# Using and comparing Rhetorical Structure Theory parsers with rst-workbench

**Arne Neumann**
Independent researcher
rst-workbench@arne.cl

## Abstract

I present rst-workbench, a software package that simplifies the installation and usage of numerous end-to-end Rhetorical Structure Theory (RST) parsers.[1] The tool offers a web-based interface that allows users to enter text and let multiple RST parsers generate analyses concurrently. The resulting RST trees can be compared visually, manually post-edited (in the browser) and stored for later usage.

## 1 Introduction

Rhetorical Structure Theory (RST) provides a formalism for hierarchical text organization that can be applied to a wide range of natural language processing tasks, ranging from text generation (Marcu, 1997; Konstas and Lapata, 2013) to the assessment of conversational patterns of Alzheimer's patients (Abdalla et al., 2018; Paulino et al., 2018).

Most research on RST parsing is focused on parser engineering, i.e. the evaluation of parsers against a "gold standard" hand-annotated dataset. Although RST corpora exist for a variety of other languages (e.g. German, Dutch and Spanish), end-to-end discourse parsers are usually only trained and evaluated on English data, with the notable exception of Braud et al. (2017a,b,c).

There are a number of RST parsers that were developed for other languages, but either are they not publicly available (e.g. Reitter (2003) for German and English as well as Pardo and Nunes (2008) for Portuguese) or they do not produce complete RST analyses, e.g. Sumita et al. (1992) for Japanese (no intra-sentence relations) and da Cunha et al. (2012) for Spanish (no inter-sentence relations).

Compared to other NLP tasks like syntax parsing, the amount of available training data is limited, with corpora being in the range from dozens to a few hundred hand-annotated texts. There is also little work evaluating RST parsers beyond the Parseval-based procedure proposed by Marcu (2000).[2]

For example, machine learning models are usually not compared with respect to their ability to detect rare rhetorical relations. Zhang and Liu (2016) found that rhetorical relations in RST-DT at different levels—i.e. between clauses within sentences, between sentences within paragraphs and between paragraphs—all follow the same Zipf's law–related distribution. Individual relations show different patterns, e.g. *Attribution* is more common in intra-sentential relations than on higher levels.

In addition, no systematic review exists of the impact of the preprocessing steps—sentence splitting, syntax parsing and segmentation into Elementary Discourse Units (EDUs)—on the quality of the resulting RST parses. There is some work in this direction, though.

For example, Surdeanu et al. (2015) implemented two RST parsers that only differ in the syntax parser used—the constituent-based RST parser produced slightly better results, but the dependency-based equivalent was 2.5 times faster. Braud et al. (2017c) evaluated the influence of syntactic information (using either constituency parses, dependency parses or only POS tags) on discourse segmentation. Rutherford et al. (2017) reviewed the impact of different neural network architectures on implicit discourse relation detection.

While Huber and Carenini (2020) showed that RST parser performance can be improved by train-

[1] The rst-workbench and all related Docker configuration files, images and REST API wrappers around the RST parsers are available from https://github.com/arne-cl/rst-workbench. An online demo is provided at https://rst-workbench.arne.cl/.

[2] In a replication study, Morey et al. (2017) found that most recently reported increases in RST parser performance (9 parsers published between 2013 to 2017) are caused by implementation differences of Marcu's evaluation procedure.



*Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations*, pages 1–6
April 19 - 23, 2021. 

ing them on large RST treebanks automatically generated using distant supervision, I hypothesize that RST parsers can profit even more from larger human-annotated training corpora.

In turn, the annotation of RST corpora can likely be sped up by leveraging RST parsers. In the same vein that translators can produce high-quality translations by post-editing machine-translated texts more quickly than by manual translation alone (Gaspari et al., 2014; Koponen, 2016), I assume that linguists can produce RST analyses faster with machine support (i.e. by selecting the best automatic analysis from a number of RST parsers and then post-editing it) than by relying on hand-annotation alone.

If the goal is to make annotators use RST parsers productively, the parsers need to be adapted to meet their needs. While the primary focus of RST parser development is improving upon state-of-the-art benchmark results, this work focuses on usability and compatibility, i.e. the parsers need to be easy to install and run while supporting the same format(s) that common RST annotation and visualization tools use.

To achieve this, I implemented rst-workbench, which:

- acts as a web-based front-end to six different RST parsers,
- provides an easy way to install the parsers on all modern desktop operating systems using Docker containers,
- facilitates their integration into NLP pipelines by wrapping them in REST APIs,
- enables the RST analyses produced by the parsers to be visualized by and edited in the rstWeb annotation tool (Zeldes, 2016) by amending it with a REST API and by providing converters from the parsers' output formats to rstWeb's input format.

The remainder of this paper is organized as follows. Section 2 gives a brief overview of related work, while Section 3 describes the architecture and usage of the system. Section 4 summarizes the main conclusions and outlines areas of future work.

## 2 Related work

To the best of my knowledge, rst-workbench is the first tool that offers a graphical user interface for and integrates several RST parsers. Besides rstWeb (Zeldes, 2016), which is integrated in this software, there are two other annotation tools specifically made for rhetorical structures: RSTTool (O'Donnell, 2000) and TreeAnnotator (Helfrich et al., 2018). For visualizing RST trees and querying RST corpora, there is ANNIS3 (Krause and Zeldes, 2014).

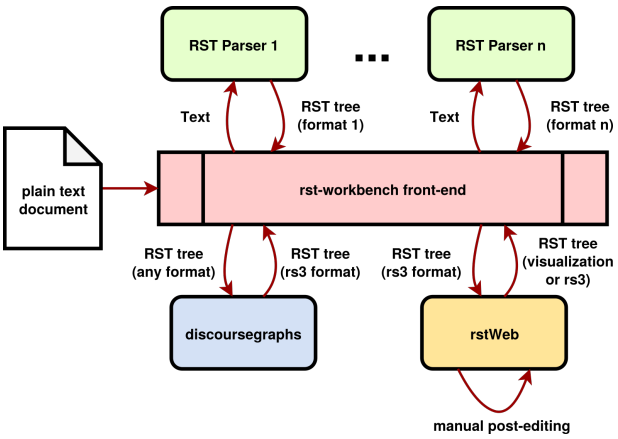


Figure 1: Components and workflow of the rst-workbench.

While I implemented very minimal REST APIs around the individual RST parsers in Python, CLAM (van Gompel and Reynaert, 2014) could be used to create REST API wrappers around command-line NLP tools by writing a configuration file. For simple cases, it is slightly more complicated to setup than my approach (cf. Section 3.2), but it comes with many additional features (e.g. user authentication, batch processing and a generic web interface for each API) and can be extended with additional format converters and visualization components.

## 3 Software architecture and usage

The rst-workbench provides a simple way to install multiple RST parsers on a computer, run them as well as visually compare and edit their analyses in a web browser. Its architecture and usage is summarized in Figure 1. Screenshots of the workflow are provided in Figures 2 and 3.

At the core, the rst-workbench consists of six existing open-source RST parsers—HILDA (Hernault et al., 2010), Feng and Hirst (2014), DPLP (Ji and Eisenstein, 2014), Heilman and Sagae (2015), CODRA (Joty et al., 2015) and StageDP (Wang et al., 2017, 2018)—packaged as Docker containers to make them easily installable without any user intervention (cf. Section 3.1).

Users do not have to learn the different

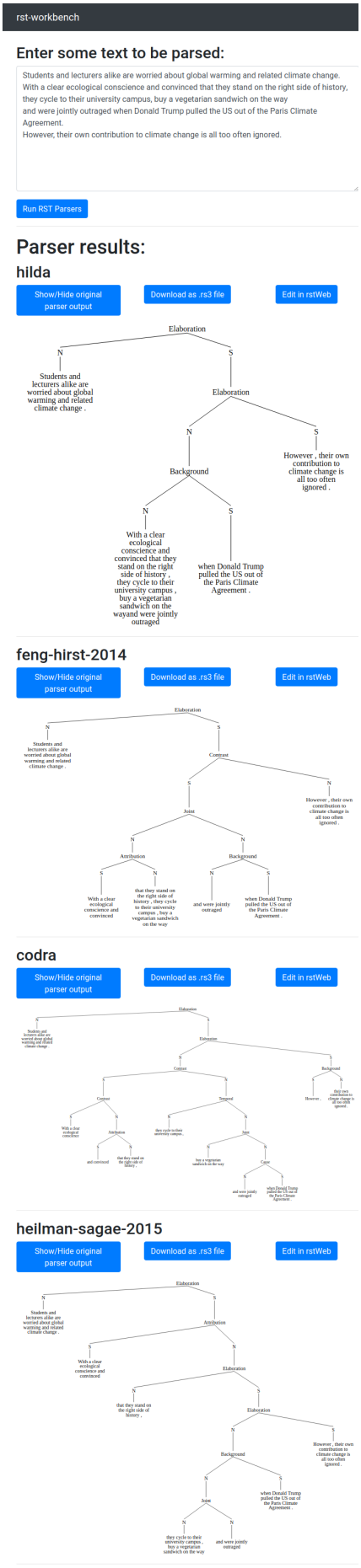


Figure 2: Screenshot of rst-workbench showing the result of parsing the beginning of a newspaper article with various RST parsers.

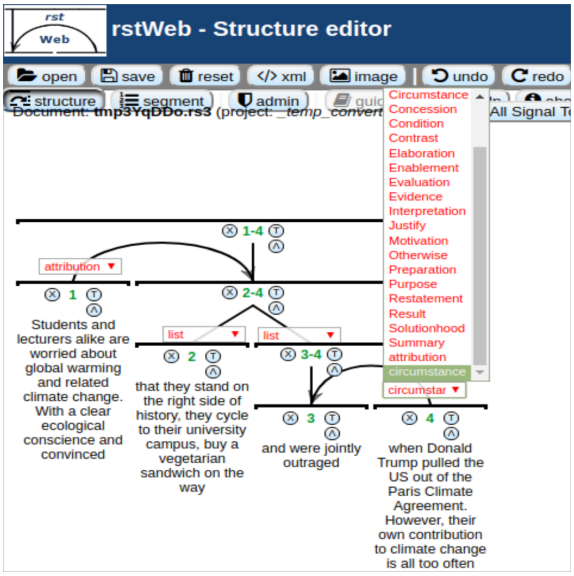


Figure 3: Post-editing a parse result in rstWeb (here: changing the relation that holds between two EDUs).

command-line interfaces of the parsers, but can simply interact with them via a web browser. To make this possible, I added a REST API to each of the parsers, which the browser can talk to (cf. Section 3.2).

In the browser, annotators can enter text or upload a plain text document. After clicking the "Run Parsers" button, all RST parsers are started concurrently to analyze the given text. The results appear asynchronously in the browser, i.e. the user sees the result of the fastest parser immediately when it is available and does not have to wait for the remaining parsers to finish processing. Users can now select the parse tree that most resembles their linguistic intuition, and click "Edit in rstWeb" to load the analysis into the rstWeb annotation tool. Here, all aspects of the RST tree can be modified, e.g. rhetorical relations between EDUs and/or larger subtrees can be replaced (Figure 3). Afterwards, the result can be saved locally for further inspection or corpus creation.

The technical setup needed to integrate all these stand-alone tools into one software package with a unified interface is described in the following subsections.

### 3.1 Docker

Docker is a tool that allows programmers to bundle a piece of software with all its dependencies into a *container*, which a user can reproducibly install on any computer with Linux, Mac OSX or Windows without having to know any details about

the software. The step-by-step installation process of a software package is described in a so-called *Dockerfile*, which is both readable by machines and humans.

Installing an RST parser from a Dockerfile will save the user the effort of finding its dependencies, installation parameters and training settings. This will not, however, reduce the run time of the installation and training process, as that will happen on the user's local machine. This process can be drastically sped up by using a Docker *image*, which is a compressed file that contains the results of running a Dockerfile. I provide Docker images for all but one of the RST parsers available in the rst-workbench.[3] If Docker is already running on the target system, the installation of an RST parser boils down to a one-line command.[4]

At this point, the parsers can be used without tedious installation procedures, but are still only available as individual command-line tools with different parameters and output formats. To improve their *usability*, I make them available as web services with a common interface (cf. Section 3.2). To improve their *comparability*, I offer a simple way to convert their output to a common format and generate visualizations of the resulting RST trees (Section 3.3).

### 3.2 Web application and REST APIs

With rst-workbench, I aim to make RST parsers more accessible to a wider audience, by providing a common (graphical) interface for them. I chose to implement this in form of a web application, which talks to the individual RST parsers via REST (Fielding, 2000), a simple text-based protocol commonly used by programs running on different computers to communicate with each other via the Internet. I implemented REST interfaces for each of the parsers using the Python hug library[5]. They receive requests containing the text to be analyzed, run the actual (command-line) RST parsers in the background on the given input, capture their outputs and send them back to the requesting program.

Using REST allows the user to run the parsers on different machines than the web application (in case one computer does not have enough RAM or processing power to run all RST parsers at the same time) and even to use the parsers as web services without the front-end, e.g. to integrate them into custom NLP pipelines. It also simplifies the process of adding more parsers to the rst-workbench, as it only needs to know where the parsers run and which output format they use.

### 3.3 discoursegraphs and rstWeb

The interoperability of RST tools is hindered by the lack of a standard format for encoding RST analyses. While corpora are either using the LISP-like `dis` or the XML-based `rs3` format, RST parsers are using a plethora of custom formats. rst-workbench is able to convert many of them into rs3—the format supported by RST annotation tools like RSTTool (O'Donnell, 2000) and rstWeb (Zeldes, 2016)— by utilizing a REST service I implemented on top of the discoursegraphs converter library (Neumann, 2015, 2016), which supports all RST file formats of the given parsers.

Using the rs3 format and integrating rstWeb into the rst-workbench allows it to leverage rstWeb's capabilities to visualize and (post)-edit RST trees.[6]

## 4 Conclusions

In this paper, I presented a software package that simplifies the installation, usage and visual comparison of RST parsers. I showed how it can help linguists to produce manual RST analyses with less effort.

I plan to integrate the rst-workbench directly into rstWeb to facilitate corpus annotation projects. In rstWeb, an "administrator" can create an annotation project, upload documents to be annotated and assign them to annotators. With an integrated rst-workbench, the administrator could precompute and store the automatic RST analyses once for all annotators, which would reduce wait time for the annotators and allow them to work without switching browser tabs and tools.

In the current setup, analyzing a text with all parsers may take up to two minutes. Most of this

[3] All Docker images for the rst-workbench are available at `https://hub.docker.com/u/nlpbox`. I can't provide an image for the HILDA RST parser (Hernault et al., 2010), as its license does not allow its source code to be freely distributed. Nevertheless, if you have access to the HILDA source code, you can simply build an image using the Dockerfile provided at `https://github.com/NLPbox/hilda-docker`.

[4] For example `docker run nlpbox/heilman-sagae-2015` for the (Heilman and Sagae, 2015) parser.

[5] `http://www.hug.rest/`

[6] To achieve this, I added a REST interface to rstWeb. For lack of time, it is not yet part of the official rstWeb source code, but is available here: `https://github.com/arne-cl/rstWeb/tree/add-rest-api`

time is spent on loading the models of the underlying syntax parsers. This can be drastically reduced by "outsourcing" the syntax parsers into their own web services, as I have already done for DPLP.[7]

## References


Mohamed Abdalla, Frank Rudzicz, and Graeme Hirst. 2018. Rhetorical structure and Alzheimers disease. *Aphasiology*, 32(1).

Chloé Braud, Maximin Coavoux, and Anders Søgaard. 2017a. Cross-lingual RST Discourse Parsing. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 292–304, Valencia, Spain. Association for Computational Linguistics.

Chloé Braud, Ophélie Lacroix, and Anders Søgaard. 2017b. Cross-lingual and cross-domain discourse segmentation of entire documents. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 237–243, Vancouver, Canada. Association for Computational Linguistics.

Chloé Braud, Ophélie Lacroix, and Anders Søgaard. 2017c. Does syntax help discourse segmentation? Not so much. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2432–2442, Copenhagen, Denmark. Association for Computational Linguistics.

Iria da Cunha, Eric SanJuan, Juan-Manuel Torres-Moreno, M. Teresa Cabré, and Gerardo Sierra. 2012. A Symbolic Approach for Automatic Detection of Nuclearity and Rhetorical Relations among Intra-sentence Discourse Segments in Spanish. In *Proceedings of the 13th International Conference in Computational Linguistics and Intelligent Text Processing (CICLing 2012)*.

Vanessa Wei Feng and Graeme Hirst. 2014. A Linear-Time Bottom-Up Discourse Parser with Constraints and Post-Editing. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 511–521, Baltimore, Maryland. Association for Computational Linguistics.

Roy Thomas Fielding. 2000. *Architectural Styles and the Design of Network-based Software Architectures*. Ph.D. thesis, University of California, Irvine.

Federico Gaspari, Antonio Toral, Sudip Kumar Naskar, Declan Groves, and Andy Way. 2014. Perception vs Reality: Measuring Machine Translation Post-Editing Productivity. In *Proc. Third Workshop on Post-Editing Technology and Practice*.


[7] See the Dockerfile in https://github.com/NLPbox/dplp-docker.


Maarten van Gompel and Martin Reynaert. 2014. CLAM: Quickly deploy NLP command-line tools on the web. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: System Demonstrations*, pages 71–75, Dublin, Ireland. Dublin City University and Association for Computational Linguistics.

Michael Heilman and Kenji Sagae. 2015. Fast Rhetorical Structure Theory Discourse Parsing. *arXiv preprint arXiv:1505.02425*.

Philipp Helfrich, Elias Rieb, Giuseppe Abrami, Andy Lücking, and Alexander Mehler. 2018. TreeAnnotator: Versatile Visual Annotation of Hierarchical Text Relations. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC-2018)*, Miyazaki, Japan. European Languages Resources Association (ELRA).

Hugo Hernault, Helmut Prendinger, David A. duVerle, and Mitsuru Ishizuka. 2010. HILDA: A Discourse Parser Using Support Vector Machine Classification. *Dialogue & Discourse*, 1(3).

Patrick Huber and Giuseppe Carenini. 2020. MEGA RST Discourse Treebanks with Structure and Nuclearity from Scalable Distant Sentiment Supervision. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7442–7457, Online. Association for Computational Linguistics.

Yangfeng Ji and Jacob Eisenstein. 2014. Representation Learning for Text-level Discourse Parsing. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13–24, Baltimore, Maryland. Association for Computational Linguistics.

Shafiq Joty, Giuseppe Carenini, and Raymond T. Ng. 2015. CODRA: A Novel Discriminative Framework for Rhetorical Analysis. *Computational Linguistics*, 41(3):385–435.

Ioannis Konstas and Mirella Lapata. 2013. Inducing Document Plans for Concept-to-Text Generation. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1503–1514, Seattle, Washington, USA. Association for Computational Linguistics.

Maarit Koponen. 2016. Is machine translation post-editing worth the effort? A survey of research into post-editing and effort. *The Journal of Specialised Translation*, 25:131–148.

Thomas Krause and Amir Zeldes. 2014. ANNIS3: A new architecture for generic corpus query and visualization. *Literary and Linguistic Computing*.

Daniel Marcu. 1997. *The Rhetorical Parsing, Summarization, and Generation of Natural Language Text*. Ph.D. thesis, Department of Computer Science. University of Toronto.

Daniel Marcu. 2000. The rhetorical parsing of unrestricted texts: a surface-based approach. *Computational Linguistics*, 26(3):395–448.

Mathieu Morey, Philippe Muller, and Nicholas Asher. 2017. How much progress have we made on RST discourse parsing? A replication study of recent results on the RST-DT. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1319–1324, Copenhagen, Denmark. Association for Computational Linguistics.

Arne Neumann. 2015. discoursegraphs: A graph-based merging tool and converter for multilayer annotated corpora. In *Proceedings of the 20th Nordic Conference of Computational Linguistics (NODALIDA 2015)*, pages 309–312, Vilnius, Lithuania. Linköping University Electronic Press, Sweden.

Arne Neumann. 2016. Merging and validating heterogenous, multi-layered corpora with discoursegraphs. *Journal for Language Technology and Computational Linguistics*, 31(1):101–115.

Michael O'Donnell. 2000. RSTTool 2.4 - A markup Tool for Rhetorical Structure Theory. In *INLG'2000 Proceedings of the First International Conference on Natural Language Generation*, pages 253–256, Mitzpe Ramon, Israel. Association for Computational Linguistics.

Thiago Alexandre Salgueiro Pardo and Maria das Graças Volpe Nunes. 2008. On the Development and Evaluation of a Brazilian Portuguese Discourse Parser. *RITA*, 15(2):43–64.

Anayeli Paulino, Gerardo Sierra, Laura Hernández-Domínguez, Iria da Cunha, and Gemma Bel-Enguix. 2018. Rhetorical relations in the speech of Alzheimers patients and healthy elderly subjects: An approach from the RST. *Computación y Sistemas*, 22(3).

David Reitter. 2003. Simple Signals for Complex Rhetorics: On Rhetorical Analysis with Rich-Feature Support Vector Models. *LDV Forum*, 18(1).

Attapol Rutherford, Vera Demberg, and Nianwen Xue. 2017. A Systematic Study of Neural Discourse Models for Implicit Discourse Relation. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 281–291, Valencia, Spain. Association for Computational Linguistics.

K. Sumita, K. Ono, T. Chino, T. Ukita, and S. Amano. 1992. A Discourse Structure Analyzer for Japanese Text. In *Proceedings International Conference on Fifth Generation Computer Systems*, pages 1133–1140.

Mihai Surdeanu, Tom Hicks, and Marco Antonio Valenzuela-Escárcega. 2015. Two Practical Rhetorical Structure Theory Parsers. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations*, pages 1–5, Denver, Colorado. Association for Computational Linguistics.

Yizhong Wang, Sujian Li, and Houfeng Wang. 2017. A Two-Stage Parsing Method for Text-Level Discourse Analysis. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 184–188, Vancouver, Canada. Association for Computational Linguistics.

Yizhong Wang, Sujian Li, and Jingfeng Yang. 2018. Toward Fast and Accurate Neural Discourse Segmentation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 962–967, Brussels, Belgium. Association for Computational Linguistics.

Amir Zeldes. 2016. rstWeb - A Browser-based Annotation Interface for Rhetorical Structure Theory and Discourse Relations. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations*, pages 1–5, San Diego, California. Association for Computational Linguistics.

Hongxin Zhang and Haitao Liu. 2016. Quantitative Aspects of RST Rhetorical Relations across Individual Levels. *Glottometrics*, 33:8–24.