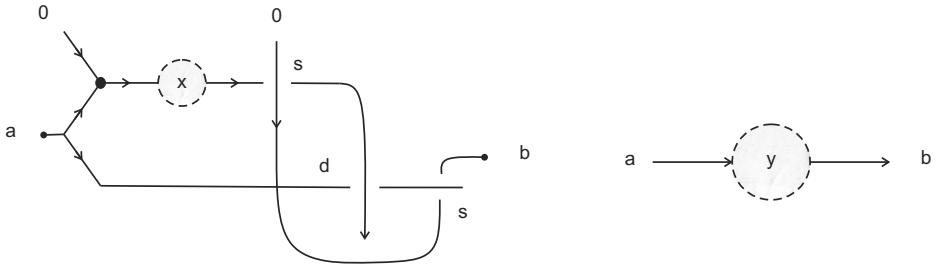
(19)

Indicator If $x \geq 1$ then$\imath(x) = 1$, if $0 < x < 1$, then $\imath(x) = x$, otherwise $\imath(x) = 0$.



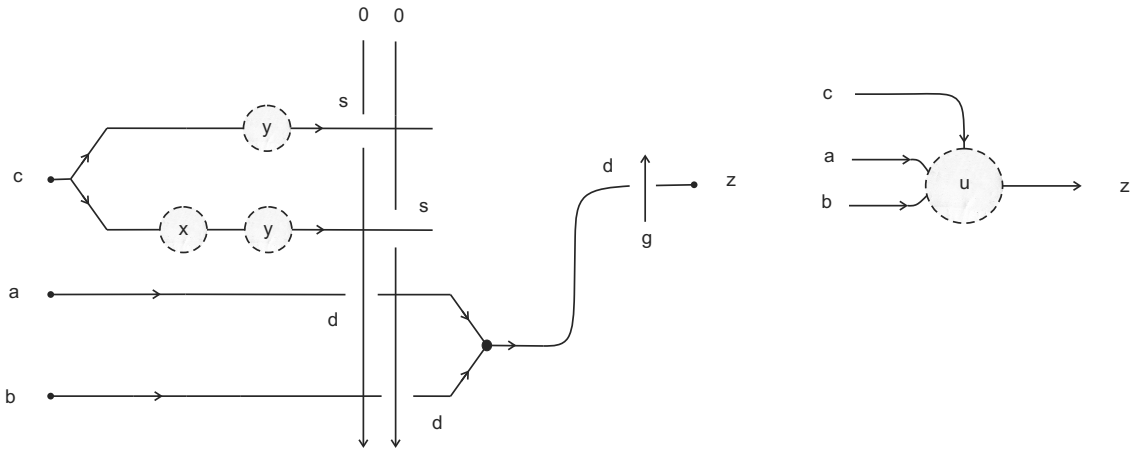
(20)

Beta A function $\beta(x)$, which satisfies $\beta(0) = -1$, $\beta(\frac{1}{2}) = 0$ and $\beta(1) = 1$.



(21)

Selector Depending on the color c of a control strand, either x or y emerges as the output, z. To be precise, $z = x$ if $c = 0$, and $z = y$ if $c = 1$.



(22)

where s is an arbitrary constant whose value is greater than two.

Mask-generating machine This sub-machine is shown in Figure 9. Its input registers are a register colored by an integer $p \in Q$, called a pointer, and a sequence of colored registers together, called a mask. Its output registers are a register colored p, and one register colored zero for all other input colored registers, except for a single register colored one in the p-th position of the output.

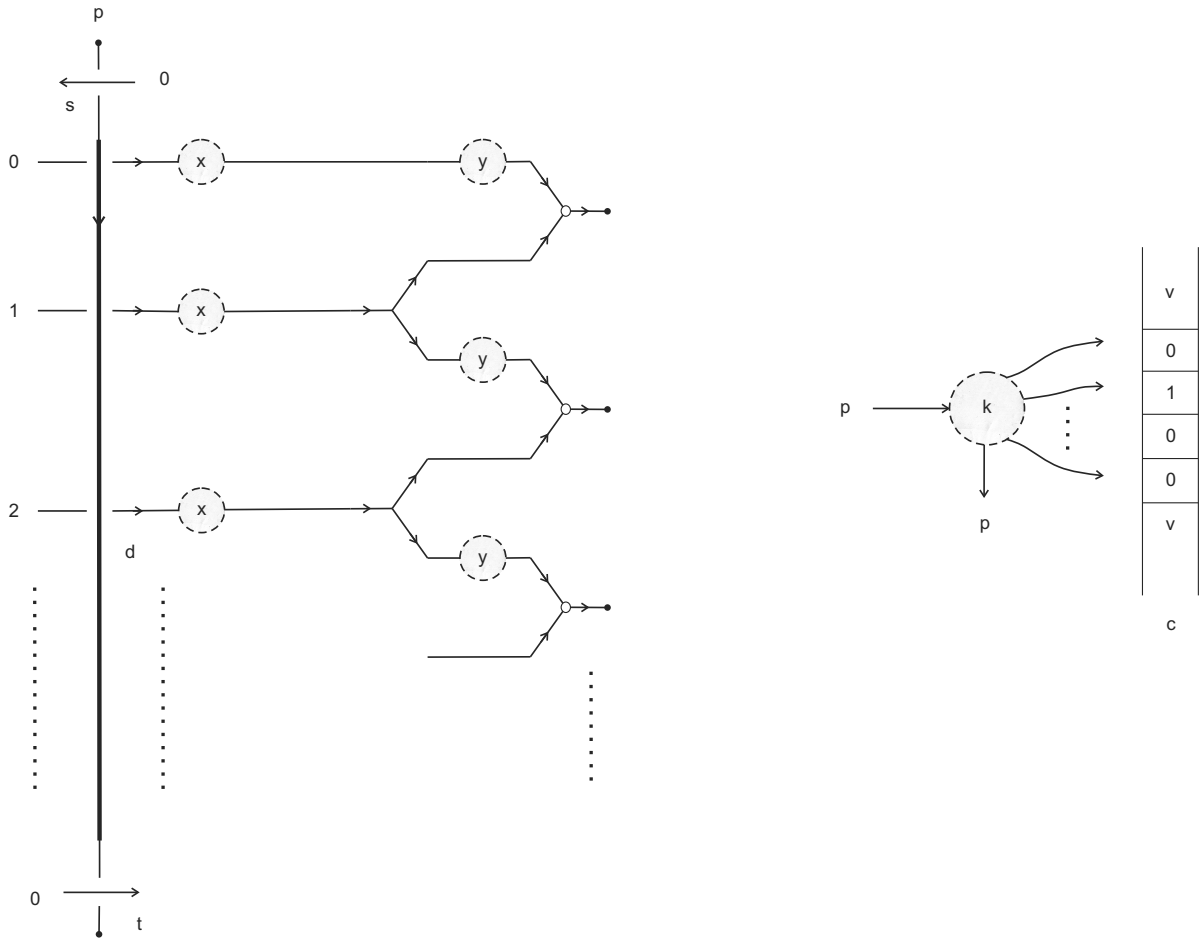


Figure 9. A mask-generating machine.

5.2. Finite Control

The first step in realizing a Turing machine is to mimic the finite control unit, *i.e.*, to simulate the transition function:

$$\delta(q(k), u(k)) = (q(k+1),\ a,\ \epsilon) \tag{23}$$

Given the current machine state $q(k)$ and the symbol currently under the R/W head $u(k)$, the transition function determines the next machine state $q(k+1)$ together with a pair of tape instructions: the symbol to be written a and an ϵ movement of the head to its next position along the tape. Without loss of generality, we shall assume henceforth that the finite control states are all natural numbers and, in particular, that $\mathcal{S} = \{1, \ldots, n\}$.

The basic building block of the finite control tangle machine is a hardwired transition, that is a function $\delta_{i,j} : \mathcal{S} \times \Sigma \to \mathbb{Z}^3$,

$$(q, u) \mapsto \begin{cases} (\bar{q}+2,\ \bar{a}+2,\ \bar{\epsilon}+2), & \text{if } q = i \text{ and } u = j+1; \\ (-\bar{q}-2,\ -\bar{a}-2,\ -\bar{\epsilon}-2), & \text{otherwise.} \end{cases} \tag{24}$$

where $\bar{q}, \bar{a}, \bar{\epsilon}$ denote, respectively, the next assumed state and the tape instructions as specified in the definition of $\delta_{i,j}$. The parameters i, j and the respective output of $\delta_{i,j}$, the triplet $(\bar{q}, \bar{a}, \bar{\epsilon})$, are hardwired into the tangle machine realization of $\delta_{i,j}$ through the quandle parameters. See Figure 10.

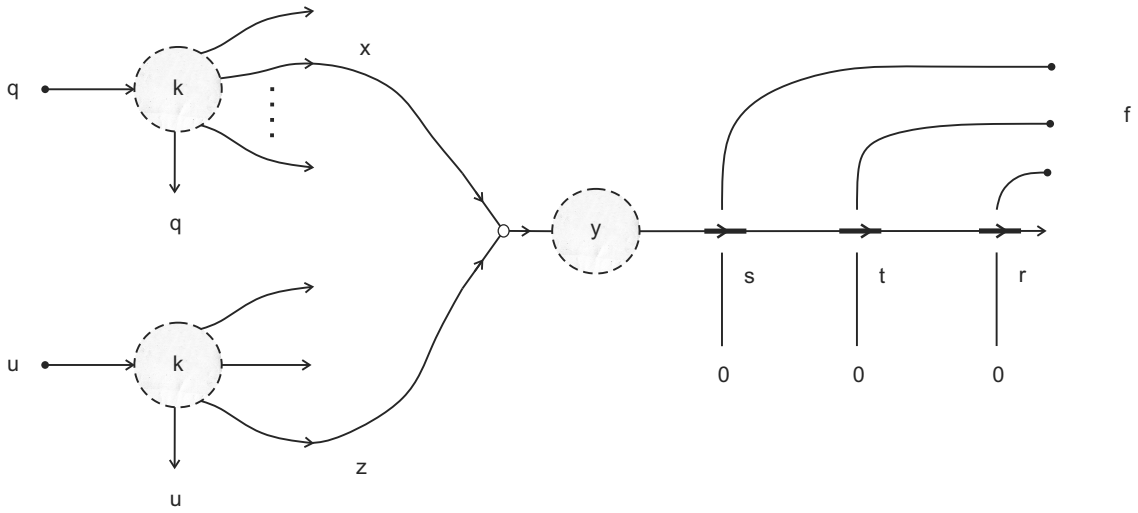
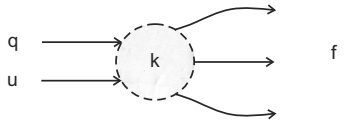


Figure 10. Hardwired transition.

The hardwired transition is symbolically represented as:



(25)

The transition function δ is constructed by combining several hardwired transitions. Note that there are no more than $3n$ possible transitions in the finite control (n states multiplied by three input symbols). Thus, the number of distinct binary operations in B does not exceed $9n + \mathcal{O}(1)$, which accounts for $3n$ triplets $(\triangleright_{\bar{q}+2}, \triangleright_{\bar{a}+2}, \triangleright_{\bar{\epsilon}+2})$ and a few other operations required for realizing the memory unit. A detailed construction of the finite control sub-machine is shown in Figure 11.

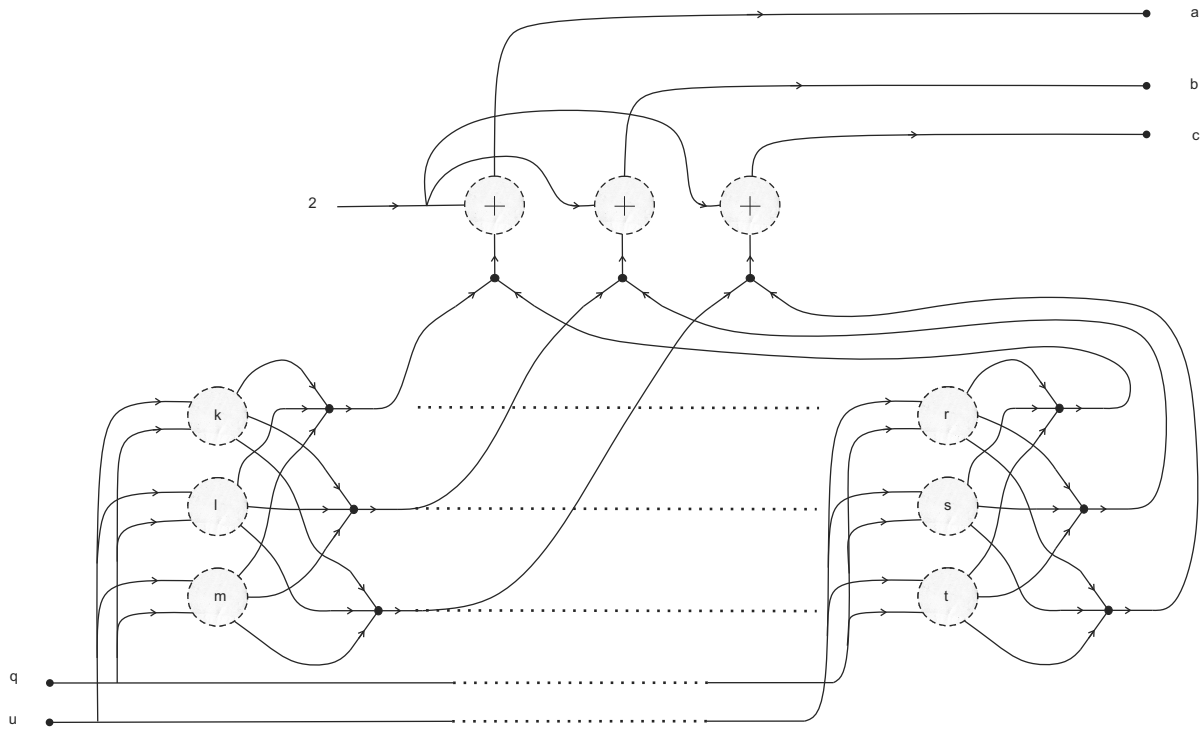


Figure 11. A tangle machine realization of a finite control unit.

5.3. Memory Unit and One-Step Computation

The memory unit consists mainly of the tape logic. It accepts a finite, possibly unbounded set of registers whose colors are manipulated in the basis of commands a and ϵ received from the finite control. The majority of the registers of the memory unit correspond to tape cells whose content is represented by colors from the set Σ.

We construct the memory (tape) reading and writing operations using a mask-generating machine together with selector machines to output the color of the $p(k)$-th register, either as it originally appeared or modified. See Figure 12.

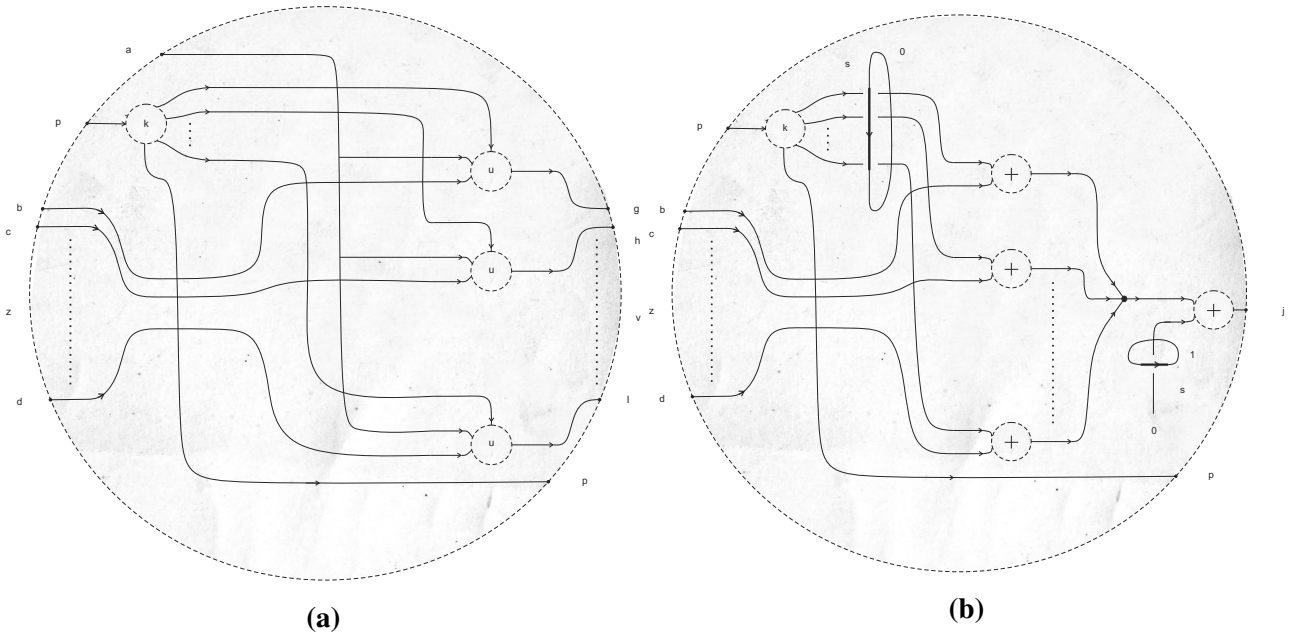


Figure 12. Memory reading and writing sub-machines. Here, $\frac{1}{2} < s < 1$ is an arbitrary parameter. (**a**) Write; (**b**) read.

Figure 13 illustrates the memory unit, connected to the finite control unit. Its inputs are:

1. An integer pointer register colored $p(k)$, indicating the current head position.
2. A finite, possibly unbounded set of registers, each of which represents a single (memory) cell on a tape. These tape registers are colored $\{c_i(k)\}_{i>0} \in \Sigma$.
3. A pair of registers colored correspondingly by a pair instructions (a, ϵ), where a denotes the symbol to be written in the current cell to which the head points (which is numbered $p(k)$), and ϵ denotes the (possibly zero) increment to be added to $p(k)$, so that $p(k+1) = p(k) + \epsilon - 1$.

The outputs of the memory unit are:

1. The updated pointer $p(k+1)$.
2. A finite, possibly unbounded set of registers whose colors $\{c_i(k+1)\}_{i>0} \in \Sigma$ have been all passed unchanged, except for a single cell whose content may have been modified.
3. A strand colored by $u(k+1)$, the content of the cell to which the head points in its new location $p(k+1)$

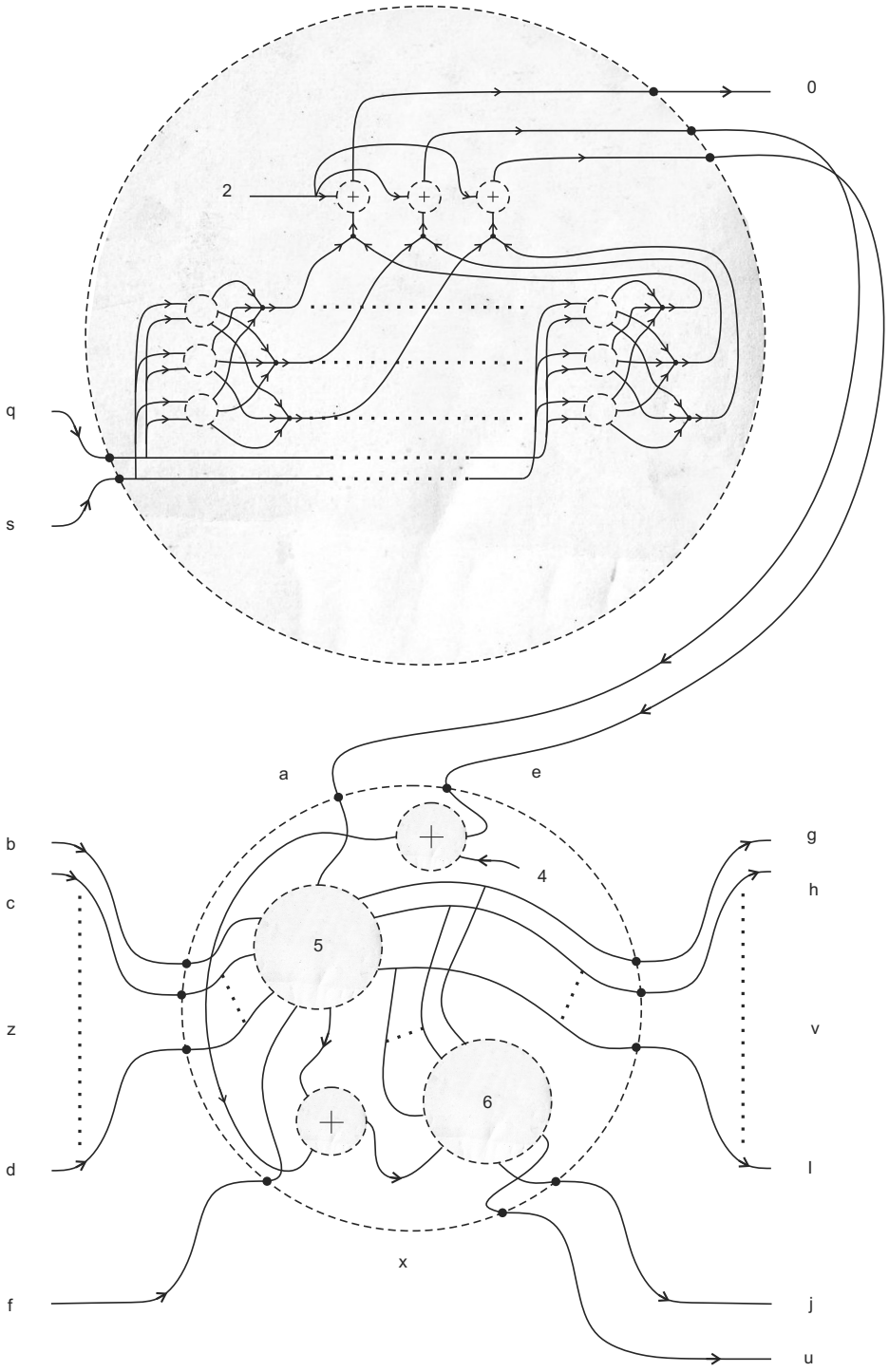


Figure 13. One-step computation of a Turing tangle.

To simulate the sequential operation of a Turing machine, copies of the finite control unit and of the memory unit are to concatenated in the obvious manner. Concatenating N copies of the machine in Figure 13 simulates N successive computations of a Turing machine (see Figure 14). This justifies naming such a procedure iteration.

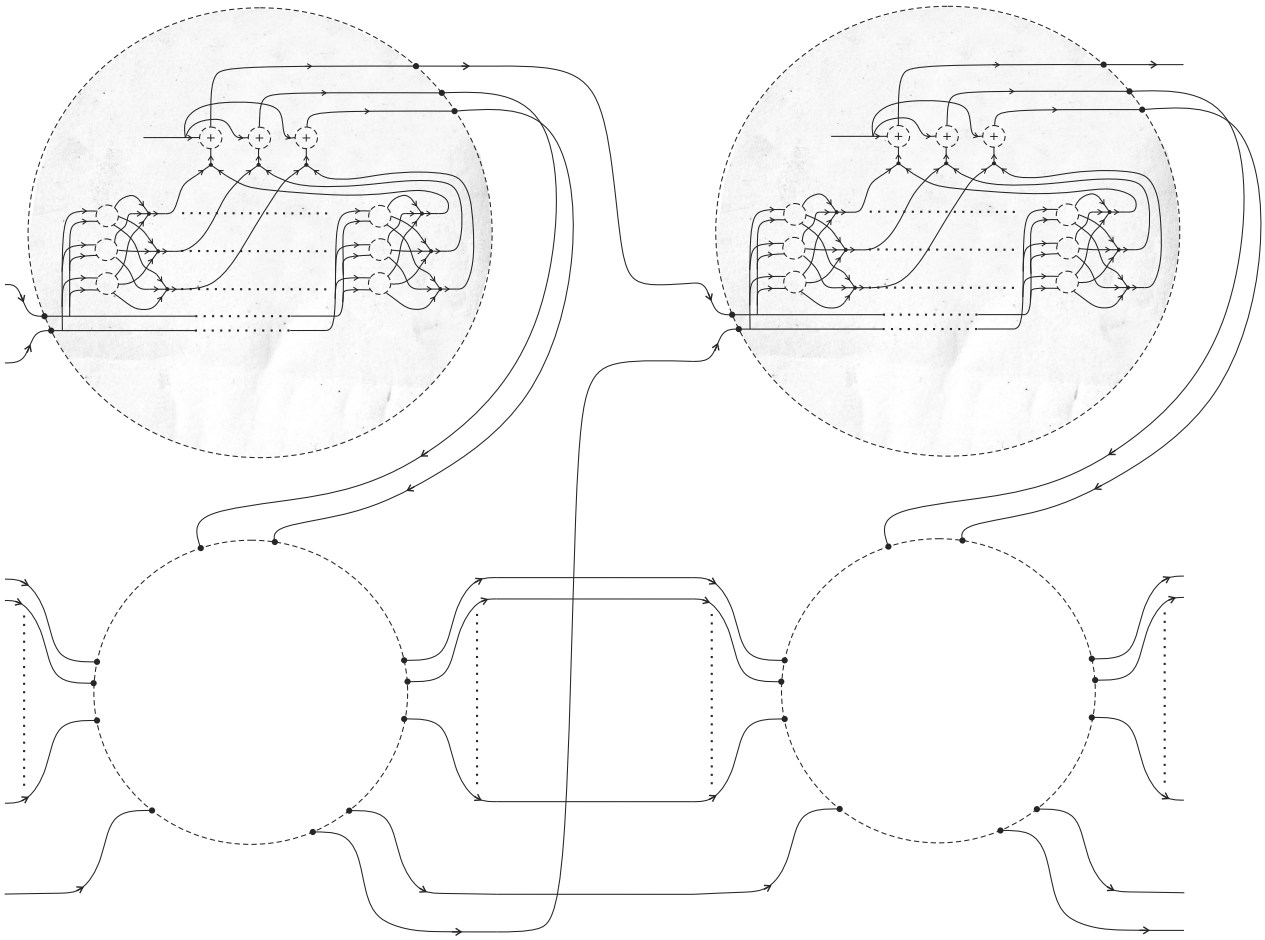


Figure 14. Iterative computation in a Turing tangle machine.

5.4. Halting

A halting state is a state for which:

$$\delta(q_h, u) = (q_h, u, 1) \tag{26}$$

Once arriving at a halting state, further iterations do not alter the memory content of the machine. The state q_h represents an equilibrium, which may or may not be reached for a given input sequence $u(0), u(1), \ldots$. A machine whose input registers are colored by a halting state can be closed by concatenating respective inputs and outputs, as shown in Figure 15.

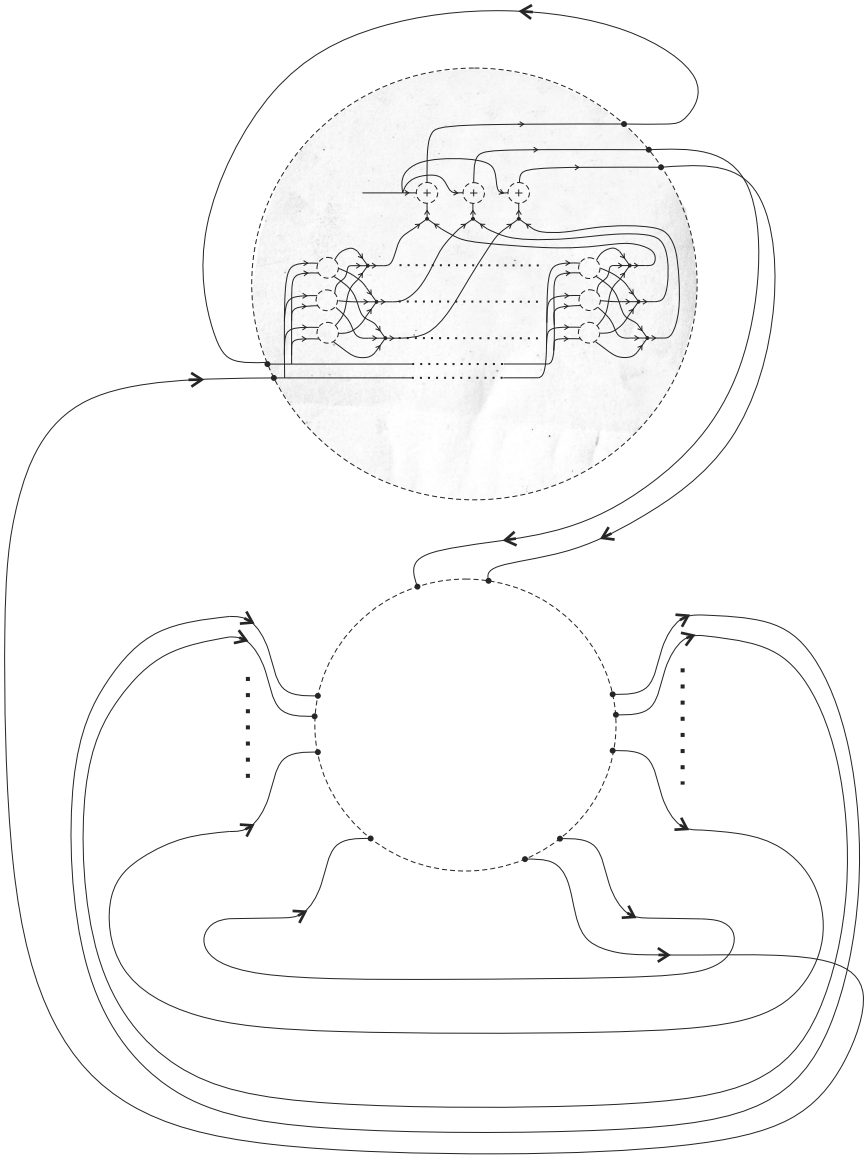


Figure 15. Closure of a Turing tangle machine in its halting state.

6. Interactive Proofs: Distribution of Knowledge by Deformation

The decision of a crowd can converge to a correct answer even when each individual has limited knowledge. This phenomenon is known as the wisdom of the crowds [46]. In line with "the many being smarter than the few", we extend the notion of interactive proof systems [26] to a system in which a collection of verifiers interact to prove a claim together. Might such a crowd of verifiers collaborate to prove more than could be proven by any individual verifier in the crowd?

6.1. Deformation of a Single Interaction

Consider a family of verifiers, each with a belief concerning whether $x \in L$ or $x \notin L$. We model the belief of each verifier W at time t as a Bernoulli random variable W_t whose realizations w_t are either

$|\text{True})$ or $|\text{False})$. We interpret $w_t = |\text{True})$ as "W believes at time t that $x \in L$", and we interpret $w_t = |\text{False})$ as "W believes at time t that $x \notin L$".

Consider an interaction at time t with agent V, one of whose patients is W. The realization w_{t+1} of W_{t+1} may equal either the belief of the agent v_t or the belief of the patient w_t. In other words, W either retains her belief or is "convinced" by V to change her belief to that of V (we use female pronouns for the verifiers, who are all "Alices"). Whether or not V "succeeds in convincing W" depends on a message ξ_t from a prover Π with access to an oracle.

Only the belief of patients changes at an interaction. The agents and the verifiers who do not participate in the interaction do not change their beliefs, so in particular, $v_{t+1} = v_t$ always.

There are three constants associated with the agent V at an interaction at time t: a completeness parameter c_V^t, a soundness parameter s_V^t with $0 < s_V^t < c_V^t \leq 1$, and a deformation parameter $\delta_V^t \in \mathbb{Q} \cap (0,1)$. For simplicity, we will assume that these three parameters are the same for all agents in the network, and s_v^t, c_V^t, δ_V^t will be written s, c, δ correspondingly.

Our basic requirement for an interaction is that the following pair of inequalities be satisfied:

$$\begin{aligned} &\text{(deformed completeness)} \quad x \in L \longrightarrow \Pr(W_{t+1} = v_t \mid V_t = v_t) \geq c\delta; \\ &\text{(deformed soundness)} \qquad x \notin L \longrightarrow \Pr(W_{t+1} = v_t \mid V_t = v_t) \leq s\delta. \end{aligned} \tag{27}$$

Remark 6. *The deformed completeness and soundness, $c\delta$ and $s\delta$, may both be below $\frac{1}{2}$ or may both be above $\frac{1}{2}$ and bounded away from one.*

In the limit $\delta \to 1$, specific values of c and of s turn the pair of inequalities (27) into familiar pairs of inequalities in interactive proof theory [47]. For example, for $s = 2^{-|x|^a}$ and $c = 1 - 2^{-|x|^b}$, where $a, b > 0$, we obtain the completeness and soundness constraints of an IP verification.

6.2. Statistics of Beliefs and Interactions

When keeping track of the beliefs of many different verifiers at many different times, it is cumbersome to work directly with Equation (27). Instead, we introduce a shorthand to keep track of the belief of a verifier, both if $x \in L$ and also if $x \notin L$, in a single expression.

The belief statistics $|W_t)$ of verifier W at time t is written:

$$|W_t) \stackrel{\text{def}}{=} a\,|\text{True}) + b\,|\text{False}) \quad , \tag{28}$$

where $a \in [0,1]$ denotes the greatest lower bound for the belief of W that $x \in L$ at time t conditioned on this belief indeed being true, and $b \in [0,1]$ denotes the greatest lower bound for the belief of W that $x \notin L$ at time t conditioned on this opposite belief indeed being true. Note that $a + b$ need not equal one. In particular:

$$\begin{aligned} x \in L \quad &\longrightarrow \quad a \leq \Pr(W_t = |\text{True})); \\ x \notin L \quad &\longrightarrow \quad b \leq \Pr(W_t = |\text{False})). \end{aligned} \tag{29}$$

An interaction between W and V at time t concludes either with W accepting the belief of V or with W sticking to her own belief. Denoting by h the probability (or more precisely, a lower bound on it) of

W switching to the belief of V in the next time frame, the distribution of possible beliefs of W in the next time frame is described by:

$$|W_{t+1}) = |W_t)^{|V_t)} \stackrel{\text{def}}{=} (1-h)\,|W_t) + h\,|V_t)\,. \tag{30}$$

Remark 7. *Belief statistics written as in Equation (28) facilitate calculations of probabilities across a network. This tool is used throughout the remainder of this note (some examples will be given shortly). Here, we explain how Equations (28) and (30) combine to give a compact way of representing two entirely different interactions, one assuming $x \in L$ and the other assuming $x \notin L$. This may be slightly confusing at first, so the reader's attention is called to this point.*

Owing to Equation (30), the probabilities anywhere in the network at time $t > 0$ depend on the parameter h. We may do all calculations and treat it as a formal parameter. Having in mind that an interaction is ultimately a procedure terminating with the statistics in Equation (27), the parameter h is set either to $c\delta$ or to $s\delta$ depending on whether or not x is in L. To verify that a network decides L, we repeat the computation twice, first for the case where $x \in L$ and then for $x \notin L$. In the former case, we will be interested only in the coefficient of $|\mathrm{True})$, whereas in the latter case, we will be interested only in the coefficient of $|\mathrm{False})$.

Here is an illustrative calculation for a single interaction. Let $W_t = |\mathrm{False})$ and $V_t = |\mathrm{True})$. Invoke Equation (30) using the completeness and soundness parameters in Equation (27): first using $h = c\delta$ and then using $h = s\delta$. Hence,

$$|W_{t+1}) = c\delta\,|\mathrm{True}) + (1-s\delta)\,|\mathrm{False}) \longrightarrow \begin{cases} c\delta\,|\mathrm{True}) + (\cdots)\,|\mathrm{False})\,, & x \in L; \\ (\cdots)\,|\mathrm{True}) + (1-s\delta)\,|\mathrm{False})\,, & x \notin L. \end{cases} \tag{31}$$

This interaction is said to decide L only if $c\delta > \frac{1}{2}$ and $1 - s\delta > \frac{1}{2}$.

6.3. Expressive Power of a Network: The Class BraidIP

Verifiers in our framework are assumed to be implemented as probabilistic polynomial-time Turing machines whose beliefs are either internal states or are stored on tapes. Similarly, an interaction is a polynomial-time procedure. Consider now a crowd of verifiers $V^1, V^2, \ldots, V^\mu$ whose initial beliefs at time $t = 0$ are $|V_0^1), |V_0^2), \ldots, |V_0^\mu)$. Allow them to interact at times $t = 0, 1, 2, \ldots, \chi$, subject to parameters $0 < c < s \leq 1$ and $\delta \in \mathbb{Q} \cap (0,1)$. We write M for this sequence of interactions. A language $L \subseteq \{0,1\}^*$ is said to be decided by M if M contains a verifier V, whose belief at time χ is $|V_\chi) \stackrel{\text{def}}{=} a\,|\mathrm{True}) + b\,|\mathrm{False})$, such that, for a fixed constant $\kappa > 0$, we have:

$$\begin{aligned} x \in L &\longrightarrow a \geq \tfrac{1}{2} + |x|^{-\kappa}\,; \\ x \notin L &\longrightarrow b \geq \tfrac{1}{2} + |x|^{-\kappa}\,. \end{aligned} \tag{32}$$

This definition depends on the choice of $\kappa > 0$. The class braided interactive polynomial time (BraidIP) consists of those languages that are decidable for any fixed $\kappa > 0$ by some network M in time χ, polynomial in $|x|$. We denote this class $\mathrm{BraidIP}\{\delta, \chi\}$, where χ is the number of interactions in M.

Remark 8. *The definition of class* BraidIP *is similar in spirit to the class* BPP. *We will see below that it includes class* IP. *Letting Equation (32) reflect the class* PP *(i.e., taking strict inequalities and right-hand constants equal to $\frac{1}{2}$) will result in networks that decide any $L \in$ IPP.*

6.4. Braid of Beliefs

Our networks admit a convenient diagrammatic description. We represent an interaction as a wire cutting through other wires (see Figure 16). The overcrossing wire, which becomes slightly thickened in an interaction, carries the belief statistics of an agent, whereas the undercrossing wires carry the belief statistics of her patients. An example of many concatenated interactions is given in Figure 17.

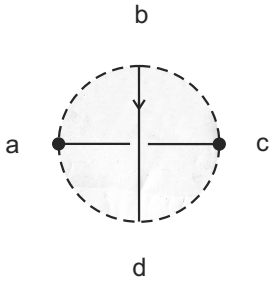


Figure 16. Diagrammatic representation of an interaction.

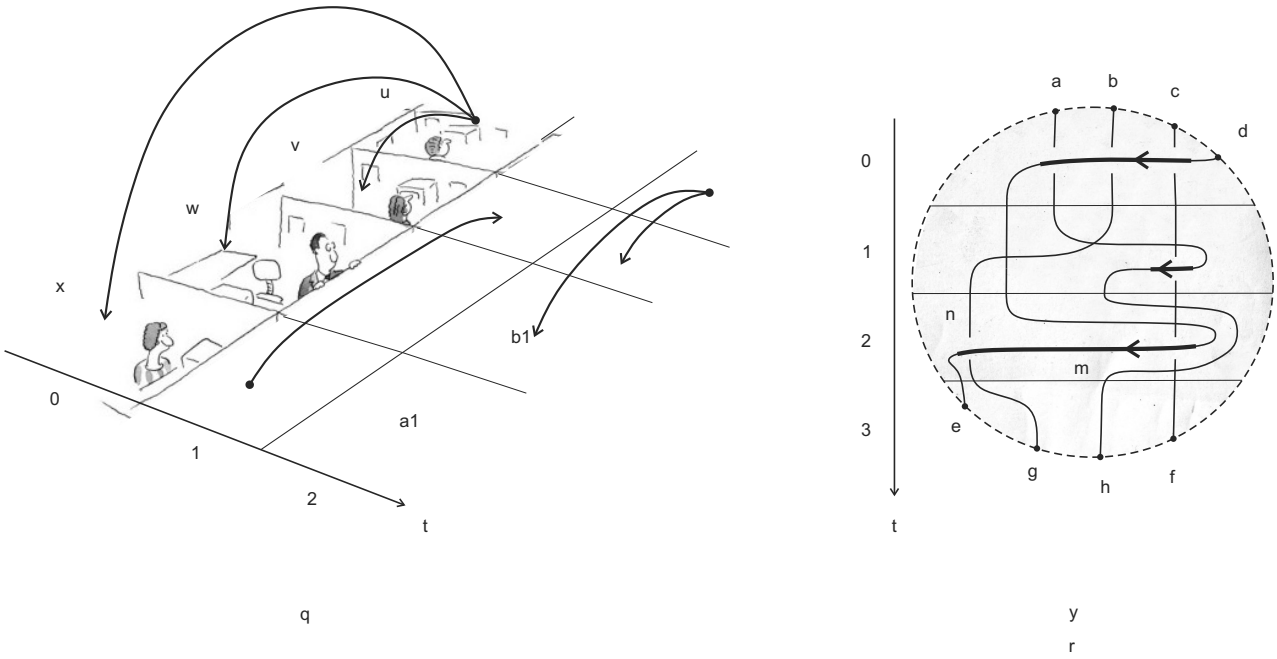


Figure 17. A rumor-passing network and its respective diagram.

The diagram representing the rumor passing network in Figure 17 is a braid, which is a special sort of tangle. Here, such diagrams represent the flow of beliefs within a network of interacting machines (verifiers).

When many patients pass under a single agent, we define this to imply that for each patient, the belief sampled from that patient is independent of the belief sampled from every other patient, and the belief sampled from the agent to update that patient's belief is independent of the belief sampled from the agent to update every other agent's belief. To say the same thing in a different way, multiple patients under the same agent are independent and unsynchronized during that time frame. We will discuss this point further in Section 10.2.

6.5. An Example

The capacity of a network to prove or disprove a claim is an emergent property. Out of a number of uncertain interactions, none of which prove the claim, the truth may eventually materialize. As an example, consider the two pairs of two consecutive interactions pictured in Figure 18. Both sequences involve three verifiers, designated X, Y and Z. Their initial beliefs are shown at the bottom.

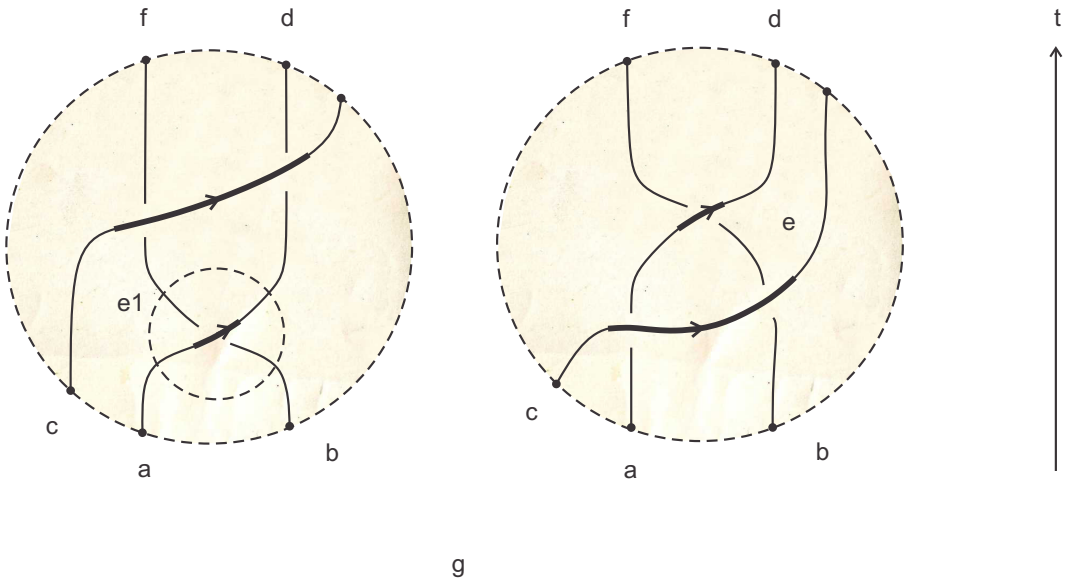


Figure 18. Equivalent networks of interactions.

Set the parameters to $s \stackrel{\text{def}}{=} \frac{1}{2}$, $c \stackrel{\text{def}}{=} 1$ and $\delta \stackrel{\text{def}}{=} \frac{1}{2}$. Allow the verifiers to interact in the left diagram. At time $t = 2$, beliefs X_0 and Y_0 of X and of Y have been updated to X_2 and to Y_2 (Z does not change his belief). The distributions are:

$$|X_2) = \left(|X_0)^{|Y_0)}\right)^{|Z_0)} \longrightarrow \begin{cases} \frac{3}{8}\,|\text{False}) + \frac{5}{8}\,|\text{True})\,, & x \in L; \\ \frac{21}{32}\,|\text{False}) + \frac{11}{32}\,|\text{True})\,, & x \notin L. \end{cases} \tag{33}$$

Hence,

$$\begin{aligned} |X_2) = \left(|X_0)^{|Y_0)}\right)^{|Z_0)} &= \tfrac{21}{32}\,|\text{False}) + \tfrac{5}{8}\,|\text{True})\,, \quad \text{(left network)}; \\ |X_2) = \left(|X_0)^{|Z_0)}\right)^{(|Y_0)^{|Z_0)})} &= \tfrac{21}{32}\,|\text{False}) + \tfrac{5}{8}\,|\text{True})\,, \quad \text{(right network)}. \end{aligned} \tag{34}$$

Similarly,

$$|Y_2) = |Y_1) = |Y_0)^{|Z_0)} = \frac{3}{8}\,|\text{False}) + \frac{3}{4}\,|\text{True})\,. \tag{35}$$

From this, we see that X decides correctly at time $\chi = 2$ with probability at least $\frac{5}{8}$ or $\frac{21}{32}$, depending on whether $x \in L$ or $x \notin L$. Both of these are greater than $\frac{1}{2}$, so the pair of inequalities Equation (32) for a suitable $\kappa > 0$, a protocol underlaid by the above interactions will succeed in deciding, at $|X_2)$, whether or not $x \in L$.

Table 1. A comparison between the meaning of a single interaction in IP and in BraidIP.

Description	IP system	Deformed IP system
Participants	Verifier, prover	Many verifiers (patient), verifier (agent), prover
Verifier "state of mind"	Accept/reject	Belief true/false
Conclusion	Verifier decides accept/reject	If the two verifiers do not agree, then the patient may change her belief.
Completeness, soundness	c, s	$c\delta, s\delta$

Note again that:

$$|Y_1) = |Y_0)^{|Z_0)} = h\,|Z_0) + (1-h)\,|Y_0)\,, \tag{36}$$

and we evaluate with $h = \frac{1}{2}$ for the coefficient of $|\mathrm{True})$ and with $h = \frac{1}{4}$ for the coefficient of $|\mathrm{False})$. This is the same for all interactions in both diagrams.

Note that both diagrams in Figure 18 have the same initial beliefs $|X_0)$, $|Y_0)$ and $|Z_0)$, the same terminal beliefs $|X_2)$, $|Y_2)$ and $|Z_2)$ and differ only in the belief of X at time $t = 1$. Thus, these two diagrams underlie equivalent deformed interactive proofs, each of which can uniquely be reconstructed from the other, which decide the same languages, but which differ at an intermediate step. This equivalence is the topic of Section 10.

7. Deformation of an IP System

In this section, we show how we may deform an IP system with any soundness parameters $0 < s < \frac{1}{2} < c \leq 1$, for any deformation parameter $\delta \in \mathbb{Q} \cap (0,1)$. The completeness and soundness parameters of the deformed system will be $s\delta$ and $c\delta$ correspondingly. The deformation parameter δ serves to introduce noise between the prover and the verifiers. In the $\delta \to 1$ limit, we recover IP, and the information obtained by a verifier at each interaction shrinks as $\delta \to 0$. However, Theorem 4 proves that we can recover IP from BraidIP by concatenating many consecutive deformed interactions.

Before describing how IP may be deformed, we outline the major differences between a single interaction in IP and BraidIP in Table 1.

7.1. Two Approaches to Deform IP

We present two approaches to deform an IP protocol. The end result is the same, but the "story" is different.

7.1.1. Agent and Patient as a Single Verifier

We may think of a patient W and an agent V of an interaction at time t as representing different aspects of a single verifier. In this approach, we conceive of W and V as being a single unit (W, V). The verifier (W, V) transmits to the prover Π the belief of both W and V. We may imagine W and V

as litigants in a court case, presenting their claims to the judge Π, where W is the defendant and V is the plaintiff. If both W and V make the same claim, then Π throws the case out (*i.e.*, $W_{t+1} = w_t$ and $V_{t+1} = v_t$). On the other hand, if V disagrees with W, then (W, V) query the prover Π according to the original interactive protocol. If, according to the original protocol, W's claim should be accepted, then Π rules in W's favor (*i.e.*, $W_{t+1} = w_t$ and $V_{t+1} = v_t$). However, if according to the original protocol, W's claim should be rejected and V's claim should be accepted, then Π picks an integer uniformly at random between one and N. If the number Π picked is less than δN, then Π rules in favor of V (*i.e.*, $W_{t+1} = v_t$ and $V_{t+1} = v_t$). Otherwise, he rules in favor of W.

Perhaps δ represents a chosen standard of "reasonable doubt". Constants s and c perhaps represent constants associated with the mechanics of the courthouse procedure. Note that as $\delta \to 1$, a single interaction involving two verifiers with opposite beliefs recovers IP.

7.1.2. Verifiers Communicating Through a Noisy Channel

The following approach has an information-theoretic interpretation. Consider the prover Π as an information source, the agent verifier V as an encoder and the patient verifier W as a decoder. The query information transmitted from W to Π is relayed via a perfect communication channel (*i.e.*, there is no loss of information in this direction). The replies from Π are passed on to V, who encodes them and transmits them back to W, this time through a noisy channel. This means that the prover replies emerge corrupted on W's end, which consequently influences her decision.

Introducing a noisy channel into the formalism restricts the information obtained by the patient verifier from the prover. It is tempting to state that the combination prover-agent-noisy channel behaves like a mendacious agent, in that the agent V decided whether to "tell the truth" or to "lie" to W. However, to think of V as a mendacious agent is inaccurate. For one thing, the agent's strategy whether or not to reliably relay Π's replies to W must account for the beliefs of both V and W, either one of which may not be correct. Her behavior does not stem from her being more knowledgeable; rather, we may think of it as a manifestation of her own beliefs.

It is important to note that although V receives the replies from Π, she is not allowed to use this information to update her own belief. One can think of protocols taking advantage of the fact that V is not aware of W's queries, wherein this restriction follows naturally. For now, it is enough to assume that V will not use the prover replies for her own benefit.

Here is how such a protocol may run. Upon disagreement between W and V, *i.e.*, $v_t \neq w_t$, the patient W sends her queries to Π. The replies to W's queries are then sent by Π to the agent V. At this point, V, who is a verifier, much like W, runs her own verification test on the prover replies. He obtains $\xi_t = 1$ for accept/true and $\xi_t = -1$ for reject/false. In the case where $\xi_t = 1$, she tampers with the prover replies, such that when they are received by W, her verification would indicate v_t with probability δ. In the case where $\xi_t = -1$, the agent V tampers with the prover replies, such that the test of W would indicate $\neg v_t$.

Implicit in the above protocol is the fact that the capacity of V to deceive W is limited by V's own belief. If her belief, v_t, coincides with the true nature of the claim, then she may potentially have more power to deceive W.

The protocol just described underlies a noisy symmetric channel between W and V. If $\xi_t = 1$, this channel is characterized by δ and has a capacity of $1 - H_2(\delta)$, where $H_2(\delta)$ is the Shannon entropy of

a Bernoulli random variable with parameter δ. It induces a maximal loss of one bit of information for $\delta = \frac{1}{2}$. An illustration of the communication between the three parties patient-agent-prover is given in Figure 19.

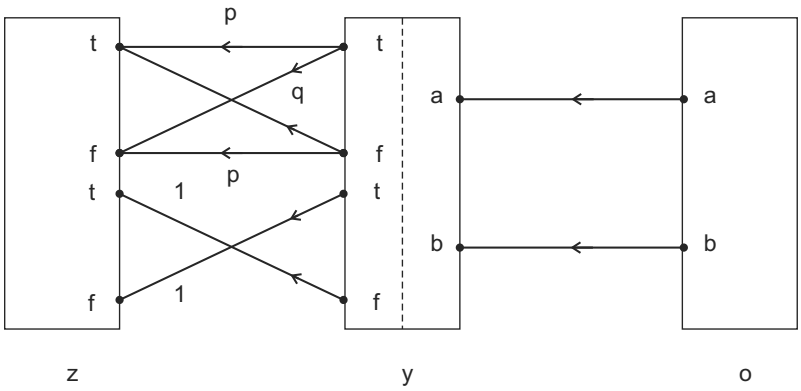


Figure 19. Communication between W and V when $v_t \neq w_t$. The channels between W and V are symmetric. The labels above edges indicate transition probabilities. The box of V is divided into two sections representing her belief v_t (left section) and the outcome of his verification ξ_t (right section).

7.1.3. Further Metaphors for Deformed Interactions

The agents in our picture all receive messages from the same oracle. This essentially suggests that a network of interactions is a construct quantizing the oracle knowledge. At every location within the network, only a quanta of this knowledge is used by way of interaction between a patient and an agent. Later on, we will show that although a single interaction may be limited in its capacity to prove the claim, the proof may yet emerge somewhere in the network, depending on its topology.

The triple patients-agent-oracle brings to mind some basic models of reasoning and information transfer. Perhaps a patient is an entity whose beliefs reflect both prior knowledge and observations. The patient is exposed to a genuine phenomenon, which the patient has not seen before. The phenomenon, which is the metaphor for an oracle, is beyond the comprehension of the patient, and hence, a number of observations are collected in an attempt to reach a definitive conclusion. These observations, however, may be distorted by limitations of the patient measuring apparatus or perhaps they contradict prevailing explanations and beliefs. In either cases, observations contain, or otherwise introduce, uncertainty. Observations are the metaphor for agents. What the patient tries to accomplish underlies the Bayesian inference paradigm.

Here is another metaphor. A patient is a decoder; an oracle is an information source; and an agent is an encoder who relays the encoded oracle message through a noisy communication channel. Alternatively, an agent-patient pair is a verifier, and the oracle is a prover who relays a message through a noisy communication channel. All metaphors reflect knowledge transfer subject to uncertainties.

7.2. Probabilistic Theorem Proving in Networks

The goal of this section is to prove Theorem 4, repeated below for convenience.

Theorem 7. IP $\subseteq$ BraidIP $\{\delta, \chi\}$ *where:*

$$I(c\delta) < \chi < \frac{1}{I(1-s\delta)}, \tag{37}$$

with $I(p) \stackrel{\text{def}}{=} -p^{-1}\log p$. *The growth rate of* χ *is* $\mathcal{O}(\frac{1}{\delta})$ *as* $\delta \to 0$.

Proof of Theorem 4. We explicitly construct a configuration of interactions that decide a language L in IP. This configuration, which is illustrated in Figure 20 for the case $\chi = 4$, is a scaled up version of that in Figure 18. It involves $\chi + 1$ verifiers W and $V^1, V^2, \ldots, V^\chi$ and χ interactions. The parameters of all interactions are the same and are c, s, δ.

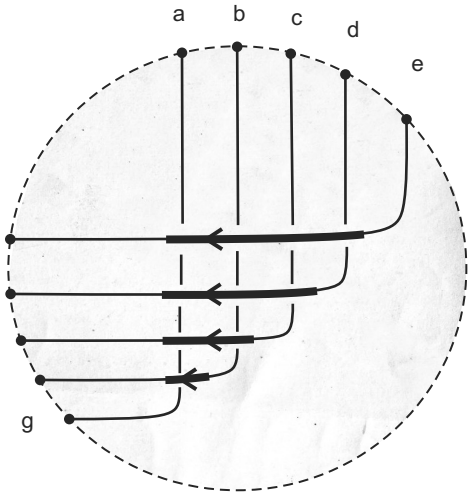


Figure 20. An interactive BraidIP theorem proving network with five verifiers W, V^1, V^2, V^3 and V^4 and four interactions.

Let $L \in$ IP. The initial beliefs at time $t = 0$ are set to $|W_0) \stackrel{\text{def}}{=} |\text{False})$ and:

$$|V_0^i) \stackrel{\text{def}}{=} \begin{cases} \frac{1}{2}|\text{False}) + \frac{1}{2}|\text{True})\,, & 1 < i < \chi; \\ |\text{True})\,, & i = \chi. \end{cases} \tag{38}$$

Thus, at time zero, there are two verifiers with opposite beliefs and $\chi - 1$ verifiers whose initial belief is that the claim $x \in L$ is 50% true and 50% false.

Calculating the output statistic $|W_\chi)$ (which occurs at time χ) yields:

$$|W_\chi) \longrightarrow \begin{cases} (1-c\delta)^\chi\,|\text{False}) + [1 - c\delta - (1-c\delta)^\chi]\left(\frac{1}{2}|\text{False}) + \frac{1}{2}|\text{True})\right) + c\delta\,|\text{True})\,, & x \in L; \\ (1-s\delta)^\chi\,|\text{False}) + [1 - s\delta - (1-s\delta)^\chi]\left(\frac{1}{2}|\text{False}) + \frac{1}{2}|\text{True})\right) + s\delta\,|\text{True})\,, & x \notin L. \end{cases} \tag{39}$$

From Equation (39), we see that the configuration in Figure 20 decides L if and only if:

$$\begin{aligned} x \in L &\longrightarrow (1-c\delta)^\chi < c\delta; \\ x \notin L &\longrightarrow (1-s\delta)^\chi > s\delta. \end{aligned} \tag{40}$$

From here, we obtain the following bounds for χ:

$$\frac{\log(c\delta)}{\log(1-c\delta)} < \chi < \frac{\log(s\delta)}{\log(1-s\delta)}. \tag{41}$$

Equation (2) follows upon noting that $\log(1-p) < -p$ for any $p \in (0,1)$. Therefore:

$$-\frac{1-p}{\log(1-p)} > \frac{\log p}{\log(1-p)} > -\frac{\log p}{p}. \tag{42}$$

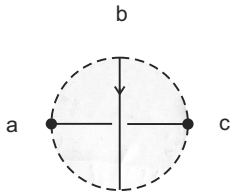
For χ within these bounds, the above configuration decides L. □

Remark 9. *Equation (2) tells us that χ has approximately the same growth rate in $|x|$ as $\frac{1}{\delta}$. By definition of* BraidIP*, χ's growth rate is polynomial in the word length $|x|$, and so, therefore, δ is asymptotically bounded below by approximately one over a polynomial in $|x|$.*

8. Efficient IP Strategies: Tangled IP

8.1. The Complexity Class TangIP

We may extend class BraidIP by allowing each verifier to have its own "local" time parameter, so that a patient belief W_t may interact with an agent belief V_s for $s \neq t$ and become updated to W_{t+1}.



(43)

We also allow verifiers to travel backwards or forwards in time and to update their previous or future beliefs, so that V_t may be updated by agent V_s to become V_{t+1}, where V_t and V_s are beliefs of one and the same verifier. Thus, each verifier may update their beliefs with past or future beliefs of itself (via feedback loops) or of other verifiers. We write M for a network of such concatenated interactions, subject to parameters $0 < c < s \leq 1$ and $\delta \in \mathbb{Q} \cap (0,1)$. An example of such a network is given in Section 8.2.

A language $L \subseteq \{0,1\}^*$ is said to be decided by M if M contains a verifier V whose belief at time χ is $|V_\chi) \stackrel{\text{def}}{=} a\,|\text{True}) + b\,|\text{False})$, such that for a fixed constant $\kappa > 0$, the inequalities Equation (32) are satisfied.

The class "tangled interactive polynomial time" (TangIP) consists of those languages that are decidable for any fixed $\kappa > 0$ by some network M, which contains χ interactions, where χ is polynomial in $|x|$. We denote this class $\text{TangIP}\{\delta, \chi\}$.

By Theorem 4, we know that:

$$\text{IP} \subseteq \text{BraidIP} \subseteq \text{TangIP}.$$

We wonder about the connection between our classes BraidIP and TangIP and multi-prover IP (MIP) [27]. In particular, we wonder whether MIP $\subseteq$ BraidIP or MIP $\subseteq$ TangIP, particularly if we allow different interactions to have different parameters (in this note, all interactions are required to have the same parameters, because that is all we need, but there is no obstruction to considering the more general case).

8.2. The Hopf–Chernoff Configuration

Consider an IP system whose soundness s is nearly equal to its completeness c, but for a small constant $\epsilon(|x|)$ that depends on the word length $|x|$, *i.e.*, $c - s = \epsilon(|x|)$. As $\epsilon(|x|) \to 0$, the IP system becomes inefficient in the sense that it accepts every word with probability nearly c regardless of its membership in L. Yet, we can still construct a deformed IP system that decides L. One may wonder how the number of interactions in such a system is affected by the decreasing gap $\epsilon(|x|)$.

The number of interactions in a network of the form given in Figure 20 is implicit in Theorem 4. Fix $\delta \in \mathbb{Q} \cap (0, 1)$, and note from Equation (2) that, as $\epsilon(|x|)$ decreases, the values bounding χ become nearly identical. However, χ is an integer, so the two bounds must have an integer between them. In general, that means that the distance between them is at least one. Therefore, the number of interactions grows as $\epsilon(|x|)$ decreases. We may need to take $\delta \to 0$ as $\epsilon(|x|) \to 0$. In fact, it can be shown that in this case $\delta(|x|) = \mathcal{O}(\epsilon(|x|))$, which means that we require $\chi = \mathcal{O}(1/\epsilon(|x|))$ interactions.

Can fewer interactions decide L ? The configuration in Figure 21 decides L for any $\epsilon(|x|)$ using substantially less than $\mathcal{O}(1/\epsilon(|x|))$ interactions. This machine is a concatenation of a number of identical smaller configurations of interactions, denoted $M_0, M_1, M_2, \ldots$. When concatenated to form a single configuration, we require approximately $\chi = \mathcal{O}(\log(1/\epsilon(|x|)))$ copies of M_0 to decide L (the precise argument is given below). If we were to trace its colors (probability-generating functions), we would notice that it behaves much like a repetition of a binary random experiment (e.g., coin flipping), hence the magnitude of χ. We have named this configuration the Hopf–Chernoff configuration, suggesting both its structure and, to some extent, its functionality.

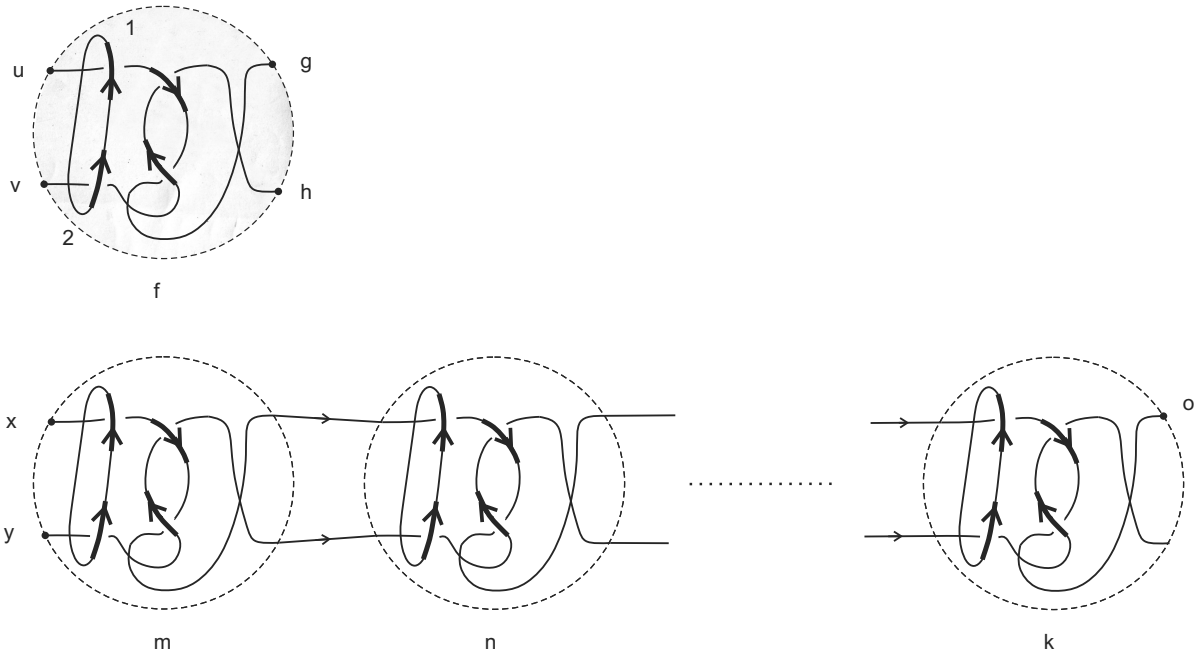


Figure 21. Hopf–Chernoff configuration (open version).

Theorem 8 (Hopf–Chernoff configuration)**.** *Consider the configuration of interactions in Figure 21, which underlies a deformed* IP *system with completeness* $c\delta$ *and soundness* $(c - \epsilon)\delta$*, where* $\epsilon > 0$*. There exists a pair of beliefs,* α *and* β*, independent of any other belief in the machine, such that for any given*

set of initial beliefs In_j^0*, the machine decides any* $L \in \mathrm{IP}$ *using* $\chi = \mathcal{O}(\log(1/\epsilon))$ *sub-machines (and* 4χ *interactions). In particular, letting:*

$$\begin{aligned} |\alpha) &= \left(\tfrac{1}{4} + \tfrac{1}{12}\epsilon\delta\right)|\mathrm{True}) + \left(\tfrac{3}{4} - \tfrac{1}{12}\epsilon\delta\right)|\mathrm{False})\,; \\ |\beta) &= \left(1 - \tfrac{1}{2}c\delta + \tfrac{1}{12}\epsilon\delta\right)|\mathrm{True}) + \left(\tfrac{1}{2}c\delta - \tfrac{1}{12}\epsilon\delta\right)|\mathrm{False})\,. \end{aligned} \tag{44}$$

yields:

$$\mathrm{Out}_1^{\chi} \longrightarrow \begin{cases} \left[\tfrac{1}{2} + \tfrac{1}{12}\epsilon\delta\right]|\mathrm{True}) + \left[\tfrac{1}{2} - \tfrac{1}{12}\epsilon\delta\right]|\mathrm{False})\,, & x \in L; \\ \left[\tfrac{1}{2} - \tfrac{1}{12}\epsilon\delta\right]|\mathrm{True}) + \left[\tfrac{1}{2} + \tfrac{1}{12}\epsilon\delta\right]|\mathrm{False})\,, & x \notin L. \end{cases} \tag{45}$$

namely, $\mathrm{Out}_1^{\chi} = \left[\tfrac{1}{2} + \tfrac{1}{12}\epsilon\delta\right]|\mathrm{True}) + \left[\tfrac{1}{2} + \tfrac{1}{12}\epsilon\delta\right]|\mathrm{False})$.

Proof. Let us begin by writing down the relations between the outputs Out_j and inputs In_j of this machine. Note that:

$$\mathrm{Out}_1 = \left(\mathrm{In}_1^{|\alpha)}\right)^{\left(\mathrm{In}_2^{|\beta)}\right)}, \quad \mathrm{Out}_2 = \left(\mathrm{In}_2^{|\beta)}\right)^{\left(\mathrm{In}_1^{|\alpha)}\right)}. \tag{46}$$

Explicitly writing Equation (46) using the formal parameter h yields:

$$\begin{bmatrix} \mathrm{Out}_1^i \\ \mathrm{Out}_2^i \end{bmatrix} = \underbrace{\begin{bmatrix} (1-h)^2 & h(1-h) \\ h(1-h) & (1-h)^2 \end{bmatrix}}_{\stackrel{\mathrm{def}}{=} A(h)} \begin{bmatrix} \mathrm{In}_1^i \\ \mathrm{In}_2^i \end{bmatrix} + \underbrace{\begin{bmatrix} h(1-h) & h^2 \\ h^2 & h(1-h) \end{bmatrix}}_{\stackrel{\mathrm{def}}{=} B(h)} \begin{bmatrix} |\alpha) \\ |\beta) \end{bmatrix}. \tag{47}$$

Letting $\mathrm{In}_j^{i+1} = \mathrm{Out}_j^i$, $j = 1, 2$, Equation (47) underlies a linear dynamical system. It is easy to verify that the eigenvalues of the transition matrix $A(h)$ all are within the unit circle, *i.e.*, $|\lambda(A(h))| < 1$ for any $h > 0$. That means that the system Equation (47) reaches a steady state as $i \to \infty$. The steady state can be obtained as follows. Rewrite Equation (47) as:

$$\begin{bmatrix} \mathrm{Out}_1 \\ \mathrm{Out}_2 \end{bmatrix} = A(h) \begin{bmatrix} \mathrm{Out}_1 \\ \mathrm{Out}_2 \end{bmatrix} + B(h) \begin{bmatrix} |\alpha) \\ |\beta) \end{bmatrix}, \tag{48}$$

and solve for Out_1 and Out_2. Thus,

$$\begin{bmatrix} \mathrm{Out}_1 \\ \mathrm{Out}_2 \end{bmatrix} = (I - A(h))^{-1} B(h) \begin{bmatrix} |\alpha) \\ |\beta) \end{bmatrix} = \frac{1}{3-2h} \begin{bmatrix} 2(1-h)\,|\alpha) + |\beta) \\ |\alpha) + 2(1-h)\,|\beta) \end{bmatrix}. \tag{49}$$

Define:

$$\begin{aligned} |\alpha) &\stackrel{\mathrm{def}}{=} a\,|\mathrm{True}) + (1-a)\,|\mathrm{False})\,; \\ |\beta) &\stackrel{\mathrm{def}}{=} b\,|\mathrm{True}) + (1-b)\,|\mathrm{False})\,. \end{aligned} \tag{50}$$

For the network to decide L, we require the steady state of Out_1 to satisfy:

$$\mathrm{Out}_1 \longrightarrow \begin{cases} \left(\tfrac{1}{2} + \sigma\right)|\mathrm{True}) + \left(\tfrac{1}{2} - \sigma\right)|\mathrm{False})\,, & x \in L; \\ \left(\tfrac{1}{2} - \sigma\right)|\mathrm{True}) + \left(\tfrac{1}{2} + \sigma\right)|\mathrm{False})\,, & x \notin L. \end{cases} \tag{51}$$

for some $\sigma > 0$. Using both Equations (49) and (50), this requirement translates into the following set of equations:

$$\begin{aligned} (3 - 2c\delta)\left(\tfrac{1}{2} + \sigma\right) &= 2(1 - c\delta)a + b, && x \in L; \\ (3 - 2(c-\epsilon)\delta)\left(\tfrac{1}{2} - \sigma\right) &= 2(1 - (c-\epsilon)\delta)a + b, && x \notin L. \end{aligned} \tag{52}$$

where the fact that $h = c\delta$ for $x \in L$ and $h = (c - \epsilon)\delta$ for $x \notin L$ has been used. Solving Equation (52) for the coefficients a and b while assuming $\sigma = \frac{1}{12}\epsilon\delta$ yields Equation (44). The underlying output probabilities in Equation (45) are given by Equation (51).

To complete the argument, we need to show that the network converges within the stated number of iterations. It is sufficient to consider the case where the output probabilities Equation (51) is attained to within the order $\mathcal{O}(\sigma) = \mathcal{O}(\epsilon)$. The growth rate of the system Equation (47) is linear in $|\lambda_1(A(h))|^\chi$ where $\lambda_1(A(h))$ denotes the largest eigenvalue of $A(h)$. Simple calculation shows that $\lambda_1(A(h)) = 1-h$, which yields $\chi = \mathcal{O}(\log(1/\epsilon))$. □

The Hopf–Chernoff configuration is a recursive structure, which is guaranteed to converge irrespective of its initial beliefs In_j^0. In fact, it represents a two-dimensional homogeneous irreducible Markov chain whose rate of convergence is $\mathcal{O}(2^{-\chi})$. Its stationary distribution, which depends on whether $x \in L$ or $x \notin L$, is given by Equation (45). By virtue of its convergence properties, we may just let it run forever (*i.e.*, $\chi \to \infty$) knowing that it will eventually reach a stationary distribution not far from Equation (45). For that reason, we may as well substitute the open network in Figure 21 with its closed counterpart in Figure 22.

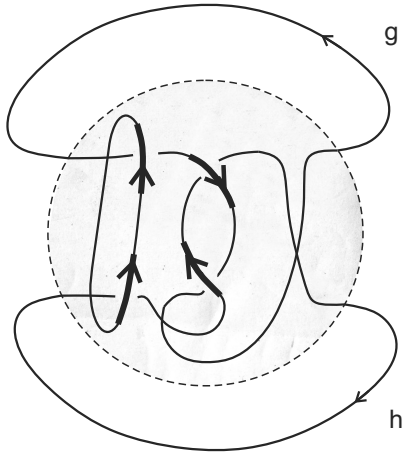


Figure 22. Hopf–Chernoff configuration (closed version).

9. PCP Networks

In this section, we specialize to non-adaptive three-bit verifiers, to exhibit that tangle machines may exhibit better performance parameters than classical systems.

We deform the Håstad PCP verifier, which has $c = 1$ and $s \approx 0.75$. Suppose the two verifiers disagree, $v_t \neq w_t$, where $v_t, w_t \in \{|\mathrm{True}), |\mathrm{False})\}$. In this case, the interaction proceeds as follows. The patient verifier W provides the addresses of three bits to Π. These bits are received by the agent verifier V, which computes a certificate ξ_t, which is equal either to one, meaning "accept", or to -1, meaning "reject". The agent then flips one out of the three bits with the following probabilities:

- $(\xi_t = 1) \wedge (v_t = |\mathrm{True})) \longrightarrow V$ flips a bit with probability $1 - \delta$.
- $(\xi_t = 1) \wedge (v_t = |\mathrm{False})) \longrightarrow V$ flips a bit with probability δ.
- $(\xi_t = -1) \wedge (v_t = |\mathrm{True})) \longrightarrow V$ flips no bit.

- $(\xi_t = -1) \wedge (v_t = |\mathrm{False})) \longrightarrow V$ flips a bit.

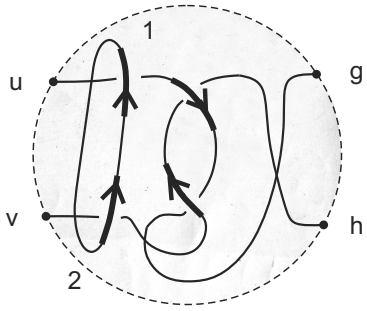
The corrupted set of bits is then sent back to W, who computes her own certificate. This protocol realizes the communication channel in Figure 19.

9.1. A Better Than Classical PCP Verifier

In this section, we construct a tangle machine whose interactions are deformed Håstad verifiers, which has perfect accuracy and a completeness of $\frac{2}{3} + \sigma \approx 0.667$. This is worse than the conjectured bound of 0.625, but better than the best-known classical non-adaptive three-bit PCP protocol, whose soundness is only around 0.741. Thus, our machine behaves like a single very good verifier.

Our machine makes use of the Hopf–Chernoff configuration in Figure 21.

1. Choose input beliefs In_j^0, $j = 1, 2$, arbitrarily from $\{|\mathrm{True}) , |\mathrm{False})\}$. Assume $c = 1$, and fix δ.
2. Let α be a Bernoulli distribution with parameter $\frac{1}{2}$. Draw a belief from α for the top agent, and set the bottom agent to the negation of that belief. We color the bottom agent as $\neg\alpha$, which equals α as a distribution, to diagrammatically signify what we are doing.
3. Do the following for $i = 1, \ldots, \chi$, where $\chi = \mathcal{O}(1)$:
   - Using the underlying deformed PCP verifier, perform the four interactions of the Hopf–Chernoff configuration to propagate the beliefs of the two verifiers from In_j^i to Out_j^i according to the diagram below.



   - Set $\mathrm{In}_j^i = \mathrm{Out}_j^{i-1}$, $j = 1, 2$.
4. If the patient belief in the output $\mathrm{Out}_1^\chi = \neg\alpha$, then return $|\mathrm{True})$; otherwise, return $|\mathrm{False})$.

Theorem 9. *The Hopf–Chernoff machine approaches perfect completeness and soundness at most* $\frac{1}{3-2s}$ *as* $\delta \to 1$. *In particular, if we put deformed Håstad verifiers at interactions, its soundness is at most* $\frac{2}{3}$.

Proof.

Completeness Assume that $x \in L$, and note that as $\delta \to 1$, so does $h = c\delta \to 1$. In the limit where $h = 1$, note that by Equation (41), the Hopf–Chernoff swaps the beliefs α and $\neg\alpha$, such that always $\mathrm{Out}_1 = \neg\alpha$ and $\mathrm{Out}_2 = \alpha$. Thus, Step (iv) of the algorithm concludes with $|\mathrm{True})$.

Soundness The soundness of the algorithm bounds the probability that $\mathrm{Out}_1 = \neg\alpha$ in the case where $x \notin L$. This probability is given by:

$$\Pr(\mathrm{Out}_1 = \neg\alpha) = \Pr(\mathrm{Out}_1 = |\mathrm{True}) \mid \alpha = |\mathrm{False})) \Pr(\alpha = |\mathrm{False})) + \Pr(\mathrm{Out}_1 = |\mathrm{False}) \mid \alpha = |\mathrm{True})) \Pr(\alpha = |\mathrm{True})). \quad (53)$$

Although not truly essential, the algorithm assumes $\Pr(\alpha = |\mathrm{True})) = \frac{1}{2}$. The conditional probabilities above can be bounded using Equation (41) as follows. Take $h = s\delta \to s$ and assume that $|\alpha) = |\mathrm{False})$. In this case, Equation (41) implies:

$$\Pr(\mathrm{Out}_1 = |\mathrm{True}) \mid \alpha = |\mathrm{False})) \leq \frac{1}{3 - 2s} \ . \quad (54)$$

On the other hand, letting $|\alpha) = |\mathrm{True})$, the same equation reads:

$$\Pr(\mathrm{Out}_1 = |\mathrm{False}) \mid \alpha = |\mathrm{True})) \leq \frac{1}{3 - 2s} \ . \quad (55)$$

This follows from the fact that the algorithm decides $x \in L$ if $\mathrm{Out}_1 = \neg\alpha$ irrespective of the beliefs themselves. For that reason, these equations coincide, though for different values of α and Out_1. Both describe a failure of the algorithm to decide $x \notin L$. The theorem now follows from Equations (53)–(55).

□

10. Low-Dimensional Topology and Bisimulation

Definition 10. *Tangle machines M and M', which each come equipped with a distinguished set of input and output registers, are bisimilar if any computation that can be carried out on M can be carried out on M' and vice versa.*

Because computations are defined only with respect to the pre-chosen sets of input and of output registers, Definition 10 encapsulates what may be thought of as a weak notion of bisimulation (no requirement is made on "silent" or "internal" interactions).

In Section 10.1, we formulate a set of local moves, such that any two machines related by these local moves are bisimilar. In Section 10.2, we discuss a feature of our formalism, that is the "unsplittability" of our agent registers. In Section 10.3, we suggest an application of machine equivalence to define a notion of zero knowledge for tangle machines and to utilize it to construct TangIP machines, which are "more secure" in a specific sense. We give an example in Section 10.4. Finally, in Section 10.5, we extend our notion of machine equivalence to machines that may have wyes.

10.1. Equivalence

The key property of tangle machines is that they admit a local notion of equivalence [1]. Two (quandle colored, without wyes) tangle machines are equivalent if they are related by a finite sequence of the moves in Figures 23 and 24. It is forbidden for these moves to involve input and output registers of a computation.

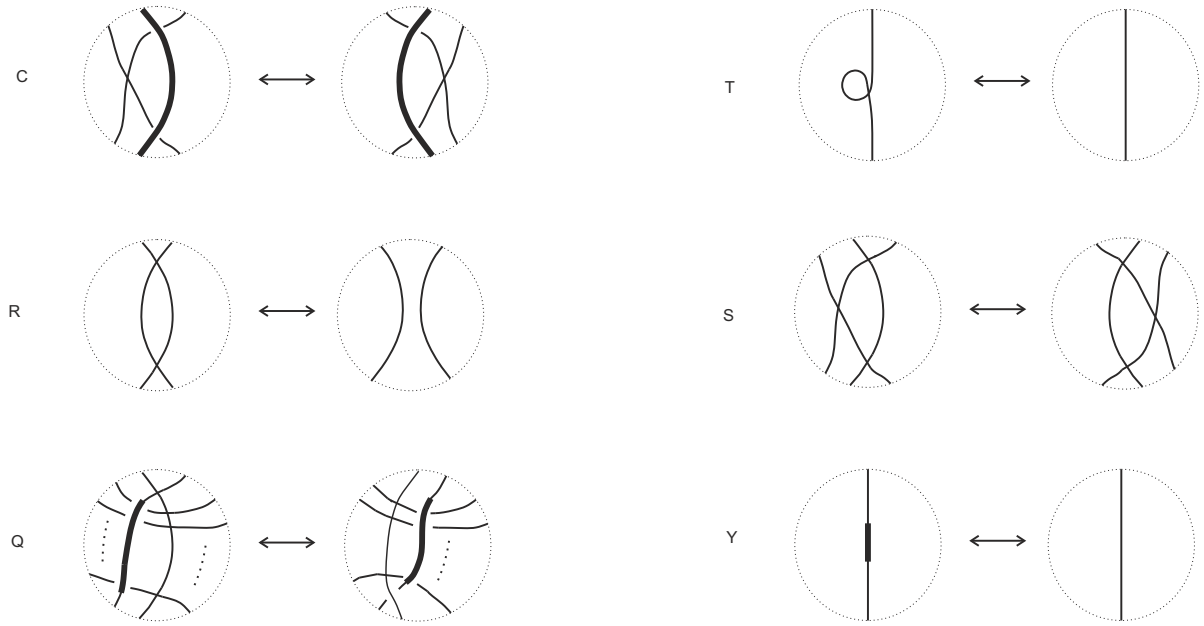


Figure 23. Cosmetic moves for machines. Directions are not indicated, meaning that the moves are valid for any direction and the same for colorings.

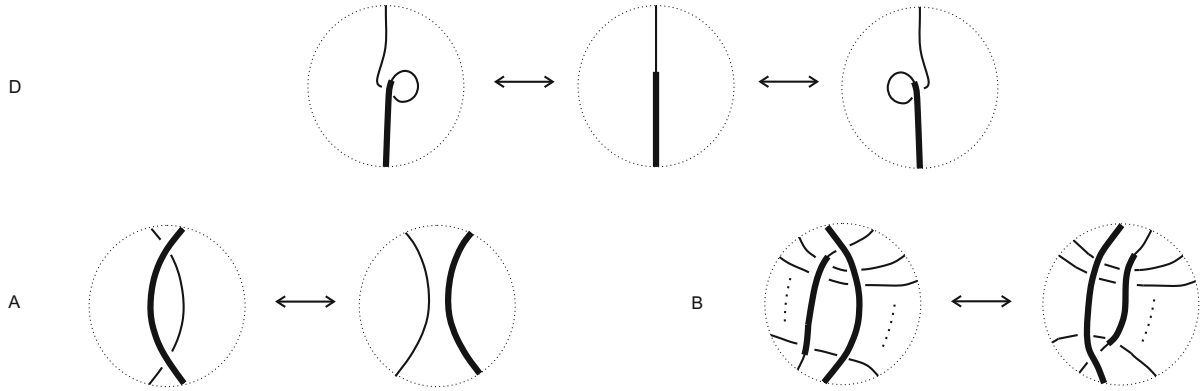
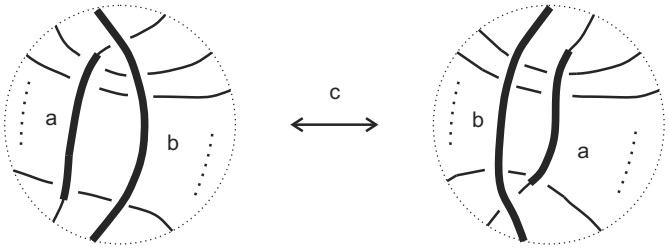


Figure 24. Reidemeister moves for machines, valid for any directions of the agents. It is forbidden for these moves to involve input and output registers of a computation.

Two machines related by the local moves in Figure 23 carry out identical computations, and it follows from the construction of an interaction that two machines related by R1 differ only by a trivial computation at which "nothing happens". Thus, the interesting moves for us are R2 and R3 in Figure 24, about which we will say more later.

Remark 10. *First note that, for* R2 *to make sense, all participating colors must be defined. This requirement is non-trivial for a machine colored by a quandloid.*

Remark 11. *For a machine colored by a quagma, the* R3 *move is replaced by the following:*



(56)

for all $\triangleright, \blacktriangleright \in B$ *satisfying* $(x \triangleright y) \blacktriangleright z = (x \blacktriangleright z) \triangleright (y \blacktriangleright z)$ *for all* $x, y, z \in Q$.

If we choose input and output registers to be machine endpoints, then equivalent machines have identical initial and terminal beliefs, which implies that both machines have the same computational power in terms of deciding a language. Nevertheless, the local behavior of equivalence may be different, in that the colors of intermediate interactions in between the same initial and terminal statistics may be different in equivalent machines. As in the earlier example in Figure 18, equivalent machines may have two different prover strategies arriving at the same proof.

To expand that example, Figure 25 features several equivalent prover strategies for the machine in Figure 20, all of which are obtained by application of R3 moves.

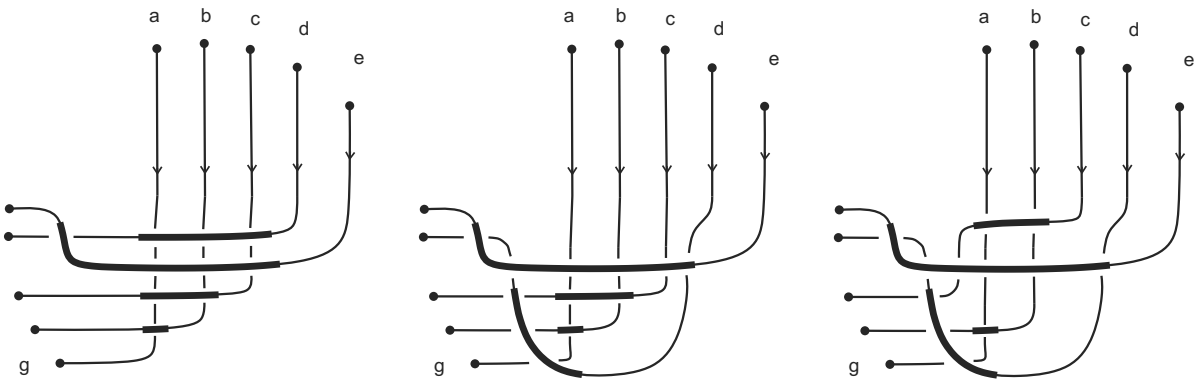
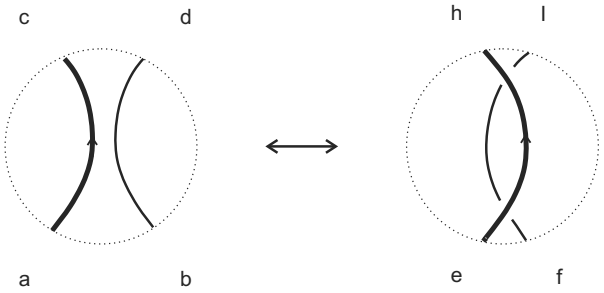


Figure 25. Equivalent prover strategies for the machine in Figure 20.

10.2. The Single Agent in R2 and R3

In this section, we discuss the single agent that acts on numerous patients and cannot be split. Such an agent features in Moves R2 and R3 and distinguishes our approach, e.g., from w-tangles [35].

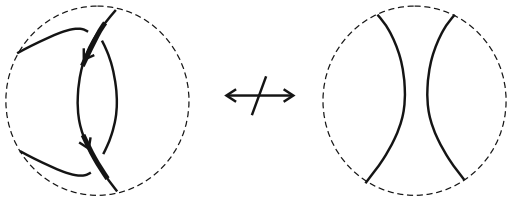
The R2 move tells us that computations are reversible, in the sense that any operation $\triangleright \in B$ has an inverse operation $\triangleleft \in B$, such that no information is computed from $(x \triangleright y) \triangleleft y$ for any $x, y \in Q$. Because we are working not only with colors, but with realizations of belief statistics, we are saying more than just $(x \triangleright y) \triangleleft y = x$. We require that there be zero knowledge gain about realizations of $(x \triangleright y) \triangleleft y$ from a realization of x.



(57)

Our formalism features agents that act on multiple patients. These actions are independent by definition. Conversely, as we saw in Section 8.2, different agents may cooperate, for example, by coordinating their realizations to be the same or to be opposed to one another.

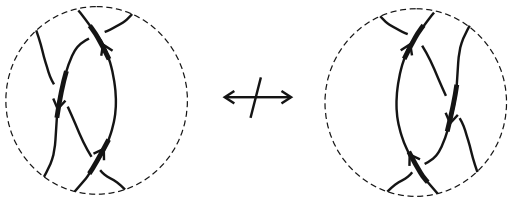
We do not impose the following *a priori* reasonable generalization of R2.



(58)

One reason that we do not impose Equation (58) can be seen by considering the example in which the two agents in the machine on the left-hand side adopt the strategy of always offering the same realization. If the realization of the patient and of the agent coincide, we can compute the realizations of both other patients. If not, then we cannot. Thus, we can compute the colors of the remaining patients in Equation (58) for some realizations, but not for others. This behavior is not shared by the machine on the right-hand side, in which colors of the realizations of patients can never be computed unless they are already given. If the choice of realization is independent for both patients, *i.e.*, if there is only a single interaction, as in the case of "honest" R2, there is no such phenomenon, and no choice of realizations for input registers is distinguished from any other. Note also that Equation (58) represents two distinct computations, each of which can be considered separately and each of which is non-trivial, which is not true for the right-hand side of the "honest" R2.

For the same reason, we do not impose the following "fake R3 move":



(59)

10.3. Zero Knowledge

The theory of IP features the notion of a zero-knowledge proof [26,48]. In a zero-knowledge proof, the information that may be gained by the verifier in the course of her interactions with the prover are restricted. This is useful when the verifier may not always be trustworthy. The definition makes use of

a simulator, which is an arbitrary feasible algorithm that is able to reproduce the transcript of such an interaction without ever interacting with the prover.

We suggest the following definition as a TangIP analogue to the notion of zero knowledge.

Definition 11 (Zero-knowledge tangle machine)**.** *A tangle machine M that decides a language $L \in$ TangIP is said to be zero knowledge if the following is satisfied.*

1. *There are no intermediate interactions in M that decide L.*
2. *There exists an equivalent machine M', which decides L at one of its intermediate interactions.*

Remark 12. *The idea of zero-knowledge tangled* IP *parallels the authors' model of fault-tolerant information fusion networks, except that there, we wanted intermediate registers to "know as much as possible", whereas here, we want them to "know as little as possible" [4].*

As a generic example, consider machines M' and M in Figure 26. Both share the same initial and terminal belief statistics. The explicit structure of the machines is mostly irrelevant, except that they both contain a sub-machine S, which we graphically represent by a blank disk, with the property that M, M', and some of S's terminal statistics $|Y_i)$ decide L. In M, the verifier Z is an agent to all initial states of S, and the resulting beliefs from this interaction are $|X_i)$, $i = 1, 2, \ldots$ In M', the same verifier is an agent to the terminal states of S, and the resulting beliefs from this interaction are characterized by $|Y_i)$.

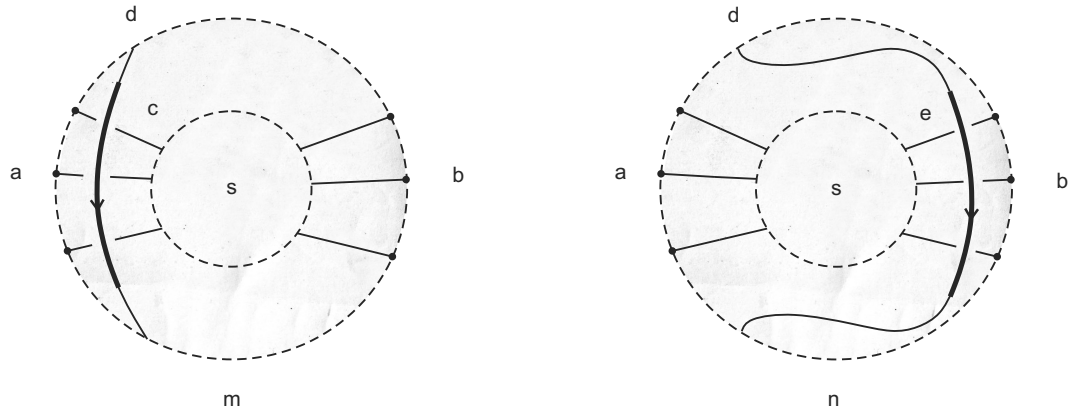


Figure 26. Two equivalent machines. The machine M is zero knowledge. The proper sub-machine S determines L.

That M is zero knowledge implies that the proof should not appear somewhere within it. Assuming none of the $\mathrm{In}'s$ decide L and neither do any intermediate belief states of S, this requirement implies that none of the $|X_i) = (\mathrm{In}_i)^{|Z)}$ decide L. On the other hand, the machine M' shows us that an interaction between the terminal states of S, some of which decide L, with the agent Z are able to produce the proof. That is, some of $\mathrm{Out}_i = |Y_i)^{|Z)}$ decide L. These requirements completely characterize the belief distribution $|Z)$. Perhaps unsurprisingly, it turns out that $|Z) = \frac{1}{2}\,|\mathrm{True}) + \frac{1}{2}\,|\mathrm{False})$.

Ideally, we would require that S reproduces the proof as if it were produced by M itself. This requirement translates into:

$$|Y_i) = \mathrm{Out}_i, \tag{60}$$

for any Out_i that decides L. As both sides of Equation (60) depend on the deformation parameter δ, this equation may be used to determine δ, such that M is zero knowledge. Below, we give an example.

10.4. Example

Consider the two machines in Figure 27. Assume they both employ a deformed IP system whose completeness and soundness are δ and $\frac{1}{2}\delta$. Let $|Z) = \frac{1}{2}\,|\mathrm{False}) + \frac{1}{2}\,|\mathrm{True})$, and let us first see what the value of δ is for the machine to decide L. The output $|X_2)$ of either machines is given by:

$$|X_2) \longrightarrow \begin{cases} \left[(1-\delta)^2 + \frac{1}{2}\delta\right]|\mathrm{False}) + \left[\delta(1-\delta) + \frac{1}{2}\delta\right]|\mathrm{True})\,, & x \in L; \\ \left[(1-\frac{1}{2}\delta)^2 + \frac{1}{4}\delta\right]|\mathrm{False}) + \left[\frac{1}{2}\delta(1-\frac{1}{2}\delta) + \frac{1}{4}\delta\right]|\mathrm{True})\,, & x \notin L. \end{cases} \tag{61}$$

from which we conclude that $\delta > \frac{1}{2}$. The machine on the right in this figure is zero knowledge, because $|X_1)$ does not decide L:

$$|X_1) \longrightarrow \begin{cases} \left[1 - \frac{1}{2}\delta\right]|\mathrm{False}) + \frac{1}{2}\delta\,|\mathrm{True})\,, & x \in L; \\ \left[1 - \frac{1}{4}\delta\right]|\mathrm{False}) + \frac{1}{4}\delta\,|\mathrm{True})\,, & x \notin L. \end{cases} \tag{62}$$

and on the other hand, the sub-machine inside the small disk on the left decides L:

$$|\bar{X}_1) \longrightarrow \begin{cases} [1-\delta]\,|\mathrm{False}) + \delta\,|\mathrm{True})\,, & x \in L; \\ \left[1 - \frac{1}{2}\delta\right]|\mathrm{False}) + \frac{1}{2}\delta\,|\mathrm{True})\,, & x \notin L. \end{cases} \tag{63}$$

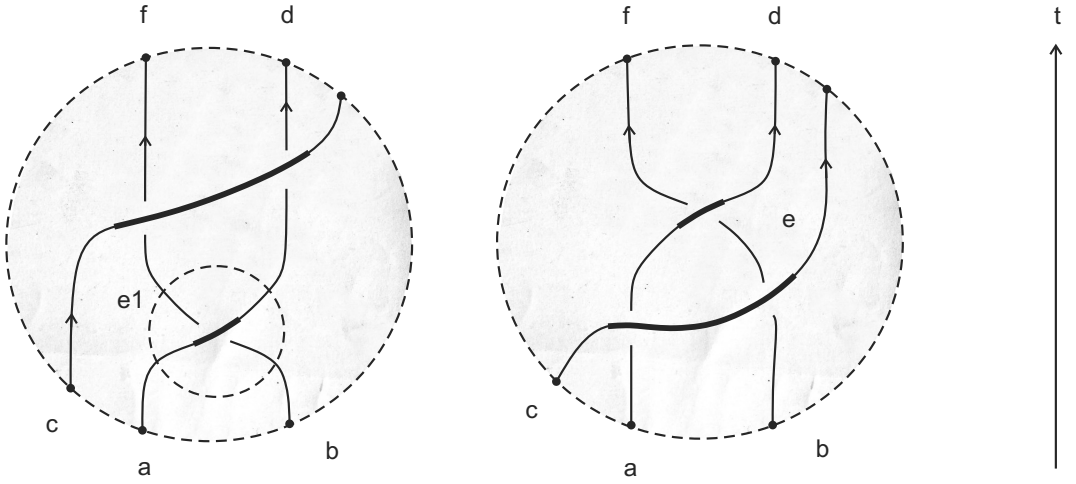


Figure 27. Example of a zero-knowledge machine.

If we further restrict the value of δ so as to satisfy Equation (60), namely,

$$|\bar{X}_1) = |X)_2 \longrightarrow \delta = \delta(1-\delta) + \frac{1}{2}\delta, \tag{64}$$

we obtain $\delta = \frac{1}{2}$, which obviously contradicts the basic requirement of deciding L. If we slightly relax this condition to allow a small discrepancy between the underlying distributions, then we may take $\delta = \frac{1}{2} + \kappa(x)$, where $\kappa(x)$ is a statistical distance, which potentially depends on x.

10.5. Equivalence for Machines with Wyes

Machines colored by quagmas as machines with wyes also have a notion of equivalence, giving them a certain flexibility as a diagrammatic language. Two trivalent machines are equivalent if they are related by a finite sequence of moves in Figures 23, 24 and 28.

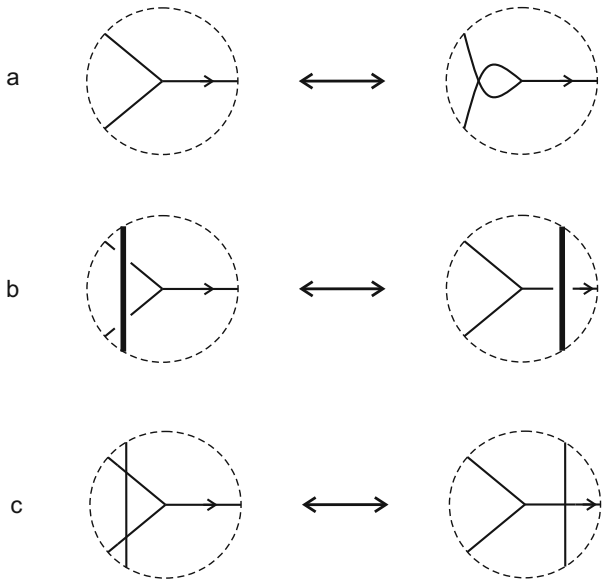


Figure 28. Local moves for wyes. Note that YR3 may reverse the label on the wye (max to min or *vice versa*), depending on what the colors are.

11. Conclusions

In this paper, we have suggested a Turing-complete diagrammatic model of computation, in which computers are drawn as tangles of decorated colored strings. With bounded resources, our "tangle machines" can decide any language in complexity IP, sometimes more efficiently than known classical single-verifier models. Our machines admit a notion of equivalence that they inherit from low-dimensional topology, with equivalent machines representing bisimilar computations. Topological invariants of our machines would be characteristic quantities for these computations, which are invariant over bisimilarity classes.

Acknowledgments

The authors thank Marius Buliga and Louis H. Kauffman for useful discussions that inspired the present note.

Author Contributions

Both authors contributed extensively to all parts of the work presented in this paper.

Conflicts of Interest

The authors declare no conflict of interest.

References

1. Carmi, A.Y.; Moskovich, D. Tangle machines. *Proc. R. Soc. A* **2015**, *471*, doi:10.1098/rspa.2015.0111.
2. Churchill, F.B. William Johannsen and the genotype concept. *J. Hist. Biol.* **1974**, *7*, 5–30.
3. Johannsen, W. The genotype conception of heredity. *Am. Nat.* **1911**, *45*, 129–159.
4. Carmi, A.Y.; Moskovich, D. Low dimensional topology of information fusion. In Proceedings of the 8th International Conference on Bio-inspired Information and Communications Technologies, Boston, MA, USA, 1–3 December 2014; pp. 251–258.
5. Shamir, A. IP = PSPACE. *J. ACM* **1992**, *39*, 869–877.
6. Elhamdadi, M. Distributivity in Quandles and Quasigroups. In *Algebra, Geometry and Mathematical Physics*; Springer: Berlin/Heidelberg, Germany, 2014; pp. 325–340.
7. Peirce, C.S. On the algebra of logic. *Am. J. Math.* **1880**, *3*, 15–57.
8. Kauffman, L.H. Knot automata. In Proceedings of the Twenty-Fourth International Symposium on Multiple-Valued Logic, Boston, MA, USA, 25–27 May 1994; pp. 328–333.
9. Kauffman, L.H. Knot logic. In *Knots and Applications*; Series of Knots and Everything 6; Kauffman, L., Ed.; World Scientific Publications: Singapore, 1995; pp. 1–110.
10. Buliga, M.; Kauffman, L. GLC actors, artificial chemical connectomes, topological issues and knots. In Proceedings of the Fourteenth International Conference on the Synthesis and Simulation of Living Systems, New York, NY, USA, 30 July–2 August 2013; pp. 490–497.
11. Meredith, L.G.; Snyder, D.F. Knots as processes: A new kind of invariant. 2010, arXiv:1009.2107. Available online: http://arxiv.org/abs/1009.2107 (accessed on 17 July 2015).
12. Kauffman, L.H.; Lomonaco, S.J., Jr. Braiding operators are universal quantum gates. *New J. Phys.* **2004**, *6*, doi:10.1088/1367-2630/6/1/134.
13. Nayak, C.; Simon, S.H.; Stern, A.; Freedman, M.; Sarma, S.D. Non-Abelian anyons and topological quantum computation. *Rev. Mod. Phys.* **2008**, *80*, 1083–1159.
14. Ogburn, R.W.; Preskill, J. Topological quantum computation. In *Quantum Computing and Quantum Communications*; Springer: Berlin, Germany, 1999; Volume 1509, pp. 341–356.
15. Kitaev, A.Y. Fault-tolerant quantum computation by anyons. *Ann. Phys.* **2003**, *303*, 2–30.
16. Mochon, C. Anyons from nonsolvable finite groups are sufficient for universal quantum computation. *Phys. Rev. A* **2003**, *67*, doi:10.1103/PhysRevA.67.022315.
17. Alagic, G.; Jeffery, S.; Jordan, S. Circuit Obfuscation Using Braids. In Proceedings of the 9th Conference on the Theory of Quantum Computation, Communication and Cryptography, Singapore, 21–23 May 2014; Volume 27, pp. 141–160.
18. Buliga, M. Computing with space: A tangle formalism for chora and difference. 2011, arXiv:1103.6007. Available online: http://arxiv.org/abs/1103.6007 (accessed on 17 July 2015).
19. Abramsky, S.; Coecke, B. Categorical quantum mechanics. In *Handbook of Quantum Logic and Quantum Structures*; Elsevier: Amsterdam, The Netherlands, 2009; Volume 2, pp. 261–323.

20. Baez, J.; Stay, M. Physics, topology, logic and computation: A Rosetta stone. *Lect. Notes Phys.* **2011**, *813*, 95–172.
21. Vicary, J. Higher Semantics for Quantum Protocols. In Proceedings of the 27th Annual ACM/IEEE Symposium on Logic in Computer Science, Los Alamitos, CA, USA, 25–28 June 2012; pp. 606–615.
22. Roscoe, A.W. Consistency in distributed databases. In *Oxford University Computing Laboratory Technical Monograph*; Oxford University Press: Oxford, UK, 1990; Volume PRG-87.
23. Turing, A.M. On computable numbers, with an application to the Entscheidungsproblem. *P. Lond. Math. Soc. Ser. 2* **1937**, *42*, 230–265.
24. Turing, A.M. On computable numbers, with an application to the Entscheidungsproblem: A correction. *P. Lond. Math. Soc. Ser. 2* **1938**, *43*, 544–546.
25. Hopcroft, J.E.; Motwani, R.; Ullman, J.D. *Introduction to Automata Theory, Languages, and Computation*, 2nd ed.; Addison-Wesley: Reading, MA, USA, 2001.
26. Goldwasser, S.; Micali, S.; Rackoff, C. The Knowledge complexity of interactive proof-systems. *SIAM J. Comput.* **1989**, *18*, 186–208.
27. Ben-Or, M.; Goldwasser, S.; Kilian, J.; Wigderson, A. Multi prover interactive proofs: How to remove intractability assumptions. In Proceedings of the 20th ACM Symposium on Theory of Computing, Chicago, IL, USA, 2–4 May 1988; pp. 113–121.
28. Babai, L.; Fortnow, L.; Lund, C. Non-deterministic exponential time has two-prover interactive protocols. *Comput. Complex.* **1991**, *1*, 3–40.
29. Arora, S.; Safra, S. Probabilistic checking of proofs: A new characterization of NP. *J. ACM* **1998**, *45*, 70–122.
30. Moshkovitz, D.; Raz, R. Two-query PCP with subconstant error. *J. ACM* **2010**, *57*, doi:10.1145/1754399.1754402.
31. Arora, S.; Lund, C.; Motwani, R.; Sudan, M.; Szegedy, M. Proof verification and hardness of approximation problems. *J. ACM* **1998**, *45*, 501–555.
32. Håstad, J. Some optimal inapproximability results. *J. ACM* **1997**, *48*, 798–859.
33. Khot, S.; Saket, R. A 3-query non-adaptive PCP with perfect completeness. In Proceedings of the 21st IEEE Conference on Computational Complexity, Prague, Czech Republic, 16–20 July 2006; pp. 159–169.
34. Zwick, U. Approximation algorithms for constraint satisfaction problems involving at most three variables per constraint. In Proceedings of the 9th ACM-SIAM Symposium on Discrete Algorithms, San Francisco, CA, USA, 25–27 January 1998; pp. 201–210.
35. Bar-Natan, D.; Dancso, S. Finite type invariants of w-knotted objects I: W-knots and the Alexander polynomial. 2014, arXiv:1405.1956. Available online: http://arxiv.org/abs/1405.1956 (accessed on 17 July 2015).
36. Kauffman, L.H. Virtual knot theory. *Europ. J. Combinatorics* **1999**, *20*, 663–690.
37. Clark, D.; Morrison, S.; Walker, K. Fixing the functoriality of Khovanov homology. *Geom. Topol.* **2009**, *13*, 1499–1582.

38. Buliga, M. Braided spaces with dilations and sub-riemannian symmetric spaces. In *Geometry, Exploratory Workshop on Differential Geometry and Its Applications*; Andrica, D., Moroianu, S., Eds.; Cluj Univ. Press: Cluj-Napoca, Romania, 2011; pp. 21–35.
39. Ishii, A.; Iwakiri, M.; Jang, Y.; Oshiro, K. A G–family of quandles and handlebody-knots. *Illinois J. Math.* **2013**, *57*, 817–838.
40. Przytycki, J.H. Distributivity versus associativity in the homology theory of algebraic structures. *Demonstr. Math.* **2011**, *44*, 823–869.
41. Joyce, D. A classifying invariant of knots: The knot quandle. *J. Pure Appl. Algebra* **1982**, *23*, 37–65.
42. Abramsky, S. No-cloning in categorical quantum mechanics. In *Semantic Techniques in Quantum Computation*; Mackie, I.; Gay, S., Eds.; Cambridge University Press: Cambridge, UK, 2010; pp. 1–28.
43. Krohn, K.; Maurer, W.D.; Rhodes, J. Realizing complex boolean functions with simple groups. *Inform. Control* **1966**, *9*, 190–195.
44. Barrington, D.A. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. *J. Comput. System Sci.* **1989**, *38*, 150–164.
45. Fredkin, E.; Toffoli, T. Conservative logic. *Int. J. Theor. Phys.* **1982**, *21*, 219–253.
46. Surowiecki, J. *The Wisdom of Crowds: Why the Many Are Smarter Than the Few and How Collective Wisdom Shapes Business, Economies, Societies, and Nations*; Random House LLC: New York, NY, USA, 2005.
47. Arora, S.; Barak, B. *Computational Complexity: A Modern Approach*; Cambridge University Press: Cambridge, UK, 2009.
48. Goldreich, O. A short tutorial of zero-knowledge. **2010**, in press. Available online: http://www.wisdom.weizmann.ac.il/ oded/zk-tut02.html (accessed on 17 July 2015).