An Algorithm for Pythonizing Rhetorical Structures

Andrew Potter
Computer Science & Information Systems Department
University of North Alabama
Florence, Alabama, USA
apotter1@una.edu

Abstract

Diagrams produced using Rhetorical Structure Theory can be both informative and engaging, providing insight into the properties of discourse structures and other coherence phenomena. This paper presents a deep dive into these diagrams and shows how an RST analysis can be reconceived as an emergent process. The paper describes an algorithm for transforming RST diagrams into Pythonic relational propositions and applies it to a set of RST analyses. The resulting expressions are isomorphic with RST diagrams as well as machine processable. As executable specifications of discourse structure, they support scalable applications in applied and theoretical studies. Several sample applications are presented. The transformation process itself suggests an alternative to the traditional view of rhetorical structures as recursive trees. The construction of coherence is shown to be a bottom-up synthesis, wherein discourse units combine to form relational propositions which in turn rendezvous with other relational propositions to create increasingly complex expressions until a comprehensive analysis is produced. This progressive bottom-up development of coherence is observable in the performance of the algorithm.

1 Introduction

An RST analysis is a picture of a discursive process. It shows how the elements of a text work together to support the writer's purpose. The purpose could be anything—to support the claim of an argument, to explain the result of a causal process, to bring an anecdote to a satisfying conclusion, to assure the punchline of a joke, or to solicit a donation from the reader. In a well-written text, every part plays a role, with each part ultimately supporting the writer's intended effect. An RST analysis depicts this process, it explains how the text does what it does. A competent analysis of a well-written text is an aesthetically pleasing appreciation of the writer's mastery. This is among the strengths of RST. It is also a limitation.

Many interesting and useful things have been accomplished, thanks to RST. Among these are automated text generation, discourse parsing, summarization, machine translation, essay scoring, coherency studies, and numerous other applications. And yet it seems the diagrams that make it distinctive tend to play only a bit part in these studies. In their survey of applications of RST, for example, Taboada and Mann (2006a) found they could recount the history of achievements in RST without need for any diagrams whatsoever. It is not unusual for papers on the topic to provide only a solitary diagram used solely for the purpose of conveying the core idea of what RST is. RST diagrams may be essential in explaining the theory, but thereafter tend to be treated as dispensable. This suggests that perhaps we have yet to fully leverage the concept of RST analyses as depictions of discursive processes. Hence the motivation for this research.

If we could develop a method for transforming RST diagrams into executable code, into a notation that would be machine processable, conceptually faithful to RST, human readable, and maybe even page-count friendly, from this it might be possible to develop systems that would enable us to more deeply explore what RST is, what it has to offer, and thus enable us to look directly into the diagrams, not just as stepping stones to some other research topic, but in and of RST itself. This could lead to a deeper understanding of discursive coherence, not only as conceived by Rhetorical Structure Theory, but as conceptualized in other discourse formalisms as well.

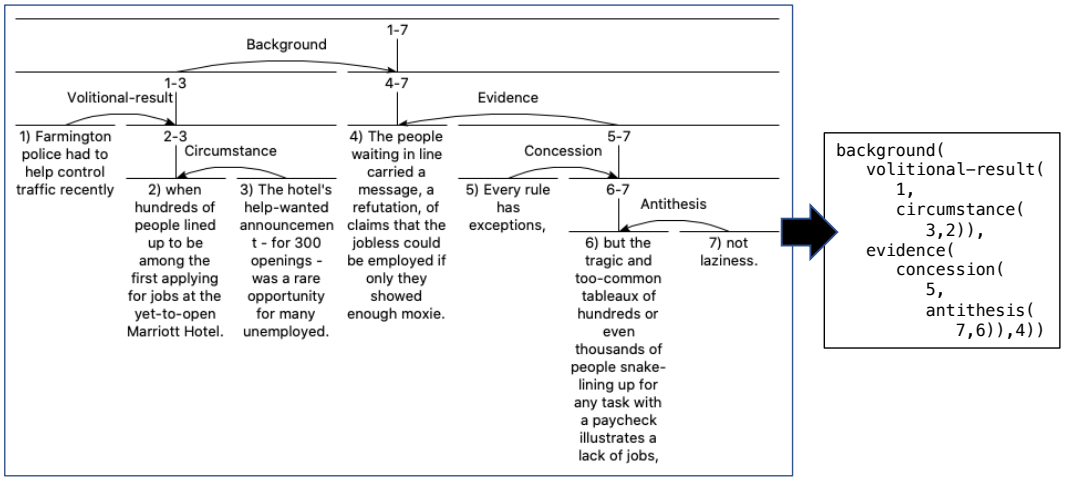


Figure 1: Pythonizing the *Not Laziness* RST analysis

The Pythonization of rhetorical structures is a process for transforming RST analyses into expressions conformant with the Python programming language, as illustrated in Figure 1. This paper describes an algorithm for making these transformations and provides direction for how these expressions can be applied to a range of research questions. I will also show how the algorithm itself sheds light on what a rhetorical structure is, how its structures come to exist, and what they mean for discursive coherence. What follows here then is a review of related literature, an overview of the motivation for developing the algorithm, and a description of the algorithm itself. This is followed by a discussion of the algorithm's potential applications and their implications. The paper concludes with a summary of the results of this study.

2 Related Research

When Rhetorical Structure Theory was originally developed by Mann and Thompson (1988) it was intended for use in automated text generation, but soon became more widely used as a descriptive theory of discourse coherence. RST is one among several theories of coherence relations; some others of note include the Penn Discourse Treebank (Webber, Prasad, Lee, & Joshi, 2019), Segmented Discourse Representation Theory (Asher & Lascarides, 2003), a taxonomic approach to coherence relations (Sanders, Spooren, & Noordman, 1992), Hobb's (1979) theory of coherence and co-reference, Polanyi's (1987) linguistic discourse model, Van Dijk's (1979) pragmatic connectives, and Grimes' (1975) rhetorical predicates. Among the distinctive characteristics of RST are its theoretical basis and its diagrammatic technique. Its theoretical basis posits that an analysis of a text will consist of a set of schema applications, subject to the constraints of completeness, connectedness, uniqueness, and adjacency. Mann and Thompson (1988) note that the first three of these constraints are sufficient to require RST analyses to take the form of tree structures. Thus as a theory of coherence relations, RST is not limited to identifying relation pairs, but provides comprehensive specifications of the functional organization of complete texts. This in turn is reflected in the RST diagramming technique, which provides a tree-shaped rendering of the organization of the analyzed text.

During its history RST has gone through several adjustments beyond the original version (Mann & Thompson, 1987, 1988), with various extensions and adaptations (Mann & Taboada, 2005; Taboada & Mann, 2006b). Carlson and Marcu (2001) extended RST with additional relations and a somewhat different approach, putting greater emphasis on syntactic devices, with the aim of increasing analytical efficiency and scalability. The annotation guidelines defined by Stede, Taboada, and Das (2017) adhere closely to those of Mann and Thompson, with minor variations.

Relational propositions, developed by Mann and Thompson (1986) prior to and concurrently with their development of RST, are propositional analogs to RST structures, with relations being expressed as implicit assertions occurring between clauses. Mann and Thompson (2000) confined their analysis of relational propositions to discourse unit pairs, and declined to apply it to more complex

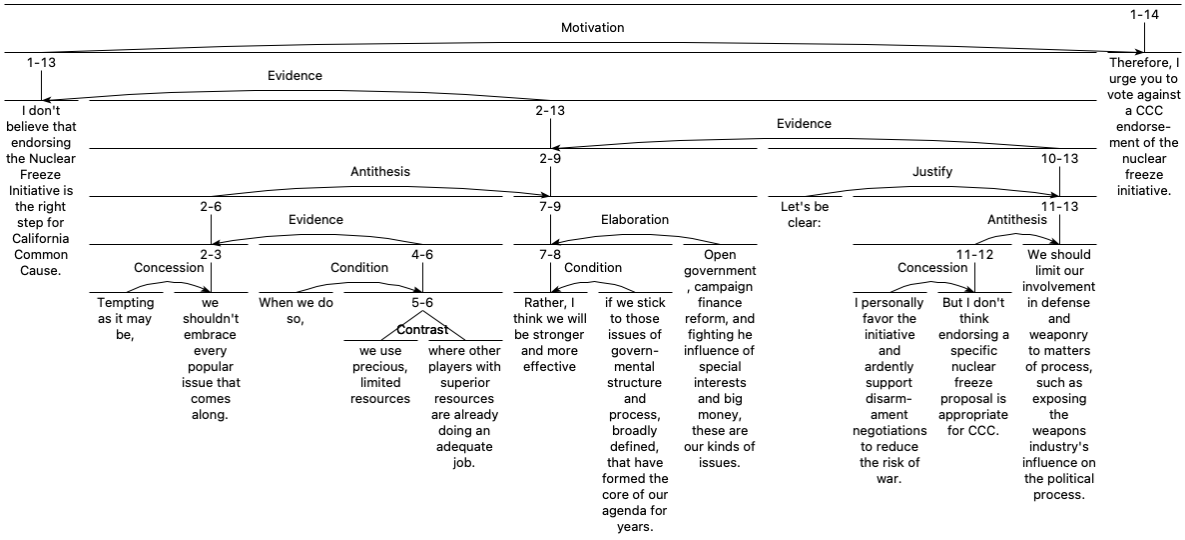


Figure 2: The *Common Cause* Analysis (Thompson & Mann, 1987)

expressions. Potter (2018) developed a notation for nested relational propositions, enabling the restatement of complete RST analyses as relational propositions. That this notation is syntactically Pythonic is fundamental to the algorithmization of RST as described in this paper.

Several tools have been developed for creating RST analyses. Among the more widely used of these are RST Tool, developed by O'Donnell (1997) and more recently rstWeb from Zeldes (2016). RST Tool is a multiplatform graphical interface for RST mark up. rstWeb is a browser-based tool developed for RST and other discourse relational formalisms. It enables annotators to work online using a browser. Both server and local versions are available. Both RST Tool and rstWeb store or export RST analyses in a common XML format.

3 Theoretical Framework

RST analyses and their respective relational propositions are structurally and semantically isomorphic, enabling transformation from one representation to the other. The interest here is in providing an automated means for transforming RST analyses into relational propositions. The motivation for doing so should be clear: while RST presents organizational properties of a text as diagrams, relational propositions present identical information in functional form. The predicates of the relational propositions may be defined as Python functions. Through transformation, the RST diagram is redefined as an Pythonic expression. Once a diagram has been transformed, it can be supported by a set of functions implementing each of the relational predicates. That is, their implementation consists in defining a set of corresponding functions. These definitions are application specific, and dependent upon the research objective. The possibilities are open-ended. Several examples are provided in Section 5.

4 Pythonizing Rhetorical Structures

The algorithm uses an RST-Tool XML file as input and generates a Pythonic relational proposition as output. While not rocket science, its behavior has yielded some interesting observations concerning the process of discourse coherence. Therefore, a look at how the algorithm works is worthwhile. (Only the core algorithm is presented here; the complete code is being made available as an open-source project.)[1]

Processing initiates at the top of the RST structure and descends recursively down each branch to the elementary discourse units. From there it constructs the leaf relational propositions and works its way back up through the structure, building the relational proposition as its goes.

Nesting structures are discovered as *span relations*. While RST-Tool uses these spans, or vertical bars, to cue visual indicators of structural subordination, for transformation they are treated as precedence operators. A span takes precedence over its satellites. So, for example, in Figure 3, the

[1] https://github.com/anpotter/pycrst

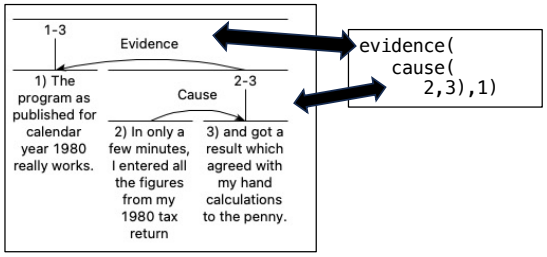


Figure 3: Span Relations as Precedence Operators

span identified as 2-3 is nested within 1-3, and therefore takes precedence over the outer span, thus defining the order of evaluation.

The core function for the transformation is simple. When called, it is passed a relational proposition object:

```
class RelProp:
  def __init__(self,rel,sat,nuc,type,text):
   self.rel = rel
   self.sat = sat
   self.nuc = nuc
   self.type = type
   self.text = text.strip() if text else ""
```

The algorithm's first order of business is to determine whether the relational proposition is the top span of the RST structure. If so, it simply steps down one level into the tree and makes a recursive call to the span's satellite:

```
def gen_exp(rp):
   if is_top(rp) and is_span_type(rp):
      return gen_exp(get_nuc(rp.sat))
```

This initiates a series of recursive calls as the function works its way down into the structure. With each call the function checks to determine whether the relational proposition under consideration is of type `span`. If so, it retrieves the span's satellites. If there is more than one satellite related to the span, the `converge` function is called to specify a `convergence` relation among the satellites with respect to the span:

```
   elif is_span_type(rp):
      if get_sat_count(rp) > 1:
         exp = converge(rp)
```

When there is only one satellite, the algorithm determines whether the proposition is multinuclear. If it is, the algorithm makes a recursive call to itself for multinuclear handling. It then links the satellite to the relational proposition. Otherwise, it makes a recursive call to the satellite and links the returned value to the span's child structure. If the span has no satellite, the satellite formats the proposition using its child structure as satellite and returns the expression:

```
      else:
         nuc_exp = gen_exp(get_span_nuc(rp))
         sat = get_sat(rp)
         if sat:
            if is_multi_type(sat):
               sat.nuc = nuc_exp
               exp = format_rp( sat.rel,
                        gen_exp(sat),nuc_exp)
            else:
               sat_rp = get_span_nuc(sat)
               if sat_rp:
                  sat.sat=gen_exp(sat_rp)
               exp = format_rp(
                        sat.rel,sat.sat,nuc_exp)
         else:
            exp=format_rp(rp.rel,nuc_exp,rp.nuc)
```

If the relational proposition is not of type span, then it must be either a `segment` or a `multinuclear`. If it is of type `segment`, the algorithm first checks to determine whether it has multiple satellites, and if so, it calls the `converge` function to perform special handling. Otherwise, the algorithm determines whether any satellites linked to the segment are multinuclear, and makes recursive calls as needed to format the relational proposition, returning that to the caller:

```
   elif is_segment(rp):
      if get_sat_count(rp) > 1:
         exp = converge(rp)
      else:
         sat = get_sat(rp)
         if not sat:
            exp = format_rp(rp)
         elif is_multi_type(sat):
            exp = format_rp(sat.rel,
                gen_exp(sat), rp.sat)
         else:
            exp = gen_exp(sat)
```

If the relational proposition is multinuclear, the algorithm makes recursive calls for each of its nuclei and formats the results. It then determines whether the multinuclear relation has satellites, and if so, performs a convergence operation similar to that performed on the span and segment types.

For each type, the resulting expression is returned to the calling code. That is the core algorithm. It has tested successfully for 265 RST analyses including the GUM Corpus (Zeldes, 2017), the STS-Corpus (Potter, 2023), as well as a miscellany of analyses from the RST literature. Many of the analyses transformed are well over 100 units in length.

Because nesting of an expression reflects the depth of its RST structure, relational propositions can be difficult to read, so a pretty-printer was developed for post-processing. Test functions are provided to assure unit continuity and span

handling. Here is the generated expression for Thompson and Mann's (1987) *Common Cause* analysis, shown in Figure 2, transformed and prettified:

```
motivation(
   evidence(
      evidence(
         justify(
            10,
            antithesis(
               concession(
                  11,12),13)),
         antithesis(
            evidence(
               condition(
                  4,
                  contrast(
                     5,6)),
               concession(
                  2,3)),
            elaboration(
               9,
               condition(
                8,7)))),1),14)
```

Formatted as such, the satellite nucleus pairs align beneath their enclosing relations, and the structural depth of the discourse is indented from left to right. Multiple levels of `evidence` support unit 1, which then is used to provide `motivation` for unit 14. The relational proposition shows the rhetorical organization of the text, but unlike the diagram it does not reflect the linearity of the discourse. A relational proposition is an abstract expression of a coherence process as reenacted by the algorithm.

5 Applying Pythonized Rhetorical Structures

An application of a relational proposition consists of a set of functions that implement the relational predicates appearing in the proposition. If, for example, a relational proposition uses `evidence` and `antithesis`, the applications must provide functions by those names. The processing performed by the functions is application specific. If an application is used simply to tabulate data about a relation proposition(s), the functions may be very simple. However, the nesting of the relational propositions defines their precedence, with each nested proposition's return values being passed to its parent. Reusable functions allow relational propositions to be treated as plug-ins within a framework. Moderately sized bulk processing can be configured by storing relational propositions as string data in Python dictionaries for runtime evaluation as Python code.

Some but not all applications are precedence sensitive. Precedence sensitive applications rely on the logic implicit in the nesting of relational propositions. For example, an application designed for a study in argument accrual may need to backtrack through discourse threads when a structural convergence is encountered. This could be used to determine the relation types of the accruing threads.

The following examples illustrate how relational propositions can be used. The first is a simple framework for measuring the frequency of argumentative relations as identified by Azar (1999). The purpose of this example is to show how readily Pythonic representations of RST analyses can be outfitted for practical applications. The second example performs an automated reduction of relational propositions to logic and then uses the logic to support examination of purported simultaneous RST analyses. The third example shows how runtime evaluations of relational propositions can be used to reenact coherence development in a discourse.

5.1 Computing an RST Metric

Using relational propositions as code requires a set of functions corresponding to the relations used in the relational proposition. Here is a set of functions for determining the Azar Score for the relations used in Thompson and Mann's (1987) *Common Cause* analysis:

```
def antithesis(*argv): return tally(argv), argv
def concession(*argv): return tally(argv), argv
def evidence(*argv): return tally(argv), argv
def motivation(*argv): return tally(argv), argv
def justify(*argv): return tally(argv), argv
def condition(*argv): return tally(argv), argv
def contrast(*argv): return tally(argv), argv
def elaboration(*argv): return tally(argv), argv
def condition(*argv): return tally(argv), argv
```

This list can be extended to include an entire RST relation set. Since every relation receives the same processing, they all call the same function:

```
def tally(argv):
    relname = sys._getframe(1).f_code.co_name
    argumentative() if relname in arg_rels \
                    else nonargumentative()
    return relname
```

From these tallies the Azar Score is as the ratio, expressed as a decimal, of argumentative to non-argumentative relations in a text. Azar (1999) designated a subset of relations as argumentative, including EVIDENCE, MOTIVATION, JUSTIFY,

ANTITHESIS, and CONCESSION. What distinguishes these relations, according to Azar, is that their loci of effect are in their nuclei and that the intended effect is to persuade, move, or otherwise influence the reader to accept the content of the nucleus. When the program is run, the *Common Cause* relational proposition (shown previously in Section 4.0) is evaluated causing the function for each relation to be executed. This in turn calls the `tally` function which increments the relation type counters as indicated by relation type. Using this, we determine that the Azar score for Thompson and Mann's (1987) *Common Cause* analysis is 0.69. This can be performed for any relational proposition.

5.2 Automating logical reductions

A method for reducing rhetorical structures to propositional logic was described by Potter (2018, 2021). Each relation was assigned a logical definition such that complex logical expressions could be constructed by mapping from the relational propositions to the logic. This can be automated by providing a set of functions where each function supports a logical interpretation of the relational predicate. Two versions of this have been developed.[2] One version consists of a set of Boolean functions that evaluate the relational proposition. The second version, which is the version presented here, returns a logical expression corresponding to the relation. The expression uses a conventional notation for propositional logic. A subset of definitions is as follows:

```
def neg(p):     return f'¬{p}'
def conj(p, q): return f'({p} ∧ {q})'
def disj(p, q): return f'({p} ∨ {q})'
def exdisj(p, q):
      return f'({disj(p, q)} ∧ {neg(conj(p, q))})'

def imp(p, q):  return f'({p} → {q})'
def mp(p, q):  return f'{imp(conj(imp(p,q),p),q)}'
def djs(p,q):  return f'{imp(conj(disj(p,q),
neg(p)),q)}'

# selected relations
def evidence(s,n):      return mp(s,n)
def concession(s,n):
    return mp(neg(imp(s,neg(n))),n)
def condition(s,n):     return imp(s,n)
def cause(s,n):         return mp(s,n)
def antithesis(s,n):    return djs(s,n)
def motivation(s,n):    return mp(s,n)
def enablement(s,n):    return mp(s,n)
def justify(s,n):       return mp(s,n)
def background(s,n):    return mp(s,n)
def elaboration(s,n):   return mp(s,n)
def evaluation(s,n):    return mp(n,s)
def contrast(s,n):      return exdisj(s,n)
def result(s,n):        return mp(s,n)
def circumstance(s,n):  return mp(s,n)
def volitionalCause(s,n): return(cause(s,n))
def volitionalResult(s,n): return(cause(n,s))
def conjunction(n,o):  return conj(n,o)
```

Two rules of inference are required: *modus ponens* and *disjunctive syllogism*. Definitions for the logical primitives, conjunction, disjunction, and negation are also provided. This is all that is necessary for the reduction. As an example, we can apply this to segments 4 through 7 of the *Not Laziness* analysis:

```
exp = evidence(concession(5,antithesis(7,6)),4)
print(exp)
```

The `antithesis` relational proposition is evaluated first, generating the disjunctive syllogism:

```
(((7 ∨ 6) ∧ ¬7) → 6)
```

The `concession` relation is evaluated next. There the writer concedes the situation presented in the satellite and asserts that, though there might appear to be an incompatibility between the satellite and the nucleus, there is no actual incompatibility. The writer holds the nucleus in positive regard, and by indicating a lack of incompatibility with its satellite, the writer seeks to increase the reader's positive regard for the nucleus (Thompson, 1987). Since the satellite does not imply the negation of the nucleus it therefore implies its affirmative. Nesting the disjunctive syllogism within the concession results in the following:

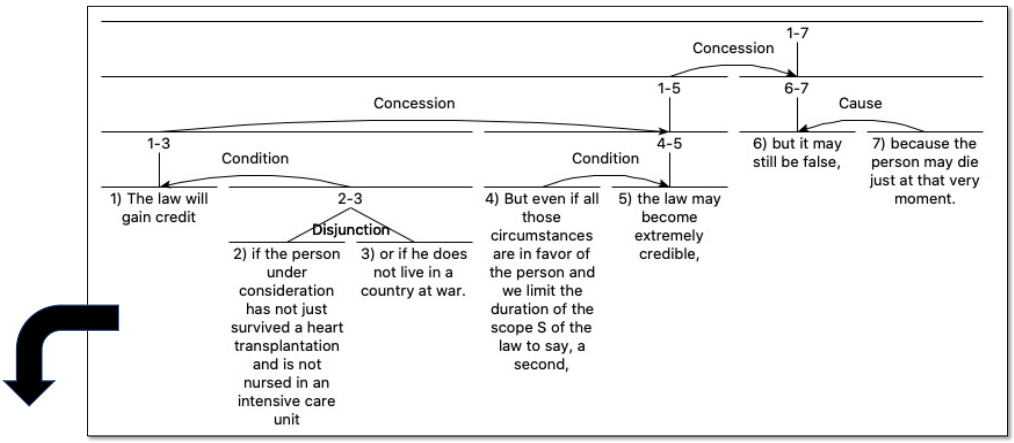
```
(((¬(5 → ¬(((7 ∨ 6) ∧ ¬7) → 6)) →
(((7 ∨ 6) ∧ ¬7) → 6)) ∧ ¬(5 → ¬(((7
∨ 6) ∧ ¬7) → 6))) → (((7 ∨ 6) ∧ ¬7)
→ 6))
```

This expression is nested as the antecedent and minor premise of the `evidence` *modus ponens*:

```
((((((¬(5 → ¬(((7 ∨ 6) ∧ ¬7) → 6)) →
(((7 ∨ 6) ∧ ¬7) → 6)) ∧ ¬(5 → ¬(((7 ∨
6) ∧ ¬7) → 6))) → (((7 ∨ 6) ∧ ¬7) →
6)) → 4) ∧ (((¬(5 → ¬(((7 ∨ 6) ∧ ¬7)
→ 6)) → (((7 ∨ 6) ∧ ¬7) → 6)) ∧ ¬(5
→ ¬(((7 ∨ 6) ∧ ¬7) → 6))) → (((7 ∨
6) ∧ ¬7) → 6))) → 4)
```

Potter (2018) claimed any text analyzable using RST could be reduced to propositional logic. The method described here shows the process can be fully automated. The results can be used to support fine-grained examination of RST analyses. For

[2] https://github.com/anpotter/RBTL




```
324 : cause(7,6)
327 : condition(4,5)
351 : disjunction(2,3)
358 : condition(disjunction(2,3),1)
311 : concession(condition(disjunction(2,3),1), condition(4,5))
311 : concession(concession(condition(disjunction(2,3),1), condition(4,5)),cause(7,6))
```

Figure 4: Reenacting the *Heart Transplant* Analysis

example, in their 1992 paper, Moore and Pollack argued that there are obvious cases where both presentational and subject matter analyses can be made of the same text. They based their claim on several examples. Here is the text of their first example:

1) George Bush supports big business.
2) He's sure to veto House Bill 1711.

Moore and Pollack say it is plausible that there is an EVIDENCE relation between unit *2*, as nucleus of the relation, and unit *1*, the satellite. So the relational proposition is `evidence(1,2)`. The intended effect of EVIDENCE is that the satellite increases the reader's belief in the nucleus. For this to hold, it would therefore be necessary that the reader already believe in the satellite, since it is an assumption of the argument. The logical reduction of the relational proposition echoes this, showing unit 2 as inferred from unit `1`: `(((1 → 2) ∧ 1) → 2)`.

In their second analysis of the same example, Moore and Pollack say that it is plausible that there is a VOLITIONAL-CAUSE relation between unit 1, as nucleus of the relation and unit 2, the satellite. So the relational proposition is now `volitional–cause(2,1)`, such that unit 2 provides a causal explanation for unit 1. As such, George Bush's support for the bill supports the inference that he supports big business: `(((2 → 1) ∧ 2) → 1)`. So in one analysis, 1 is inferred from 2, and in the other, 2 is inferred from 1. This does not affirm that multiple analyses must be supported, but rather that there are two quite different readings of the text. And once we allow arbitrary assumptions necessary for multiple decontextualized readings, all bets are off as to the correct analysis. For all we know, the bill might have been something strongly disfavored by big business, but that President Bush intended to support it anyway, making the relation between the two units CONCESSION. Similar issues arise with Moore and Pollack's second example:

1) Come home by 5:00.
2) Then we can go to the hardware store before it closes.
3) That way we can finish the bookshelves tonight.

The first of their analyses for this example uses the MOTIVATION relation: Finishing the bookshelves motivates going to the hardware store, and taken together these motivate coming home by 5:00: `motivation(motivation(3,2),1)`:

`((((((3 → 2) ∧ 3) → 2) → 1) ∧ (((3 → 2) ∧ 3) → 2)) → 1)`

The second analysis uses the CONDITION relation: coming home by 5:00 is a condition on going to the hardware store, and together these are a condition for finishing the bookshelves: `condition(condition(1,2),3)`, or

```
((1 → 2) → 3)
```

For the MOTIVATION analysis to be realizable, it is necessary that the reader accept the initial premise of the relation, the bookshelves can be finished tonight. So in one case, there is a line of reasoning leading from unit 3 to unit 1, and in the other, leading from 1 to 3. Once again, the analyses are not simultaneous. Any possibility of simultaneous analysis relies on an insufficiency of information. Decontextualized, obscure, or ambiguous texts are hard to understand, and this should be expected to impede analysis. The use of semantic relations for pragmatic purposes is identified by means of a determination of purpose, and therefore there is not really an overlap at all. If there is a problem here, it is with the limiting circumstances under which the theory is applied, not with the theory itself.

5.3 Reenacting Rhetorical Structures

The transformation algorithm can be used to reenact the process of structure formation. This process initiates with the innermost relations of each branch and works its way upward. To demonstrate this, I instrumented the algorithm with debug prints and applied it to the *Heart Transplant* analysis shown above in Figure 4. As the algorithm descends into the tree it seeks the precedence, ultimately finding it in the leaves and their relations. These low-level relational propositions are transformed first. The algorithm continues upward, constructing more complex expressions from the bottom up, until a complete relational proposition is formulated. With each relational proposition, there is a transference of intended effect from satellite to nucleus. Without the satellite-nucleus transfer, we would have merely an empty structure. The only way to a nucleus is through its satellites. But all this is at odds with the view of RST trees as recursive.

Recursion, it has been said, is pervasive in discourse, semantically, rhetorically, structurally, grammatically, and thematically (e.g., Hwang, 1989; Muhammad, 2011; Pinker & Jackendoff, 2005; Polanyi, 1988). And of rhetorical structures, it has been widely observed that not only are they are tree-shaped (Bateman, 2001; Grasso, 2002; Mann & Thompson, 1988), but that the units comprising the tree are linked to one another recursively (Das & Taboada, 2018; Demberg, Asr, & Scholman, 2019; Guerini, Stock, & Zancanaro, 2004; Peldszus & Stede, 2016; Taboada & Mann, 2006b). While these observations are structurally correct, they are functionally incomplete. As the reenactment of rhetorical structures shows, RST tree structures define themselves from the bottom up. Elementary units combine to form relational propositions and these propositions rendezvous with other propositions to create increasingly complex expressions. The tree is the result of a pragmatic process. Through this process rhetorical intentionality develops.

This becomes more obvious when analyzing a nonsensical text, where the RST linkage is discernible, but the satellite-nucleus transfers fail, as shown in Figure 5. The structure is discoverable even when the intention is unachievable. Texts may be analyzable, and if so, they will be transformable and reducible, and yet at the same time nonsensical.

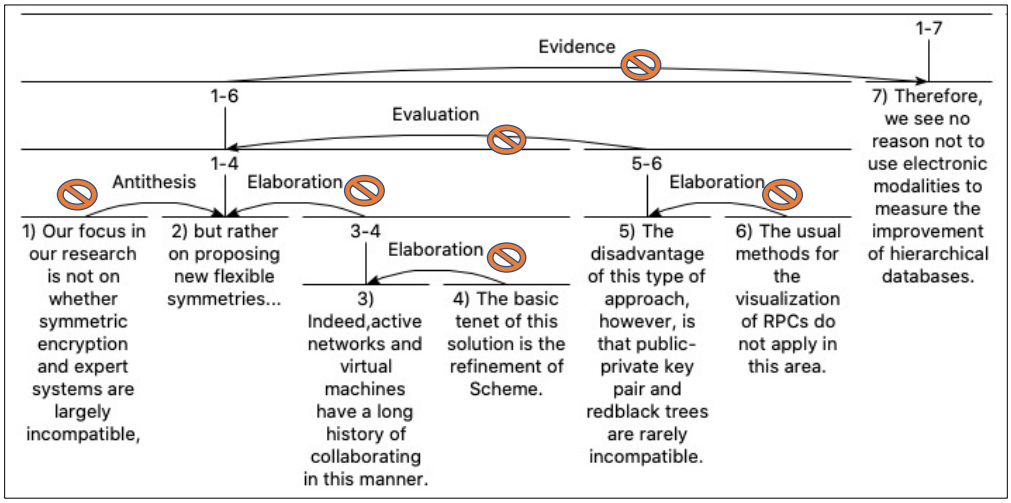


Figure 5: An Analysis of Nonsense

This analysis is of a passage from a paper created using the SCIgen nonsense paper generator (Stribling, Krohn, & Aguayo, 2005). The analysis is superficially plausible, it transforms correctly, and builds up just like any other:

```
evidence(
   evaluation(
      elaboration(
         6,5),
      conjunction(
         antithesis(
            1,2),
         elaboration(
            elaboration(
               4,3),2))),7)
```

And yet the text is nonsensical. If such nonsense is analyzable, what does this say about RST? Is coherence as defined by RST merely window dressing? On the contrary, the inferences within the text, if read with attention to content, are *non sequitur* to the point of being ridiculous. The ELABORATIONS are not really elaborations, the EVALUATION is not evaluative, the EVIDENCE is not evidential. The superficiality of the analysis mirrors that of the text. For an RST analysis to be sound, the bottom-up transfer of intention from satellite to nuclei must be assured. This echoes Marcu's (2000) strong nuclearity thesis, but from a bottom-up perspective. A nucleus acquires its "strength" through its relationship with its satellite. Transference of intention upward shows that, in a coherent text, each relation subsumes its underlying structure. An RST analysis is the realization of a discursive process. The constituents of a text organize from the bottom up to realize the writer's purpose.

6 Conclusion

The algorithm presented here provides a tool for transforming RST analyses into machine processable code. As such, an RST analysis need not be regarded as an end product, but rather as a starting point for deeper investigation. Of particular interest are studies using Pythonic relational propositions to investigate threads of coherence. The algorithm is scalable to large analysis sets.

The bottom-up synthesis of relational propositions generates purely abstract renditions of coherence processes. This validates the theory of relational propositions. Relational propositions implicitly assert the intentionality between discourse units. Coherence arises out of the instantiation of these propositions, not only at the unit level but among the complex spans that bring structure to the rhetorical space. Within this space, a span is a container of an intentional effect. It is through spans that structure arises. While we may view the process from the top down, as is the tendency with RST, intentionality develops from the bottom up. The tree-structures characteristic of RST are the end-result of this process.

References

Nicholas Asher, & Alex Lascarides. 2003. *Logics of conversation*. Cambridge, UK: Cambridge University Press.

Moshe Azar. 1999. Argumentative text as rhetorical structure: An application of rhetorical structure theory. *Argumentation, 13*(1), 97-114.

John A. Bateman. 2001. Between the leaves of rhetorical structure: Static and dynamic aspects of discourse organisation. *Verbum, 23*(1), 31-58.

Lynn Carlson, & Daniel Marcu. 2001, September. Discourse tagging reference manual. Retrieved from ftp://ftp.isi.edu/isi-pubs/tr-545.pdf

Debopam Das, & Maite Taboada. 2018. RST Signalling Corpus: A corpus of signals of coherence relations. *Language Resources and Evaluation, 52*(1), 149-184. doi:10.1007/s10579-017-9383-x

Vera Demberg, Fatemeh Torabi Asr, & Merel Scholman. 2019. How compatible are our discourse annotations? Insights from mapping RST-DT and PDTB annotations.

Floriana Grasso. 2002. Towards computational rhetoric. *Informal Logic, 22*(3), 195-229.

Joseph E. Grimes. 1975. *The thread of discourse*. Berlin: Mouton.

M. Guerini, O. Stock, & M. Zancanaro. 2004. Persuasive Strategies and Rhetorical Relation Selection. In *Proceedings of the ECAI Workshop on Computational Models of Natural Argument*. Valencia, Spain.

Jerry R. Hobbs. 1979. Coherence and coreference. *Cognitive Science, 3*, 67-90.

Shin Ja Joo Hwang. 1989. Recursion in the paragraph as a unit of discourse development. *Discourse Processes: A Multidisciplinary Journal, 12*(4), 461-478.

William C. Mann, & Maite Taboada. 2005, October. An introduction to rhetorical structure theory (RST). Retrieved from http://www.sil.org/~mannb/rst/rintro99.htm

William C. Mann, & Sandra A. Thompson. 1986. Relational propositions in discourse. *Discourse Processes, 9*(1), 57-90.

William C. Mann, & Sandra A. Thompson. 1987. *Rhetorical structure theory: A theory of text organization* (ISI/RS-87-190). Retrieved from Marina del Rey, CA:

William C. Mann, & Sandra A. Thompson. 1988. Rhetorical structure theory: Toward a functional theory of text organization. *Text - Interdisciplinary Journal for the Study of Discourse, 8*(3), 243-281.

William C. Mann, & Sandra A. Thompson. 2000. *Toward a theory of reading between the lines: An exploration in discourse structure and implicit communication.* Paper presented at the Seventh International Pragmatics Conference, Budapest, Hungary.

Johanna Doris Moore, & Martha E Pollack. 1992. A problem for RST: The need for multi-level discourse analysis. *Computational Linguistics, 18*(4), 527-544.

Manaal Jassim Muhammad. 2011. *The use of Rhetorical Structure Theory in political editorials: A contrastive study of text analysis with special reference to its application as text-based generation.* University of Pune, Pune, India.

Michael O'Donnell. 1997. RST-Tool: An RST analysis tool. In *Proceedings of the 6th European Workshop on Natural Language Generation*. Duisburg, Germany: Gerhard-Mercator University.

Andreas Peldszus, & Manfred Stede. 2016. Rhetorical structure and argumentation structure in monologue text. In *Proceedings of the 3rd Workshop on Argument Mining* (pp. 103-112). Berlin, Germany: Association for Computational Linguistics.

Steven Pinker, & Ray Jackendoff. 2005. The faculty of language: what's special about it? *Cognition, 95*(2), 201-236.

Livia Polanyi. 1987. *The linguistic discourse model: Towards a formal theory of discourse structure*. Cambridge, MA: Bolt, Beranek, and Newman, Inc.

Livia Polanyi. 1988. A formal model of the structure of discourse. *Journal of Pragmatics, 12*(5-6), 601-638.

Andrew Potter. 2018. Reasoning between the lines: A logic of relational propositions. *Dialogue and Discourse, 9*(2), 80-110.

Andrew Potter. 2021. Text as tautology: an exploration in inference, transitivity, and logical compression. *Text & Talk*. doi:doi:10.1515/text-2020-0230

Andrew Potter. (2023). *STS-Corpus*. Retrieved from: https://github.com/anpotter/STS-Corpus

Ted J M Sanders, W P M Spooren, & L G M Noordman. 1992. Toward a taxonomy of coherence relations. *Discourse Processes, 15*, 1-35.

Manfred Stede, Maite Taboada, & Debopam Das. 2017. *Annotation guidelines for rhetorical structure*. Retrieved from Potsdam and Burnaby: http://www.sfu.ca/~mtaboada/docs/research/RST_Annotation_Guidelines.pdf

Jeremy Stribling, Maxwell Krohn, & Daniel Aguayo. 2005. SCIgen - An automatic CS paper generator. Retrieved from https://pdos.csail.mit.edu/archive/scigen/

Maite Taboada, & William C. Mann. 2006a. Applications of rhetorical structure theory. *Discourse Studies, 8*(4), 567-588.

Maite Taboada, & William C. Mann. 2006b. Rhetorical structure theory: Looking back and moving ahead. *Discourse Studies, 8*(3), 423-459.

Sandra A. Thompson. 1987. 'Concessive' as a discourse relation in expository written English. In B. Joseph & A.M. Zwickey (Eds.), *A Festschift for llse Lehiste* (pp. 64-73). Columbus, Ohio: Ohio State University.

Sandra A. Thompson, & William C. Mann. 1987. Antithesis: A study in clause combining and discourse structure. In Ross Steele & Terry Threadgold (Eds.), *Language Topics: Essays in Honour of Michael Halliday, Volume II* (pp. 359-381). Amsterdam: John Benjamins.

Teun A. Van Dijk. 1979. Pragmatic connectives. *Journal of Pragmatics*, 447-456.

Bonnie Webber, Rashmi Prasad, Alan Lee, & Aravind Joshi. 2019. The Penn Discourse Treebank 3.0 annotation manual.

Amir Zeldes. 2016. rstWeb – A browser-based annotation interface for Rhetorical Structure Theory and discourse relations. In *Proceedings of NAACL-HLT 2016 (Demonstrations)* (pp. 1-5). San Diego, California: Association for Computational Linguistics.

Amir Zeldes. 2017. The GUM corpus: Creating multilayer resources in the classroom. *Language Resources and Evaluation, 51*(3), 581-561.