★ BRIEFS ★

All Tied in Knots

David J. Lobina

In this note, I wish to critique a proposal put forward by Camps & Uriagereka (2006; C&U) and Balari *et al*. (2011; BEA) regarding the study of the evolution of language. In particular, I intend to cast doubt on the connection they draw between the computational properties of the language faculty and those involved in the conceptualization of a knot. In what follows, I will offer a rather negative commentary, in the sense that no alternative will be forthcoming. In fact, one of the main points of this paper will be that there is no phenomenon to explain at all; or at least it has not been properly formulated.

The general idea underlying what C&U and BEA propose is clear enough. Considering that the language faculty is underlain by a computational system that generates sound/meaning pairs, it ought to be possible to outline some of its computational properties. Furthermore, it is at least a possibility for some other cognitive domain of the human mind to share some of these very computational properties. If that is the case, C&M claim, such a domain would constitute a "cognitive base" that can be said to be in a "causal correlation" with the linguistic capacity (p. 35) — that is, the computational properties of such a 'base' would be parasitic on those of the language faculty. If this holds, the behaviors associated with this cognitive base could plausibly constitute indirect evidence for an "underlying linguistic prerequisite" (*ibid*.).

I pretty much doubt that such an inference is in fact sound, but let it stand for the sake of the argument. The specific behavior that engages C&U and BEA relates to the ability to tie a knot, a skill that must have originated in modern man, given that evidence for it in the fossil record — such as in the binding of projectiles to their shafts (C&U: 58) — is only present in the archaeology of Homo Sapiens (p. 45). Naturally, language and knot-tying are phenomena that at first sight appear to be completely unrelated, but C&U and BEA assure us that they may in fact share underlying properties. The overall argument is more or less as follows. According to mathematical linguistics, the expressive power of natural language is context-sensitive (or more accurately, mildly context-sensitive; Joshi *et al*. 1990); thus the computational system underlying language is of such power. Further, the mathematical structure of knots may be studied by employing the tools of Knot Theory, a subfield of mathematical topology. According to C&U, and citing a work from this literature (*viz*. Mount 1989), knots can only be created/described by a context-sensitive system, a conclusion they take to be "not subject to rational debate" (p. 63). In a related manner, BEA conclude, citing another

I am grateful to Mark Brenchley, José E. García-Albea, Aravind Joshi, and Mario Santos-Sousa for helpful comments on a longer version of this paper. Part of this research was funded by an AGAUR grant 2009 SGR-401 awarded by the Catalan administration.

study from Knot Theory (Hass *et al*. 1999), that determining whether a string is knotted or not is of a computational complexity comparable to the processing of linguistic expressions (p. 11). Given that evolution doesn't, apparently, generate identical structures (C&U: 45), the ability to entertain and create knots may indeed be parasitic on computational properties of the language faculty.

Note, first of all, that in the above paragraph there is a leap from the expressive power of a language to the computational complexity of processing it; whilst these two factors are closely related, they should not be conflated. Such computational properties point, of course, to the classification of formal grammars and languages that Chomsky (1956, 1963) delineated — the so-called Chomsky Hierarchy.[1] In those publications, Chomsky ranked different classes of formal languages (where a language is defined as a set of strings of symbols) in terms of the formal grammars (i.e. string rewriting systems) that are said to *generate* these languages. The expressive power of a grammar, then, refers to the precise set of strings that it can generate. Moreover, to say that a grammar is context-sensitive is to specify a particular set of constraints on the form of its rewriting rules that differentiates them from, for example, a context-free grammar. Concurrently, mathematical linguistics has also focused on the automata that are said to *recognize* each of the languages of the Chomsky Hierarchy (Hopcroft *et al*. 2007). In particular, it has been amply demonstrated that for each language class there is an automaton that recognizes all sets of strings of this class; in this sense, each grammar is equivalent to a specific automaton. In a perhaps more neutral vocabulary, one could state that automata and grammars *specify* languages.

Even though both automata and grammars describe the same reality — *viz*. a ranking of different language classes — there is a clear difference in perspective between employing a grammar and an automata in the study of computational properties.[2] Indeed, it is no surprise that it is the latter construct that has featured more extensively in the study of the "rate of growth of the time or space" required to solve a problem (Aho *et al*. 1974: 2); that is, the study of the computational complexity of a problem is much more amenable for study by employing abstract machine devices such as automata than it is with a grammatical

[1] It seems to me that this leap and the subsequent conflation of these two properties stems from the manner in which C&U and BEA interpret the Chomsky Hierarchy. In fact, these two publications follow the (in my opinion entirely misbegotten) 're-interpretation' of the Chomsky Hierarchy Uriagereka conducts in chapter 7 of his 2008 book. This is rather surprising, for a number of reasons. First of all, even though C&U (p. 36) state that their description of the Hierarchy is based on Uriagereka (2008) — which they define, rather conceitedly, as a "current linguistic perspective" — this book was not even published at the time. More importantly, Uriagereka himself would surely admit that his re-interpretation is not only non-standard, but a very speculative exercise indeed. Why would these scholars, then, assume its validity as a framework upon which to draw a comparison between language and knot-tying? Be that as it may, the main mistake of this re-interpretation lies in Uriagereka's belief that focusing on the different automata that specify the different language classes gives you an account of structure generation, but this is quite simply not true; see infra for more details.

[2] Hopcroft & Ullman (1969: 5) call these two perspectives the "recognition point of view" and the "generative point of view". Similarly, Wintner (2010: 17) talks of the "dual view of language".

formalism.[3] If this is so, it is the case that the preoccupations of Automata Theory revolve (mainly) around discovering the inherent computational difficulty of various problems.

Consequently, when it is stated that natural language has a particular computational complexity, this is supposed to refer to the inherent difficulty involved in processing linguistic structures. As it happens, what computational complexity is in fact involved in the processing of language is very uncertain. In a review article explicitly devoted to this question, and even though one of its section is titled "Parsing and recognition", Pratt-Hartmann (2010) focuses his attention on a much narrower issue: the recognition problem. That is, given a grammar *G* within a specific formalism *F*, the recognition problem aims to ascertain the amount of time and space that a Turing Machine would require in order to determine if a given string defined over the alphabet of *G* belongs to the language specified by *G* (Pratt-Hartmann 2010: 55). According to Pratt-Hartmann, the computational complexity of a grammar can only be determined within a specific formalism, and therefore different formalisms are likely to involve different measures of complexity. For example, the recognition of a context-free grammar specified in Chomsky normal form (see *infra*) can be achieved by the so-called CYK algorithm in time $O(mn^3)$, where *m* is the number of production rules, *n* is the length of the string, and *O* refers to the upper bound on the growth rate of this specific function (pp. 57–58). In other formalisms, the measures of complexity differ considerably: For a language represented with a *tree adjoining grammar* (TAG; see Frank 2004 for a brief description),[4] the recognition problem can be solved in time $O(n)^6$ (p. 60); with a *government and binding* formalism, the problem is in the class PSPACE[5]; and, finally, in the case of an *Aspects*-based grammar, the recognition problem is quite simply undecidable (p. 63).

At first sight, these results would appear to be far removed from the interests of a psycholinguist, and in a sense, they clearly are. After all, language processing is not at all like the problem of determining whether a string is part of a language (putting aside the ability to judge the grammaticality of sentences to one side, obviously). That this is so follows, in my opinion, from the rather incontrovertible fact that formal language theory, strictly speaking, focuses on the properties of sets of strings of symbols, and not, or at least not as much, on the structural descriptions that are assigned to these strings. The issues at hand, however, are rather subtle and significant care must be employed in their discussion.

Consider a grammar formalized in (a simplified version of) Chomsky normal form; namely, a 3-tuple composed of a set of non-terminals, a set of terminals and a set of production rules. Assume that there is a 'start' symbol that can be expanded by employing one of the production (that is, rewriting) rules of the grammar. In turn, the resultant string — a composition of terminal and non-

[3] Cf. Aho *et al*. (1974: iii): "[T]o analyze the performance of an algorithm some model of a computer is necessary". Similarly, Pratt-Hartmann (2010) employs a multi-tape Turing Machine in order to summarize the main results of studies in computational complexity.

[4] Incidentally, the expressive power of TAG is mildly context-sensitive, which is argued to be the correct expressive power of natural language.

[5] For a problem to be in the class PSPACE means that the Turing Machine would require a polynomial amount of space to solve the recognition problem.

terminal symbols — can be further expanded by using other rewriting rules until a string consisting only of terminals is derived. The history of these rule applications is usually called a derivation, and it is possible to use a tree representation as a visual aid in order to depict a given derivation in graphic form. In this sense, the 'derivation tree' so devised would specify the structure of the string so generated.

Note, however, that there are in fact two structural descriptions at hand here, what Simon (1962) called a state description (an object as sensed) and a process description (an object as is constructed), respectively. While obviously related (a process generates an object), the internal structure of each construct may not coincide piecemeal. Perhaps it is the case that there is a direct connection between the applications of rewriting rules and the intrinsic structure of a linguistic object so generated — surely the devise of linguists —, but such a nexus is not quite so transparent in other formalisms. As Miller (1999) points out, there is a difference between a 'derived tree' and a 'derivation tree' in TAG; whilst the former describes a linguistic object as postulated by the linguist (as in the syntactic trees so common in many a linguistic paper), the latter specifies the operations that TAG employs (viz. the adjunction and substitution of 'elementary trees'). Further, Miller (1999) proceeds, it is to the latter than we ought to focus if we are interested in the structures that TAG generates (usually called its strong generative capacity).[6] That is, it is the derivation tree that specifies the structural descriptions of a formalism, not the derived tree.

My present point is that there is a difference between a string rewriting system as employed in mathematical linguistics and a tree rewriting system such as TAG in respect to what products these two systems return (cf. Miller 1999: 29). Strictly speaking, the rewriting systems of formal language theory generate strings (weak generation); they don't generate structures. That is, there is nothing in the formalism of a rewriting system itself that even hints at the possibility of generating structure. Certainly, it is possible to modify such systems so that they generate structures, and this is precisely what obtains in systems such as those employed in a TAG. In my view, then, it is entirely correct to state that a rewrite rule generates a string to which a structural description is associated (surely an assignment that the linguist carries out; cf., again, Miller 1999: 2), but it is simply fantasy to suppose that they literally generate structures.

The same point applies to automata, which for present purposes can be described, in a somewhat simplifying manner, as being composed of an input tape, a control operation and a finite set of states (such as the initial state and the final, accepting state; Hopcroft *et al*. 2007: 45–46). It is certainly true that a properly characterized automaton would be able to accept a string of symbols of which we predicate a structure, but the automaton itself would not reflect in any way the internal constitution the set of symbols it receives is supposed to underlie. To suppose otherwise, it would quite simply be a figment of someone's imagination; or worse, metaphor.[7]

---

[6] Stabler (forthcoming) makes the very same point regarding the merge-based derivations of his minimalist grammars.

[7] Hopcroft *et al*. (2007: 243–244) point out that a pushdown automaton (the automaton that specifies/recognizes context-free languages) can simulate the derivations of a grammar, in

Consequently, even if mathematical linguistics may have been able to study the computational complexity exemplified in various automata or grammatical formalisms to a significant extent, none of this may bear any resemblance to what goes on in the mind of speakers and hearers when they produce or process linguistic material. Admittedly, a hearer receives a chain of elements in a temporal sequence, but it is rather obvious that this input is not treated as if it were a string of symbols; rather, a structure is imposed on this material in some manner. Accordingly, the computational complexity of natural language processing will have to consider properties of human psychology such as memory limitations, the strategies that are employed in parsing, the use of the immediate context and many other factors. All in all, it is simply not known what computational complexity our mental machinery exhibits in the processing of language. Consequently, a comparison with other computational tasks in these very terms seems to me rather flimsy.

Even if this weren't the case, it is very easy to show that the mathematical theory of knots is in fact not informative about either the expressive power or the computational complexity involved in tying a knot. Further, it also has nothing to say regarding how to determine if a string is knotted. This is unquestionable the case because the subject matter of such a field involves something else altogether. A fortiori, no relation can at present be drawn between the ability to tie a knot and the conceptualization/processing of language.

As in any other subfield of mathematics, Knot Theory is a rather narrow and technical discipline, a factor that should make anyone skeptical of the possibility of adapting it to the purposes of studying human cognition. As it turns out, the knots that Knot Theory studies bear no relation to real knots. Basically, a mathematical knot is a closed structure, an embedding of a circle into Euclidean 3-space (Burde & Zieschang 2003: 1). Moreover, the main line of research in this field is extremely narrow; what these theorists attempt to do is figure out which two knots are isotopic and which are not, where two knots are regarded as isotopic if one of them can be transformed into the other by following step-by-step moves. This, the knot recognition problem, involves working out the formal equivalence of two knots. A special case of this problem concerns the so-called 'unknot', a closed loop without any knot in it, as shown on the left-hand side of Figure 1 below. The 'unknotting' problem, in turn, involves specifying an algorithm that can recognize the unknot in a figure like the one found on the right-hand side of Figure 1 (that is, convert the knot on the right-hand side into an unknot).

---

the sense that the material the automaton is inputted can be manipulated in a manner that mimics the rule applications of the grammar (that is, the derivation). The same point follows: the structure that is assigned to the derivation (by the linguist) plays no role in the operations of such formalisms. Furthermore, the structure of a given derivation may not coincide with the structure of the object that is constructed. After all, simulation and mimicking do not stand on the same ground as a mechanism that is literally constructed as to generate structures, such as a tree rewriting system or an embedding mechanism like Merge (this situation is a bit like the attempt to model language comprehension — or indeed the whole of cognition— as a type of Bayesian inference; virtually anything can be so modeled, but this doesn't imply that mental mechanisms in fact effect such inferences).

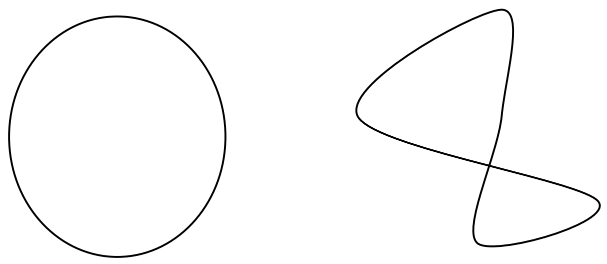

Figure 1: The unknot and a non-trivial knot

Relevant to the issues I am unearthing here is the so-called Reidemeister moves, a set of well-defined combinatorial moves that can disentangle a knot without damaging it (note that these moves *disentangle* a knot, they don't untie it). There are three such moves: twist/untwist; move one strand over another; and move one strand over/under a crossing (Manturov 2004: 12).

Naturally, none of this has anything to do with how you go about tying a knot, let alone the computational complexity required to do so. Perhaps unsureprisingly, the actual details of Knot Theory go completely unmentioned in C&U; its relevance to real-life knot-tying abilities just assumed. BEA do point out that Knot Theory deals with "elastic, closed, and tangled knots" (p. 11), but they go on to claim that "formal details aside" (as if they were of no importance), "the task of determining whether any string is knotted is known to have a complexity comparable to the one needed to process linguistic expressions" (*ibid.*; reference: Hass *et al.* 1999). And a bit later, they say, "(un)tying knots (or determining whether a tangled string is knotted) seems to require an underlying computational system of Type 1" (*ibid.*; Type 1 in the CH: context-sensitive).

There are two things at fault here. First is the claim that Knot Theory involves "determining whether a string is knotted", something that is clearly *not* the case, as Knot Theory takes tied knots as its starting assumptions — indeed, this field's sole concern is the equivalence problem outlined above. The other problem is to treat (un)tying a knot and determining if a string is knotted as if they are equivalent, but there are no reasons whatsoever to believe so. Furthermore, the reference BEA include in relation to this (*viz.* Hass *et al.* 1999) is clearly misrepresented. Rather, what Hass *et al.* (1999) proved is that an algorithmic solution for the unknotting problem is in the complexity class NP, which is to say that the algorithm will define multiple ways of processing the input without specifying which one it will take, in polynomial time. Quite clearly, this has *no* relation to either the mildly context-sensitive expressive power of language or the complexity involved in language processing; moreover, it also has *no* relation to the complexity of (un)tying a knot.

Nevertheless, this is not to say that (un)tying a knot may well involve a non-trivial computational system, but we don't have an account of this.[8] At one point, BEA do envision what may actually be involved in making a knot; one must relate, they tell us, a segment of the knot with the background, and this may

[8] In this and the next paragraph, I implicitly assume that knot tying can be modeled as a computational task, but I do not actually think this is so obvious. In any case, it would have to be demonstrated, not presumed.

well involve "grouping and long-distance-like relations" (p. 11). This insight comes from C&U, in fact; therein, the authors briefly describe a possible transformation of a string into a knot by assigning a specific number to each segment so that these symbols can in turn be manipulated by a (context-sensitive) grammar. They don't provide a proof of this, but the underlying idea is not incoherent. For example, Turing (1954) discusses a similar issue in relation to solvable and unsolvable problems in Knot Theory. As noted earlier, a knot is a closed curve in three dimensions, but it can also be accurately described, Turing tells us, "as a series of segments joining the points given in the usual (x, y, z) system of coordinates" (p. 585). Further, a set of symbols can be employed to represent unit steps in each coordinate direction (say, *a's* and *d's* for the X-axis, and so forth) so that transformation moves can be modeled by substitution rules of the production systems variety.

These are, in fact, the terms in which I assume C&U claimed that Mount (1989) showed the necessity of a context-sensitive system to create knots; a conclusion, it will be recalled, supposedly "not subject to rational debate". Somewhat amusingly, Mount (1989) turns out to be an unpublished computer manual for a program devised to assist mathematicians in the study of Knot Theory. At one point (p. 4), this author discusses the Reidemeister moves I outlined above, and remarks that the transformation of one knot into another may be reduced to a grammar problem, in precisely the terms Turing (1954) discusses. Later on, it is again remarked that "the Reidemeister moves could be rephrased as some kind of context-sensitive grammar" (p. 5).

Note what is actually being claimed here. First, that the Reidemeister moves *could* be modeled by a context-sensitive grammar; obviously, this is not a demonstration, but mere supposition. Secondly, such a supposition is exclusively meant to relate to the (narrow) purposes of Knot Theory; that is, Mount is wondering whether a production system may be employed to study the knot recognition problem. Again, this has nothing to do with the computational complexity or expressive power of (un)tying a knot in real life. Nor could it be construed as even suggesting such a connection. It is rather astonishing that the passing comments of an unpublished computer manual can become, on anyone's reading, a conclusion "not subject to rational debate".

In short, as it stands there is no fact of the matter regarding what relation there is, if there must be one, between the computational properties of the language faculty and whatever capacity underlies our ability to conceptualize and indeed tie a knot. This is not to say that there might not be a fruitful way to study such a relationship, but neither C&U nor BEA have provided any reason whatsoever, plausible or speculative, to believe that there is anything in need of explanation here.

In order to put an end to this brief examination, I should also add that C&U and BEA raise many other issues that are certainly worth discussing, such as the application of the Chomsky Hierarchy in the study of cognitive domains, the role of the different levels of analysis in such a study, and general features of mental architecture. In my opinion, there are significant shortcomings in the manner in which they treat all these issues, but this is not the place to discuss any of this; I do note, however, that I have done so elsewhere (Lobina 2012).

References

Aho, Alfred V., John E. Hopcroft & Jeffrey D. Ullman. 1974. *The Design and Analysis of Computer Algorithms*. London: Addison-Wesley Publishing Company.

Balari, Sergio, Antonio Benítez-Burraco, Marta Camps, Víctor M. Longa, Guillermo Lorenzo & Juan Uriagereka. 2011. The archaeological record speaks: Bridging anthropology and linguistics. *International Journal of Evolutionary Biology* 2011, doi:10.4061/2011/382679.

Burde, Gerhard & Heiner Zieschang. 2003. *Knots*. Berlin: Walter de Gruyter.

Camps, Marta & Juan Uriagereka. 2006. The Gordian knot of linguistic fossils. In Joana Rosselló & Jesús Martín (eds.), *The Biolinguistic Turn. Issues on Language and Biology*, 34–65. Barcelona: Publications of the University of Barcelona.

Chomsky, Noam. 1956. Three models for the description of language. *IRE Transactions of Information Theory IT-2*, 113–124.

Chomsky, Noam. 1963. Formal properties of grammars. In R. Duncan Luce, Robert R. Bush & Eugene Galanter (eds.), *Handbook of Mathematical Psychology*, vol. 2, 323–418. New York, NY: John Wiley.

Frank, Robert 2004. Restricting grammatical complexity. *Cognitive Science* 28, 669–697.

Hass, Joel, Jeffrey C. Lagarias & Nicholas Pippenger. 1999. The computational complexity of knot and link problems. *Journal of the ACM* 46, 185–211.

Hopcroft, John E. & Jeffrey D. Ullman. 1969. *Formal languages and their Relation to Automata*. London: Addison-Wesley Publishing Company.

Hopcroft, John E., Rajeev Motwani & Jeffrey D. Ullman. 2007. *Introduction to Automata Theory, Languages and Computation*. London: Addison-Wesley.

Joshi, Aravind K., K. Vijay Shanker & David Weir. 1990. The convergence of mildly context-sensitive formalisms. Department of Computer and Information Science Technical Report (University of Pennsylvania), 1–65.

Lobina, David. 2012. Conceptual structure and the emergence of the language faculty: Much ado about knotting. Ms., Tarragona: Universitat Rovira i Virgili. *LingBuzz*, http://ling.auf.net/lingBuzz/001397.

Manturov, Vassily. 2004. *Knot Theory*. Florida: Chapman & Hall/CRC.

Mount, John. 1985. *KnotEd: A program for studying knot theory*. [http://mzlabs.com/JohnMount] (27 January 2012)

Miller, Phillip H. 1999. *Strong Generative Capacity*. Stanford, CA: CSLI Publications.

Pratt-Hartmann, Ian. 2010. Computational complexity in natural language processing. In Alexander Clark, Chris Fox & Shalom Lappin (eds.), *The Handbook of Computational Linguistics and Natural Language Processing*, 43–73. Malden, MA: Blackwell.

Simon, Herbert. 1962. The architecture of complexity. *Proceedings of the American Philosophical Society* 106, 467–482.

Stabler, Edward. Forthcoming. Recursion in grammar and performance. [http://www.linguistics.ucla.edu/people/stabler/Stabler10-Recurs.pdf] (27 January 2012)

Turing, Alan M. 1954. Solvable and unsolvable problems. In B. Jack Copeland (ed.), *The Essential Turing*, 576–595. Oxford: Oxford University Press.

Uriagereka, Juan. 2008. *Syntactic Anchors*. Cambridge: Cambridge University Press.

Wintner, Shuly. 2010. Formal language theory. In Alexander Clark, Chris Fox & Shalom Lappin (eds.), *The Handbook of Computational Linguistics and Natural Language Processing*, 11–42. Malden, MA: Blackwell Publishing.

David J. Lobina
Universitat Rovira i Virgili
Departament de Psicologia
Centre de Recerca en Avaluació i Mesura de la Conducta
Ctra. de Valls s/n
43007 Tarragona
Spain
davidjames.lobina@urv.cat