Interactive Visualization for Linguistic Structure

Aaron Sarnat, Vidur Joshi, Cristian Petrescu-Prahova
Alvaro Herrasti, Brandon Stilson, and **Mark Hopkins**
Allen Institute for Artificial Intelligence
Seattle, WA

Abstract

We provide a visualization library and web interface for interactively exploring a parse tree or a forest of parses. The library is not tied to any particular linguistic representation, but provides a general-purpose API for the interactive exploration of hierarchical linguistic structure. To facilitate rapid understanding of a complex structure, the API offers several important features, including expand/collapse functionality, positional and color cues, explicit visual support for sequential structure, and dynamic highlighting to convey node-to-text correspondence.

1 Introduction

Interpreting visual representations of linguistic structure can be challenging and time-consuming. Consider the examples provided in Figure 1, which visualize syntactic parses of the sentence "Although some people have provided negative reviews, her restaurants have reliably great music, good food, and excellent service, and they deliver!" These representations can result in cognitive overload, due to several concrete issues:

- **The visualizations are static.** If one is using the visualization for debugging purposes, then typically one cares about only a part of the linguistic structure, not every last detail. Unfortunately, a static visualization must include all information that could possibly be relevant.

- **The visualizations are large.** The screen real estate in Figure 1 is dominated by arrows that must be carefully tracked by the eye, in order to understand the relationships between nodes. Visualizations of long sentences often run off the side of the screen, requiring the user to scroll to discover the endpoints of these arrows.

- **All node relationships look identical.** Predicate-argument relationships (e.g. "restaurants" as the subject of "have") get the same visual treatment as modifier relationships (e.g. "good" as a modifier of "food") and sequence elements (e.g. "good food" as an element of the sequence "great music, good food, and excellent service").

In this work, we provide a library and web interface for the interactive visualization of linguistic structure, seeking to minimize the cognitive load required to understand syntactic and semantic parses. Figure 2 shows a screenshot of our web UI for the same sentence as Figure 1. It reflects several of our visualization strategies (described in more detail in Section 3):

- Instead of a static visualization, we provide an interactive visualization with an expand/collapse functionality that allows the user to focus on what is relevant to her. For instance, the subclause "although some people have provided negative reviews" is collapsed in Figure 2, but can be expanded by clicking on it.

- To reduce the size of the visualization, we use a box layout, and use badging to eliminate simple leaf nodes (e.g. instead of creating a separate node for the article "her", it is represented by a badge on the node "restaurant").

- We use positional cues to highlight predicate-argument relationships (subjects appear to the left, objects to the right, modifiers attach

Proceedings of the 2017 EMNLP System Demonstrations, pages 49–54
Copenhagen, Denmark, September 7–11, 2017.

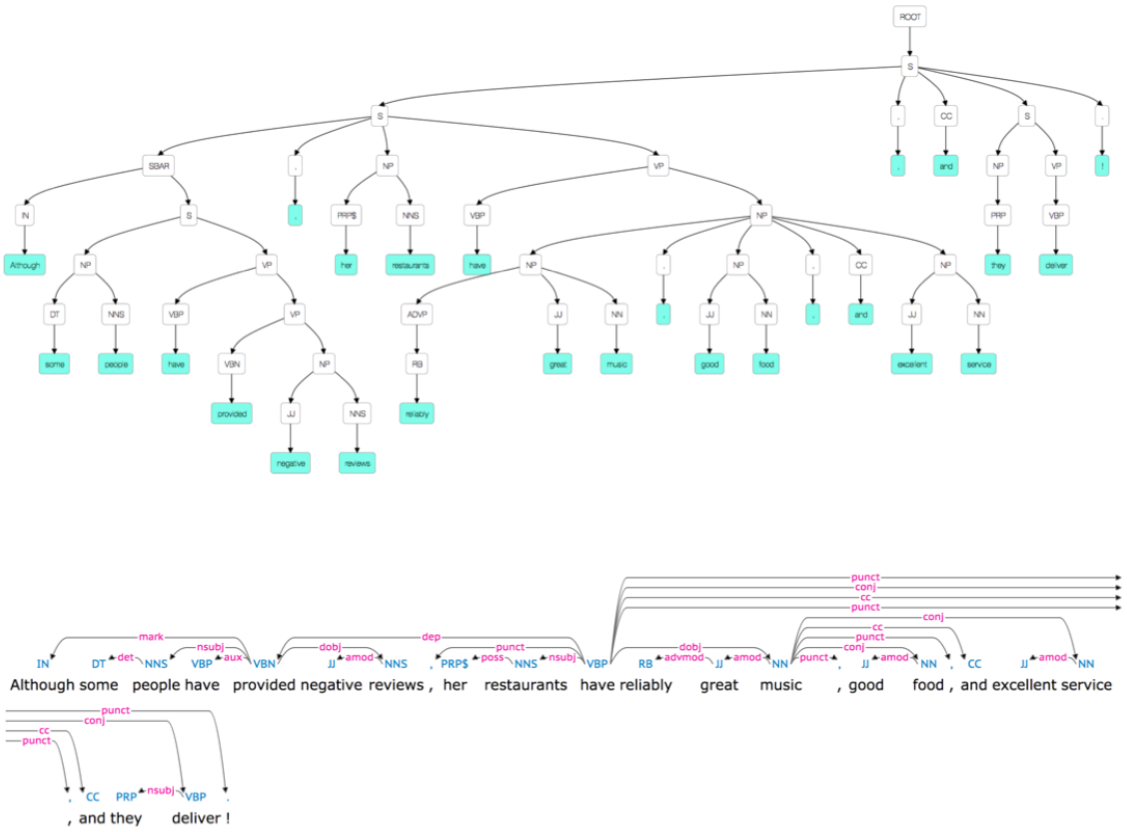

Figure 1: Typical visualizations of a constituency parse (top) and a dependency parse (bottom) for the sentence "Although some people have provided negative reviews, her restaurants have reliably great music, good food, and excellent service, and they deliver!"

to the bottom) and we provide specialized visualization for sequential structures (this occurs at two different levels in Figure 2, both for the clause sequence as well as the entity sequence "reliably great music, good food, and excellent service").

Our library is not tied to any particular linguistic formalism. In Section 4, we describe the flexible and configurable API. While our primary use case has been for creating custom grammars for mathematical language (Hopkins et al., 2017), we also provide a demonstration of how our API can be used to visualize output from the Stanford Parser (Socher et al., 2013).

2 Related Work

Most parse visualizations use the source sentence as an anchor, leaving it readable left-to-right. For constituency parses (where the words of the sentence are the leaves of the tree), this gives a representation like Figure 1 (top). For dependency parses (where the words of the sentences are the nodes of the tree), this gives a representation like Figure 1 (bottom).

Figure 1 was generated from (Podgursky, 2015), which provides open-source code and a web interface for a static rendering of a constituency parse. There are a number of libraries (Stenetorp et al., 2012; Montani, 2016; Athar, 2010; Yimam et al., 2013) that provide static renderings of dependency parses similar to Figure 1 (bottom). Among these, Brat (Stenetorp et al., 2012) provides some interactive elements (like mouseover highlighting of subtrees, and the ability to add dependencies via drag-and-drop). Displacy (Montani, 2016) provides a general API for the static rendering of various dependency parsing schemes, e.g. Universal Dependencies (Nivre et al., 2016) and Stanford Dependencies (De Marneffe and Manning, 2008).

We are not aware of any recent tools for exploring parse forests. Historically, the REDWOODS annotation environment (Oepen et al., 2004) produced a static parse forest from a hand-built grammar, and then allowed users to select the best

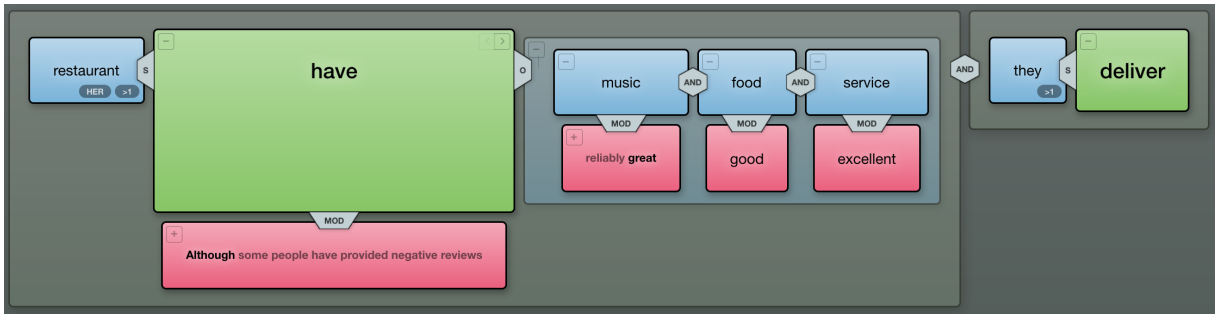

Figure 2: A screenshot of our parse visualization tool for the sentence "Although some people have provided negative reviews, her restaurants have reliably great music, good food, and excellent service, and they deliver!"

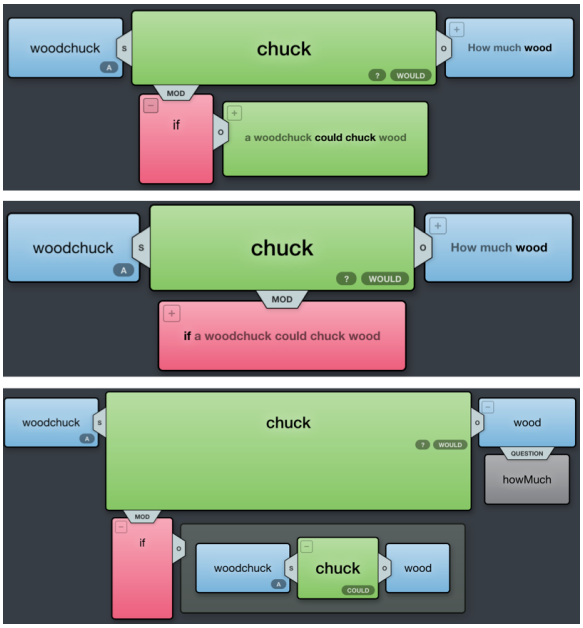

Figure 3: Three screenshots of our visualizer for the question "How much wood would a woodchuck chuck, if a woodchuck could chuck wood?": partially expanded (top), fully collapsed (middle), and fully expanded (bottom).

parse from this forest by repeatedly specifying constraints (called *discriminants* in the paper).

3 Key Features

In this section, we describe our key features for facilitating a rapid understanding of linguistic structure. As a companion to this section, we invite readers to use our web demonstration at `http://hierplane.allenai.org/explain` to interactively explore examples.[1]

[1]The user may experience some slowness when parsing certain sentences. This is due to the speed of the back-end parser, not the visualization library.

3.1 Expand/Collapse Functionality

Rather than a static rendering, our dynamic expand/collapse functionality allows the user to focus on relevant parts of the linguistic structure, while the rest is conveniently summarized at a coarser granularity. In Figure 3, the top screenshot shows a partially expanded structure that delves into the internal structure of the if-clause. The middle and bottom screenshots respectively show the fully collapsed and expanded visualizations.

3.2 Positional Cues to Distinguish Node Relations

Linguistic relationships can often be organized into intuitive clusters. For instance, subjects and objects can be roughly viewed as required arguments of a verb (exactly one per verb), while modifiers are optional (a verb can take any number of them, including zero). Our library allows the visual expression of these "relationship families" through positional cues. In Figure 3, the subject is attached to the left of its verb, the object is attached to the right of its verb, and all modifiers are attached beneath. For consistency, the object of the preposition "if" is also attached to its right. At a glance, this representation allows the user to read a gloss of the main clause by simply skimming the top line of the visualization (i.e. "a woodchuck would chuck how much wood?").

3.3 Color Cues to Distinguish Node Types

Our library provides support for using color to distinguish node types. In Figure 3, green represents events, blue represents entities, red represents modifiers, and gray is a catch-all for anything else.

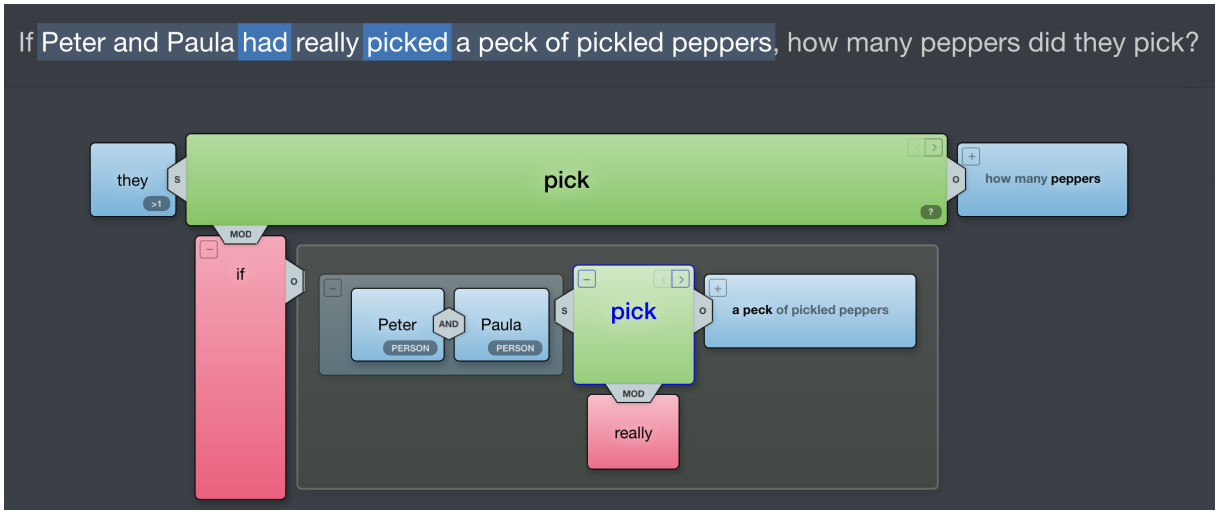

Figure 4: Dynamic highlighting of the correspondence between the linguistic structure and the source text: "If Peter and Paula had really picked a peck of pickled peppers, how many peppers did they pick?"

3.4 Badging

Some leaf nodes of linguistic structures convey very simple information about their parent node. We allow these nodes to be represented as badges rather than separate nodes. In Figure 3, the determiners ("a" for "a woodchuck") and modals ("would" for "would chuck") are represented as badges.

3.5 Sequence Support

Sequential structure is a fundamental linguistic element that is often difficult to access in a parse visualization. We make sequences visually explicit as a container of linked nodes. Examples include "Peter and Paula" in Figure 4 or the sequence of top-level clauses "her restaurants have...excellent service" and "they deliver" in Figure 2. Contrast this to the undistinguished representations of sequential structure found in Figure 1.

3.6 Dynamic Node-to-Text Highlighting

It can be difficult (particularly in semantic parses) to ascertain how the source text and the linguistic representation correspond. Upon mouse-over of a node, our visualizer highlights the part of the source text corresponding to the subtree rooted at that node. Parts that do not also correspond to some descendant node in the subtree are strongly highlighted. In Figure 4, upon mouse-over of the lower "pick" node, the corresponding segment "Peter and Paula had really picked a peck of pickled peppers" is weakly highlighted. The tokens "had" and "pecked" are strongly highlighted, because they do not correspond to any of the node's descendants.

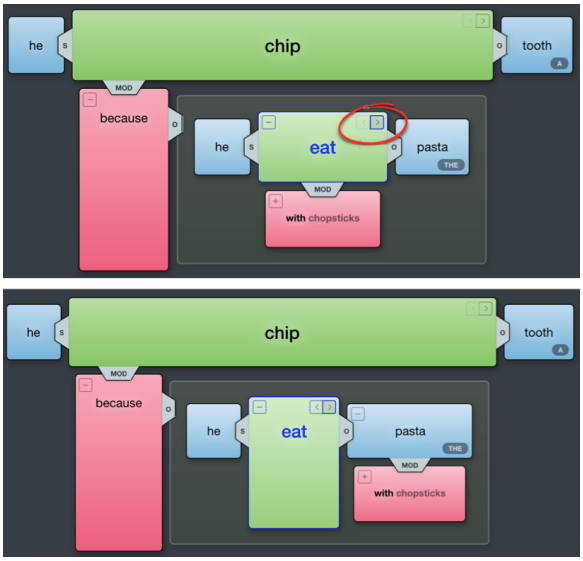

Figure 5: Before-and-after screenshots of forest exploration for the sentence "Because he ate the pasta with chopsticks, he chipped a tooth."

3.7 Modular Forest Exploration

Our interactive setting permits a convenient exploration of parse forests. For instance, see Figure 5. The subtree rooted at "eat" has two interpretations, attaching "with chopsticks" to the verb or to the object. An indicator in the node's upper-right allows the user to browse these interpretations. We localize ambiguities to the lowest possible node to allow the user to explore the forest in a convenient, modular fashion (rather than cycling through an

exponential number of parses at the root).

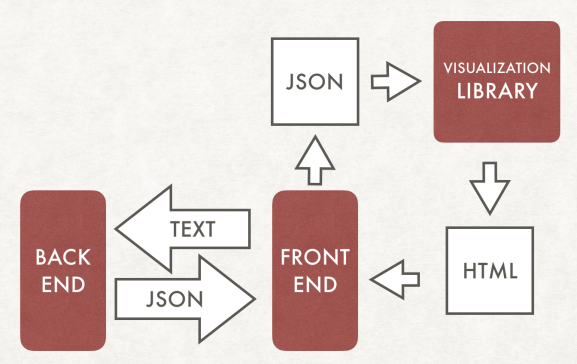

Figure 6: System architecture for our web interface. Similar architectures are employed by Brat and WebAnno.

4 APIs

Our web interface, hosted at http://hierplane.allenai.org/explain, has the architecture shown in Figure 6. The front-end accepts text input from the user. It forwards this text to the back-end and receives a JSON that contains an annotated tree and styling instructions. Figure 7 shows a simplified version of the JSON that renders the "He ate pasta with chopsticks" subtree in Figure 5. This JSON is passed to the visualization library, which renders it in HTML and returns this HTML to the front-end.

4.1 Basic Features

A key feature of our architecture is that the back-end is in charge of defining the node and edge types, and specifying how these should be displayed. This allows the visualization library to be independent of any particular linguistic annotation scheme. To demonstrate this flexibility, our demo provides two alternative back-ends. By default, it uses a custom grammar that we have been developing for parsing mathematical language (e.g.

http://hierplane.allenai.org/explain/He%20ate%20pasta%20with%20chopsticks

utilizes the default back-end). However, we have also wrapped the Stanford Parser (Socher et al., 2013) as an alternative back-end. For any given sentence, one can try out this alternative by replacing the URL prefix


```
{
  "text": "He ate pasta with chopsticks.",
  "kindToStyle": [{
    "key": "event",
    "value": ["color1"]
  }, {
    "key": "entity",
    "value": ["color2"]
  }, {
    "key": "detail",
    "value": ["color3"]
  }],
  "linkToPosition": [{
    "key": "subj",
    "value": "left"
  }, {
    "key": "obj",
    "value": "right"
  }],
  "root": {
    "kind": "event",
    "word": "eat",
    "children": [{
      "kind": "entity",
      "word": "he",
      "link": "subj"
    }, {
      "kind": "entity",
      "word": "pasta",
      "link": "obj"
    }, {
      "kind": "detail",
      "word": "with",
      "link": "adv",
      "children": [{
        "kind": "entity",
        "word": "chopstick",
        "attributes": [">1"],
        "link": "parg"
      }]
    }]
  }
}
```


Figure 7: A simplified version of the JSON that describes the parse tree for "He ate pasta with chopsticks."

http://hierplane.allenai.org/explain/

with

http://hierplane.allenai.org/explain/stanford/

For instance:

http://hierplane.allenai.org/explain/stanford/

`He%20ate%20pasta%20with%20chopsticks`

will parse "He ate pasta with chopsticks" using the Stanford parser.

In the JSON returned by the back-end, each node in the tree is annotated with its kind and label ("word"), the type of edge ("link") connecting it to its parent, and its badges ("attributes"). The linkToPosition map allows each node to be positioned according to its relation to its parent (e.g. subjects are positioned to the left of their parents, according to the example JSON). The kindToStyle map specifies colors for the various node types.

4.2 Advanced Features

To enable interactive node-to-text highlighting, nodes in the input JSON tree can each be annotated with a span field that contains the indexes of the start and end characters of the substring corresponding to the node. To enable modular forest navigation, each node with a next or previous subtree can be annotated with codes identifying these subtrees. Navigation arrows are only rendered when they lead to another subtree. When a user clicks on one, the front-end sends the code identifying the desired subtree to the back-end, and expects the requested subtree as a response.

5 Conclusion

In this work, we have tried to rethink the visualization of hierarchical linguistic structure from the ground up, first identifying the problems that cause cognitive load (large, static visualizations with no cues to distinguish node or edge families), and then designing new tactics to counter these problems (e.g. expand/collapse functionality, positional cues to distinguish node relations, explicit sequence visualization, and dynamic node-to-text highlighting). We have also created the first tool to explore parse forests using modern web design. We plan to make our visualizer freely available as an open-source library and a web interface.

References

Awais Athar. 2010. Dependensee: A dependency parse visualization tool. http://chaoticity.com/dependensee-a-dependency-parse-visualisation-tool. Accessed: 2017-04-27.

Marie-Catherine De Marneffe and Christopher D Manning. 2008. Stanford typed dependencies manual. Technical report, Technical report, Stanford University.

Mark Hopkins, Cristian Petrescu-Prahova, Roie Levin, Ronan Le Bras, Alvaro Herrasti, and Vidur Joshi. 2017. Beyond sentential semantic parsing: Tackling the math sat with a cascade of tree transducers. In *EMNLP*.

Ines Montani. 2016. Displacy dependency visualizer. https://demos.explosion.ai/displacy. Accessed: 2017-04-27.

Joakim Nivre, Marie-Catherine de Marneffe, Filip Ginter, Yoav Goldberg, Jan Hajic, Christopher D Manning, Ryan McDonald, Slav Petrov, Sampo Pyysalo, Natalia Silveira, et al. 2016. Universal dependencies v1: A multilingual treebank collection. In *Proceedings of the 10th International Conference on Language Resources and Evaluation (LREC 2016)*, pages 1659–1666.

Stephan Oepen, Dan Flickinger, Kristina Toutanova, and Christopher D Manning. 2004. Lingo redwoods. *Research on Language and Computation*, 2(4):575–596.

Ben Podgursky. 2015. Nlpviz. http://nlpviz.bpodgursky.com. Accessed: 2017-04-27.

Richard Socher, John Bauer, Christopher D Manning, and Andrew Y Ng. 2013. Parsing with compositional vector grammars. In *ACL (1)*, pages 455–465.

Pontus Stenetorp, Sampo Pyysalo, Goran Topic, Tomoko Ohta, Sophia Ananiadou, and Jun'ichi Tsujii. 2012. brat: a web-based tool for nlp-assisted text annotation. In *EACL*.

Seid Muhie Yimam, Iryna Gurevych, Richard Eckart de Castilho, and Chris Biemann. 2013. Webanno: A flexible, web-based and visually supported system for distributed annotations. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 1–6, Sofia, Bulgaria. Association for Computational Linguistics.