A Survey of Phrase Projectivity in the *Antigone*

Jonathan Sterling

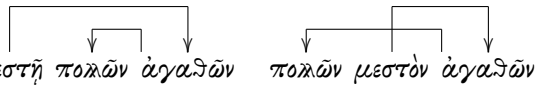
April 2013

Abstract

In this paper, I demonstrate how, and to what degree, phrase projectivity corresponds with register and meter in Sophocles's *Antigone*, by developing a quantitative metric for projectivity and comparing it across lyrics, trimeters and anapaests using the data provided by the Perseus Ancient Greek Dependency Treebank (Bamman and Crane, 2011). In the appendices, the formal algorithm for the computations done herein is developed in the programming language Haskell (Marlow, 2010).

1 Dependency Trees and Their Projectivity

A dependency tree encodes the head-dependent relation for a string of words, where arcs are drawn from heads to their dependents. We consider a phrase *projective* when these arcs do not cross each other, and *discontinuous* to the extent that any of the arcs intersect. Figure 1 is a minimal pair that demonstrates how hyperbaton introduces a projectivity violation; in this case, a path of dependency "wraps around itself".



(a) "Full of plentiful supplies" (Xenophon, *Anabasis* 3.5.1) is fully projective.

(b) "Full of many good things" (Plato, *Laws* 906a) has one projectivity violation.

Figure 1: Examples drawn from Devine and Stephens (2000, p. 11).

In addition to the above, adjacent phrases (that is, phrases at the same level in the tree) may interlace, causing projectivity violations. This is commonly introduced by Wackernagel's Law, as in Figure 2, where the placement of μὲν δὴ interlaces with the τὰ...πόλεος phrase.

We consider a violation to have occured for each and every intersection of lines on such a drawing; thus, the hyperbaton of one word may introduce multiple violations. Consider, for instance, Figure 3, in which five violations are brought about by the displacement of φονώ-σαισιν. In this way, the number of intersections is a good heuristic for judging the severity of hyperbata.

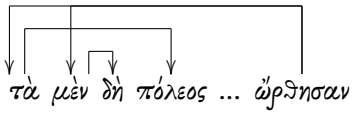


Figure 2: "[The gods] righted the matters of the city..." (*Antigone*, ll. 162–163) has one projectivity violation, due to the μὲν δὴ falling in Wackernagel's Position.

Figure 3: "And he stood over the rooftops, gaped in a circle with murderous spears around the seven-gated mouth, and left" (*Antigone*, ll. 117–120) has six projectivity violations, five of which are induced by the hyperbaton of φονώσαι-σιν, and one from the usual placement of δ' in Wackernagel's Position.

1.1 Counting Violations

Drawing trees and counting intersections is time-consuming and error-prone, especially since the number of intersections may vary if one is not consistent with the relative height of arcs and placement of endpoints. It is clear, then, that a computer ought to be able to do the job faster and more accurately than a human, given at least the head-dependent relations for a corpus.

The formal algorithm for counting the number of intersections is given in Appendix A.2, but I shall reproduce an informal and mostly nontechnical version of it here. First, we index each word in the sentence by its linear position, and cross-reference it with the linear position of its head:

στὰς δ'	ὑπὲρ	μελάθρων	φονώσαισιν	ἀμφικανὼν	κύκλῳ	λόγχαις	ἑπτάπυλον	στόμα
1:11 2:11	3:1	4:3	5:8	6:11	7:6	8:6	9:10	10:6

ἔβα
11:_

Next, arrange the dependencies into a tree as in Figure 4. Then, counting upwards from the lowest edges (i.e. the lines) in the tree up to the topmost ones, make a list of edges indexed by vertical level as in Table 1.

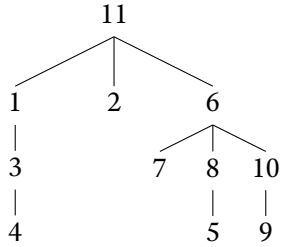


Figure 4: The dependency relations arranged in a non-linear tree.

LEVEL	EDGES
1	$3 \leftrightarrow 4, 5 \leftrightarrow 8, 9 \leftrightarrow 10$
2	$1 \leftrightarrow 3, 6 \leftrightarrow 7, 6 \leftrightarrow 8, 6 \leftrightarrow 10$
3	$1 \leftrightarrow 11, 2 \leftrightarrow 11, 6 \leftrightarrow 11$

Table 1: Edges of the tree in Figure 4 arranged by level.

$2 \leftrightarrow 11$	$1 \leftrightarrow 3$	1
$6 \leftrightarrow 11$	$5 \leftrightarrow 8$	1
$6 \leftrightarrow 7$	$5 \leftrightarrow 8$	2
$6 \leftrightarrow 8$	$5 \leftrightarrow 8$	1
$6 \leftrightarrow 10$	$5 \leftrightarrow 8$	1
		$\mathsf{total} = 6$

Table 2: Projectivity violations which arise from the edges in Table 1.

Then, each edge in our table must be checked for violations against all the other edges in the table except those which are in a level higher than it. The level of the edge corresponds with the height at which we drew the arcs; this condition arises out of the fact that an arc cannot cross an arc that is above it.

Next, we must figure out all the possible ways for an arc to intersect another at given levels. These are enumerated in detail in the function checkEdges in Appendix A.2, but suffice it to say that they fall into a few main categories:

1. Both vertices of the higher edge are within the bounds of the lower edge. This is a double violation, as both sides of an arc will extrude through another.
2. One vertex of the higher edge is within the bounds of the lower edge, and the other vertex is not; this vertex is allowed to be equal to the second vertex of the lower edge. In either case, this is a single violation, as just one intersection occurs.
3. The edges are at the same level, and one vertex of the higher edge is neither within bounds of the other, nor equal to any of the vertices of the other. This is a single violation.

Using this procedure, we shall have found the edge violations which are listed in Table 2, which are 6 in total.

1.2 $\wp$: a metric of projectivity

In order for our view of a text's overall projectivity to not be skewed by its length, we must have a ratio. For the purposes of this paper, we shall call this metric $\wp$, as given by the following ratio:

$$\wp = \frac{\text{number of violations}}{\text{number of arcs}}$$

Now, this metric applies just as much to a single sentence as it does to a larger body of text. So, averages of $\wp$ should not be taken; rather, total numbers of violations and total numbers of arcs should be accumulated until $\wp$ may be computed for the entire body of text being examined.

2 The Perseus Treebank

The Perseus Ancient Greek Dependency Treebank is a massive trove of annotated texts that encode all the dependency relations in every sentence. The data is given in an XML (**E**xtensible **M**arkup **L**anguage) format resembling the following:

```
<sentence id="2900759">
  <word id="1" form="χρὴ" lemma="χρή" head="0" />
  <word id="2" form="δὲ" lemma="δέ" head="1" />
  ...
</sentence>
```

Every sentence is given a unique, sequential identifier; within each sentence, every word is indexed by its linear position and coindexed with the linear position of its superordinate head. Hence, the treebanks are an invaluable source for a scholar who wishes to confirm intuitions about hyperbaton-frequency with real data on a large scale.

Indeed, it is not a new proposition to analyze the Perseus Treebanks for hyperbata; for instance, Bamman and Crane (2006) report an experiment run against the Latin Treebank to compare the level of hyperbaton across Jerome, Caesar, Cicero and Vergil. Bamman and Crane's design, however, is both more limited and more fine-grained than the one presented in this paper: on the one hand, they exclusively observe hyperbata that involve a dependent being transposed out from under a preposition (and ignore the syntactically parallel account for other categories); on the other hand, they distinguish between two different such cases, namely those of *memorem ob īram* and *īram ob memorem*, which are respectively the Y_2 and Y_1 hyperbata of Devine and Stephens.

3 Projectivity in the *Antigone*

To observe the variation of projectivity within a text, then, one may make a selection of sentences that have something in common, compute their trees and thence derive a cumulative $\wp$ for the entire selection. Then that figure may be compared with that of other selections.

I have chosen to compare projectivity in lyrics, anapaests and trimeters. Lyrics I have divided into two categories: choral odes and laments, whereas I divide trimeters into medium-to-long speeches and stichomythia. Appendix B.1 deals with parsing the Perseus XML representations of the *Antigone* into dependency trees for which we can compute $\wp$.

To that end, I have selected passages from the *Antigone* and organized them by type. Table 3 enumerates the lyric passages of the play, along with their computed $\wp$ values, and a cumulative $\wp$ value for the entire set. Table 4 does the same for anapaests. Lastly, Table 5

(a) Choral Odes

LINES		$\wp$
100 · · · 154	*First choral ode*	0.83
332 · · · 375	*Second choral ode*	0.58
583 · · · 625	*Third choral ode*	0.71
781 · · · 800	*Fourth choral ode*	0.67
944 · · · 987	*Fifth choral ode*	0.47
1116 · · · 1152	*Sixth choral ode*	0.64
	cumulative $\wp = 0.66$	

(b) Laments

LINES		$\wp$
806 · · · 816	*Antigone's Lament*	1.37
823 · · · 833	*Antigone's Lament (cntd.)*	0.78
839 · · · 882	*Antigone's Lament (cntd.)*	0.63
1261 · · · 1269	*Kreon's Lament*	0.38
1283 · · · 1292	*Kreon's Lament (cntd.)*	1.34
1306 · · · 1311	*Kreon's Lament (cntd.)*	0.37
1317 · · · 1325	*Kreon's Lament (cntd.)*	1.13
1239 · · · 1246	*Kreon's Lament (cntd.)*	0.60
	cumulative $\wp = 0.78$	

Table 3: Lyrics

gives selections of dialogue (which is in iambic trimeters), divided between medium-to-long speeches and stichomythia.

As can be seen from the data, lyrics have the highest degree of non-projectivity, followed by speeches, then anapaests, and then stichomythia. To try and understand why this is the case, it will be useful to discuss Greek hyperbaton in more general terms.

Whereas in prose, hyperbaton corresponds to *strong focus*, which "does not merely fill a gap in the addressee's knowledge but additionally evokes and excludes alternatives", hyperbaton in verse only entails weak focus, which emphasizes but does not exclude (Devine and Stephens, 2000, p. 107, 303).

As a result, hyperbaton in verse may be used to evoke a kind of elevated style without incidentally entailing more emphasis and other pragmatic effects than intended. And so it should not be surprising that lyric passages, which reside in the most poetic and elevated register present in tragic diction, should have proved in the *Antigone* to have the highest proportion of projectivity violations.

Within the lyric passages, the laments appear to have consistently higher $\wp$ than the choral odes, which may stem from their being much more emotive and personal in nature; I have not come to a firm conclusion on that particular matter. It should be noted that, whilst the

LINES		$\wp$
155 · · · 161	*Kreon's Entrance*	0.33
376 · · · 383	*Antigone's Entrance*	0.91
526 · · · 530	*Ismene's Entrance*	0.05
626 · · · 630	*Haimon's Entrance*	0.91
801 · · · 805	*Antigone's Entrance*	1.16
817 · · · 822	*Chorus to Antigone*	0.57
834 · · · 838	*Chorus to Antigone*	0.05
929 · · · 943	*Chorus, Kreon and Antigone*	0.25
1257 · · · 1260	*Chorus before Kreon's Kommos*	0.00
1347 · · · 1353	*Final anapaests of the Chorus*	0.31
	cumulative $\wp$ =	0.47

Table 4: Anapaests.

individual odes conform tightly to the cumulative $\wp$ of their category, there is a fair degree of variation among the laments. Likewise, the anapaests vary so wildly in their $\wp$ that it may be difficult to say very much about them that is relevant to the questions we are considering.

As for dialog, longer-form speeches are largely conformant in their $\wp$, with stichomythias varying a bit more. Speeches are a somewhat less projective than the stichomythias, being typically more eloquent and long-winded than their argumentative, choppy counterparts.

So far, the most surprising thing about the data is the degree to which certain verse-types vary in $\wp$ (or, if you like, the degree to which other types *don't*). The data draw us, then, to the following conclusions:

1. Non-projectivity varies within a single metrical type (lyrics, iambic trimeters, anapaests).
2. Certain registers seem to be more conventionalized with respect to $\wp$ than others; that is, choral odes and speeches do not vary greatly amongst themselves, but laments and anapaests do.

Lyric passages are in general less projective than anything else, but some laments reach a degree of non-projectivity that exceeds the most elliptical odes in the *Antigone*. Further, within the trimeters, speeches are less projective than stichomythias. From these things, then, we can say that that meter itself would not seem to be a primary factor for predicting incidence and severity of hyperbaton, but rather a secondary one at best.

That is to say, we know for a fact that passages in lyric meters have greater $\wp$ than passages in other meters. Yet, the variation of $\wp$ within that very meter indicates that there is some other factor involved, which very likely has to do with register along two different dimensions, which is to say, relative "dignity of style" and emotive force.

With regard to the very low $\wp$ found in the stichomythias, I suggest that it is the necessary shortness of each utterance which is at fault here. That is, the maximum "damage" that a hyperbaton can do is greatly lessened, when the ultimate depth of the phrase structure is limited

(a) Speeches and Dialogue

LINES		$\wp$
162 · · · 210	Kreon: ἄνδρες, τὰ μὲν δὴ...	0.40
249 · · · 277	Guard: οὐκ οἶδ'· ἐκεῖ γὰρ οὔτε...	0.57
280 · · · 314	Kreon: παῦσαι, πρὶν ὀργῆς...	0.45
407 · · · 440	Guard: τοιοῦτον ἦν τὸ πρᾶγμ'...	0.56
450 · · · 470	Antigone: οὐ γάρ τί μοι Ζεὺς...	0.56
473 · · · 495	Kreon: ἀλλ' ἴσθι τοι...	0.45
639 · · · 680	Kreon: οὕτω γὰρ, ὦ παῖ...	0.55
683 · · · 723	Haimon: πάτερ, θεοὶ φύουσιν...	0.45
891 · · · 928	Antigone: ὦ τύμβος, ὦ νυμφεῖον...	0.48
998 · · · 1032	Teiresias: γνώσῃ, τέχνης σημεῖα...	0.57
1033 · · · 1047	Kreon: ὦ πρέσβυ, πάντες...	0.24
1064 · · · 1090	Teiresias: ἀλλ' εὖ γέ τοι...	0.82
1155 · · · 1172	Messenger: Κάδμου πάροικοι καὶ...	0.51
1192 · · · 1243	Messenger: ἐγώ, φίλη δέσποινα...	0.40
	cumulative $\wp$ = 0.50	

(b) Stichomythia

LINES		$\wp$
536 · · · 576	*Ismene, Antigone and Kreon*	0.29
728 · · · 757	*Haimon and Kreon*	0.38
991 · · · 997	*Kreon and Teiresias*	0.70
1047 · · · 1063	*Kreon and Teiresias*	0.12
1172 · · · 1179	*Chorus and Messenger*	0.29
	cumulative $\wp$ = 0.32	

Table 5: Dialogue (Trimeters)

by its length (whence, for instance, it is unlikely for a single hyperbaton to cause more than a few projectivity violations).

3.1 Representational Distortions

The particular format and conventions adopted by the Perseus Project in their dependency annotation can cause some distortions in the analysis of hyperbaton. The first and most easily dispatched of these is that in addition to words, they also include punctuation in the dependency trees (such as commas, periods and question marks).

This is problematic, since such a mark may induce a technical hyperbaton, simply by virtue of what the Perseus annotators have chosen to mark as its "head", to the extent that it means anything at all for a punctuation mark to have a head. To compensate, we simply filter out all punctuation during the parsing stage (see edgeFromXML on p. 15 in Appendix B.1).

Wackernagel's Law: Syntax or Phonology?

Another potential source of distortion is the choice of the annotators to label the members of postpositional particle chains in Wackernagel's Position as being heads of each other in a chain from left to right, such as where μὲν is given as the head of δὴ in Figure 2. I am unconvinced either way as to whether this is the proper relation for particle chains in Dependency Grammar, and simply would observe for the sake of argument that an alternative analysis, where the verb is the head of each, might yield a greater number of projectivity violations, as in Figure 5.

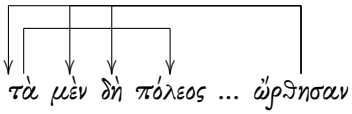


Figure 5: An alternative analysis of the dependency relations in Figure 2 yields a greater number of projectivity violations.

Further, if we allow ourselves to tiptoe outside the bounds of Dependency Grammar for a moment into a more orthodox, derivational approach, we will see that "hyperbata" which arise from enclitics are likely of a very different kind of displacement than that which occurs in, for instance, prepositional phrases or noun phrases. Agbayani and Golston (2010) argue convincingly that the placement of enclitics in so-called "second position" is phonological, and not syntactic. I shall follow their analysis, which holds that the enclitics are *syntactically* in first position, and mandate *phonologically* that they have a word to their left which is from the modified phrase.

According to the Y-Model of Linguistics (Figure 6), phonological concerns cannot affect semantic interpretation, and vice versa. So, when a movement occurs along the path from syntax to phonology, it cannot have any semantic force. Therefore, the *phonological* movement of a word to the position behind a postpositive particle cannot confer any particular focus, which is consistent with our understanding that postpostive conjunctions, asseveratives and

other particles may apply semantic force to their complements, and not *per se* to the words onto which they are enclitic (i.e. the emphasis in τὰ μὲν ... πόλεος is on τὰ πόλεος, not just τά).

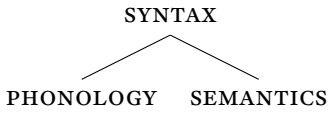


Figure 6: The Y-Model of Linguistics, in which Syntax is interpreted separately into Phonological Form (PF) and Logical Form (LF).

Yet, the other kinds of displacement do indeed induce focus, whether it be weak or strong. And so, whether these hyperbata are to be taken as a kind of movement or not, it is untenable to analyze them as phonological movements: they must be present in the syntax prior to translation to PF. Thus these are distinct from those displacements which arise from Wackernagel's Law.

In this way, a general analysis of hyperbaton which uses Dependency Grammar as its basis will invariably fail to recognize the difference between displacements which are *phonological* in nature and those which are *syntactic*, where the latter are the true target of our investigation. This confounding factor, then, must be kept in mind, when analyzing data from such an experiment.

References

Brian Agbayani and Chris Golston. Second-position is first-position: Wackernagel's law and the role of clausal conjunction. *Indogermanische Forschungen: Zeitschrift für Indogermanistik und allgemeine Sprachwissenschaft*, 115:1–21, 2010.

David Bamman and Gregory Crane. The design and use of a latin dependency treebank. In *Proceedings of the Fifth Workshop on Treebanks and Linguistic Theories (TLT 2006)*, pages 67–78, 2006. URL `http://ufal.mff.cuni.cz/tlt2006/pdf/110.pdf`.

David Bamman and Gregory Crane. The ancient greek and latin dependency treebanks. In Caroline Sporleder, Antal Bosch, and Kalliopi Zervanou, editors, *Language Technology for Cultural Heritage*, Theory and Applications of Natural Language Processing, pages 79–98. Springer Berlin Heidelberg, 2011.

A.A.M. Devine and L.D. Stephens. *Discontinuous syntax: hyperbaton in Greek*. Oxford University Press, Incorporated, 2000.

Euripides and D.J. Mastronarde. *Euripides: Medea*. Cambridge Greek and Latin Classics. Cambridge University Press, 2002.

Simon Marlow. The Haskell 2010 report, 2010. `http://www.haskell.org/onlinereport/haskell2010/`.

Conor McBride and Ross Paterson. Functional pearl: Applicative programming with effects. *Journal of functional programming*, 18(1):1–13, 2008.

Sophocles and M. Griffith. *Sophocles: Antigone*. Cambridge Greek and Latin Classics. Cambridge University Press, 1999.

Appendices

The functions used in parsing and computing the data for this paper are developed here in the programming language Haskell. Haskell is a typed lambda calculus with inductive data types and type classes; the listings below use standard Haskell syntax with the exception of some infix operators to improve readability, and the addition of so-called "idiom brackets", which allow a more syntactically clean presentation of function application within a context (McBride and Paterson, 2008).

A Algorithm & Data Representation

Dependency trees are a recursive data structure with a head node, which may have any number of arcs drawn to further trees (this is called a *rose tree*). We represent them as a Haskell data-type as follows:

$$\textbf{data}\ \mathsf{Tree}\ \alpha = \alpha \curvearrowright [\mathsf{Tree}\ \alpha]$$

This can be read as "For all types α, a Tree of α is constructed from a *label* of type α and a *subforest* of Trees of α," where brackets are a notation for lists.

Given a tree, we can extract its root label or its subforest by pattern matching on its structure as follows:

$$\begin{aligned}
&\mathsf{getLabel} :: \mathsf{Tree}\ \alpha \to \alpha \\
&\mathsf{getLabel}\ (l \curvearrowright _) = l \\
&\mathsf{getForest} :: \mathsf{Tree}\ \alpha \to [\mathsf{Tree}\ \alpha] \\
&\mathsf{getForest}\ (_ \curvearrowright ts) = ts
\end{aligned}$$

A.1 From Edges to Trees

We shall consider each word index to be a *vertex*, and each pair of vertices to be an Edge, which we shall write as follows:

$$\textbf{data}\ \mathsf{Edge}\ \alpha = \alpha \leftrightarrow \alpha\ \textbf{deriving}\ \mathsf{Eq}$$

An Edge α is given by two vertices of type α; the **deriving** Eq statement generates the code that is necessary to determine whether or not two Edges are equal using the $(\equiv)$ operator. In order to perform our analysis, we should wish to transform the raw list of edges into a tree structure. The basic procedure is as follows:

First, we try to find the root vertex of the tree. This will be a vertex that is given as the head of one of the words, but does not itself appear in the sentence:

$$\begin{aligned}
&\mathsf{rootVertex} :: \mathsf{Eq}\ \alpha \Rightarrow [\mathsf{Edge}\ \alpha] \to \mathsf{Maybe}\ \ \alpha \\
&\mathsf{rootVertex}\ es = \mathsf{find}\ (\notin \mathsf{deps})\ \mathsf{heads}\ \textbf{where}
\end{aligned}$$

```
heads = ⟦ (λ(x ↔ y) → x) es ⟧
deps  = ⟦ (λ(x ↔ y) → y) es ⟧
```

If the data that we are working with are not well-formed, there is a chance that we will not find a root vertex; that is why the type is given as Maybe.

Then, given a root vertex, we look to find all the edges that it touches, and try to build the subtrees that are connected with those edges.

```
onEdge :: Eq α ⇒ α → Edge α → 𝔹
onEdge i (x ↔ y) = x ≡ i ∨ y ≡ i

oppositeVertex :: Eq α ⇒ α → Edge α → α
oppositeVertex i (x ↔ y)
  | x ≡ i       = y
  | otherwise = x
```

This is done recursively until the list of edges is exhausted and we have a complete tree structure:

```
treeFromEdges :: Ord α ⇒ [Edge α] → Maybe (Tree α)
treeFromEdges es = ⟦ (buildWithRoot es) (rootVertex es) ⟧ where
  buildWithRoot es root = root ↷ sortedChildren where
    roots           = ⟦ (oppositeVertex root) localVertices ⟧
    children        = ⟦ (buildWithRoot foreignVertices) roots ⟧
    localVertices   = filter (onEdge root) es
    foreignVertices = filter (¬ ∘ onEdge root) es
    sortedChildren  = sortBy (compare `on` getLabel) children
```

A.2 Counting Violations

The basic procedure for counting projectivity violations is as follows: flatten down the tree into a list of edges coindexed by their vertical position in the tree; then traverse the list and see how many times these edges intersect each other.

```
type Level = ℤ
```

The vertical position of a node in a tree is represented as its Level, counting backwards from the total depth of the tree. That is, the deepest node in the tree is at level 0, and the highest node in the tree is at level n, where n is the tree's depth.

```
levels :: Tree α → [[α]]
levels t = fmap (fmap getLabel) $
  takeWhile (¬ ∘ null) $
  iterate (=≪getForest) [t]

depth :: Tree α → ℤ
depth = length ∘ levels
```

We can now annotate each node in a tree with what level it is at:

$$
\begin{aligned}
&\mathsf{annotateLevels} :: \mathsf{Tree}\ \alpha \to \mathsf{Tree}\ (\mathsf{Level}, \alpha)\\
&\mathsf{annotateLevels} = [\![\ \mathsf{aux}\ \mathsf{depth}\ \mathsf{id}\ ]\!]\ \textbf{where}\\
&\quad \mathsf{aux}\ l\ (x \curvearrowright ts) = (l, x) \curvearrowright [\![\ (\mathsf{aux}\ (l-1))\ ts\ ]\!]
\end{aligned}
$$

Then, we fold up the tree into a list of edges and levels:

$$
\begin{aligned}
&\mathsf{allEdges} :: \mathsf{Ord}\ \alpha \Rightarrow \mathsf{Tree}\ \alpha \to [(\mathsf{Level}, \mathsf{Edge}\ \alpha)]\\
&\mathsf{allEdges} = \mathsf{aux} \circ \mathsf{annotateLevels}\ \textbf{where}\\
&\quad \mathsf{aux}\ ((\_, x) \curvearrowright ts) = ts \gg\!= \mathsf{go}\ \textbf{where}\\
&\qquad \mathsf{go}\ t@((l, y) \curvearrowright \_) = (l, \mathsf{edgeWithRange}\ [x, y]) : \mathsf{aux}\ t
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{edgeWithRange} :: \mathsf{Ord}\ \alpha \Rightarrow [\alpha] \to \mathsf{Edge}\ \ \alpha\\
&\mathsf{edgeWithRange} = [\![\ \mathsf{minimum}\ \ \leftrightarrow\ \ \mathsf{maximum}\ ]\!]
\end{aligned}
$$

A handy way to think of edges annotated by levels is as a representation of the arc itself, where the vertices of the edge are the endpoints, and the level is the height of the arc. Now, we can count the violations that occur between two arcs.

$$
\begin{aligned}
&\mathsf{checkEdges} :: \mathsf{Ord}\ \alpha \Rightarrow (\mathsf{Level}, \mathsf{Edge}\ \alpha) \to (\mathsf{Level}, \mathsf{Edge}\ \alpha) \to \mathbb{Z}\\
&\mathsf{checkEdges}\ (l, xy@(x \leftrightarrow y))\ (l', uv@(u \leftrightarrow v))\\
&\quad \mid x \in_E uv \wedge ((y \geqslant v \wedge l > l') \vee y > v) = 1\\
&\quad \mid y \in_E uv \wedge ((x \leqslant u \wedge l > l') \vee u < u) = 1\\
&\quad \mid u \in_E xy \wedge ((v \geqslant y \wedge l < l') \vee v > y) = 1\\
&\quad \mid v \in_E xy \wedge ((u \leqslant x \wedge l < l') \vee u < x) = 1\\
&\quad \mid x \in_E uv \wedge y \in_E uv \wedge l \geqslant l' \qquad\quad = 2\\
&\quad \mid u \in_E xy \wedge v \in_E xy \wedge l \leqslant l' \qquad\quad = 2\\
&\quad \mid otherwise \qquad\qquad\qquad\qquad = 0
\end{aligned}
$$

We determine whether a vertex is in the bounds of an edge using $\cdot \in_E \cdot$.

$$
\begin{aligned}
&\cdot \in_E \cdot :: \mathsf{Ord}\ \alpha \Rightarrow \alpha \to \mathsf{Edge}\ \alpha \to \mathbb{B}\\
&z \in_E x \leftrightarrow y = z > \mathsf{minimum}\ [x, y]\\
&\qquad\qquad\quad \wedge\ z < \mathsf{maximum}\ [x, y]
\end{aligned}
$$

We can now use what we've built to count the intersections that occur in a collection of edges. This is done by adding up the result of checkEdges of the combination of each edge with the subset of edges which are at or below its level:

$$
\begin{aligned}
&\mathsf{edgeViolations} :: \mathsf{Ord}\ \alpha \Rightarrow [(\mathsf{Level}, \mathsf{Edge}\ \alpha)] \to \mathbb{Z}\\
&\mathsf{edgeViolations}\ xs = \mathsf{sum}\ [\![\ \mathsf{violationsWith}\ xs\ ]\!]\ \textbf{where}\\
&\quad \mathsf{rangesBelow}\ (l, \_) = \mathsf{filter}\ (\lambda(l', \_) \to l' \leqslant l)\ xs\\
&\quad \mathsf{violationsWith}\ x \quad = \mathsf{sum}\ [\![\ (\mathsf{checkEdges}\ x)\ (\mathsf{rangesBelow}\ x)\ ]\!]
\end{aligned}
$$

A.3 Computing $\wp$

We introduce a data type $\wp$ of integer-to-integer ratios which may be computed into a rational.

$$\begin{aligned}
&\textbf{data}\ \mathfrak{P} = \mathfrak{P}\ \{\mathsf{violationCount} :: \mathbb{Z}, \mathsf{edgeCount} :: \mathbb{Z}\}\\
&\mathsf{compute}_{\wp} :: \mathfrak{P} \to \mathbb{Q}\\
&\mathsf{compute}_{\wp} = [\![\ \frac{\mathsf{violationCount}}{\mathsf{edgeCount}}\ ]\!]
\end{aligned}$$

Furthermore, $\wp$s generate a monoid, which is an algebraic structure that abstracts out the notion of an identity and an associative binary operation that respects that identity. In this way, we can combine $\wp$ values:

$$\begin{aligned}
&\textbf{instance}\ \mathsf{Monoid}\ \mathfrak{P}\ \textbf{where}\\
&\quad \mathcal{E} = \mathfrak{P}\ 0\ 0\\
&\quad (\mathfrak{P}\ x\ y) \oplus (\mathfrak{P}\ u\ v) = \mathfrak{P}\ (x+u)\ (y+v)
\end{aligned}$$

Finally, $\wp$ may be computed for trees.

$$\begin{aligned}
&\wp :: \mathsf{Ord}\ \alpha \Rightarrow \mathsf{Tree}\ \alpha \to \mathfrak{P}\\
&\wp = [\![\ \mathfrak{P}\ \mathsf{edgeViolations}\ \mathsf{length}\ ]\!] \circ \mathsf{allEdges}
\end{aligned}$$

B Working with the Perseus Treebank

B.1 Parsing the XML

We can express the general shape of a treebank document as follows:

$$\begin{aligned}
&\textbf{type}\ \mathsf{Document} = [\mathsf{Sentence}]\\
&\textbf{data}\ \mathsf{Sentence} = \mathsf{Sentence}\ \{\mathsf{sentenceId} :: \mathbb{Z}, \mathsf{sentenceEdges} :: [\mathsf{Edge}\ \mathbb{Z}]\}
\end{aligned}$$

To construct a Document from the contents of an XML file, it suffices to find all of the sentences.

$$\begin{aligned}
&\mathsf{documentFromXML} :: [\mathsf{Content}] \to \mathsf{Document}\\
&\mathsf{documentFromXML}\ xml = \mathsf{catMaybes}\ [\![\ \mathsf{sentenceFromXML}\ \mathsf{elems}\ ]\!]\ \textbf{where}\\
&\quad \mathsf{elems} = \mathsf{onlyElems}\ xml \ggg\!= \mathsf{findElements}\ (\mathsf{simpleName}\ \texttt{"sentence"})
\end{aligned}$$

Sentences are got by taking the contents of their `id` attribute, and extracting edges from their children.

$$\begin{aligned}
&\mathsf{sentenceFromXML} :: \mathsf{Element} \to \mathsf{Maybe}\ \mathsf{Sentence}\\
&\mathsf{sentenceFromXML}\ e = [\![\ \mathsf{Sentence}\ (\mathsf{readAttr}\ \texttt{"id"}\ e)\ (\mathsf{pure}\ \mathsf{edges})\ ]\!]\ \textbf{where}\\
&\quad \mathsf{edges} \quad = \mathsf{catMaybes}\ [\![\ \mathsf{edgeFromXML}\ \mathsf{children}\ ]\!]\\
&\quad \mathsf{children} = \mathsf{findChildren}\ (\mathsf{simpleName}\ \texttt{"word"})\ e
\end{aligned}$$

An edge is got from an element by taking the contents of its `id` attribute with the contents of its `head` attribute. We make sure to filter out punctuation which would skew our data.

```
edgeFromXML :: Element → Maybe (Edge ℤ)
edgeFromXML e =
  case findAttr (simpleName "form") e of
    Just x | x ∈ [".",",",";",":"] → Nothing
    otherwise → ⟦ (readAttr "head" e) ↔ (readAttr "id" e) ⟧
```

Thence, turn a sentence into a tree by its edges using the machinery from Section A.1.

```
treeFromSentence :: Sentence → Maybe (Tree ℤ)
treeFromSentence (Sentence _ ws) = treeFromEdges ws
```

By applying treeFromSentence to every sentence within a document, we can generate all the trees in a document.

```
treesFromDocument :: Document → [Tree ℤ]
treesFromDocument ss = catMaybes ⟦ treeFromSentence ss ⟧
```

By combining the above, we also may derive a document structure from a file on disk.

```
documentFromFile :: FilePath → IO Document
documentFromFile path = ⟦ (documentFromXML ∘ parseXML) (readFile path) ⟧
```

B.2 Analysis of Data

We compute the cumulative $\wp$ of the trees contained in a document as follows:

```
analyzeDocument :: Document → 𝔓
analyzeDocument doc = mconcat ⟦ ℘ (treesFromDocument doc) ⟧
```

We will wish to compare the $\wp$ for parts of the *Antigone*. A section is given by a two sentence indices (a beginning and an end):

```
data Section = ℤ ··· ℤ
```

Then, the entire document can be cut down into smaller documents by section:

```
restrictDocument :: Section → Document → Document
restrictDocument (start ··· finish) = filter withinSection where
  withinSection (Sentence i _) = i ⩾ start ∧ i ⩽ finish
```