Knots as processes: a new kind of invariant

L.G. Meredith and David F. Snyder

Partner, Biosimilarity LLC
505 N72nd St, Seattle, WA 98103, USA,
lgreg.meredith@biosimilarity.com

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.

Keywords: Process calculi, knots, invariants

1. Introduction

Knot and link tabulation continues to be a lively area of scientific research and promises to be useful to areas of science such as quantum computing and DNA unpacking [?] [?] [?]. The past 20 years have seen major advances in knot classification, the development of knot invariants, and computational methods for tabulating knots and links. There are tables of the alternating prime knots of up to 23 crossings [?] [?] [?] [?]. While current algorithms provide complete tables of knots (and links), winnowing these tables of duplicates is a time consuming task. Moreover, as knot tables have proved of use to researchers in genetics, and may prove to be to researchers in quantum computation, the need to search these tables in meaningful ways presents itself. For example, of the 4,976,016,485 prime, non-oriented, alternating knots with minimal crossing number of 22, which contain the tangle corresponding to 5/3 (if any)? Of course, knot invariants are useful to distinguish knots, but few provide the basis for a formal language of knot properties with which to identify classes of knots *via* logical expressions in the language.

1.1. *Summary of contributions and outline of paper*

Here we present a newly-found strong knot invariant that does give rise to such a formal language of knot properties. Knots are invariantly associated to expressions in Milner's π-calculus [?], a member of a family of dynamic calculi known as the mobile process calculi. These calculi were developed for the analysis of concurrent computation [?] [?]. The invariant introduced here interprets knot equivalence up to ambient isotopy as an equivalence of process dynamics known as bisimulation [?] [?] in the concurrency theory literature. Of critical importance, and somewhat surprisingly, these two notions of equivalence correspond exactly on the image of the encoding: the main result of this paper is that two knots are ambient isotopic if, and only if, their images under the encoding are weakly bisimilar.

Building on this result, we observe that dual to the process calculi are a family of logics, the Hennessy-Milner logics (HMLs), providing a logical language capable of the classification of processes via logical properties. Factoring this capability through our encoding leads to the development of a logical language identifying classes of knots with logical properties. In particular, the spatial logics discovered by Caires and Cardelli [?] constitute a particularly interesting sub-family of the HMLs having logical connectives enabling us to take full advantage of key features of our encoding. We illustrate the application of the logic in [?] primarily *via* examples of predicates that select features of knots.

The paper is organized as follows. First, we provide a brief overview of knot presentations relevant to this paper, given in the context of the history of knot tabulation. Next, we give a brief overview of process calculi, highlighting achievements in that field pertinent to this paper. Section 5 begins the technical exegesis with an introduction to Milner's polyadic π-calculus via demonstration of process expressions that, much like the combinators of Conway's tangle calculus, reflect aspects of knot structure (crossings *etc.*). The section concludes with an example encoding of the trefoil knot. Section 5.5 gives a condensed but complete formal presentation of the process calculus. The main theorem is proved in section 6. This is followed by an account of spatial logic via knots in section 7 and how this can be applied in the study and inquiry of knot tables, with potential application in knot tabulation algorithms themselves. Finally, new techniques are judged not solely by the questions they answer but by the new questions to which they give rise. In the conclusion, we identify areas of further investigation.

2. Graphs

Definition 2.1. A *graph* consists of the following data $(V, E, \mathsf{src}, \mathsf{trgt})$:

(1) a set, V, of *vertices* (alternately, *nodes*);
(2) a set, E, of *edges* (alternately, *arrows* or *links*);
(3) a map, $\mathsf{src} : E \to V$, determining the *source* of an edge;
(4) a map, $\mathsf{trgt} : E \to V$, determining the *target* of an edge;

This definition is general enough to capture both directed and undirected graphs. In particular, nothing in the definition dictates that an edge is oriented from source to target. Rather, each edge, e, has two endpoints, source and target, accessed by the functions, src, and trgt, respectively. And, speaking of accessing, we will avail ourselves of standard 'accessor' function notation. Thus, if $G = (V, E, \mathsf{src}, \mathsf{trgt})$, then $V(G) = V$ and $E(G) = E$, etc.

Example 2.2.

(1) The *discrete graph*, $G(n)$, is given by $(\{0, ..., n-1\}, \emptyset, \perp, \perp)$ where $\perp$ is the function with empty domain.
(2) A *chain*, $C(n)$, is given by
    (a) $V(C(n)) = \{0, ..., n-1\}$;
    (b) $E(C(n)) = \{e_0, ..., e_{n-2}\}$;
    (c) $\mathsf{src}(e_i) = e_{i-1}, 0 \leq i \leq n-2$;
    (d) $\mathsf{trgt}(e_i) = e_{i+1}, 0 \leq i \leq n-2$.
(3) A *Loop*, $L(n)$, is constructed from a chain, $C(n)$, by adding an edge, e_{n-1} with $\mathsf{src}(e_{n-1}) = e_{n-2}$ and $\mathsf{trgt}(e_{n-1}) = e_0$.

Example 2.3. A *knot* is an embedding, $K : S^0 \to \mathbb{R}^3$, of the circle into 3-space. A *knot diagram* is a projection of a knot onto the plane such that every crossing is of the form .... A *knot shadow* is a graph derived from a knot diagram by associating vertices to crossings and edges between connected crossings. Note that a knot shadow is always a tetravalent graph, i.e. for any vertex, v, the cardinality of the cone of v is $|\mathsf{e}(v)| = 4$.

Example 2.4.

$$E + S \rightleftharpoons ES \to E + P \tag{2.1}$$

Definition 2.5. A *subgraph*, H, of a graph, G, is a graph such that

(1) $V(H) \subseteq V(G)$;
(2) $E(H) \subseteq E(G)$;
(3) $\mathsf{src}(H) = \mathsf{src}(G) \backslash E(H)$;
(4) $\mathsf{trgt}(H) = \mathsf{trgt}(G) \backslash E(H)$;

More generally, given graphs, G, H, a graph *homomorphism*, $m : G \to H$, from G to H is a pair of functions, $m_V : V(G) \to V(H)$, $m_E : E(G) \to E(H)$ such that $\mathsf{src}(m_E(e)) = m_V(\mathsf{src}(e))$ and $\mathsf{trgt}(m_E(e)) = m_V(\mathsf{trgt}(e))$. And a graph *isomorphism* is a graph homomorphism in which the vertex and edge maps are bijections.

As usual, we overload $\subseteq$ for the subgraph relation. Thus, $G(n) \subseteq C(n) \subseteq L(n)$ holds for the examples above. We write $G \sim H$ when there exists a graph isomorphism between G and H.

Definition 2.6. A *path*, p, in graph, G, is an ordered collection of edges, $p = (e_0, ..., e_N)$, such that $\mathsf{src}(e_i) = \mathsf{trgt}(e_{i-1})$. We use the notation $p[i]$ to access the ith edge in the path, p, and $|p|$ to denote the length of the path. Like an edge, a path, p, has a source, $\mathsf{src}(p)$, and a target, $\mathsf{trgt}(p)$, calculated as $\mathsf{src}(p) = \mathsf{src}(p[0])$, and $\mathsf{trgt}(p) = \mathsf{trgt}(p[|p| - 1])$.

Definition 2.7. A *connected component*, C, of a graph, G, is a subgraph of G such that there is a path between every distinct pair of vertices. That is, for all v_0, v_1 such that $v_0 \neq v_1$ there exists a p in C such that either $\mathsf{src}(p) = v_0$ and $\mathsf{trgt}(p) = v_1$ or $\mathsf{src}(p) = v_1$ and $\mathsf{trgt}(p) = v_0$.

Remark 2.8. Note that while chains and loops have a single connected component the discrete graph $G(n)$ enjoys n connected components.

Definition 2.9. The *slice* of a vertex, v, in a graph, G, is given by

$$\mathsf{slice}_G(v) \triangleq \{e \mid \mathsf{src}(e) = v\} \tag{2.2}$$

and the *co-slice* is given by

$$\mathsf{co\text{-}slice}_G(v) \triangleq \{e \mid \mathsf{trgt}(e) = v\} \tag{2.3}$$

and the *cone* is given by $\mathsf{e}(v) \triangleq \mathsf{slice}_G(v) \cup \mathsf{co\text{-}slice}_G(v)$.

As usual the subscript, G, will be dropped when context minimizes the risk of confusion.

3. Concurrent process calculi and spatial logics

In the last thirty years the process calculi have matured into a remarkably powerful analytic tool for reasoning about concurrent and distributed systems. Process-calculus-based algebraical specification of processes began with Milner's Calculus for Communicating Systems (CCS) [?] and Hoare's Communicating Sequential Processes (CSP) [?] [?] [?] [?], and continue through the development of the so-called mobile process calculi, e.g. Milner, Parrow and Walker's π-calculus [?], Cardelli and Caires's spatial logic [?] [?] [?], or Meredith and Radestock's reflective calculi [?] [?]. The process-calculus-based algebraical specification of processes has expanded its scope of applicability to include the specification, analysis, simulation and execution of processes in domains such as:

- telecommunications, networking, security and application level protocols [?] [?] [?] [?];
- programming language semantics and design [?] [?] [?] [?];
- webservices [?] [?] [?];
- and biological systems [?] [?] [?] [?].

Among the many reasons for the continued success of this approach are two central points. First, the process algebras provide a compositional approach to the specification, analysis and execution of concurrent and distributed systems. Owing to Milner's original insights into computation as interaction [?], the process calculi are so organized that the behavior —the semantics— of a system may be composed from the behavior of its components [?]. This means that specifications can be constructed in terms of components —without a global view of the system— and assembled into increasingly complete descriptions.

The second central point is that process algebras have a potent proof principle, yielding a wide range of effective and novel proof techniques [?] [?] [?] [?]. In particular, *bisimulation* encapsulates an effective notion of process equivalence that has been used in applications as far-ranging as algorithmic games semantics [?] and the construction of model-checkers [?]. The essential notion can be stated in an intuitively recursive formulation: a *bisimulation* between two processes P and Q is an equivalence relation E relating P and Q such that: whatever action of P can be observed, taking it to a new state P', can be observed of Q, taking it to a new state Q', such that P' is related to Q' by E and vice versa. P and Q are *bisimilar* if there is some bisimulation relating them. Part of what makes this notion so robust and widely applicable is that it is parameterized in the actions observable of processes P and Q, thus providing a framework for a broad range of equivalences and up-to techniques [?] all governed by the same core principle [?] [?] [?].

3.1. *The general encoding*

Given a graph, G, a pair of injective maps, $\widehat{\cdot}, \cdot^{\circ} : V(G) \hookrightarrow \mathcal{N}$, satisfying $\widehat{V(G)} \cap V(G)^{\circ} = \emptyset$, we encode G by

$$[\![G]\!]_{\pi} \triangleq \Pi_{v \in V(G)} [\![v]\!]_{\pi} \tag{3.4}$$

$$[\![v]\!]_{\pi} \triangleq \Sigma_{e \in \mathsf{slice(v)}} \hat{v}(s).\widehat{\mathsf{trgt}(e)}[s].[\![v]\!]_{\pi} + (\nu\ t) v^{\circ}(s).s[t] \tag{3.5}$$

Remark 3.1. An intuitively appealing encoding would, for each vertex, record its participation as the endpoint of an edge as potential interaction. Discrete graphs, however, have vertices that are distinguishable *despite* the absence of any edge structure. Thus, the encoding must also accomodate this additional capacity for distinguishing vertices as additional possible behavior at each process representing a vertex. This fact is methodologically noteworthy: if the mathematical data makes exploitable distinctions a faithful process representation must reflect these distinctions as distinguishable behavior. Put another way, the world-view operative in process calculi dictates that any difference that makes a difference must show up as a distinct way of kicking or prodding some process.

On the other hand, it is easy to see that

Lemma 3.2. *If G is such that for all $v \in V(G), \mathsf{e}(v) \neq \emptyset$, then for any distinct $v_0, v_1 \in V(G)$ $(\nu\ v_0^{\circ})[\![v_0]\!]_{\pi} \not\simeq (\nu\ v_1^{\circ})[\![v_1]\!]_{\pi}$.*

That is, in every graph in which every vertex participates as the endpoint of at least one edge, then even if we hide behavior at v_i°, the processes $(\nu\ v_i^{\circ})[\![v_i]\!]_{\pi}$ are distinct for distinct nodes, v_0, v_1. For, in the worst case, when the cone of v_0 equals the cone of v_1 there can never be an edge in which they are simultaneously the source or simultaneously the target. Hence, there is always some distinguishing behavior. In fact, the adventurous reader is encouraged to write down the distinguishing Hennessy-Milner formula.

This fact allows us to employ a simpler encoding for graphs of this type, i.e.

$$[\![v]\!]_\pi \triangleq \Sigma_{e \in \mathsf{slice(v)}} \hat{v}(s).\widehat{\mathsf{trgt}(e)}(s).[\![v]\!]_\pi \tag{3.6}$$

In a similar vein we have

Theorem 3.3. $[\![G]\!]_\pi \simeq [\![H]\!]_\pi \iff G \sim H$.

Proof. By construction. □

Remark 3.4. We observe that there is an dual encoding given by

$$[\![G]\!]^\bullet_\pi \triangleq \Pi_{v \in V(G)} [\![v]\!]^\bullet_\pi \tag{3.7}$$

$$[\![v]\!]^\bullet_\pi \triangleq \Sigma_{e \in \mathsf{co\text{-}slice(v)}} \widehat{\mathsf{src}(e)}(s).\hat{v}[s].[\![v]\!]^\bullet_\pi + (\nu\ t) v^\circ(s).s[t] \tag{3.8}$$

3.1.1. *Examples*

Example 3.5.

(1) $[\![G(n)]\!]_\pi = \Pi_{i=0}^{n-1} (\nu\ t) i^\circ(s).s[t]$
(2) $[\![C(n)]\!]_\pi = \Pi_{i=0}^{n-2} \hat{i}(s).\widehat{i+1}[s]$
(3) $[\![L(n)]\!]_\pi = \Pi_{i=0}^{n-1} \hat{i}(s).(\widehat{i+1\ mod}\ n)[s]$

4. A summary of knot presentations

The knot presentations we focus upon herein are the Gauss code [?], the Dowker-Thistlethwaite (DT) code [?], the Conway notation [?], and the Master Code of Rankin, Schermann and Smith [?] [?]. Other presentation schemes, such as braid representatives and Morse links, are not directly useful to the development of this paper and thus are overlooked. We also give a "wish-list" of properties one would desire from one's knot notation scheme.

4.1. *Knot presentations*

This section briefly discusses various knot notations and establishes some terminology, with references for readers needing further details.

4.1.1. *Regular knot projections*

In 1847, J.B. Listing classified knots up to 5 crossings by analyzing knot projections [?]. Listing was the first to publish an article containing a drawing of a regular knot projection (or *knot diagram* or *planar diagram*). A *knot* is taken to be a polygonal simple closed curve in 3-space. Choose any plane on which projection onto the plane results in a curve whose only singularities are transverse double points which are not the image of any vertex from the polygonal curve. Using a canonical normal vector to the plane for line of sight, for a given double point d we find the point c in the projection's preimage of d farthest from our line of sight, and we indicate an undercrossing by erasing the image of a small neighborhood of c from the planar projection. A careful presentation is given in [?].

Out of all possible regular projections of a knot, there is (at least) one with minimal number of crossings.

4.1.2. *The Gauss code.*

Gauss used regular knot projections to arrive at what is now called the Gauss Code of a knot [?]. The Gauss code is the first knot notation system. P. G. Tait used an encoding in the 1870's to classify knots up to 7 crossings [?]. Tait's encoding may be regarded as an extension to the Gauss Code. One begins somewhere on the knot projection (not at a crossing point), then proceeds along the knot applying labels to the first, third, fifth etc. crossing until all crossings are labeled; one then traverses the knot once more, writing the label of each crossing in the order that you reach it, attaching a plus or minus sign, depending on whether you are crossing over or under.

4.1.3. *Tait, Little, and Kirkman.*

Using Kirkman's classification of certain polygons [?], Tait (and Little, using similar methods) was able to tabulate knots up to 11 crossings [?] [?] [?] [?]. Tait's system included using a simple reduction rewrite strategy within his notation system. This notation was improved by Dowker and Thistlethwaite, as discussed in some detail below.

4.1.4. *Reidemeister moves.*

Reidemeister developed a reduction rewrite system for knot projections (the three Reidemeister moves) [?]. We illustrate the three moves. The convention in these (and in other graphically-based invariants) is that we show only the relevant part of the knot. In our diagrams here, we have placed dashed circles, which the reader should imagine as viewing through scope lens only a portion of the knot, with the remainder of the knot outside of the scope of view and unchanged.

The import of this system is Reidemeister's Theorem: an ambient isotopy of two knots may be exhibited as sequences of rewrites of (any of) their diagrams, and

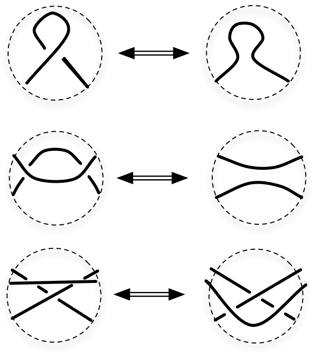

Fig. 1. The three Reidemeister moves. The topmost move will be referred to as R_1, the middle one as R_2, and the bottommost as R_3. For any one of these moves R, we write $R^{\rightarrow}(K)$ (resp. $R^{\leftarrow}$(K)) if the move has been applied left-to-right (resp. right-to-left) to the knot diagram K.

vice versa. That is, two knots are ambient isotopic if and only if a diagram of one may be rewritten, via a sequence of Reidemeister moves, to a diagram of the other [?].

4.1.5. *Knotation*

Conway developed a clever notational system for tangles, finding an algebraic-like system for this system that led to several methods of reduction rewrites [?]. He used this system to retabulate knots and links of 11 crossings by hand, in one afternoon (an effort that took Tait and Little years of work), discovering one omission and a few duplications in the process. Conway's system can be used to classify all arithmetic (also called algebraic, or rational) knots. The Conway code for a knot originally was given as a basic polyhedron followed by a sorted list of arithmetic tangles. See Conway's paper [?] for details. This system has extensions by Caudron [?] and by Bangert [?]. Conway has investigated further structures such as bangles and bracelets.

4.1.6. *Dowker-Thistlethwaite Codes (DT codes)*

Dowker created a variation of Tait's notational system that is easier to implement computationally. Dowker and Thistlethwaite made it the basis for an algorithm that successfully enumerated knots of up to 13 crossings [?]. Not every DT code is valid, i.e. an arbitrary DT code may not correspond to an actual knot, and two distinct composite knots may share the same DT Code. However, a valid DT code for a prime knot specifies the knot uniquely [?]. In their paper, Dowker and Thistlethwaite develop an algorithm to filter out invalid cases. They also give a reduction system to remove duplicates from their enumeration.

We now recall the definition of the DT code of a regular knot projection [?]. Begin at a selected point on the diagram (excepting crossing points) and traverse the diagram in a selected direction. At each encounter of a crossing, label the crossing with the next available counting number; for an even label, prepend a negative sign if traversing the overcross. Returning to the beginning point, each crossing has been labeled twice, once with an odd number and once with an even number. Let $S_{\pm}$ denote the set of labels generated.

If the knot projection has n crossings, then there are $2n$ labels (from 1 to $2n$ in absolute value) and at each crossing an unique odd counting number paired with an even integer. This induces a parity-reversing function σ on $S = \{1, \ldots, 2n\}$, where for each $i \in S$ we find the crossing with label whose absolute value is i and let $\sigma(i)$ be the absolute value of the other label at that crossing. Note that $\sigma(\sigma(i)) = i$ for any $i \in S$. Note that there is also a bijective translation $\tau : S \to S_{\pm}$ where $|\tau(s)| = s$ for all $s \in S$. Then the DT code of the knot projection is determined by $\delta = \tau \circ \sigma$, usually given as the list $\{\delta(1), \delta(3), \ldots, \delta(2n-1)\}$. Note that $\delta \circ \sigma = \tau$. Finally, for each $i \in S$, let $sgn(i)$ denote the sign of $\tau(i)$ (so $sgn(i) < 0$ implies i is even).

For each $i \in S_{\pm}$, let $C(i)$ denote the crossing with label i. Then for any $i \in S$ we know $C(i)$ is adjacent to the two crossings $C(i \pm 1)$ and $C(\delta(i) \pm 1))$, where addition is modulo $2n$. This fact is applied later when giving an explicit algorithm for encoding a knot.

4.1.7. *From Calvo to Rankin, Flint, and Schermann*

Calvo developed an inductive knot tabulation algorithm, thereby sidestepping the need to check validity of DT codes [**?**]. However, detection of duplication within the tables generated then becomes the issue. Calvo had the insight that understanding the deeper flype structure of prime, non-alternating diagram led to greater efficiencies in his algorithm. The Calvo algorithm was essentially refined in the development of a notational system by Rankin, Flint, and Schermann , based on what they call the group code which reminds one somewhat of the Gauss code but, instead, using a Conway-like insertion scheme to allow for easy reduction of flype structures in the notation [**?**] [**?**] [**?**]. This results in a master array that provides a unique identifier for prime alternating knots. This forms part of the basis for inductive construction of all prime alternating knots of crossing size n from those of crossing size $n-1$(see the above papers for details). Composite knots remain problematic, due principally to the issue of detection of duplicates.

4.2. *Desirable properties for a knot presentation system*

In this section, we list some properties that one would wish for a knot presentation system to enjoy, followed by a discussion of the knot presentations of the last subsection in the light of these wishes.

Due to complexity considerations, one may well despair of a knot notation system that would allow for mathematical classification of all knots. However, in designing a system of knot notation or presentation, informed by the history of knot tabulation and classification we can enumerate the following properties such a notation system may enjoy. These properties invariably correspond to demands of functoriality on the encoding when considered as a map from the category of knots or braids to some suitable target category while others are demands on the faithfulness (respectively, fullness) of the encoding considered as functor.

- Surjection (realizability). Each code in the notation represents a knot (or if this is not the case, those codes that do not represent a knot are easily recognizable). A code for a knot should be easily obtained from one of its planar diagrams.
- Reduction. The notation enjoys a calculus with which to reduce and simplify encodings. Each step of simplification or reduction results in an encoding representing a knot isotopy equivalent to the originating knot.
- Minimality. The notational encoding of a given knot can be reduced to a (finite non-empty set of equivalent) minimal encoding(s).
- Well-definedness. If a knot has two (or more) minimal reduced encodings, each of these encodings is equivalent to the notation for a knot equivalent *via* ambient isotopy to the original.
- Injection. Two knots that have equivalent minimal reduced encodings are ambient isotopic.
- Economy. The notation is computationally cheap and easily constructed from a diagram of the knot or from a simple code, such as the DT code.
- Compositionality. Operations on knots, e.g. knot composition, correspond to natural operations on elements in the image of the encoding.
- Separation. The notation can be used to classify a class X of knots, where X contains a previously classified class of knots (for example, arithmetic knots (also known as algebraic knots) and bracelets) but is not previously classified itself.
- Extensibility. The notation enjoys a formal language in which to describe properties and invariants of notation objects that reflect interesting properties of knots. This language should also be useful in selecting specific sets of knots (such as the set of all 21-crossing prime alternating links containing the tangle 5/3). This language ideally should be scaleable and be applicable to tables of indefinite size.

The DT code, with proper care taken, satisfies the first six properties but breaks under compositionality. Conway's tangle notation enjoys all of the properties listed, though the Conway encoding requires foreknowledge of the classes of basic polygons with n vertices. The master array of Rankin, Flint, and Schermann satisfies the first five properties, but appears unlikely to support the last four properties to a degree useful for anything other than tabulation.

4.2.1. *A meta-knotation*

Due to Reidemeister's Theorem, in order to establish our main theorem we need only consider diagrams of knots and not knots themselves. Hence we adopt the following convenient notation and terminology. The meta-variable K ranges over knot diagrams and we reserve $\mathcal{K}$ to range over knots. For simplicity, we refer to K as a knot. We write $\#(K)$ to mean the number of crossing points in the diagram K but will call this value the crossing number of the knot K, which the reader is asked not to confuse with the minimal crossing number of the knot (it is in this sense that we will later say that "two knots from the same isotopy class can have different crossing numbers"). Formally speaking $\sim_{\text{iso}}$ will denote equivalence of knots under ambient isotopy and $\sim_R$ will denote equivalence of diagrams under (finite) sequences of Reidemeister transforms. However, when there is little chance of confusion we drop the subscript and write $\sim$.

5. Knots as processes

This section bootstraps intuitions about the target calculus by introducing process expressions for key aspects of a knot's structure. An n crossing knot K is modeled as a system $[\![K]\!]$ of concurrently executing processes. More specifically, $[\![K]\!]$ is a *parallel composition* $\Pi_{i=0}^{n-1}[\![C(i)]\!]|W$ of $n+1$ processes consisting of n crossing processes $[\![C(i)]\!]$ and a process W constituting a "wiring harness." The latter process can be thought of as the computational equivalent to Conway's "basic polygon," if the knot is in minimal crossing number form. The complete expression of the encoding is

$$\begin{aligned}[\![K]\!] := (v_0 ... v_{4n-1})(\Pi_{i=0}^{n-1}(\nu\ u)[\![C(i)]\!](v_{4i}, ..., v_{4i+3}, u) \\ |\Pi_{i=0}^{n-1} W(v_{\omega(i,0)}, v_{\omega(i,1)})|W(v_{\omega(i,2)}, v_{\omega(i,3)}))\end{aligned}$$

Here, $C(i)$ represents the ith crossing in the knot diagram K. The wiring process, $\Pi_{i=0}^{n-1} W(v_{\omega(i,0)}, v_{\omega(i,1)})|W(v_{\omega(i,2)}, v_{\omega(i,3)})$, is itself a parallel composition of wire processes that correspond to edges in the 4-valent graph of the knot shadow [?] the constraints of which are reflected in the indexing function ω. The wiring process may be calculated from other knot notations. For example, we later show how the indexing function ω may be calculated from δ, the DT code of the knot projection. The crossing and wire processes have further substructure, outlined below.

Remark 5.1 (knots as abstractions). The reader familiar with process calculi will observe that the encoding actually produces an *abstraction* [?] in $4\#(K)$ names (see the first choice in the production labeled agent in the grammar of 5.5). This is actually a way of demarking that the encoding should be insensitive to the particular set of names chosen to represent the ports of the crossings. Some caution must be exercised, however, as the encoding only preserves knot structure if the abstraction is applied to a vector of $4\#(K)$ distinct names.

5.1. *Crossing processes*

A crossing is conceived in the diagram below as a black-box having four points of external interaction (*ports*) with the remainder of the knot process and as having two internal wires each connecting a pair of ports. As a process, the crossing has four possible behaviors, as shown in the defining encoding below.

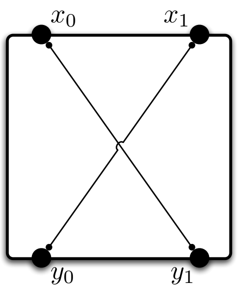

Fig. 2. Crossing process

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

A crossing process has four ports x_0, x_1, y_0, y_1 and a hidden synchronizer u. Each port has a partner port, linked as shown in the diagram (note the relationship to Conway's ± 1 tangles [**?**]). For example, the first behavior (indicated by the first term of the summand) is that the process listens at port x_1 for a signal s (which will come, if at all, via the wiring process). Having heard s, the signal is passed directly to the port y_0 where the signal is then broadcast via the wiring process. Then the process alerts the hidden synchronizer u that a signal has been passed between the ports, while concurrently preparing itself for further signal processing. The second summand represents a signal passing along the same strand in the opposite direction. The third and fourth summands are similar to the first two, except that before passing any received signal to its partner port, the process waits for a signal from the synchronizer u before allowing the signal to pass. So the role of u is that of a traffic controller who gives priority to traffic over the route between x_1 and y_0, mimicking an over-crossing.

5.2. *Wirings*

As an illustration of the expressive power of the formalism, taken together with the short description of the process calculus in the next section, the definitions below fully equip the interested reader to verify that wire processes are lossless, infinite capacity buffers.

$$
\begin{aligned}
W(x,y) &\leftarrow (\nu\ n\ m)(Waiting(x,n,m)|Waiting(y,m,n))\\
Waiting(x,c,n) &\leftarrow x?(v).(\nu\ m)(Cell(n,v,m)|Waiting(x,c,m))\\
&\quad +c?(w).c?(c).Ready(x,c,n,w)\\
Ready(x,c,n,w) &\leftarrow x?(v).(\nu\ m)(Cell(n,v,m)|Ready(x,c,m,w))\\
&\quad +x!(w).Waiting(x,c,n)\\
Cell(c,v,n) &\leftarrow c!(v).c!(n).0
\end{aligned}
$$

The ports x and y in which W is *parameterized* may be intuitively considered splice points in the knot diagram. We adopt this terminology in the sequel.

One may well wonder why perfect buffers are chosen to represent wires. For example, the following process intuitively captures a notion of wire.

$$Relay(x,y) := x?(s).y!(s).Relay(x,y) + y?(s).x!(s).Relay(x,y)$$

The problem is one of composition. Foreshadowing the method of proof, in the sequel we will need to compose wires and crossings and have the result act as a wire. For example, if $W(x,y)$ represents a candidate for wire behavior, to model the first Reidemeister move we have a demand that

$$W(y_0,y_1) \simeq (\nu\ x_0\ x_1\ w_0\ w_1)(W(y_0,w_0)|(\nu\ u)C(x_0,x_1,w_0,w_1,u)|W(x_0,x_1)|W(y_1,w_1))$$

Again, the reader may verify that while this is true of buffers, it is not true of the *Relay* process defined above.

5.3. *An example: the trefoil knot as a process*

The above intuitions are illustrated with an example, showing one way to represent the trefoil knot as a process. Following the schema above, the process encoding of the trefoil knot is a parallel composition of three crossing circuits with a wiring harness whose design ensures 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.

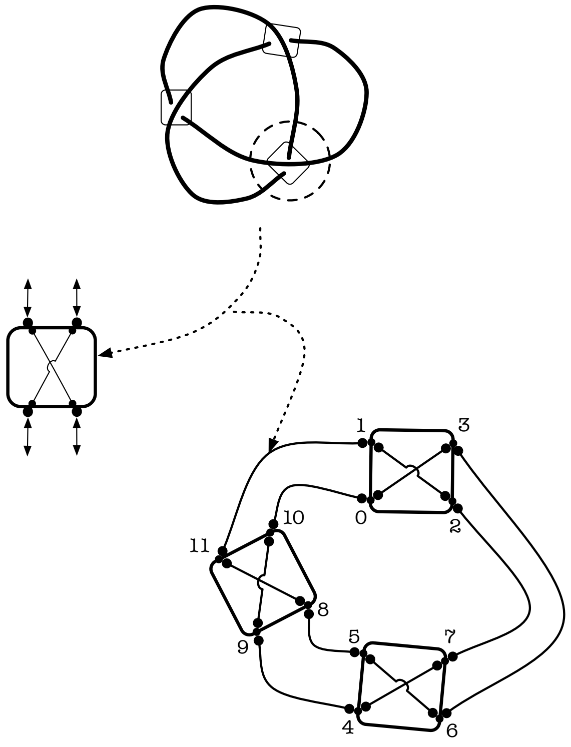

Fig. 3. The 3_1 (trefoil) knot as process. The ports in the circuit diagram have been labeled with the corresponding subscripted index in the process expression of the text.

$$\begin{aligned}
[\![3_1]\!] = (v_0...v_5)(\nu\ u_0)&C(v_0, v_1, v_2, v_3, u_0)\\
&|W(v_2, v_7)|W(v_3, v_6)\\
&|(\nu\ u_1)C(v_4, v_5, v_6, v_7, u_1)\\
&|W(v_4, v_9)|W(v_5, v_8)\\
&|(\nu\ u_2)C(v_8, v_9, v_{10}, v_{11}, u_2)\\
&|W(v_{10}, v_0)|W(v_{11}, v_1)
\end{aligned}$$

5.4. *The distinguishing power of dynamics*

In summary, to each knot $\mathcal{K}$ the encoding associates an invariant $[\![K]\!]$, an expression in a calculus of message-passing processes via an encoding of a diagram of the knot. More precisely, given the set of knot diagrams $\mathbb{K}$ and the set of processes modulo structural equivalence $\mathbb{P}$ (see section 5.5), the encoding induces a map, $[\![-]\!] : \mathbb{K} \to \mathbb{P}$.

Most importantly, the notion of equivalence of knots coincides perfectly with the notion of equivalence of processes, i.e. bisimulation (written here and in the sequel $\simeq$). Stated more formally,

Theorem 5.2 (main).

$$K_1 \sim K_2 \iff [\![K_1]\!] \simeq [\![K_2]\!].$$

In particular, in contrast to other invariants, the alignment of process dynamics with knot characteristics is what enables the invariant identified here to be perfectly distinguishing. As discussed below, among the other beneficial consequences of this alignment is the ability to apply process logics, especially the spatial sub-family of the Hennessy-Milner logics, to reason about knot characteristics and knot classes.

We see moreover the possibility of a deeper connection. As mentioned in the previous section, bisimulation has turned out to a powerful proof technique in the theory of computation adaptable to a wide range of situations and admitting a number of potent up-to techniques [**?**]. One of the central aims of this research is to broaden the domain of applicability of bisimulation-based proof methods.

5.5. *The syntax and semantics of the notation system*

Having bootstrapped an intuitive account of the target calculus *via* the encoding, we now summarize a technical presentation of this calculus. The typical presentation of such a calculus follows the style of giving generators and relations on them. The grammar, below, describing term constructors, freely generates the set of processes, $\mathcal{P}$. This set is then quotiented by a relation known as structural congruence, yielding the set $\mathbb{P}$ mentioned above in the type signature of the map constituting the encoding.

5.5.1. *Process grammar*

SUMMATION
$$M, N ::= 0 \mid x.A \mid M + N$$

AGENT
$$A ::= (\vec{x})P \mid [\vec{x}]P$$

PROCESS
$$P, Q ::= N \mid P|Q \mid X\langle \vec{y} \rangle \mid (\mathsf{rec}\ X(\vec{x}).P)\langle \vec{y} \rangle \mid (\nu\ \vec{x})P$$

Note that $\vec{x}$ denotes a vector of names of length $|\vec{x}|$. In the encodings for knots, crossings and wires given above we adopted the following standard abbreviations.

$$x?(\vec{y}).P := x.(\vec{y})P \qquad x!(\vec{y}).P := x.[\vec{y}]P \qquad X(\vec{y}) \leftarrow P := (\vec{y})(\mathsf{rec}\ X(\vec{x}).P)\langle \vec{y} \rangle$$

$$\Pi_{i=0}^{n-1} P_i := P_0 | \dots | P_{n-1}$$

5.5.2. *Structural congruence*

Free and bound names and alpha-equivalence. At the core of structural equivalence is alpha-equivalence which identifies process that are the same up to a change of variable. Formally, we recognize the distinction between free and bound names. The free names of a process, $\mathcal{FN}(P)$, may be calculated recursively as follows:

$$\mathcal{FN}(0) := \emptyset$$

$$\mathcal{FN}(x?(\vec{y}).P) := \{x\} \cup (\mathcal{FN}(P) \setminus \{\vec{y}\}) \qquad \mathcal{FN}(x!(\vec{y}).P) := \{x\} \cup \{\vec{y}\} \cup \mathcal{FN}(P)$$

$$\mathcal{FN}(P|Q) := \mathcal{FN}(P) \cup \mathcal{FN}(Q) \qquad \mathcal{FN}(P+Q) := \mathcal{FN}(P) \cup \mathcal{FN}(Q)$$

$$\mathcal{FN}((\nu\ \vec{y})P) := \mathcal{FN}(P) \setminus \{\vec{y}\}$$

$$\mathcal{FN}((\mathsf{rec}\ X(\vec{x}).P)\langle\vec{y}\rangle) := \{\vec{y}\} \cup \mathcal{FN}(P) \setminus \{\vec{x}\}$$

The bound names of a process, $\mathcal{BN}(P)$, are those names occurring in P that are not free. For example, in $x?(y).0$, the name x is free, while y is bound.

Definition 5.3. Then two processes, P, Q, are alpha-equivalent if $P = Q\{\vec{y}/\vec{x}\}$ for some $\vec{x} \in \mathcal{BN}(Q), \vec{y} \in \mathcal{BN}(P)$, where $Q\{\vec{y}/\vec{x}\}$ denotes the capture-avoiding substitution of $\vec{y}$ for $\vec{x}$ in Q.

Definition 5.4. The *structural congruence* [?] , $\equiv$, between processes is the least congruence containing alpha-equivalence, satisfying the abelian monoid laws (associativity, commutativity and 0 as identity) for parallel composition | and for summation +, in addition to the following axioms:

$$(\nu\ x)0 \equiv 0 \qquad (\nu\ x)(\nu\ x)P \equiv (\nu\ x)P \qquad (\nu\ x)(\nu\ y)P \equiv (\nu\ y)(\nu\ x)P$$

$$P|(\nu\ x)Q \equiv (\nu\ x)(P|Q),\ \textit{if}\ x \notin \mathcal{FN}(P)$$

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

5.5.3. *Operational semantics*

Finally, we introduce the computational dynamics. What marks these algebras as distinct from other more traditionally studied algebraic structures, e.g. vector spaces or polynomial rings, is the manner in which dynamics is captured. In traditional structures, dynamics is typically expressed through morphisms between such structures, as in linear maps between vector spaces or morphisms between rings. In algebras associated with the semantics of computation, the dynamics is expressed as part of the algebraic structure itself, through a reduction reduction relation typically denoted by $\rightarrow$. Below, we give a recursive presentation of this relation for the calculus used in the encoding.

$$\textsc{Comm}\quad \frac{\vec{y} \cap \vec{v} = \emptyset \qquad |\vec{y}| = |\vec{z}|}{x.(\vec{y})P \mid x.(\nu\vec{v})[\vec{z}]P \to (\nu\vec{v})(P\{\vec{z}/\vec{y}\}|Q)}$$

$$\textsc{Par}\quad \frac{P \to P'}{P|Q \to P'|Q} \qquad \textsc{Equiv}\quad \frac{P \equiv P' \qquad P' \to Q' \qquad Q' \equiv Q}{P \to Q} \qquad \textsc{New}\quad \frac{P \to P'}{(\nu\, x)P \to (\nu\, x)P'}$$

We write $\Rightarrow$ for $\to^*$, and $P \to$ if $\exists Q$ such that $P \to Q$.

In closing this summary, we take the opportunity to observe that it is precisely the dynamics that distinguishes this encoding. The equivalence that coincides with knot equivalence is a *behavioral* equivalence, i.e. an equivalence of the dynamics of processes in the image of the encoding. In a marked departure from Gauss codes or DT codes or Conway's "knotation," this facet of the encoding affords the *conflation* of notation scheme with invariant, providing a framework in which to establish the distinguishing power of the invariant and a language in which to express classes of knots as logical properties, as discussed in subsection 7.

5.5.4. *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 derived from weak barbed bisimulation [**?**]. We must introduce an "up to" [**?**] [**?**] strategy to deal with the fact that Reidemeister moves can not only introduce or eliminate crossings (see R_1 R_2), but "reorder" them (see R_3).

Definition 5.5. 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 5.6. A *barbed bisimulation* is a symmetric binary relation $\mathcal{S}$ between agents such that $P\ \mathcal{S}\ Q$ implies:

(1) If $P \to P'$ then $Q \Rightarrow Q'$ and $P'\ \mathcal{S}\ Q'$, for some 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 a barbed bisimulation $\mathcal{S}$.

5.5.5. *Contexts*

One of the principle advantages of computational calculi like the π-calculus is a well-defined notion of context, contextual-equivalence and a correlation between contextual-equivalence and notions of bisimulation. The notion of context allows the decomposition of a process into (sub-)process and its syntactic environment, its context. Thus, a context may be thought of as a process with a "hole" (written $\Box$) in it. The application of a context M to a process P, written $M[P]$, is tantamount to filling the hole in M with P. In this paper we do not need the full weight of this theory, but do make use of the notion of context in the proof the main theorem. As will be seen, the Reidmeister moves amount to decomposing the representation of the knot into some collection of crossings or wires and the rest of the knot.

SUMMATION

$$M_M, M_N ::= \Box \mid x.M_A \mid M_M + M_N$$

AGENT

$$M_A ::= (\vec{x})M_P \mid [\vec{x}]M_P$$

PROCESS

$$M_P ::= M_N \mid P|M_P \mid (\mathsf{rec}\ X(\vec{x}).M_P)\langle\vec{y}\rangle \mid (\nu\ \vec{x})M_P$$

Definition 5.7 (contextual application). Given a context M, and process P, we define the *contextual application*, $M[P] := M\{P/\Box\}$. That is, the contextual application of M to P is the substitution of P for $\Box$ in M.

Example 5.8. For example, if we take

$$\begin{aligned} M_{3_1} := (v_0 ... v_5)(\nu\ u_0)&C(v_0, v_1, v_2, v_3, u_0)\\ &|W(v_2, v_7)|\Box\\ &|(\nu\ u_1)C(v_4, v_5, v_6, v_7, u_1)\\ &|W(v_4, v_9)|W(v_5, v_8)\\ &|(\nu\ u_2)C(v_8, v_9, v_{10}, v_{11}, u_2)\\ &|W(v_{10}, v_0)|W(v_{11}, v_1) \end{aligned}$$

then $[\![3_1]\!] = M_{3_1}[W(v_3, v_6)]$

6. Main theorem: proof sketch

We have a couple of technicalities to dispatch. To motivate the first of these we wish to note that the arguments for the forward direction require some care. The aim is to capture the intuitive equivalence-preserving nature of the Reidemeister moves as corresponding bisimularity-preserving transformations on processes (in the image of the encoding). Because of the encoding of crossing information of wires in terms of synchronization of signal-flow, we have to introduce "enough signal" to keep the knot "firing", as it were, to establish that the process transformations corresponding to the knot transformations are bisimulation-preserving. Rather than seeing this extra condition as a weakness of the approach we submit that this feature provides evidence to our claim that the characterization of ambient isotopy of knots at work in this encoding is in terms of process dynamics.

We will say that the encoding of a knot is *alive* as long as it is firing, i.e. the process is enabled to make a reduction step. If it ever ceases to push signal through, i.e. process cannot make a reduction step, then it is *dead*.

We can ascertain an upperbound on initial signal that guarantees liveness of the encoding. Surely, the parallel composition of $2\#(K)$ barbs, i.e. two barbs for each crossing, will guarantee the liveness of the encoding. More declaratively, we simply demand that $[\![K]\!]|initSignal$ be live before we are willing to admit it as a representation of the knot.

Definition 6.1. More precisely, we will call the pair $([\![K]\!], initSignal)$ *alive* if $initSignal$ is an abstraction over a parallel composition of barbs, with $|initSignal| = |[\![K]\!]|$, we demand that for any vector of distinct names, v with $|v| = |[\![K]\!]|$ and any state, K', such that $[\![K]\!]\langle v\rangle|initSignal\langle v\rangle \Rightarrow K'$ we have that $K' \rightarrow$.

Different crossing numbers mean different numbers of free names. Knots in the same isotopy class may have diagrams with different numbers of crossings. Different numbers of crossings lead to different arities in the abstractions, so in interpreting these knots we haveto work to properly capture the notion of equivalence. While the precise statement is somewhat technical, the intuition is simple: it is possible to find a common “interface”, i.e. argument list, between the two knots such that restricting to that argument list obtains bisimilar processes.

Imagine (the processes that interpret) knots as programs housed in boxes with ports on the perimeter. Two knot diagrams $K_i, i \in \{1, 2\}$, from the same isotopy class may have different crossing numbers and thus their boxes have different numbers $\#(K_i)$ of ports on the outside. We can find a number of ports, call it n, somewhere between the minimal crossing number of the isotopy class and the lesser of $\#(K_i)$ such that if – using restriction – we close off $\#(K_1) - n$ ports on the first box and $\#(K_2) - n$ on the second we get two boxes that perform the same observable set of signal processing steps.

Formally, suppose $K_1 \sim K_2$ and let $\#_{Min}(K) := min\{\#(K') : K' \sim K\}$. We assert that there is an n such that $4\#_{Min}(K_1) \leq n \leq 4 * min\{\#(K_1), \#(K_2)\}$ and for any vector of names, $\vec{v}$, with $|\vec{v}| = n$ and $v[i] \neq v[j] \iff i \neq j$, there exists two vectors of names, $\vec{w_1}, \vec{w_2}$, also all distinct, such that

$$(\nu\ \vec{w_1})[\![K_1]\!]\langle \vec{v} : \vec{w_1}\rangle \simeq (\nu\ \vec{w_2})[\![K_2]\!]\langle \vec{v} : \vec{w_2}\rangle$$

with $|\vec{w_i}| = 4\#(K_i) - n$.

Prime versus composite knots. Finally, we need to say a little about how we obtain a π-calculus expression for a given knot diagram. We use another bootstrapping procedure, beginning with another knot notation (Dowker-Thistlethwaite codes is used here) and exhibiting an algorithm for calculating a process expression from the chosen notation scheme. The reader will note that DT codes are unique only when restricted to the class of prime knots. It turns out that our encoding preserves knot composition. In fact, knot composition turns out to be a specialized form of a procedure, parallel composition + hiding, long-investigated in the process-algebraic setting. So, it is sufficient to demonstrate the encoding for prime knots.

6.1. *Forward direction.* $K_1 \sim K_2 \implies [\![K_1]\!] \simeq [\![K_2]\!]$

Strategy and intuitions. Since $K_1 \sim K_2$ we know there is a sequence of Reidemeister moves converting K_1 to K_2. Each move is proved to correspond to a bisimilarity-preserving transformation on a related process. We establish context and substitution lemmas and as a consequence obtain that the process operations corresponding to the Reidemeister moves preserve bisimularity. As noted above, a small amount of bookkeeping is required to iteratively apply these transformations to mirror Reidemeister moves as applied in a proof of ambient isotopy of two knots.

We begin by observing that the intuition behind the Reidmeister moves is fundamentally about performing local operations, i.e. on some subset of wires or crossings, while leaving the rest of the process unchanged. We interpret this notion of a "local operation", say R, on a knot diagram K schematically as follows.

- factor the process, $[\![K]\!]$, as $M[\Pi_i C_i | \Pi_j W_j]$ where $\Pi_i C_i | \Pi_i W_i$ encodes the set of crossings or wires to be modified by R, i.e. the left side of the move to be performed, and M is the context representing the unchanged portion of the process;
- letting $R^{\rightarrow}(K)$ (resp. $R^{\leftarrow}(K)$) denote the left-to-right (resp. right-to-left) application of the move R to K, then $[\![R^{\rightarrow}(K)]\!]$ is calculated as $M[\Pi_{i'} C'_{i'} | \Pi_{j'} W_{j'}]$, where $\Pi_{i'} C'_{i'} | \Pi_{j'} W_{j'}$ are the processes interpreting the modified set of crossings or wires, i.e. the right side of the move to be performed.

The mathematical content of these statements is that the encoding naturally extends to an encoding $[\![R_i\{L, R\}]\!]$ of the left and right hand sides of each Reidemeister move such that $[\![K]\!] = M[[\![R_i L]\!]]$ (resp. $M[[\![R_i R]\!]]$) and $[\![R_i^{\rightarrow}(K)]\!] = M[[\![R_i R]\!]]$ (resp. $[\![R_i^{\leftarrow}(K)]\!] = M[[\![R_i L]\!]]$). Here, a picture really is worth a thousand words (see figure 5).

Example 6.2. For example, as in 5.8 taking M_{3_1} to be the encoding of $R_1(3_1, W(v_3, v_6))$, the application of R_1 to 3_1 at to the strand corresponding to the wire process $W(v_3, v_6)$ is given by $[\![R_1(3_1, W(v_3, v_6))]\!] = (\nu\ x_0\ x_1\ y_0\ y_1)M_{3_1}[((\nu\ u)C(x_0, x_1, y_0, y_1, u)|W(x_0, x_1)|W(y_0, v_3)|W(y_1, v_6))]$. See figure 4.

A certain discipline is required in the extended encoding. To establish our substitution and context lemmas we have to keep the *interface* (*i.e.* the ports in the process expression corresponding to the splice points) of the left and right hand sides of the R-move the same. So, for $R_1 L$ and $R_2 L$ we must restrict the ports that

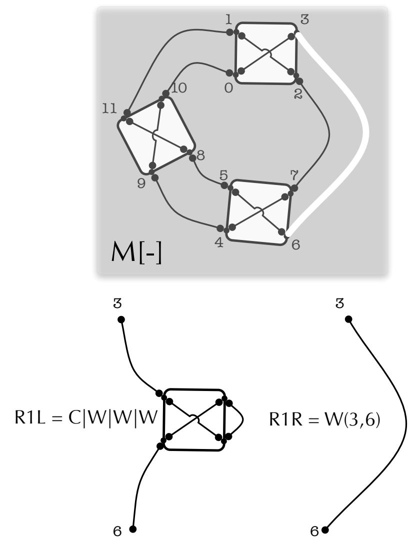

Fig. 4. The figure illustrates a Reidemeister move of the first type as reflected in the context M of the 3_1 knot, as given in Ex. 5.8.

are not the splice points. One way to address this is to embed the restrictions into the encodings of R_1L and R_2L. Algebraically,

$$\begin{aligned}
&[\![R_1L]\!](y0, y1) = \\
&(\nu\ x_0\ x_1\ w_0\ w_1)(W(y_0, w_0) \\
&\qquad |(\nu\ u)C(x_0, x_1, w_0, w_1, u)|W(x_0, x_1) \\
&\qquad |W(y_1, w_1)) \\
&[\![R_2L]\!](x_{00}, x_{01}, x_{10}, x_{11}) = \\
&(\nu\ y_{00}, y_{01}, y_{10}, y_{11}, w_{00}, w_{01}, w_{10}, w_{11})(W(x_{00}, w_{00})|W(x_{01}, w_{01}) \\
&\qquad |(\nu\ u_0)C(w_{00}, w_{01}, y_{00}, y_{01}, u0) \\
&\qquad |W(y_{00}, y_{11})|W(y_{01}, y_{10}) \\
&\qquad |(\nu\ u_1)C(w_{10}, w_{11}, y_{10}, y_{11}, u_1) \\
&\qquad |W(x_{10}, w_{10})|W(x_{11}, w_{11}))
\end{aligned}$$

Keeping to the notion of equivalence outlined in the section above, however, it will be convenient to factor out the restrictions as in the example above.

Lemma 6.3 (context). $\forall i \in \{1, 2, 3\}$ *if* $K_1 \xrightarrow{R_i} K_2$ *then there exists a context* M *and (possibly empty) vector of distinct names* $\vec{w}$ *such that*

$$\begin{aligned}
(\nu\ \vec{w})[\![K_1]\!]\langle v : w\rangle &= (\nu\ \vec{w})M[[\![R_iL]\!]] \\
[\![K_2]\!] &= M[[\![R_iR]\!]]
\end{aligned}$$

Proof. This follows directly from the definition of the encoding. □

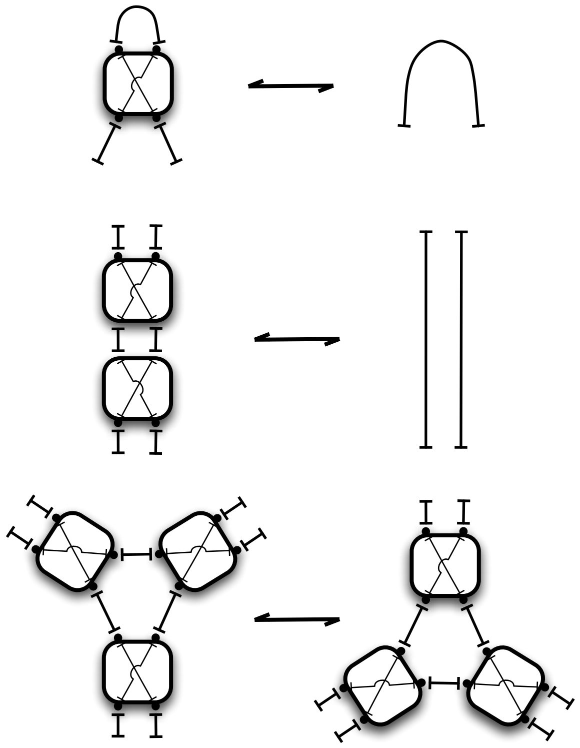

Fig. 5. Reidemeister moves as bisimilarity-preserving transformations. Cf. Fig. 1

Lemma 6.4 (substitution). *For all $i \in \{1,2,3\}$ R_iL is bisimilar to R_iR in the context of a live encoding. That is, if*

- $[\![K]\!] | initSignal$ *is alive, and*
- $[\![K]\!] | initSignal = M[[\![R_iL]\!]]$

for some context M then we can substitute $[\![R_iR]\!]$ in its place without change of behavior, i.e.

$$(\nu\ \vec{w})M[[\![R_iL]\!]] \simeq M[[\![R_iR]\!]]\ ,\ \forall i \in \{1,2,3\}$$

Proof. This follows from the definitions of $[\![R_i\{L,R\}]\!]$ plus the requirement that M derives from a live encoding. Even if one side has additional synchronizations, there is always enough signal to overcome spurious blocking. □

An immediate consequence of these two lemmas is

Lemma 6.5 (1-step). *if K_1 can be derived from K_2 by application of one Reidemeister move, then*

$$[\![K_1]\!]\langle v\rangle \simeq (\nu\ w)[\![K_2]\!]\langle v : w\rangle$$

We are almost in a position to complete the proof of the forward implication. The next step is to show that you can iterate the performance of these bisimilarity-preserving transformations in a way that precisely mimics iterated application of Reidemeister moves. Because the moves R_1 and R_2 change the number of crossings the number of ports in the interface of the (encoding of the) knot to which they are applied changes. This is the core issue in the formal statement of the theorem. Because the splice interface remains constant across the two sides of a Reidemeister move all one has to do is keep track of the ports to be hidden. The complete proof uses a case analysis of the composition of any two types of Reidemeister moves. We illustrate the analysis in the case of a simplifying (crossing-elimination) R_1 step followed by a complicating (crossing-introducing) R_2 step.

- The $R_1L \rightarrow R_1R$ step means we have a context M such that

$$\begin{aligned}(\nu\ x_0\ x_1\ w_0\ w_1)[\![K_1]\!]\langle\vec{v_0} : x_0 : x_1 : w_0 : w_1\rangle&\\ &= (\nu\ x_0\ x_1\ w_0\ w_1)M[[\![R_1L]\!]]\\ &\simeq M[[\![R_1R]\!]]\\ &= [\![K_2]\!]\langle\vec{v_0}\rangle.\end{aligned}$$

- The $R_2R \rightarrow R_2L$ step means we have a context M' such that

$$\begin{aligned}(\nu\ y_{00}\dots y_{11}\ w_{00}\dots w_{11})[\![K_3]\!]\langle\vec{v_1} : y_{00} : \dots : y_{11} : w_{00} : \dots : w_{11}\rangle&\\ &= (\nu\ y_{00}\dots y_{11}\ w_{00}\dots w_{11})M'[[\![R_2L]\!]]\\ &\simeq M'[[\![R_1R]\!]]\\ &= [\![K_2]\!]\langle\vec{v_1}\rangle.\end{aligned}$$

- Since $\vec{v_0}, \vec{v_1}$ are just lists of distinct names with $|\vec{v_0}| = |[\![K_2]\!]| = |\vec{v_1}|$, just pick $\vec{v_0} = \vec{v_1}$. Dropping the subscript, we conclude

$$\begin{aligned}&(\nu\ x_0\ x_1\ w_0\ w_1)[\![K_1]\!]\langle\vec{v} : x_0 : x_1 : w_0 : w_1\rangle \simeq\\ &\quad(\nu\ y_{00}\dots y_{11}\ w_{00}\dots w_{11})[\![K_3]\!]\langle\vec{v_1} : y_{00} : \dots : y_{11} : w_{00} : \dots : w_{11}\rangle\end{aligned}$$

- Moreover, we have $[\![K_2]\!]\langle\vec{v}\rangle$ forming the shared core.

6.2. *Reverse direction.* $K_1 \sim K_2 \Leftarrow [\![K_1]\!] \simeq [\![K_2]\!]$

We prove this by contradiction, assuming two knots in distinct isotopy classes with bisimilar images and arriving at absurdity. Not surprisingly, the argument in this direction is considerably less complicated as it is nonconstructive.

Without loss of generality (by application of the lemmas of the forward direction as needed), we assume knots are given in minimal crossing diagrams. Due to this minimality, if the crossing numbers of these diagrams are different, then the number of free names (or arity of the abstractions interpreting the knots) in $[\![K_1]\!]$ differs from those in $[\![K_2]\!]$ –contradicting bisimilarity. Thus, the crossing numbers must be the same.

Now we employ the form of the encoding: $\Pi_{i=0}^{n-1}[\![C(i)]\!](...)|\Pi_{i=0}^{n-1}W(...)|W(...) \simeq \Pi_{j=0}^{n-1}[\![C(j)]\!](...)|\Pi_{j=0}^{n-1}W'(...)|W'(...)$. Notice that a consequence of sharing the same crossing number is that the crossing process part of the two encodings is *identical.* Because bisimulation is a congruence this implies $\Pi_{i=0}^{n-1}W(...)|W(...) \simeq \Pi_{j=0}^{n-1}W'(...)|W'(...)$. This says that the only difference can be in the "wiring harnesses". Obviously, if any of these wires differ (up to α-equivalence), then there is a distinguishing barb, contradicting our assumption of bisimulation. Hence none of the wires differ, their respective sets of crossings are wired identically, which means the diagrams are identical. Thus, the knots are ambiently isotopic – a contradiction.

7. Unlikely characters: spatial logic for knots

Associated to the mobile process calculi are a family of logics known as the Hennessy-Milner logics. These logics typically enjoy a semantics interpreting formulae as sets of processes that when factored through the encoding outlined above allows an identification of classes of knots with logical formulae. In the context of this encoding the sub-family known as the spatial logics [?] [?] [?] are of particular interest providing several important features for expressing and reasoning about properties (i.e. classes) of knots. We hint here at how this may be done.

7.0.1. *Structural connectives*

The spatial logics enjoy structural connectives corresponding, at the logical level, to the parallel composition ($P|Q$) and new name ($(\nu\ x)P$) connectives for processes. As illustrated in the examples below, these connectives are extremely expressive given the shape of our encoding.

7.0.2. *Decideable satisfaction*

In [?] the satisfaction relation is shown to be decideable for a rich class of processes. It further turns out that the image of the our encoding is a proper subset of that class. This result provides the basis for an algorithm by which to search for knots enjoying a given property.

7.0.3. *Characteristic formulae*

In the same paper [?] , Caires presents a means of calculating characteristic formulae, selecting equivalence classes of processes up to a pre–specified depth limit on the support set of names. Composed with our encoding, this characteristic formula can be used to select characteristic formulae for knots.

7.0.4. *Spatial logic formulae*

The grammar below (segmented for comprehension) summarizes the syntax of spatial logic formulae. We employ illustrative examples in the sequel to provide an intuitive understanding of their meaning referring the reader to [**?**] for a more detailed explication of the semantics.

BOOLEAN	SPATIAL
$A, B ::= T \mid \neg A \mid A \wedge B \mid \eta = \eta'$	$\mid\ 0\ \mid A \vert B \mid x Ⓡ A \mid \forall x.A \mid Hx.A$

BEHAVIORAL	RECURSION	ACTION	NAME
$\mid \alpha.A$	$\mid X(\vec{u}) \mid \mu X(\vec{u}).A$	$\alpha ::= \langle x?(\vec{y})\rangle \mid \langle x!(\vec{y})\rangle \mid \langle \tau \rangle$	$\eta ::= x \mid \tau$

7.1. *Example formulae*

7.1.1. *Crossing as formula.*

$$\begin{aligned}\mu C(x_0, x_1, y_0, y_1, u).(&\langle x_0?(z)\rangle(\langle u!\rangle\langle y_1!z\rangle C(x_0, x_1, y_0, y_1, u))\\ &\wedge \langle y_1?(z)\rangle(\langle u!\rangle\langle x_0!z\rangle C(x_0, x_1, y_0, y_1, u))\\ &\wedge \langle x_1?(z)\rangle(\langle u?\rangle\langle y_0!z\rangle C(x_0, x_1, y_0, y_1, u))\\ &\wedge \langle y_0?(z)\rangle(\langle u?\rangle\langle x_1!z\rangle C(x_0, x_1, y_0, y_1, u)))\end{aligned}$$

The lexicographical similarity between the shape of this formulae and the shape of definition of the process representing a crossing reveals the intuitive meaning of this formulae. It describes the capabilities of a process that has the right to represent a crossing. For example it picks out processes that may perform an input on the port x_0 in its initial menu of capabilities. What differentiates the formula from the process, however, is that the crossing process is the smallest candidate to satisfy the formula. Infinitely many other processes – with internal behavior hidden behind this interface, so to speak – also satisfy this formula. Even this simple formula, then, can be seen to open a new view onto knots, providing a computational interpretation of *virtual* knots.

Note that this formula is derived by hand. A similar formula can be derived by employing Caires' calculation of characteristic formula [**?**] to the process representing a crossing. In light of this discussion, we let $[\![C]\!]_\phi(x0, x1, y0, y1, u)$ denote a formula specifying the dynamics we wish to capture of a crossing. To guarantee we preserve the shape of the interface and minimal semantics we demand that $[\![C]\!]_\phi(x0, x1, y0, y1, u) \Rightarrow \mathbf{C}(x0, x1, y0, y1, u)$ where $\mathbf{C}(x0, x1, y0, y1, u)$ denotes the formula above.

7.1.2. *Crossing number constraints.*

The moral content of the context lemma (Lemma 6.3) is that the notion of "locality" in the Reidemeister moves is effectively captured by the parallel composition operator of the process calculus. This intuition extends through the logic. Given a formula, $[\![C]\!]_\phi(x0, x1, y0, y1, u)$, we can use the structural connectives to specify constraints on crossing numbers, such as at least n crossings, or exactly n crossings.

AT-LEAST-N

$$K_\phi^{\geq n}(\vec{xs}, \vec{ys}) := \Pi_{i=0}^{n-1} Hu.[\![C]\!]_\phi(xs_i, ys_i, u)|T$$

EXACTLY-N

$$K_\phi^{=n}(\vec{xs}, \vec{ys}) := \Pi_{i=0}^{n-1} Hu.[\![C]\!]_\phi(xs_i, ys_i, u)|\neg(\forall x_0, y_0, x_1, y_1, u.[\![C]\!]_\phi(x_0, y_0, x_1, y_1, u)|T)$$

To round out this section, recall that the encoding of an n-crossing knot decomposes into a parallel composition of *n copies* of a crossing process together with a wiring harness. To specify different knot classes with the same crossing number amounts to specifying logical constraints on the wiring harness. In the interest of space, we defer examples to a forthcoming paper. Suffice it to say that both the conditions “alternating knot” and “contains the tangle corresponding to 5/3” are expressible. For example, it is possible to calculate the characteristic formula of a process corresponding to the tangle 5/3 and conjoin it into the classifying formula via the composition connective of the logic.

Finally, we wish to observe that it is entirely within reason to contemplate a more domain-specific version of spatial logic tailored to the shape of processes in the image of the encoding. Such a domain-specific logic would have a better claim to the title formal language of knot properties.

8. Conclusions and future work

8.1. *Knotation system properties*

Rather than list the properties of section 4.2 and tick them off, as we may do, let us take a step back. In the end it is the act of calculating over knots – tabulating, searching, making new ones from old – that places demands on the notation system used to represent them. These are the practical considerations that give rise to the properties listed in 4.2. Establishing the tight correspondence between processes in the image of the encoding and the knots they represent is ultimately in service of showing this is a reasonable proxy for knots because calculations on the representation have a correspondence to calculations on the domain. Moreover, the last few decades of logic in computer science has taught us that there is fidelity and then there is *fidelity.* It’s one thing map elements of one domain to elements of another, but it’s another to map the procedures of one domain (roughly, operations on elements) to the procedures of another. Whether one sees a demand like this through the lens of “functoriality” or through a correspondence of proofs (of say, ambient isotopy, in a proof system made of Reidmeister moves) and programs (that transform programs), the practical side of the demand is about making calculation *tractable*, so that “local” operations in one domain correspond to “local” operations in the other. This makes it possible to structure calculations in terms of the structure of the elements over which the calculations are being performed, which has far-ranging consequences on the complexity of calculation in the representative domain. Of note, we have been careful to restrict attention to the image of the encoding. We do not, as of yet, have an effective characterization of processes that will decide whether or not a process is in the image of the encoding. This restriction notwithstanding, the image is the only system thus far proposed that does enjoy all of the properties listed, including possessing (actually, being) a language for classifying knots via logical properties.

8.2. *Knot queries*

The interest in knot tabulation coming from the physical sciences is a strong motivation to investigate the design of a specialized form of spatial logic tailored to reasoning about processes in the image of this encoding. Specifically, the current authors are in the process of devising an executable knot query language that runs on top of a translation of spatial logic predicates to XQuery, the XML query language [?] [?]. In this application, a member, K, of an isotopy class $[K]_{\sim}$ is stored as an XML document $[\![[\![K]\!]]\!]_{XML}$. As the notation indicates, the document may be calculated from the process expression, $[\![K]\!]$. Pleasantly, all such documents will conform to an XML schema [?] [?] [?] that may be formally derived from the grammar for processes described in section 5.5.1. Thus, the mathematics extends to a formal specification of the software implementation of a "knot database" and the compositional nature of the specification makes tractable formal verification of the correctness of the software with regards to the specification.

8.3. *Braids, tangles, virtual knots, racks and quandles*

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. More generally, 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, that line of investigation is very much aligned to the guiding intuitions of the encoding presented here. Namely, the crossing circuit *a priori* could be any π-calculus process that respects the interface of the wiring circuit. We conjecture that many forms of *virtualization* may be faithfully interpreted as *simulation*.

Moreover, it seems likely that the rack and quandle frameworks may be mirrored in the process calculus, as these structures can be used to model the *action* of the Reidemeister moves on a knot. The relation between this type of action and the dynamics of the process calculus presented here remains to be investigated.

8.4. *Other calculi, other bisimulations and geometry as behavior*

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 appears to be nothing particularly remarkable about the representation. Though expressed in a seemingly idiosyncratic way, the encoding is built from 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 knot equivalence. 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 the π-calculus is just one of many 'computational calculi' —the λ-calculus being another paradigmatic example— that may be thought of as a *computational dynamics+algebra* 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 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 *weltanschauung.* On the other hand, unlike the particular mathematical framework of continuity in which Einstein worked out his programme, behavior and its 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 Kauffman bracket and quantum groups. Specifically, it appears that the kinematic picture of loop quantum gravity derived from spin networks can be faithfully encoded in a manner analogous to one used here to encode knots, but the process structure offers an account of dynamics somewhat different from spin foams [**?**].

9. Appendix: From DT-codes to processes

What follows is an algorithm for calculating the ω indexing function. Note that the function *over* can be implemented easily using the function δ, or the function *sgn*, described in section 4.1.6. Exercise for the reader: upgrade the algorithm so that wires are not duplicated during the process.

```
(* ------------------------------------------------------------------ *)
(*                                                                    *)
(* Given types Knot and Wire, the omega function is typed as follows  *)
(* omega : int -> (int -> int) -> (int -> int) -> Knot -> (Wire list) *)
(* With dt a table representing the Dowker-Thistlethwaite mapping     *)
(* from odds to evens, and dti the inverse. The function keeps an     *)
(* accumulator, acc, in which to collect the wires it calculates.     *)
(* The function assumes helper functions C and x0,x1,y0,y1. C takes   *)
(* and instance of type Knot and the odd index of the DT code and     *)
(* returns the crossing. The other helpers are accessors of the ports *)
(* of the crossing. The algorithm visits every crossing and so        *)
(* generates many wires more than once, which is why the accumulator  *)
(* is updated with union rather than cons on the recursive calls. As  *)
(* an exercise, modify the algorithm to avoid wire duplication.       *)
(*                                                                    *)
(* ------------------------------------------------------------------ *)

let omega i dt dti knot acc =
  if (i <= (numCrossings knot))
     then
       let ic = (2*i - 1) in
         (omega
            (i+1) dt dti knot
            (union acc
               [ W(x1(C(knot,ic)),
                 (if (over dt ic-1)
                     then y0(C(knot,ic-1))
                     else y1(C(knot,ic-1))));
                 W(y0(C(knot,ic)),
                 (if (over dt ic+1)
                     then x1(C(knot,ic+1))
                     else x0(C(knot,ic+1))));
                 W(x0(C(knot,ic)),
                 (if (over dt (dti ((dt ic)-1)))
                     then y0(C(knot,(dti ((dt ic)-1))))
                     else y1(C(knot,(dti ((dt ic)-1))))));
                 W(y1(C(knot,ic)),
                 (if (over dt (dti ((dt ic)+1)))
                     then x1(C(knot,(dti ((dt ic)+1))))
                     else x0(C(knot,(dti ((dt ic)+1)))))) ]))
     else acc
```

Acknowledgments. The authors wish to thank Matthias Radestock for suggestions for coding wires, and early advocation of the wire-saturated form the final encoding took and his probing questions regarding proof strategies in earlier drafts of this paper. One of the co-author wishes to acknowledge his longstanding debt to Samson Abramksy for making so accessible his foundational insights into the Curry-Howard isomorphism. Additionally, the authors would like to thank the reviewers of ICALP 2006 for their helpful comments on an earlier write up of these results.