Knots as processes: knot theory embeds in the π-calculus

L.G. Meredith[1] and David F. Snyder[2]

[1] Biosimilarity
505 N72nd St, Seattle, WA 98103, USA,
lgreg.meredith@gmail.com
[2] Department of Mathematics
Texas State University–San Marcos
601 University Drive
San Marcos, TX 78666
dsnyder@txstate.edu

Abstract. We exhibit an encoding of knots into processes in the π-calculus such that knots are ambient isotopic if and only their encodings are weakly bisimilar.

1 Introduction and motivation

Recent research in concurrency theory has led many to consider geometric interpretations of concurrent computation such as Goubault's investigations of higher-order automata [2] [1] and Herlihy's application of homology theory [3]. Exploiting a duality hidden within that framework, the authors have switched the roles of the domains and the tool of investigation, returning to a more traditional mathematical activity of finding invariant features of spatial entities. More specifically, in this paper we use theπ-calculusto construct an isotopy invariant of knots, exhibiting a π-calculusencoding of any given knot K as a process $[\![K]\!]_\pi$. The encoding is dependent upon the knot presentation. Nonetheless, the encoding has the property that two knots are isotopic if and only if their encodings as processes are weakly bisimilar. Our main theorem is:

Theorem 1 (Main Theorem) *Two knots, K_0 and K_1 are ambient isotopic, written here $K_0 \sim K_1$, iff their encodings as processes are weakly bisimilar, i.e.*

$$K_0 \sim K_1 \iff [\![K_0]\!]_\pi \simeq [\![K_1]\!]_\pi$$

While this makes clear that the encoding is an invariant up to isotopy, we see another motivation for investigating knots from this perspective. Bisimulation has proven to be an powerful and flexible proof principle, adaptable to a wide range of situations and admitting a few significant, potent equivalence class ("up-to") techniques [9] [8]. Thus, we seek to apply to topological inquiries what we have learned from the past several decades of investigation into notions of the equivalence of computational behavior. The dual notion, laid bare here in its simplest-to-grasp context, may bear fruit in the study of the equivalence of spaces which have simplicial models.

1.1 Paper summary

In section 2 we give a brief review of the polyadic π-calculus, immediately following that with a review of the primary results from knot theory needed to state and prove our main theorem. Next, we introduce the intuitions behind the encoding, sketching the general shape and walking through the procedure in the case of the trefoil. We follow this with a section on the details of the encoding. This puts us in a position to state and prove the main theorem in section 5. In the penultimate section we discussion some of the results following from this method of encoding, illustrating a way to interpret knot composition as parallel composition and a way to interpret the Kauffman bracket (and hence a number of other knot invariants) in this setting. In the final section we state some conclusions and foreshadow some directions for future research.

2 The π-calculus in a nutshell

While the rapidly expanding literature on mobile process calculi is both wide and deep, unfortunately it does not as of yet offer a gentle introduction for the investigator who is not already steeped in the mathematical culture of programming language semantics. Due to space constraints we cannot rectify the situation here. We will, however attempt to provide guiding intuition where possible.

2.1 Note for mathematicians

What follows is to aid mathematicians in reading the next two sections. Since Milner's seminal paper *Functions as processes* [5] [1], process calculi are typically presented in a manner mimicking the presentation of an algebra via operators, generators and relations. The grammar generating the set of processes may be seen as a generalization of generators and the structural equivalence may be seen as the relations over the freely generated set, using the allowable operation(s). [2]

In this sense, the process calculi fit loosely into the standard zoo of algebraic structures, resembling vector spaces in the sense that they are built out of two kinds of fundamental building blocks – i.e. vector spaces are built out of scalars and vectors (and the operations amongst these) while process calculi are built out of names and processes. What principally distinguishes these calculi from classical algebraic structures is the introduction of an additional relation, called reduction, that is neither a priori reflexive or symmetric, representing the evolution or dynamics of a process. Thus, these structures have an explicit representation of computation, unlike structures such as vector spaces where dynamics is introduced via functionals between such structures.

[1] which title inspires this paper's.

[2] This view is somewhat non-standard from the process calculi literature where structural equivalence is really a stepping stone and the equational theories considered are built to coincide with bisimulation.

It is noteworthy that several other 'computational calculi' also admit similar algebraic presentations. The lambda calculi, as a family, provide a canonical example: term grammar as generalized generators; alpha-equivalence as relations; and beta-reduction as the explicit account of dynamics. We submit that, by utilizing bisimulation techniques, some such computational calculi may generally constitute an interesting mine for topological invariants.

2.2 π-calculus

As mentioned above, the grammar generating the set of processes may be seen as a kind of generalized set of generators in a presentation of an algebra via generators and relations.

• summation	$N ::= \Sigma_{i\in I} x_i.A_i$
• agent	$A ::= F \mid C \mid (\nu\ \boldsymbol{x})A$
• abstraction	$F ::= (\boldsymbol{x})P \mid (\nu\ \boldsymbol{x})F$
• concretion	$C ::= [\boldsymbol{x}]P \mid (\nu\ \boldsymbol{x})C$
• process	$P, Q ::= N \mid P\vert Q \mid X\langle \boldsymbol{y}\rangle \mid (\mathsf{rec}\ X(\boldsymbol{x}).P)\langle \boldsymbol{y}\rangle \mid (\nu\ \boldsymbol{x})P$

WHAT IS REC?

Note, we identify summation over an empty index set with the null process, 0, and prefixing with a summation over a singleton index set. We adopt vector notation, $\boldsymbol{x}$, for finite sequences of names, $x_0, ..., x_{n-1}$, and define the *arity* of sequences, $|x_0, ..., x_{n-1}| = n$, extending it in a natural way to abstractions and concretions, e.g. $|(\boldsymbol{x})P| = |\boldsymbol{x}|$.

Further, we define *pseudo-application* of an abstraction to a concretion via

$$(\boldsymbol{y})P \circ (\nu\boldsymbol{v})[\boldsymbol{z}]Q \triangleq (\nu\boldsymbol{v})(P\{\boldsymbol{z}/\boldsymbol{y}\}\vert Q) \tag{1}$$

provided that $\boldsymbol{y} \cap \boldsymbol{v} = \emptyset$ and $|\boldsymbol{y}| = |\boldsymbol{z}|$.

We also adopt the following standard abbreviations.

$$x?(\boldsymbol{y}).P \triangleq x.(\boldsymbol{y})P \tag{2}$$

$$x!(\boldsymbol{y}).P \triangleq x.[\boldsymbol{y}]P \tag{3}$$

We also define:

$$X(\boldsymbol{y}) := P \triangleq (\boldsymbol{y})(\mathsf{rec}\ X(\boldsymbol{x}).P)\langle \boldsymbol{y}\rangle \tag{4}$$

2.3 Structural congruence

In keeping with the generalized generators and relations view espoused in 2.1, the structural congruence can be seen as the set of relations over the 'free' algebra generated by the grammar above.

Definition 1. *The* structural congruence, $\equiv$, *between processes is the least congruence closed with respect to alpha-renaming, satisfying the abelian monoid laws (associativity, commutativity and* 0 *as identity) for parallel composition as well as summation, and the following axioms:*

1. *the scope laws:*

$$\begin{aligned}(\nu\ x)0 &\equiv 0,\\ (\nu\ x)(\nu\ x)P &\equiv (\nu\ x)P,\\ (\nu\ x)(\nu\ y)P &\equiv (\nu\ y)(\nu\ x)P,\\ P|(\nu\ x)Q &\equiv (\nu\ x)(P|Q),\ \textit{if } x \notin \mathcal{FN}(P)\end{aligned}$$

2. *the recursion law:*

$$(\mathsf{rec}\ X(\boldsymbol{x}).P)\langle \boldsymbol{y}\rangle \equiv P\{\boldsymbol{y}/\boldsymbol{x}\}\{(\mathsf{rec}\ X(\boldsymbol{x}).P)/X\}$$

Note that the third scoping law inspires the shorthand $(\nu\ xy) \triangleq (\nu\ x)(\nu\ y)$.

2.4 Operational semantics

Finally, we introduce of the computational dynamics through the reduction relation $\rightarrow$.

$$\frac{|F| = |C|}{x.F \mid x.C \rightarrow F \circ C} \qquad (\textsc{Comm})$$

In addition, we have the following context rules:

$$\frac{P \rightarrow P'}{P|Q \rightarrow P'|Q} \qquad (\textsc{Par})$$

$$\frac{P \rightarrow P'}{(\nu\ x)\,P \rightarrow (\nu\ x)\,P'} \qquad (\textsc{New})$$

$$\frac{P \equiv P' \qquad P' \rightarrow Q' \qquad Q' \equiv Q}{P \rightarrow Q} \qquad (\textsc{Equiv})$$

We write $\Rightarrow$ for $\rightarrow^*$.

2.5 Bisimulation

The computational dynamics gives rise to another kind of equivalence, the equivalence of computational behavior. As previously mentioned this is typically captured via some form of bisimulation. The notion we use in this paper is weak barbed bisimulation [6].

Definition 2. *An agent, B, occurs* unguarded *in A if it has an occurence in A not guarded by a prefix x. A process P is observable at x, written here $P \downarrow x$, if some agent $x.A$ occurs unguarded in P. We write $P \Downarrow x$ if there is Q such that $P \Rightarrow Q$ and $Q \downarrow x$.*

Definition 3. *A* barbed bisimulation *is a symmetric binary relation $\mathcal{S}$ between agents such that $P\ \mathcal{S}\ Q$ implies:*

1. *If $P \rightarrow P'$ then $Q \Rightarrow Q'$ and $P'\ \mathcal{S}\ Q'$.*
2. *If $P \downarrow x$, then $Q \Downarrow x$.*

P is barbed bisimilar to Q, written $P \simeq Q$, if $P\ \mathcal{S}\ Q$ for some barbed bisimulation $\mathcal{S}$.

One of the principal advantages of this framework is the co-algebraic proof method for establishing bisimilarity between two processes: exhibit a bisimulation [9].

2.6 Linking abstractions and other process constructions

In the sequel we will find it useful to adapt some of Milner's additional process constructions, such as the one for linking abstractions. Given abstractions $F = (\boldsymbol{x})P$, and $G = (\boldsymbol{y})Q$ with $|F|, |G| \geq j$ we may compose them, gluing channels $x_{|F|-j+1}, ..., x_{|F|-1}$ to channels $y_0, ..., y_{i'+j}$ by

$$F \#_j G \triangleq (z_0, ..., z_{|F|+|G|-(j+1)})(F\langle z_0, ..., z_{|F|-1}\rangle | G\langle z_{|F|-j+1}, ..., z_{|F|+|G|-(j+1)}\rangle) \quad (5)$$

We may omit the subscript, $_j$, when it is clear from context. Note that the operation is associative, allowing us to write $F\#G\#H$ unambiguously.

Additionally, for an abstraction of the form, $F = (\boldsymbol{z})\Pi_{i=0}^{k} P_i\langle z_{f(i,0)}, ..., z_{f(i,n_i)}\rangle$ with f and indexing function into the formals, $\boldsymbol{z}$, of the abstraction, we write

$$F - P_j \triangleq (\boldsymbol{v})\Pi_{i=0}^{j-1} P_i\langle v_{f(i,0)}, ..., v_{f(i,n_i)}\rangle | (\boldsymbol{v})\Pi_{i=j+1}^{k} P_i\langle v_{f(i,0)}, ..., v_{f(i,n_i)}\rangle \quad (6)$$

to denote the deletion of the process P_j from the parallel composition. Thus, we have

$$F \equiv (F - P_j)\#P_j \quad (7)$$

3 Knots

In this section, we review a bit of knot theory and establish some connections useful for proving our lemmas and theorems. By a *knot*, we mean an embedding, up to isotopy equivalence, of the unit circle into three-dimensional Euclidean space. Whether two given knots are of the same isotopy class or not, and how this can be detected, is the primary interest in knot theory. The most common way of representing a knot is as a carefully chosen orthogonal projection of the knot onto a plane, where any crossing point in the resulting diagram corresponds to at most two points of the knot [4]. Reidemeister had the insight that isotopies of maps can be decomposed into a sequence of successive isotopies which, when projected, result in simple moves being performed upon a small region of the knot diagram.

3.1 Reidemeister moves

There are three basic Reidemeister moves that may be performed upon a knot projection, which we denote by $\Omega_1, \Omega_2, \Omega_3$. These are illustrated in the Fig. 1. Note that each move has an inverse move to "undo" it, and there is a mirror-image of each move.

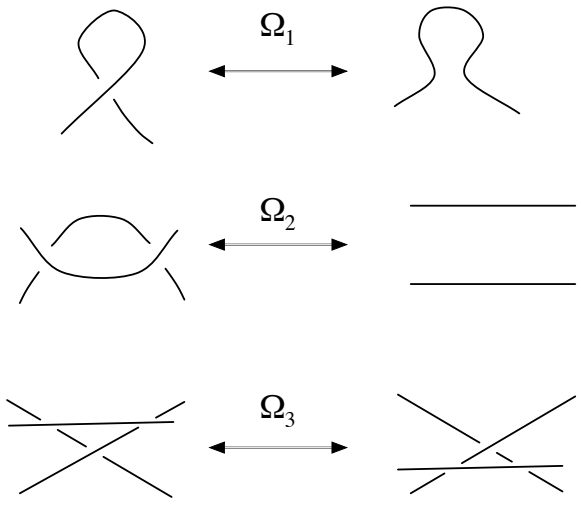

Fig. 1. The three Reidemeister moves

Reidemeister established that two knot presentations are ambient isotopic if and only if there is a sequence of Reidemeister moves transforming one presentation to the other ([7]). This allows us to establish the following useful little lemma.

We first establish a lemma to our purposes, that shows we can clean up our knot diagram in a particularly suitable way.

Given a knot K and a diagram $D(K)$ of K, we call the application of a Reidemeister move ρ a *neatening* move if the resulting diagram $D'(K)$ has a crossing number less than or equal to the crossing number of K i.e. any neatening move of the type Ω_1 or Ω_2 can occur in only one direction. Let $\hat{\rho} = \rho_1 \cdots \rho_n$ be a sequence of successive Reidemeister moves on the diagram $D(K)$ (with $n = 0$ being the identity move). We call $\hat{\rho}$ a *neatening isotopy* if either $n = 0$ or each ρ_i is a neatening move. If, in addition, at least one of the ρ_i is a move of type Ω_1 or Ω_2, we call $\hat{\rho}$ a *cleaning isotopy* of $D(K)$. If ρ_n is move of type Ω_1 or Ω_2, we call $\hat{\rho}$ a *concise* cleaning isotopy of $D(K)$

For a given diagram $D(K)$, the collection of all its neatening isotopies can be given a partial ordering: given neatening isotopies $\hat{\rho} = \rho_1 \cdots \rho_n$ and $\hat{\sigma} = \sigma_1 \cdots \sigma_m$, we put $\hat{\rho} \leq \hat{\sigma}$ when $n \leq m$ and $\rho_i = \sigma_i$ for all $i \leq n$. This partial order restricts to the the subcollection of concise cleaning isotopies.

Lemma 1. *For any knot K and knot diagram $D(K)$, we may reduce $D(K)$ via a neatening isotopy to a diagram $D'(K)$ having the property that no neatening isotopy of $D'(K)$ is a cleaning isotopy of $D'(K)$.*

Proof. If the collection of cleaning isotopies of $D(K)$ is an empty one, then $D(K)$ is the desired diagram. So assume there is at least one cleaning isotopy of $D(K)$.

The collection of concise cleaning isotopies of $D(K)$ forms a partially ordered set. Since each concise cleaning isotopy reduces the number of crossings in the diagram by at least 1, each chain in the partially order set has length bounded by the number of crossings in $D(K)$. Select any maximal element $\hat{\rho}$ of the partially ordered set. Let $D'(K)$ be the diagram derived from $D(K)$ by applying $\hat{\rho}$.

The lemma allows us to assume without loss of generality that we are working with minimal crossing diagrams.

3.2 The Dowker-Thistlethwaite code

The Dowker-Thistlethwaite code ("DT") of a knot K is obtained as follows:

Choose a point on K and begin traversing K , counting each crossing you pass through. If K has n crossings, then (since every crossing is visited twice) the count ends at $2n$. Label each crossing with the value of the counter when it is visited (each crossing is labeled twice). Finally, when labeling a crossing with an even number, prepend with the label with a minus sign if traversing "under" the crossing. All crossings end up being labeled by a pair of integers whose absolute values run, *in toto*, from 1 to $2n$. It is easy to see that each crossing is labeled with one odd integer and one even integer. For each odd integer j between 1 and $2n - 1$ inclusive, let pairedWith(j) be the even integer with which it is paired. The DT code is the sequence $1, \text{pairedWith}(1), 3, \text{pairedWith}(3), \ldots, 2n-1, \text{pairedWith}(2n-1)$.

4 The encoding

The guiding intuition at work in the encoding is to think of a knot diagram as representing the schematic of a signal flow. The crossings represent gates or circuits. The arcs between the crossings represent wires between the circuits. Thus, the whole encoding is really a problem in how to represent this signal flow as a process.

This problem then decomposes into representing the crossing circuits and wiring the circuits appropriately. As we will see in the sequel, the crossing circuit of the i-th crossing of a knot, K, is a process, $[\![C(i)]\!]_\pi(x_0, x_1, y_0, y_1, u)$, parameterized in $4+1$ ports corresponding to the in-coming and out-going arcs between each crossing together with an additional port representing the synchronizer for letting signals 'cross over' in the gates. From this perspective, the shape of the encoding may be expressed as follows.

$$[\![K]\!]_\pi = (v_0 ... v_{2n-1}) \Pi_{i=0}^{n-1} (\nu\ u) [\![C(i)]\!]_\pi (v_{\omega(i,0)}, ..., v_{\omega(i,3)}, u) \tag{8}$$

where n is the crossing number of K, and $\omega : n \times 4 \to 2n$, gives the index into the list of ports, $v_0 ... v_{2n-1}$ used to wire the crossing circuits together in a manner consistent with the chosen knot diagram of K.

4.1 An example: the trefoil as a process

Before giving a formal account of the encoding we begin with an example, showing how to represent the trefoil knot as a process.

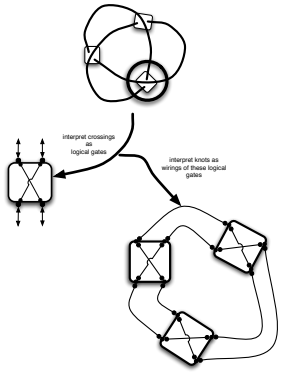

Fig. 2. The figure illustrates encoding the trefoil as a process

The natural starting point is the process encoding of a crossing circuit. To arrive at this construction we begin by observing that a wire, $W(x_0, x_1)$, between two end-points, x_0 and x_1, is coded as the process

$$W(x_0, x_1) := x_0?(s).x_1!(s).W(x_0, x_1) + x_1?(s).x_0!(s).W(x_0, x_1) \tag{9}$$

that receives a signal at either one of its end-points and sends it out the other and then resumes being a wire.

Next, we note that a crossing circuit will have somewhat more structure than a pair of wires between x_0 and y_1 and x_1 and y_0. In particular, it must somehow code what it means for a wire to cross over or under. We interpret this as a synchronization. The wire crossing over is allowed to transmit the signal without waiting while the under-crossing wire must wait for an additional input on a synchronization channel. To alert the under-crossing wire that it may now proceed, the over-crossing wire must fire off an output. Thus, a crossing circuit is coded as the following process.

$$\begin{aligned} C(x_0, x_1, y_0, y_1, u) := {} & x_1?(s).y_0!(s).(C(x_0, x_1, y_0, y_1, u)|u!) \\ & + y_0?(s).x_1!(s).(C(x_0, x_1, y_0, y_1, u)|u!) \\ & + x_0?(s).u?.y_1!(s).(C(x_0, x_1, y_0, y_1, u)) \\ & + y_1?(s).u?.x_0!(s).(C(x_0, x_1, y_0, y_1, u)) \end{aligned} \tag{10}$$

Then the process encoding of the trefoil knot is essentially a parallel composition of three crossing circuits. The main property we must ensure is that the crossing circuits are connected to each other in a way that respects the knot diagram. Additionally, we make each synchronization channel local to each crossing via a restriction on that channel.

$$\begin{aligned} [\![K_3]\!]_\pi = {} & (v_0 ... v_5)(\nu\ u_0)C(v_0, v_1, v_2, v_3, u_0) \\ & |(\nu\ u_1)C(v_2, v_3, v_2, v_4, u_5) \\ & |(\nu\ u_2)C(v_4, v_5, v_0, v_1, u_1) \end{aligned} \tag{11}$$

4.2 The general encoding

As we have already seen in the example how to code up a crossing circuit, the only thing that remains is to show how to wire the crossing circuits together in a manner consistent with a knot diagram. To do this we make use of the DT code. We recognize that the (lexicographically least) DT code is only unique for prime knots. This is not really a limitation of our approach as we will demonstrate in a subsequent section.

From a DT code, $DT(K)$, of a knot K we recover a parity reversing involution $p : 2n \to 2n$, where n is the crossing number of K, as well as an ordering on the crossings.

Observe that the process encoding of the knot will consume exactly $2n$ ports for wiring up the $x0, ..., y1$ positions of the crossing circuits.

Let $C(i)$ denote the i-th crossing of K in the ordering implied by the DT code, $DT(K)$. The constraints in the figure below (DT-constraints) uniquely

$$y0(C(i)) = x0(C(i-1)) \tag{12}$$
$$y1(C(i)) = x1(C(p(i)-1)) \tag{13}$$
$$x0(C(i)) = y0(C(i+1)) \tag{14}$$
$$x1(C(i)) = y1(C(p(i)+1)) \tag{15}$$

Fig. 3. DT-constraints

determine a wiring of the crossing circuits, expressed in the map $\omega : n \times 4 \to 2n$, that respects the connections of the crossings. where we use $s_i(C(i))$ to denote the port used in the s_i position of the encoding of the ith crossing and all arithmetic is mod $2n$.

This result may be derived from a small tweak to Scharein's derivation of the graph sometimes called the knot shadow from a DT code.

Remark 1. We observe that it is possible to compress considerably the expression of these constraints. Let s range over $\{x, y\}$, and define the involution $\hat{(-)} : \{x, y\} \to \{x, y\}$ by $\hat{x} = y$, $\hat{y} = x$, and let $\chi : \{x, y\} \to \{0, 1\}$ be defined by $\chi(x) = 1$, $\chi(y) = 0$. Our constraints may be expressed by the single equation

$$s_i(C(i)) = \hat{s_i}(C(p^i(i) + (-1)^{\chi(s)+1})) \tag{16}$$

The advantage of expressing the constraints in this more compressed way is that the two 'switching conditions' are more prominently evident. Firstly, x positions are always wired to y positions. Secondly, left-hand-side ports are always wired to predecessor (respectively sucessor) crossings using the index, while right-hand-side ports are always wired to predecessor (respectively successor) crossings using p of the index.

More generally, the encoding respects the discipline that both inter- and intra-crossing wires are always x to y while inter-crossing wires take $\_i$ positions to $\_i$ positions and intra-crossing wires take $\_i$ positions to $\_{i+1}$ positions. In some very real sense, this discipline is the essence of what it means to cross.

5 Ambient isotopy as weak bisimilarity

Theorem 1 (main). *Two knots, K_0 and K_1 are ambient isotopic, written here $K_0 \sim K_1$, iff their encodings as processes are weakly bisimilar, i.e.*

$$K_0 \sim K_1 \iff [\![K_0]\!]_\pi \simeq [\![K_1]\!]_\pi \tag{17}$$

5.1 One direction

In this section we show that if two knots are ambient isotopic then their encodings as processes are bisimilar. Since two knot(presentation)s are ambient

isotopic if and only if there is a sequence of Reidemeister moves transforming one presentation to the other ([7]), it suffices to show that the encodings of the Reidemeister moves as operations on processes preserve bisimilarity.

Lemma 2 (Reidemeister preserves bisimilarity). *For each Reidemeister move, Ω_i if $K \xrightarrow{\Omega_i} K'$ then $[\![K]\!]_\pi \simeq [\![K']\!]_\pi$.*

Encoding the Reidemeister moves We need to add a few additional processes to our toolbelt. The next one is a short-circuit. Its primary difference in behavior from a wire is that it is also willing to offer a communication on the synchronizer channel associated with a crossing circuit. As we will see this behavior is useful for composing with a crossing circuit to turn it into a wire.

$$\begin{aligned} S(x_0, x_1, u) := x_0?(s).x_1!(s).S(x_0, x_1, u) + x_1?(s).x_0!(s).S(x_0, x_1, u) \\ +u!.S(x_0, x_1, u) \end{aligned} \tag{18}$$

Ω_1 We give $[\![L(\Omega_1)]\!]_\pi$ (resp. $[\![R(\Omega_1)]\!]_\pi$), the process encodings of the left (resp. right) hand side of the Reidemeister move, Ω_1.

$$[\![L(\Omega_1)]\!]_\pi(x_0, x_1) := (\nu\ z_0 z_1 u_0)(S(z_0, z_1, u_0)|C(z_0, z_1, x_0, x_1, u_0)) \tag{19}$$
$$[\![R(\Omega_1)]\!]_\pi(x_0, x_1) := W(x_0, x_1) \tag{20}$$

It is straightforward to calculate the bisimulation that verifies

$$[\![L(\Omega_1)]\!]_\pi(x_0, x_1) \simeq [\![R(\Omega_1)]\!]_\pi(x_0, x_1) \tag{21}$$

Ω_2

$$\begin{aligned} [\![L(\Omega_2)]\!]_\pi(x_0, x_1, y_0, y_1) := (\nu\ z_0 z_1 u_0)(C(x_0, x_1, z_0, z_1, u_0) \\ |C(z_0, z_1, y_0, y_1, u_0)) \end{aligned} \tag{22}$$
$$[\![R(\Omega_2)]\!]_\pi(x_0, x_1, y_0, y_1) := W(x_0, y_0)|W(x_1, y_1) \tag{23}$$

Likewise that

$$[\![L(\Omega_2)]\!]_\pi(x_0, x_1, y_0, y_1) \simeq [\![R(\Omega_2)]\!]_\pi(x_0, x_1, y_0, y_1) \tag{24}$$

is a straightforward calculation.

Ω_3

$$\begin{aligned} [\![L(\Omega_3)]\!]_\pi(x_0, x_1, y_0, y_1, w_0, z_0) := (\nu\ u_0 v_0 w_1 z_1)(C(x_0, x_1, v_0, z_1, u_0) \\ |C(w_0, v_0, z_0, w_1, u_0) \\ |C(w_1, z_1, y_0, y_1, u_0)) \end{aligned} \tag{25}$$

$$\begin{aligned}[\![R(\Omega_3)]\!]_\pi(x_0,x_1,y_0,y_1,w_0,z_0) := (\nu\ u_0v_0w_1z_1)(&C(x_0,x_1,v_0,z_1,u_0)\\ &|C(v_0,w_1,y_0,y_1,u_0)\\ &|C(z_1,w_0,w_1,z_0,u_0)) \qquad (26)\\ & \qquad (27)\end{aligned}$$

Again, straightforward, if tedious calculation will show that

$$[\![L(\Omega_2)]\!]_\pi(x_0,x_1,y_0,y_1,w_0,z_0) \simeq [\![R(\Omega_2)]\!]_\pi(x_0,x_1,y_0,y_1,w_0,z_0) \qquad (28)$$

Remark 2. Up to this point, when referring to the encoding of a knot as a process, we have been ignoring the fact that a given knot may have many diagrams. Because the Reidemeister moves (as operations on processes) preserve bisimilarity this ignorance can remain a blissful state.

5.2 The other direction

The intuitive argument runs as follows. Suppose K_0, K_1 such that $[\![K_0]\!]_\pi \sim [\![K_1]\!]_\pi$. For any crossing $C_i(x_0,x_1,y_0,y_1,u)$ of $[\![K_{i \bmod 2}]\!]_\pi$ there are exactly four transitions possible. W.l.o.g. we may consider only two of them: one for the over-crossing and one for the under-crossing. Of the two, only one of them (the over-crossing transition) immediately enables a transition in another crossing. Because of the DT-constraints we know that only one such crossing is so enabled, and only one of its transition is enabled. We think of this as the 'successor' crossing of the current one. Note that the DT-constraints do not determine whether the over- or under-crossing is enabled, but *eventually* (i.e. in the presence of at least two signals or pulses to the knot-process) the successor crossing enables its successor. The DT-constraints ensure that we may continue in this way visiting every crossing exactly twice.

Because $[\![K_i]\!]_\pi \sim [\![K_{i+1 \bmod 2}]\!]_\pi$, $[\![K_{i+1 \bmod 2}]\!]_\pi$ must mimick the path we have traced. Further, bisimulation is symmetric and so our argument may be run from $[\![K_{i+1 \bmod 2}]\!]_\pi$ to $[\![K_i]\!]_\pi$. Thus, the bisimulation establishes a bijection between the crossings of K_0 and K_1 that preserves the polarity (i.e. over/under relationships) and the connections between the crossings. Thus, K_0 is in the same ambient isotopy class as K_1.

6 Discussion

6.1 From prime knots to composites via parallel composition

We can compose knots in the process representation by rewiring crossings in a manner that respects the DT-constraints.

Thus if we have

$$[\![K_0]\!]_\pi = (v_0 ... v_{2n_0-1}) \Pi_{i=0}^{n_0-1} (\nu\ u)[\![C_0(i)]\!]_\pi(v_{\omega(i,0)}, ..., v_{\omega(i,3)}, u) \qquad (29)$$

$$[\![K_1]\!]_\pi = (x_0 ... x_{2n_1-1}) \Pi_{i=0}^{n_1-1} (\nu\ u)[\![C_1(i)]\!]_\pi(x_{\omega(i,0)}, ..., x_{\omega(i,3)}, u) \qquad (30)$$

and we want to wire $[\![C_0(k)]\!]_\pi(v_{\omega(k,0)}, v_{\omega(k,1)}, v_{\omega(k,2)}, v_{\omega(k,3)}, u_i)$ to $[\![C_1(k')]\!]_\pi(x_{\omega(k',0)}, x_{\omega(k',1)}, x_{\omega(k',2)}, x_{\omega(k',3)}, u_j)$ we can construct a new knot

$$\begin{aligned}
[\![K_0 + K_1]\!]_\pi = (w_0 w_1)(v_0 ... \widehat{v_{\omega(i,2)}} ... \widehat{v_{\omega(i,3)}} ..., v_{2n_0-1})(x_0 ... \widehat{x_{\omega(i,0)}} ... \widehat{x_{\omega(i,1)}} ... x_{2n_1-1}) \\
\Pi_{i=0}^{k-1}(\nu\ u)[\![C_0(i)]\!]_\pi(v_{\omega(i,0)}, ..., v_{\omega(i,3)}, u) \\
|\Pi_{i=k+1}^{n_0-1}(\nu\ u)[\![C_0(i)]\!]_\pi(v_{\omega(i,0)}, ..., v_{\omega(i,3)}, u) \\
|[\![C_0(k)]\!]_\pi(v_{\omega(k,0)}, v_{\omega(k,1)}, w_0, w_1, u_i)|[\![C_1(k')]\!]_\pi(w_0, w_1, x_{\omega(k',2)}, x_{\omega(k',3)}, u_j) \\
|\Pi_{i=0}^{k'-1}(\nu\ u)[\![C_1(i)]\!]_\pi(x_{\omega(i,0)}, ..., x_{\omega(i,3)}, u) \\
|\Pi_{i=k'+1}^{n_1-1}(\nu\ u)[\![C_1(i)]\!]_\pi(x_{\omega(i,0)}, ..., x_{\omega(i,3)}, u) \qquad (31)
\end{aligned}$$

where $\widehat{\alpha_{\omega(a,b)}}$ denotes the deletion of that name from the sequence.

Taking advantage of the compositional nature of the encoding, and recalling equation 7 in 2.6 we see that

$$[\![K]\!]_\pi \equiv ([\![K]\!]_\pi - [\![C(j)]\!]_\pi) \# [\![C(j)]\!]_\pi \qquad (32)$$

(eliding the restriction scope on $[\![C(j)]\!]_\pi$ and its arguments).

Similarly, we can provide a more compact notation for wiring two crossing circuits together.

$$\begin{aligned}
[\![C_0(k)]\!]_\pi \backsim [\![C_1(k')]\!]_\pi \\
\triangleq \\
(w_0 w_1)[\![C_0(k)]\!]_\pi(v_{\omega(k,0)}, v_{\omega(k,1)}, w_0, w_1, u_i) \\
|[\![C_1(k')]\!]_\pi(w_0, w_1, x_{\omega(k',2)}, x_{\omega(k',3)}, u_j) \qquad (33)
\end{aligned}$$

allowing us to write 31 as

$$\begin{aligned}
[\![K_0 + K_1]\!]_\pi \\
= \\
([\![K_0]\!]_\pi - [\![C_0(k)]\!]_\pi) \# ([\![C_0(k)]\!]_\pi \backsim [\![C_1(k')]\!]_\pi) \# ([\![K_1]\!]_\pi - [\![C_1(k')]\!]_\pi) \qquad (34)
\end{aligned}$$

This construction not only illustrates, as previously promised, that the method of encoding is not restricted to prime knots, but that there is a good conceptual fit between the domain and the representation: the notion of composition of knots lines up with a compositor, namely parallel composition, of processes. It should be mentioned in this connection that there is nothing in the proof of the main theorem that in any way depends on primality of the knots being compared.

6.2 Old invariants from new: the Kauffman bracket

We also observe that the representation is particularly natural for working with skein relations as in the Kauffman bracket. In particular, we may recursively define a function, $\langle\!\langle - \rangle\!\rangle$, taking a process, P in the image of our encoding of knots to a Laurent polynomial (the Kauffman bracket of the knot P encodes).

To make the encoding more intuitive and well-aligned with the standard definition it will be useful to add a few more items to our toolbox. First, we have the following operators on crossing circuits

$$\begin{aligned} C^{\asymp}(j) &\triangleq W(v_{\omega(j,0)}, v_{\omega(j,1)}) | W(v_{\omega(j,2)}, v_{\omega(j,3)}) \\ C^{)(}(j) &\triangleq W(v_{\omega(j,0)}, v_{\omega(j,2)}) | W(v_{\omega(j,1)}, v_{\omega(j,3)}) \end{aligned} \tag{35}$$

representing the two ways to wire the ports of the crossing circuit to avoid the crossing.

Next, we characterize a *cycle*, which will represent for us the class of processes corresponding to the unknot, as any process R, structurally equivalent to a process of the form $\Pi_{i=0}^{k-1} W(v_i, v_{i+1 \bmod k})$, for some k.

Then our function, $\langle\!\langle - \rangle\!\rangle$, is determined by the constraints

1.
$$\langle\!\langle (K - j) \# C(j) \rangle\!\rangle = a \langle\!\langle (K - j) \# C^{\asymp}(j) \rangle\!\rangle + a^{-1} \langle\!\langle (K - j) \# C^{)(}(j) \rangle\!\rangle \tag{36}$$

2. for any cycle, R, that
$$\langle\!\langle (v_0 ... v_{2n-1}) R \rangle\!\rangle = 1 \tag{37}$$

3.
$$\langle\!\langle (v_0 ... v_{2n-1})(P|R) \rangle\!\rangle = (-a^2 - a^{-2}) \langle\!\langle (v_0 ... v_{2n-1}) P \rangle\!\rangle \tag{38}$$

corresponding to the usual equations defining the Kauffman bracket:

1. $\langle \times \rangle = a \langle \;) \; (\; \rangle + a^{-1} \langle \cup \rangle$
2. $\langle \bigcirc \rangle = 1$
3. $\langle \, L \sqcup \bigcirc \rangle = (-a^2 - a^{-2}) \langle \, L \, \rangle$

7 Conclusions and future work

7.1 Generalizing crossings and virtual knots

Kauffman posits an intriguing new member to the knot family by virtualizing the crossings in a knot diagram. We note that while we arrived at this encoding before becoming aware of Kauffman's work, this line of investigation is very much in line with the intuitions guiding the encoding presented here. Specifically, we see no reason why the crossing circuit cannot be any π-calculus process that respects the interface of crossing circuit presented here.

7.2 Braids and tangles

As may be seen from the encoding of the Reidemeister moves, nothing in this approach restricts it to knots. In particular, the same techniques may be lifted to braids and tangles.

7.3 Knot invariants as program invariants

To what class of programs do knots belong? Can various knot invariants be usefully employed as quick calculations of program equivalence for that class of programs?

7.4 Other calculi, other bisimulations and geometry as behavior

Of course, the astute reader may have noticed that the encoding described here is not much more than a linear notation for a minimal graph-based representation of a knot diagram. In this sense, there is nothing particularly remarkable about the representation. Though expressed in a seemingly idiosyncratic way, it's more or less the same information that any freely downloadable program for calculating knot polynomials uses routinely. What is remarkable about this representational framework is that it enjoys an *independent* interpretation as the description of the behavior of concurrently executing processes. Moreover, the notion of the equivalence of the behavior of two processes (in the image of the encoding) coincides exactly with the notion of ambient isotopy. It is the precise alignment of independently discovered notions that often indicates a phenomena worth investigating.

This line of thought seems particularly strengthened when we recall, as we did in the introduction, the π-calculus is just one of many 'computational calculi' that may be thought of as an *algebra+computational dynamics* and that virtually every such calculus is susceptible to a wide range of bisimulation and bisimulation up-to techniques. As such, we see the invariant discussed here as one of many potential such invariants drawn from these relatively new algebraic structures. It is in this sense that we see it as a new kind of invariant and is the inspiration for the other half of the title of this paper.

Finally, if the reader will permit a brief moment of philosophical reflection, we will conclude by observing that such a connection fits into a wider historical context. There is a long-standing enquiry and debate into the nature of physical space. Using the now familiar signposts, Newton's physics – which sees space as an absolute framework – and Einstein's – which sees it as arising from and shaping interaction – we see this connection as fitting squarely within the Einsteinian weltenschauung. On the other hand, we cannot help but notice that unlike the particular mathematical framework in which Einstein worked out his programme – a framework that required continuity – behavior and the implied notions of space and time are entirely discrete in this setting, built out of names and acts of communication. In this light we look forward to revisiting the now well-established connection between the various knot invariants such as the Jones polynomial and quantum groups.

Acknowledgments. The author wishes to acknowledge his longstanding debt to Samson Abramksy for making so accessible his foundational insights into the Curry-Howard isomorphism.

References

1. Eric Goubault. Geometry and concurrency: a user's guide. *Mathematical Structures in Computer Science*, 10(4):411–425, 2000.
2. Eric Goubault and Thomas P. Jensen. Homology of higher dimensional automata. In Rance Cleaveland, editor, *CONCUR*, volume 630 of *Lecture Notes in Computer Science*, pages 254–268. Springer, 1992.
3. Maurice Herlihy, Sergio Rajsbaum, and Mark R. Tuttle. An overview of synchronous message-passing and topology. *Electr. Notes Theor. Comput. Sci.*, 39(2), 2001.
4. Charles Livingston. *Knot Theory*. Carus Mathematical Monographs. MAA, 1993.
5. Robin Milner. Functions as processes. *MSCS*, 2(2):119–141, 1992.
6. Robin Milner. The polyadic π-calculus: A tutorial. *Logic and Algebra of Specification*, Springer-Verlag, 1993.
7. V. V. Prasolov and A. B. Sossinsky. *Knots, links, braids and 3-manifolds*, volume 154 of *Translations of Mathematical Monographs*. American Mathematical Society, Providence, RI, 1997. An introduction to the new invariants in low-dimensional topology, Translated from the Russian manuscript by Sossinsky [Sosinskiĭ].
8. Davide Sangiorgi. On the proof method for bisimulation (extended abstract). In Jirí Wiedermann and Petr Hájek, editors, *MFCS*, volume 969 of *Lecture Notes in Computer Science*, pages 479–488. Springer, 1995.
9. Davide Sangiorgi. Bisimulation: From the origins to today. In *LICS*, pages 298–302. IEEE Computer Society, 2004.