# Exploring emergence with substance-free categories

Jamie Douglas (jamie.a.douglas89@gmail.com), January 2024

## Abstract

This essay adopts a substance-free *differentiation tree* model of syntactic categories where each category is defined by its *differentiation code*. The differentiation code of a category is a binary code reflecting the developmental sequence of splits from a single, ancestor category, with the length of the code being equal to the amount of information borne (measured in bits). Labelling of merged categories is achieved through a simple, additive evaluation algorithm, which evaluates two input differentiation codes term by term and outputs a differentiation code (the label).

The essay then explores some abstract, emergent properties of this system based on how the amount of information in the labels rises and falls in a syntactic structure, as well as exploring what happens if we assume the mapping between merge sequences and label sequences must be bi-unique. Rather than attempting to derive existing theoretical concepts or natural language phenomena, the aim is to show that simple, substance-free models are interesting and fruitful bottom-up ways of exploring emergent phenomena, some of which may or may not be reminiscent of existing ideas.

## 1 Introduction

'Emergentist' approaches study the way that phenomena (often highly complex and hard to intuit) arise from the interactions of simpler rules within a system, and they have become increasingly popular in a range of fields in both the natural and social sciences, including linguistics. One way to study emergence is 'bottom-up', that is building models or formalised systems of rules and states that represent aspects of interest, and seeing what sorts of behaviours arise. These models may vary in complexity depending on their purpose, from the highly complex models used to simulate the climate or epidemics, to very simple models, such as the Prisoners' Dilemma (see Axelrod 1997). While very simple models may not be particularly realistic, they may nevertheless be useful in exploring difficult problems. As Axelrod (1997) says:

> "… Models that aim to explore fundamental processes should be judged by their fruitfulness, not by their accuracy. For this purpose, realistic representation of many details is unnecessary and even counterproductive … The primary aim is to undertake the exploration in a manner so general that many possible settings could be illuminated."

In this essay, I will explore some of what emerges from a simple model of categories and labelling.

## 2 Differentiation trees and differentiation codes

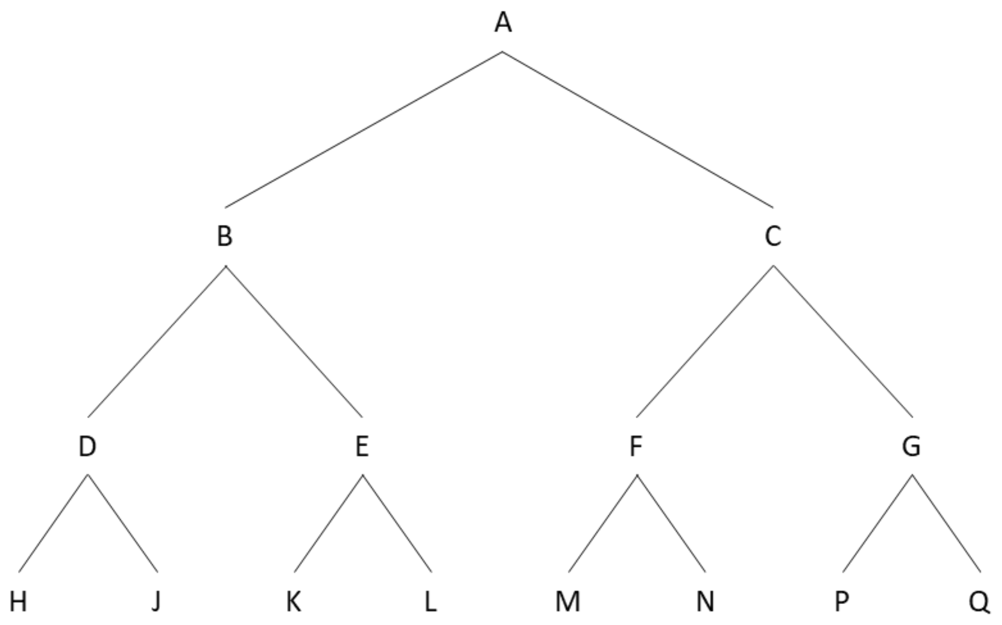
One of the benefits of very simple and abstract models is that they can be applied to many possible settings, and hence tend to be substance-free (see also Boeckx 2015). The model I adopt below is in fact taken from the field of embryology, but is in the abstract applicable to syntax.

In certain fundamental respects, some of the problems posed by embryologists and syntacticians are very similar. An embryo does not contain the different cell types found in a mature organism – it is not a homunculus – yet all these cell types arise reliably through cell division and differentiation at the right time and in the right place for each individual within any given species, presumably in response to conditions in the external (chemical) environmental of the embryo. Similarly for syntacticians, there is a growing consensus that infants are not born with grammars containing all the different categories found in an adult grammar, yet somehow all these categories arise reliably through category differentiation during language acquisition for each individual within any given linguistic community, presumably in response to statistical cues in the external environment of the learner. Indeed, the importance of differentiation as a 'biolinguistic' process goes back at least as far as Lenneberg (1967).

While even the basic details of the differentiation process (such as the conditions under which it occurs, and how it proceeds) are difficult and active research questions in both embryology and linguistics (and in a range of other fields facing formally similar problems), I will simply assume that differentiation exists. At an abstract level, we can model the differentiation of cells or syntactic categories using a *differentiation tree* (Björklund & Gordon 1994; Gordon 2016).

A differentiation tree is effectively a family tree (see Figure 1). At the top (the root) is the original 'ancestor' category (A). This splits (or *differentiates*) into two 'descendant' categories (B and C), which may themselves split into further categories (D and E, and F and G, respectively), and so on. In developmental terms, this could be that grammars initially contain a single category (A), and on the basis of some statistical cue or contrast detected in the external linguistic environment, category A differentiates into subcategories B and C, and so on.[1] Whatever the substance of these categories is, each differentiation represents a contrast; for example, B and C are both subcategories of A, but B and C contrast with one another.

Figure 1: Differentiation tree

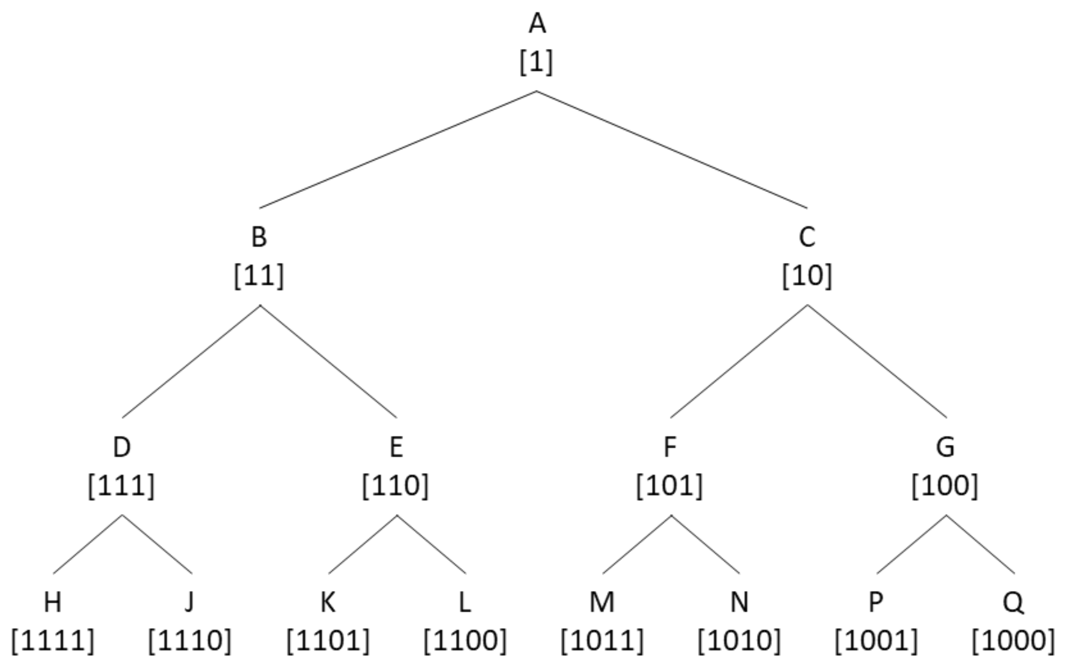


Arranging categories hierarchically in this way is not a new idea (for example, feature geometries in phonology, morphology and syntax, and inheritance hierarchies in computational linguistics); nor is the idea that such hierarchies reflect the development or emergence of categories during acquisition (see Hurford 2012: Section 4.4 for an overview). However, if each split is represented in binary terms, i.e. one branch is designated [1] and the other [0], then every node or category can be represented uniquely as a binary sequence – its *differentiation code* (Björklund & Gordon 1994; Gordon 2016). The differentiation code of a category, read left-to-right, thus reflects its developmental pathway

[1] It could be that the first contrast detected is between 'linguistic things' and 'non-linguistic things', but as this does not matter for the purposes of this essay, I will not pursue this question of category substance further.

from the original ancestor category, which in this example is A with the differentiation code [1] (see Figure 2, where for illustration every left-branch is [1] and every right-branch is [0]).

Figure 2: Differentiation tree with differentiation codes



The differentiation codes encode the sequence of differentiations or contrasts that lead to each category, and hence the formal relationships between categories. It also captures the amount of information (measured in bits) that each category contains. The longer the code, the more information it contains and the more specific the category is, for example, category N [1010] contains 4 bits of information and so is more specific than category C [10], which contains 2 bits of information. I will return to this in Section 4, after the introduction of an evaluation algorithm used for labelling in the next section.

## 3 The evaluation algorithm

When two categories are merged, what is the resulting category or label? Or in other words, what information is projected, if any? As Hurford (2012: Section 4.4.5) notes, many different theoretical frameworks have converged on the idea that the information in the label is *endocentric*, i.e. the information in the label comes exclusively from information in the constituent categories of the structure being labelled.[2] As such, I will pursue the intuition that, when two categories are merged, they effectively contribute as much of their information as they can to the label, i.e. labelling is *maximally endocentric*.

I propose an *evaluation algorithm*, which is an additive procedure that takes two input differentiation codes and outputs a differentiation code (the label). Operationally, the algorithm compares the first (leftmost) term of the two input differentiation codes. If the values are inconsistent, the algorithm stops; if the values are consistent, that value becomes the value of the first term of the output differentiation code. If all terms of both input differentiation codes have been evaluated, the

[2] In the Chomskyan tradition, the term 'endocentric' is essentially synonymous with 'head-centric'. However, it is a non-trivial problem to identify or define the 'head' of a phrase, and I will not complicate the system being developed here by adding such a concept.

algorithm stops. Otherwise, it moves on to compare the next term of the input differentiation codes, and so on.

Before going through some examples, note that there are three possible relationships that two categories can have with one another based on their relative positions in a differentiation tree. I will call these relationships 'match', 'ancestor-descendant' and 'cousins'. The application of the evaluation algorithm has a different outcome in each case, as illustrated below.

In 'match' relationships, the two categories are identical. In this case, the output differentiation code will be the same as the input differentiation codes. An example is given in (1):

(1) Merge [101] and [101]

a. Evaluate the first terms (i.e. 1 and 1). The values are consistent, so 1 becomes the value of the first term of the output differentiation code, and we move on to the second term.

b. Evaluate the second terms (i.e. 0 and 0). The values are consistent, so 0 becomes the value of the second term of the output differentiation code, and we move on to the third term.

c. Evaluate the third terms (i.e. 1 and 1). The values are consistent, so 1 becomes the value of the third term of the output differentiation code. There are no more terms to evaluate so the algorithm stops. The output differentiation code is [101].

Summary: Evaluate([101],[101]) → [101]

In 'ancestor-descendant' relationships, one category is a descendant of the other (or conversely, one category is the ancestor of the other). This means the two input differentiation codes will be of different lengths, and at some point the evaluation algorithm will be evaluating a value (1 or 0) and nothing (denoted as Ø), for example, evaluating [1] and [101] is equivalent to evaluating [1ØØ] and [101]. Since Ø means the absence of a contrast, Ø is effectively consistent with either 1 or 0. Therefore, in the same way that positive + zero = positive, and negative + zero = negative, I will assume that, when Ø is evaluated against 1 or 0, the more specific value (i.e. 1 or 0) becomes the output value. In other words, the evaluation in these cases is *additive*. An example is given in (2):

(2) Merge [1] and [101]

a. Evaluate the first terms (i.e. 1 and 1). The values are consistent, so 1 becomes the value of the first term of the output differentiation code, and we move on to the second term.

b. The input differentiation code [1] has no second term, but the input differentiation code [101] does. This means we are effectively evaluating Ø and 0. These are consistent in the sense defined above, and the more specific value (in this case, 0) becomes the value of the second term of the output differentiation code, and we move on to the third term.

c. The input differentiation code [1] has no third term, but the input differentiation code [101] does. This means we are effectively evaluating Ø and 1. These are consistent in the sense defined above, and the more specific value (in this case, 1) becomes the value of the third term of the output differentiation code. There are no more terms to evaluate so the algorithm stops. The output differentiation code is [101].

Summary: Evaluate([1],[101]) → [101]

Finally, in 'cousins' relationships, the categories are neither identical, nor in an 'ancestor-descendant' relationship; instead, they belong to different category lineages of the differentiation tree. However, they do still share a common ancestor category. In this case, the output differentiation code will be the same as the last common ancestor category, as shown in (3):

(3) Merge [101] and [11]

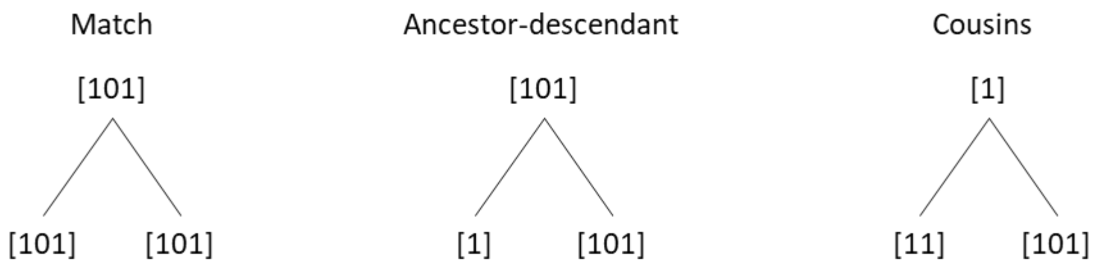
a. Evaluate the first terms (i.e. 1 and 1). The values are consistent, so 1 becomes the value of the first term of the output differentiation code, and we move on to the second term.

b. Evaluate the second terms (i.e. 0 and 1). The values are inconsistent, so the algorithm stops. The output differentiation code is [1].

Summary: Evaluate([101],[11]) → [1]

As can be seen, the output differentiation codes in all scenarios are maximally endocentric, with each of the two input differentiation codes contributing as much as they can to the output differentiation code (within the bounds of consistency). While the 'match' and 'ancestor-descendant' scenarios yield output differentiation codes that are equivalent to at least one of the input differentiation codes (giving the appearance of being 'head-centric'), the 'cousins' scenario does not (and is not possible under a 'head-centric' labelling approach) (see Figure 3). This will play an important role in the discussion in the next section.

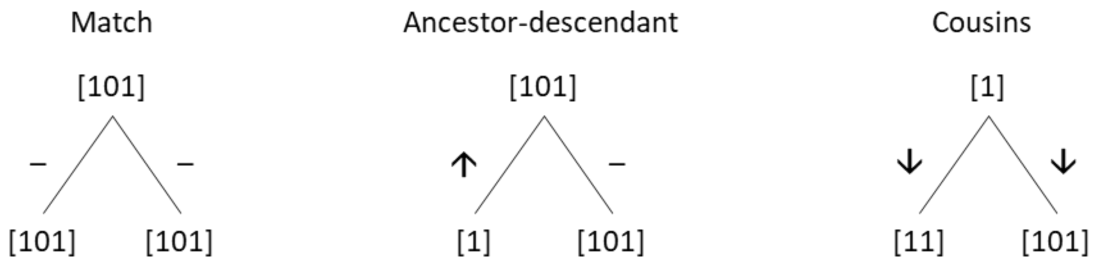
Figure 3: The three possible scenarios involving two merging categories, and the resulting label



## 4 Information dynamics

In Section 2, we saw how differentiation codes capture the amount of information in a category, with the length of the differentiation being equal to the number of bits it contains. Combining this with the evaluation algorithm from the previous section, we can view 'match', 'ancestor-descendant' and 'cousins' scenarios in terms of increases, decreases or no change in information going from a constituent category to the resulting label. Consider Figure 4, where '↑' and '↓' indicate increases and decreases in information, respectively, and '–' indicates no change:

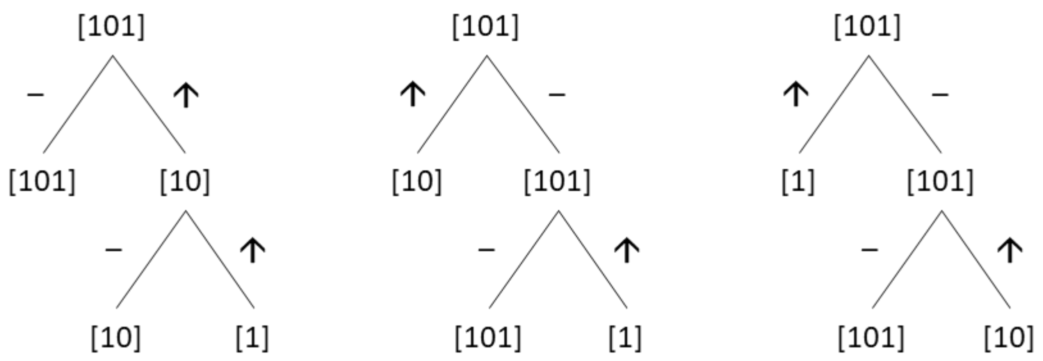
Figure 4: Information changes in the three possible labelling scenarios



Under the 'match' scenario, the output label is as specific as each of the constituent categories. I will describe this as there being no net change in information relative to the constituent categories. Under the 'ancestor-descendant' scenario, the output label is more specific than one of the constituent categories (the ancestor category) and as specific as the other (the descendant category). I will describe this as there being a net increase in information relative to the constituent categories. Finally, under the 'cousins' scenario, the output label is less specific than either of the constituent categories. I will describe this as there being a net decrease in information relative to the constituent categories. The mechanics of the evaluation algorithm rule out other logical possibilities, such as the output label being more specific than either of the constituent categories, or the output label being as specific as one of the constituent categories and less specific than the other.

If we have a sequence of distinct categories from the same *category lineage* (i.e. for any two categories chosen from the sequence, they will be in an 'ancestor-descendant' relationship), then no matter the sequence in which these categories are merged, the output labels will exhibit a net increase in information relative to their constituent categories at each step. For example, if we have three distinct categories from the same category lineage, such as [1], [10] and [101], there are three possible *merge sequences*, assuming merge between two categories is unordered, i.e. {[10],[1]} = {[1],[10]}. As can be seen in Figure 5, there is a net increase in information relative to the constituent categories at each step, regardless of the particular merge sequence.

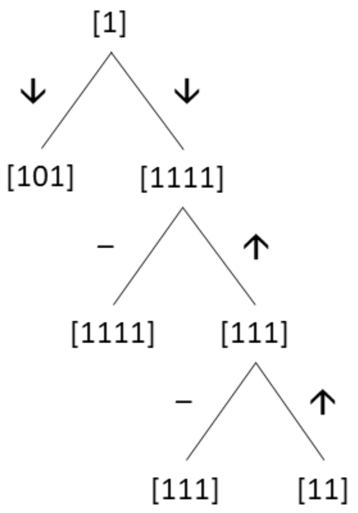
Figure 5: Net increase in information in merge sequences of distinct categories from the same category lineage



In the current system, situations where all the distinct categories being merged come from the same category lineage, which are reminiscent of extended projections (Grimshaw 1997) or domains of functional sequences in syntactic cartography (see e.g. Cinque 1999),[3] are defined by *monotone increasing information* in the labels.

No matter how extensive a sequence of categories from the same category lineage is, as soon as a 'cousin' category is merged, there is a net decrease in information relative to both of the constituent categories, with the output label being the last common ancestor category. This means that as the syntactic structure switches between sequences of distinct category lineages, there is a collapse in the amount of information in the label. This is illustrated in Figure 6:

Figure 6: Decrease in information when categories from distinct category lineages are merged

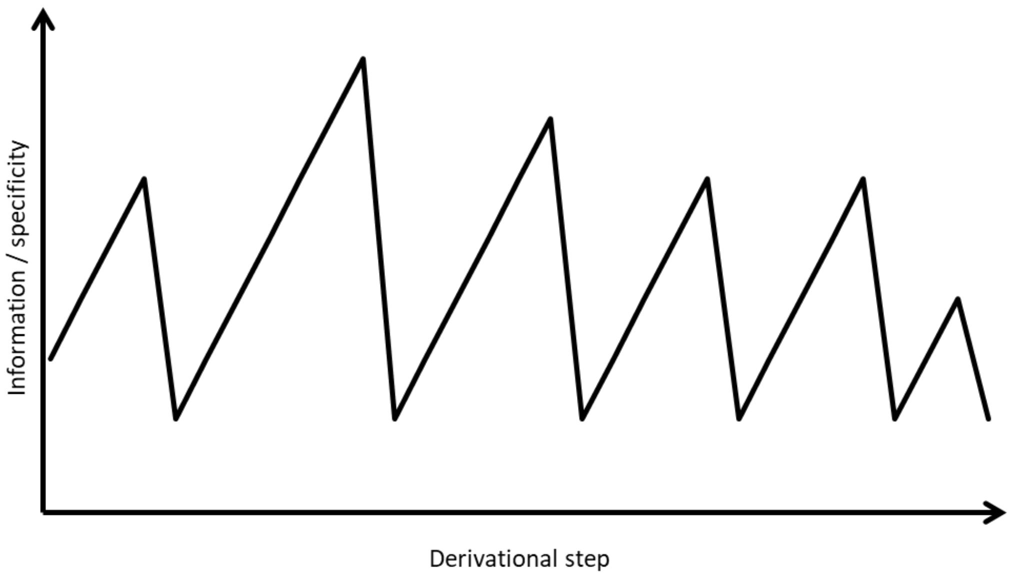


The build-up of information when categories of the same category lineage are merged and the collapse of information when categories of different category lineages are merged, yields information periods or cycles, a bit like a syntactic pulse. I will refer to these structures, which emerge as a result of the way information builds and collapses as categories from the same or different category lineages are merged, as *syntactic periods*, as schematically illustrated in Figure 7. If we suppose interfacing systems are sensitive to changes in information in the formal syntactic structure, these syntactic periods are somewhat reminiscent of phases (Chomsky 2000), with edges marked by collapses in information at the end of sequences of categories from the same category lineage.[4]

[3] While extended projections or functional sequences are typically considered to have strict sequences, I have not imposed this as a requirement on the formal system being developed here for simplicity. Even if sequences are strict, it need not automatically follow from a property of the formal system. For example, strict sequences could reflect the order of acquisition of categories, or could arise through some evolutionary dynamic preference for strict, or near-strict, merge sequences (see Section 5).

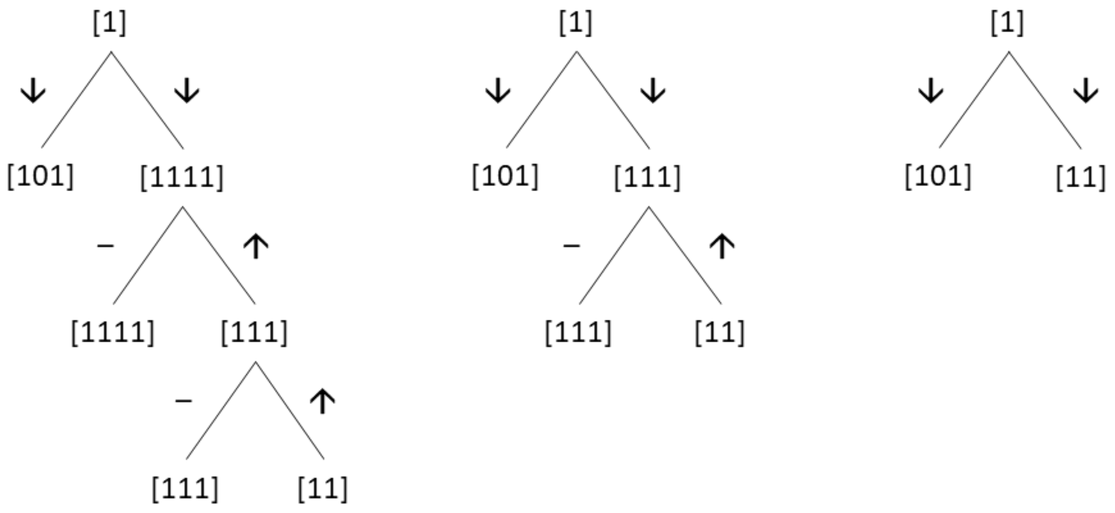
[4] I say 'reminiscent' because I am not trying to derive Chomskyan phases. If periods are phases, they are quite different from Chomskyan phases in key ways, for example, periods are based on information theoretic dynamics among categories (rather than an appeal to computational efficiency); they are emergent (rather than a primitive of the theory or posited on the basis of any empirical effects); they are dynamic and make no reference to phase heads or phase edges (rather than being properties inherent to particular phase heads).

Figure 7: Schematic illustration of periodic increases and decreases in information or specificity



Furthermore, in the current system, these syntactic periods are inherently *dynamic*. Periodicity is not a property of any particular category or class of categories; instead, a syntactic period ends as soon as a 'cousin' category is merged, where 'cousin' is by definition a relative term (as shown above). In this way, syntactic periods may vary in size, and (almost) any category may in principle serve as the syntactic period boundary, i.e. the last category merged in a sequence of categories from the same category lineage.[5] This is illustrated in Figure 8, and is reminiscent of dynamic phases (Bošković 2014):

Figure 8: Syntactic periods vary in size depending on when a 'cousin' category is merged



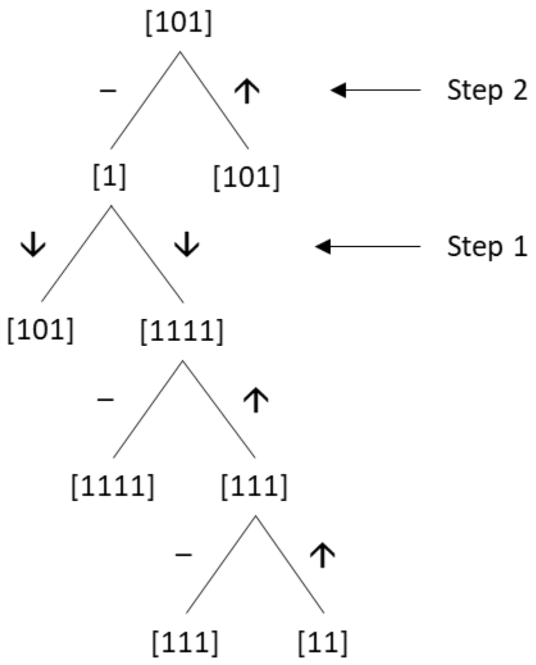
In Figure 8, [101] is the 'cousin' category triggering information collapse at different stages, resulting in periods of different sizes. Incidentally, the current system is insensitive to whether this 'cousin'

[5] The only systematic exception is the original ancestor category, which can never serve as the period boundary, because it has no 'cousin' categories.

category is an input category or an output category, and whether it is externally or internally merged, suggesting a variety of configurations where information collapses could be found.

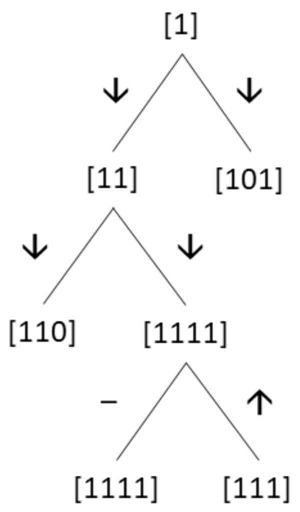
Furthermore, according to the current system, the transition from one syntactic period to another is a two-step process. The first step is the merger of a 'cousin' category. This triggers the information collapse that signals the end of a syntactic period. However, this is not sufficient to establish a new syntactic period. The second step is therefore the merger of a category from a new category lineage. This is illustrated in Figure 9, but note that the category in the first and second steps need not be the same according to the logic of the system:

Figure 9: Two-step process to transition from one syntactic period to another



Another situation that could arise in the current system is a sequence of information collapses. These would arise when a 'cousin' category is merged, triggering collapse to the last common ancestor category, followed by merger of a category that is a  'cousin' category of that last common ancestor, triggering collapse to an even more distant last common ancestor. The limit of how long such a sequence of information collapses could continue would depend on the number of 'cousin' categories available to the output category at each stage. Figure 10 illustrates two consecutive steps of information collapse – [110] and [1111] are 'cousin' categories, triggering collapse to [11]; and [11] and [101] are 'cousin' categories, triggering a further collapse to [1].

Figure 10: Sequence of information collapses



Whether the interface subsystems are sensitive to differences between single points of information collapse and sequences of information collapses is an intriguing possibility to explore, for example, if this aspect were built into the model, would we see phenomena emerge that are reminiscent of conditions on extraction, or effects where the degree of category relatedness between a moving element and the structure it is being extracted from have an impact on accessibility. I will not pursue such questions here, but note that the current approach to categories and labelling offers a fruitful way to explore what sorts of interactions might emerge from the system.

## 5 Mapping between merge sequences and label sequences

As noted in the previous section, the current system places no order on the merger of categories; categories are free to merge in any order, i.e. all merge sequences are possible. However, in a system with labelling, merge sequences map to *label sequences* and vice versa, which opens the possibility that constraints on the mapping may constrain the types of sequence that might come to predominate in the system due to evolutionary dynamics or natural selection.

Specifically, in this section, I explore how many merge sequences satisfy a *bi-unique* mapping, i.e. a one-to-one mapping between merge sequences and label sequences, and some of the behaviours they exhibit. While bi-unique mappings introduce a high degree of redundancy between merge sequences and label sequences, they also make information transfer more resilient in the face of random errors or noise, so I think this serves as a reasonable first avenue to explore.[6] In what follows, I will also limit the discussion to merge sequences involving categories from the same category lineage, and will assume throughout that elements are merged once and only once.

If we have *n* distinct categories from the same category lineage, and if (as above) merge is unordered, i.e. Merge(a,b) = Merge(b,a), the number of possible merge sequences for those *n* categories, $M_n$, is given as:

(4)

$$M_n = \frac{n!}{2}$$

[6] This is analogous to the bi-unique mapping between nucleotide base pairs in DNA, which increases resilience against random errors when replicating genetic information.

For example, if we have four distinct categories from the same category lineage, such as [10], [101], [1011] and [10110], which I will call ①, ②, ③ and ④ respectively to reflect their relative specificity,[7] we have 12 (= 4!/2) possible merge sequences. These are listed below:

(5)

| | | |
|---|---|---|
| {{{① ②} ③} ④} | {{{① ④} ②} ③} | {{{② ④} ①} ③} |
| {{{① ②} ④} ③} | {{{① ④} ③} ②} | {{{② ④} ③} ①} |
| {{{① ③} ②} ④} | {{{② ③} ①} ④} | {{{③ ④} ①} ②} |
| {{{① ③} ④} ②} | {{{② ③} ④} ①} | {{{③ ④} ②} ①} |

Given the evaluation algorithm from Section 3, each merge sequence maps onto a label sequence. The label sequences corresponding to the merge sequences in (5) are given below. Note that, while there are twelve possible merge sequences, there are only five distinct label sequences. In other words, while any single merge sequence maps onto a single label sequence, multiple merge sequences may map onto the same label sequence.

(6)

| | | |
|---|---|---|
| ② ③ ④ | ④ ④ ④ | ④ ④ ④ |
| ② ④ ④ | ④ ④ ④ | ④ ④ ④ |
| ③ ③ ④ | ③ ③ ④ | ④ ④ ④ |
| ③ ④ ④ | ③ ④ ④ | ④ ④ ④ |

If we have *n* distinct categories from the same category lineage, then the number of terms in the label sequences will be *n – 1* because there are *n – 1* branching nodes. It is possible to calculate the number of distinct label sequences for *n* categories, $L_n$, using the following formula (see Appendix A for how this is derived):

(7)

$$L_n = \frac{(2(n-1))!}{(n)!\,(n-1)!} = \frac{1}{n}\binom{2(n-1)}{n-1}$$

For example, when $n = 4$, $L_n = 5$, i.e. there are five distinct label sequences, as shown above (namely ②③④, ②④④, ③③④, ③④④ and ④④④).

As mentioned above, label sequences may map back onto several different merge sequences. However, there will always be some label sequences which map back onto a single merge sequence, i.e. there is a bi-unique mapping between the merge sequence and the label sequence. I will refer to such label sequences as *unique label sequences*. For example, of the five distinct label sequences shown above, two are unique label sequences (namely, ②③④ and ②④④). In the general

[7] Note that ①, ②, ③, etc. could stand for the members of any chosen group of categories from the same category lineage. The important point here is that ① stands for the least specific category among the group, ② for the second least specific category among the group, ③ for the third least specific category among the group, and so on.

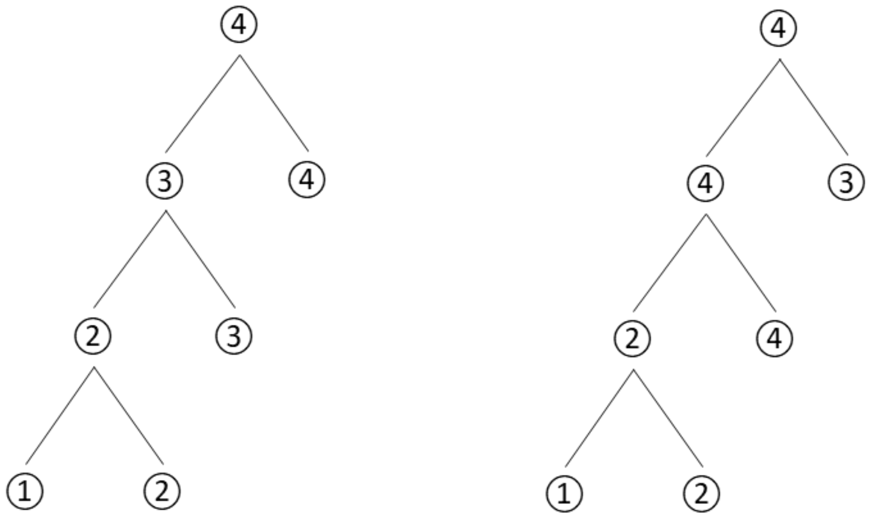
case, if we have *n* distinct categories from the same category lineage, then the number of unique label sequences for *n* categories, $U_n$, is given by (see Appendix B for how this is derived):[8]

(8)

$$U_n = 2^{n-3}$$

The set of unique label sequences always contains the label sequence that is *strict monotone increasing*, i.e. every term of the label sequence is more specific than the previous term. However, when $n > 3$, there are other possibilities. In these cases, there are unique label sequences that are what I will call *near-strict monotone increasing*. Consider the two unique label sequences ②③④ and ②④④ from the example above: the first is strict monotone increasing, but the second is not (since there is no increase between the second and third terms). The difference lies in the order in which ③ and ④ are merged relative to each other in the merge sequence, which in turn results in a different label sequence (and potentially quite a different interpretation at the interfaces). As shown in Figure 11, in the left-hand structure corresponding to the strict monotone increasing label sequence ②③④, category ③ appears as the *complement* of ④ (to use conventional configurational terms), while in the right-hand structure corresponding to the near-strict monotone increasing label sequence ②④④, category ③ appears as the *specifier* or *adjunct* of ④.

Figure 11: Strict monotone increasing and near-strict monotone increasing unique label sequences



Such pairs of unique label sequences are systematic for merge sequences where $n > 3$. To illustrate, I have listed the four unique label sequences when $n = 5$ and the eight unique label sequences when $n = 6$, but will leave the reader to work through such sequences if they wish to do so.

(8) $n = 5$, $U_n = 4$

a. ②③④⑤
b. ②③⑤⑤

[8] Since there cannot be non-integer numbers of label sequences, the exponent must be greater than or equal to 0, i.e. $n - 3 \geq 0$. Therefore, we could write: $U_n = 2^x$ where $f(x) = max\{n-3,0\}$. This would capture the case where $n = 2$ where there is only one possible label sequence and hence one unique label sequence. For $n = 1$, unary Merge could apply and the labelling algorithm presented here would yield one possible label sequence and hence one unique label sequence, assuming unary Merge is essentially binary Merge where an element merges with nothing.

c. ②④④⑤
d. ②④⑤⑤

(9) $n = 6, U_n = 8$

a. ②③④⑤⑥
b. ②③④⑥⑥
c. ②③⑤⑤⑥
d. ②③⑤⑥⑥
e. ②④④⑤⑥
f. ②④④⑥⑥
g. ②④⑤⑤⑥
h. ②④⑤⑥⑥

As can be seen, the near-strict unique label sequences deviate from the strict unique label sequence in a systematic way. The system developed here means there are restrictions on which values a given term in a label sequence may have. The value of the last term must be the most specific category, the value of the second to last term may be either the most specific category or the second most specific category, the value of the third to last term may be the most specific category, the second most specific category or the third most specific category, and so on. For near-strict unique label sequences, there is at least one pair of consecutive identical values. Such pairs of values are only allowed to occur at the highest term for that value, and the term immediately preceding it.

## 6 Conclusion

In this essay, I adopted a substance-free differentiation tree model of syntactic categories, where each category is designated by its own unique differentiation code. I proposed a simple, endocentric evaluation algorithm that reads the input differentiation codes and outputs a differentiation code (the label). I then explored some of the information dynamics that emerge from such a system, including findings that are potentially and intriguingly reminiscent of ideas in the literature, such as extended projections and dynamic phases. Finally, I explored some consequences of assuming the mapping between merge sequences and label sequences is bi-unique, at least for structures involving categories from the same category lineage.

Further questions may also be explored in this system, or the system may be refined to model further aspects of interest. For example: what effects emerge if some categories do not map onto natural classes, or are ignored by the system? What evolutionary dynamics might be at play in the survival of newly created and existing categories? What other properties could emerge from bi-unique or other types of mapping between merge sequences and label sequences? Given the highly general system developed here, are there any other systems with (newly-exposed) formal similarities that could shed light on syntactic problems?

The discussion and open questions highlight how rich and complex behaviours may emerge from a simple, abstract and substance-free system under simple assumptions. As such, although not aiming to derive natural language phenomena or existing theoretical ideas, I take the model developed here to be both interesting and fruitful in approaching certain fundamental questions in syntax and beyond.

## Appendices

### Appendix A: Calculating the number of distinct label sequences

The equation for calculating the number of distinct label sequences for *n* distinct categories from the same category lineage is the equation for generating the *Catalan numbers*, which is a sequence found in the solution to a range of problems in combinatorics.[9]

(A1)

$$L_n = \frac{(2(n-1))!}{(n)!\,(n-1)!} = \frac{1}{n}\binom{2(n-1)}{n-1}$$

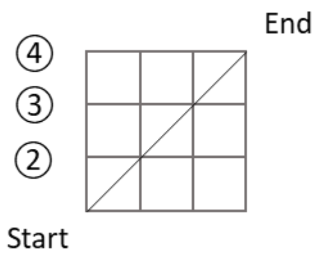
To show why this is the case, I will show that the problem of calculating the number of distinct label sequences can be approached in one of the various ways that has been used to prove this equation.

To start, if we have *n* distinct categories from the same category lineage (and for simplicity, assuming that each category can be merged once and only once), then the merge sequences will have *n* terms, and the corresponding label sequences will have *n – 1* terms. In what follows, I will denote the number of terms in a label sequence as *t*, where *t = n – 1*.

The problem of calculating the number of distinct label sequences of length *t* can be modelled as finding the number of monotonic lattice paths along the edges of a grid with *t x t* square cells from the bottom-left to the top-right which only consist of upwards and rightwards moves, and where all moves are above the diagonal line that bisects the grid (see Wikipedia: “Catalan number”). The reasons for these conditions will be explained below.

Figure A1 illustrates with a 3 x 3 grid. Assuming we have four distinct categories from the same category lineage, call them ①, ②, ③ and ④ reflecting their relative specificity (where ① is the least specific category and ④ is the most specific category), the terms of the corresponding label sequences will have three possible values, namely ②, ③ and ④ (as the least specific category among those being merged, ① would never be an output category and hence would not be a possible value of the corresponding label sequences given the evaluation algorithm laid out above). The three possible values of the label sequence are given down the side of the grid, while the bottom of the grid may be interpreted as the first, second and third terms of the label sequence.

Figure A1:



[9] Note the equation is typically given as:

$$C_n = \frac{(2n)!}{(n+1)!\,n!} = \frac{1}{n+1}\binom{2n}{n} \; for\; n \geq 0$$

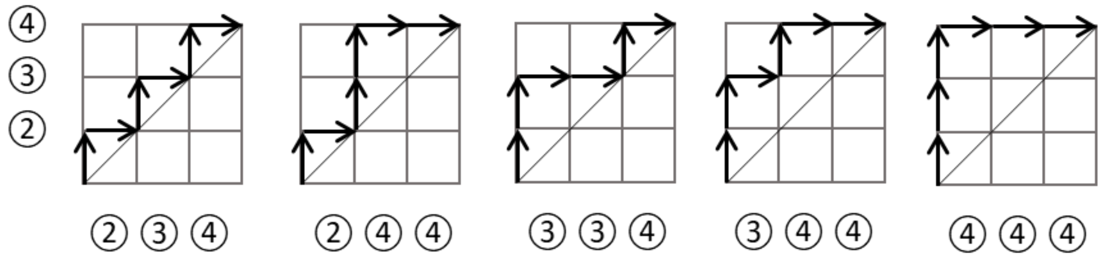
Since a merge sequence with *n* categories will correspond to a label sequence with *n – 1* terms, I substitute *n – 1* in place of *n* in (A1). Alternatively, the equation for $L_n$ is for *n ≥ 1*, while the equation for $C_n$ is for *n ≥ 0*.

As mentioned above, we are only interested in the paths which consist of upwards and rightwards moves. The absence of downwards moves reflects the fact that, when dealing with categories from the same category lineage, the values in a label sequence will be either the same as or more specific than the previous value. The absence of leftwards moves reflects the assumption that labels cannot be overwritten at a later stage.

As also mentioned above, we are only interested in those paths that are entirely in the part of the grid above the bisecting diagonal line. This reflects the constraints on the values that the label sequence terms may take. Specifically, for label sequences with three terms and possible values ②, ③ and ④, the first term may be ②, ③ or ④, the second term may be ③ or ④, and the third term may only be ④. The value ② is possible in the first term if ① and ② are merged first; if ② is merged with ③ or ④ first, the resulting label will be ③ or ④, and ② could never be the value of subsequent terms since it is less specific than ③ or ④. Furthermore, even if the first term is ②, subsequent terms cannot be because the only remaining possibilities would be to merge ② with ③ or ④, both which are more specific than ②, and hence the next value will be ③ or ④, and so on.

Figure A2 gives all possible paths that satisfy the conditions mentioned above, and their corresponding label sequences. The leftmost path represents the label sequence ②③④, the path second from left represents the label sequence ②④④, and so on. As expected, there are five possible paths, corresponding to the five distinct label sequences for four distinct categories from the same category lineage.

Figure A2:



The reason for showing that our problem can be represented in this way is that this forms the first step of one of the many established proofs for the equation above. In the rest of this section, I will briefly go through the key elements of the proof (but for full details, see Wikipedia: “Catalan number”).

All of the paths in Figure A2 have an *exceedance* equal to 3, i.e. the number of upward arrows above the diagonal in each of the paths above is 3. It can be shown that the number of paths of exceedance 3 is equal to the number of paths of exceedance 2, which is equal to the number of paths of exceedance 1, which is equal to the number of paths of exceedance 0. In fact, it can be shown in the general case that, for a *t x t* grid, the number of paths of exceedance *t* is equal to the number of paths of exceedance *t – 1*, *t – 2*, and so on, through to *t – t* (i.e. exceedance 0). Furthermore, it can also be shown that the total number of paths of any exceedance through a *t x t* grid, $P_t$, is equivalent to the *central binomial coefficient*, i.e. the binomial coefficient that appears exactly in the middle of the $t^{th}$ even-numbered row in Pascal’s triangle, conventionally denoted as:

(A2)

$$P_t = \binom{2t}{t}$$

Since there are *t + 1* exceedance groups for any *t x t* grid, and since the number of paths in each exceedance group is the same, the number of distinct label sequences of length *t* is the central binomial coefficient divided by the number of exceedance groups:

(A3)

$$L_t = \frac{1}{t+1}\binom{2t}{t}$$

Since *t = n – 1*, we can substitute to express the number of distinct label sequences in terms of *n*:

(A4)

$$L_n = \frac{1}{n}\binom{2(n-1)}{n-1}$$

This is equivalent to:

(A5)

$$L_n = \frac{(2(n-1))!}{(n)!\,(n-1)!}$$

## Appendix B: Calculating the number of unique label sequences

The number of unique label sequences can be calculated by considering the possible values of each term in the label sequence.

The final term in *any* label sequence (including unique label sequences) must be the most specific value, since as soon as the most specific category is merged, that will be the output category, and the very last opportunity to merge the most specific category is at the end of the merge sequence.

The first term in a *unique* label sequence must be the least specific value, i.e. ② (not the least specific category, i.e. ①, since ① can never be the label in a label sequence where all categories are from the same category lineage). If the value of the first term were anything more specific, there would be ambiguity about the merge sequence: in the simplest case, suppose the first term of the label sequence was ③. This would be ambiguous between the merge sequence {① ③} and {② ③}, so all label sequences where the first term is anything other than ② would map onto at least two merge sequences, and therefore would not be unique.

For each of the remaining terms in unique label sequences, i.e. those that are not the first or last terms, there are two possible values. For a label sequence of length *t*, the second term may be either ③ or ④, the third term may be either ④ or ⑤, the fourth term may be either ⑤ or ⑥, and so on, through to term *t – 1*. As in Appendix A, this partly reflects the constraints on the values that the label sequence terms may take. But it also reflects the constraint that, if the label sequence is to be unique, values cannot occur too early in the label sequence depending on the specificity.

For example, consider the label sequences ②④⑤⑤ and ②⑤⑤⑤. The label sequence ②④⑤⑤ is unique. The first term is ②, so we know the merge sequence starts with {① ②}. The second term is ④, so we know the next category to have been merged must have been ④ (if the next merged category had been ③, the second term in the label sequence would have been ③; if the next merged category had been ⑤, the second term in the label sequence would have been ⑤). The third term is ⑤, so we know the next category to have been merged must have been ⑤.

Finally, the fourth term is ⑤, and we know the final category to have been merged was ③ because that is the only category left to account for.

Turning to label sequence ②⑤⑤⑤, this is not unique. From the first and second terms, we know the merge sequence must start with {{① ②} ⑤}. However, as subsequent terms of the label sequence are also ⑤, we cannot be sure whether the merge sequence was {{{{① ②} ⑤} ③} ④} or {{{{① ②} ⑤} ④} ③}. This would be true where the difference between any two consecutive terms in a label sequences is greater than 2 degrees of specificity. This constraint, combined with the constraints on the values that terms in any label sequence may take, means that for all unique label sequences, the first term and last term may each have only one possible value, while all other terms may have two possible values. The number of unique label sequences of length *t*, $U_t$, is thus:

(B1)

$$U_t = 1.1.2^{t-2} = 2^{t-2}$$

Since $t = n - 1$, we can substitute to express the number of unique label sequences in terms of *n*:

(B2)

$$U_n = 2^{n-3}$$

## References

Axelrod, R. (1997). *The Complexity of Cooperation*. Princeton, NJ: Princeton University Press.

Björklund, N. K. & Gordon, R. (1994). Surface contraction and expansion waves correlated with differentiation in axolotl embryos. I. Prolegomenon and differentiation during the plunge through the blastopore, as shown by the fate map. *Computers & Chemistry 18*(3), 333-345. doi: 10.1016/0097-8485(94)85027-5

Boeckx, C. (2015). *Elementary Syntactic Structures: Prospects of a Feature-Free Syntax*. Cambridge: Cambridge University Press.

Bošković, Ž. (2014). Now I'm a phase, now I'm not a phase: On the variability of phases with extraction and ellipsis. *Linguistic Inquiry 45*(1), 27-89.

Chomsky, N. (2000). Minimalist inquiries: The framework. In Martin, R., Michaels, D. & Uriagereka, J. (eds.), *Step by Step: Essays on Minimalist Syntax in Honor of Howard Lasnik* (pp. 89-155). Cambridge, MA: MIT Press.

Cinque, G. (1999). *Adverbs and Functional Heads*. Oxford: Oxford University Press.

Gordon, R. (2016). Walking the tightrope: The dilemma of hierarchical instabilities in Turing's morphogenesis. In Cooper, S.B. & Hodges, A. (eds.), *The Once and Future Turing: Computing the World* (pp. 144-159). Cambridge: Cambridge University Press.

Grimshaw, J (1997). Projection, Heads, and Optimality. *Linguistic Inquiry 28*(3), 373-422.

Hurford, J.R. (2012). *The Origins of Grammar: Language in the Light of Evolution*. Oxford: Oxford University Press.

Lenneberg, E. (1967). *Biological Foundations of Language*. New York: Wiley.

https://en.m.wikipedia.org/wiki/Catalan_number