RepGraph: Visualising and Analysing Meaning Representation Graphs

Jaron Cohen[*] **Roy Cohen**[*] **Edan Toledo**[*] **Jan Buys**
Department of Computer Science
University of Cape Town
{CHNJAR003, CHNROY002, TLDEDA001}@myuct.ac.za jbuys@cs.uct.ac.za

Abstract

We present **RepGraph**, an open source visualisation and analysis tool for meaning representation graphs. Graph-based meaning representations provide rich semantic annotations, but visualising them clearly is more challenging than for fully lexicalized representations. Our application provides a seamless, unifying interface with which to visualise, manipulate and analyse semantically parsed graph data represented in a JSON-based serialisation format. **RepGraph** visualises graphs in multiple formats, with an emphasis on showing the relation between nodes and their corresponding token spans, whilst keeping the representation compact. Additionally, the web-based tool provides NLP researchers with a clear, visually intuitive way of interacting with these graphs, and includes a number of graph analysis features. The tool currently supports the DMRS, EDS, PTG, UCCA, and AMR semantic frameworks. A live demo is available at https://repgraph.vercel.app/.

1 Introduction

Broad-coverage semantic graphs provide richer representations of sentence meaning than surface-level syntax or lexicalised semantic dependencies (Oepen et al., 2019). The breadth of meaning representation approaches now includes a large number of semantic graph frameworks — each with their own respective strengths and weaknesses at encoding the meaning of natural language (Koller et al., 2019). Recently, a growing body of work has focused on parsing to or generating from graph-based meaning representations (Hershcovich et al., 2017; Buys and Blunsom, 2017; Zhang et al., 2019; Song et al., 2018). The outputs of many other syntactic and semantic analysis tasks can also be represented as graphs, where labelled nodes correspond to token spans and edges to relations between these spans (Jiang et al., 2020).

[*] These authors contributed equally to this work

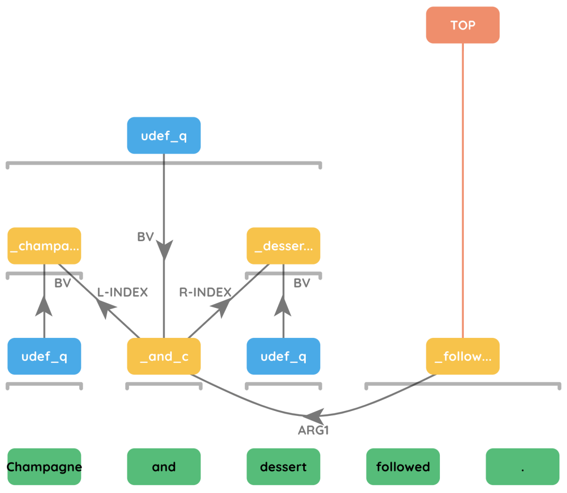


Figure 1: Hierarchical visualisation of the EDS graph for the sentence “Champagne and dessert followed.”

Visualisations of constituency trees, syntactic dependency trees and semantic dependency graphs are well established for teaching, analysing and presenting the outputs of those representations. However there is no similar established standard for visualising broad-coverage semantic graphs, due to diverging approaches to representing semantics in different frameworks, as well as the challenges involved in visualising both the graph and the correspondence to the sentence it represents clearly.

To stem the “Balkanisation” caused by framework-specific approaches, CoNLL 2019 and 2020 hosted shared tasks on Cross-Framework Meaning Representation Parsing (MRP; Oepen et al., 2019, 2020). These tasks provided a uniform abstract graph representation with a JSON-based serialization format and standardized datasets. However, current visualisation tools are either framework-specific or fail to represent graphs clearly and consistently across frameworks (§5).

RepGraph is an open source, web-based visualisation and analysis tool for meaning representation graphs, with support for multiple frameworks.

The tool provides three novel visualisation formats, each designed to better elicit specific inherent qualities of the graphs (§3). In contrast to previous approaches, two of our visualisations (*hierarchical* and *tree-like*) represent the graph and sentence tokens in an integrated manner, clearly showing the relationship between nodes and the tokens or token spans they correspond to. The third (*flat*) is akin to existing dependency graph visualisations.

The tool provides a unified, intuitive and feature-rich platform for interacting with meaning representation graphs. Analysis functionality includes: displaying subgraphs, searching graphs by node label or subgraph, comparing graphs visually, testing graph properties such as planarity, and providing dataset-level statistics (§4).

RepGraph is targeted towards everyone working with meaning representation graphs, including: researchers developing parsers or generators; computational linguists performing semantic analysis; NLP practitioners using graphs in downstream applications; and students learning about these representations. Parsing is not currently integrated, but parser output can easily be processed. RepGraph is also not intended to be an annotation tool; such functionality is orthogonal to what we provide.

Five semantic graph frameworks are currently supported: Dependency Minimal Recursion Semantics (**DMRS**; Copestake, 2009), Elementary Dependency Structures (**EDS**; Oepen and Lønning, 2006), Prague Tectogrammatical Graphs (**PTG**; Hajič et al., 2012), Universal Conceptual Cognitive Annotation (**UCCA**; Abend and Rappoport, 2013) and Abstract Meaning Representation (**AMR**; Banarescu et al., 2013). Some framework-specific normalisations are performed to improve compatibility and enable a unified approach to visualisation. While our development focused on English datasets, most of the frameworks also support other languages, and the tool can easily be extended to support additional frameworks.[1]

2 System Description

A semantic graph is a triple (T, N, E), where N is a set of nodes, $E \subseteq N \times N$ is a set of directed edges, and $T \subset N$ is a set of *top* nodes (Oepen et al., 2019).

Nodes may optionally have zero or more *properties* with associated values. The relationship between the graph nodes and the input string is referred to as *anchoring* or *alignment*. We assume that the input is tokenized; a graph node may be anchored to a token, a token span, or a set of token spans.[2] The alignment is not annotated in all frameworks; we assume that it can be obtained, using an aligner, for (most) graph nodes.

The alignment between the nodes and input tokens forms the basis of the design of our *hierarchical* and *tree-like* visualisation formats. We also distinguish between *surface* nodes, which represent the lexical items (tokens) they are aligned to directly, and *abstract* nodes, which represent the semantic contribution of grammatical constructions.

This distinction is annotated explicitly in DMRS and EDS (which are based on the same underlying annotations), but we extend it to the other frameworks based on the alignments and framework properties.

2.1 System Architecture

The system consists of a web-based front-end user-interface through which users upload a file containing a bank of MRP graphs in the Uniform Graph Interchange Format, which is serialised in JSON Lines format (Oepen et al., 2019). The file is then uploaded to and parsed by the back-end of the **RepGraph** application to create a transitory structure of the dataset. The user can then proceed to the main screen of the application (Fig. 2). The main libraries used are **React**[3], **visx**[4], and **material-UI**[5] for the front end, and **Spring-Boot**[6] for the backend. The source code can be found at `https://github.com/RepGraph/RepGraph`.

3 Graph Visualisation

Our application provides three distinct visualisations of meaning representation graphs in all the supported frameworks, providing users with multiple perspectives of the same semantic information.

As can be seen in Figures 1–3, the visualisations use colour and annotations to represent the various elements of the graphs — for example, surface and abstract nodes are differentiated and shown in different colours. An expandable legend is provided

[1]A demo video is available at `https://vimeo.com/user136369092/repgraph`

[2]The anchoring is annotated at character level in the MRP data. We tokenise the input in a manner that is consistent with the given annotations.

[3]`https://reactjs.org/`

[4]`https://github.com/airbnb/visx`

[5]`https://material-ui.com/`

[6]`https://spring.io/`

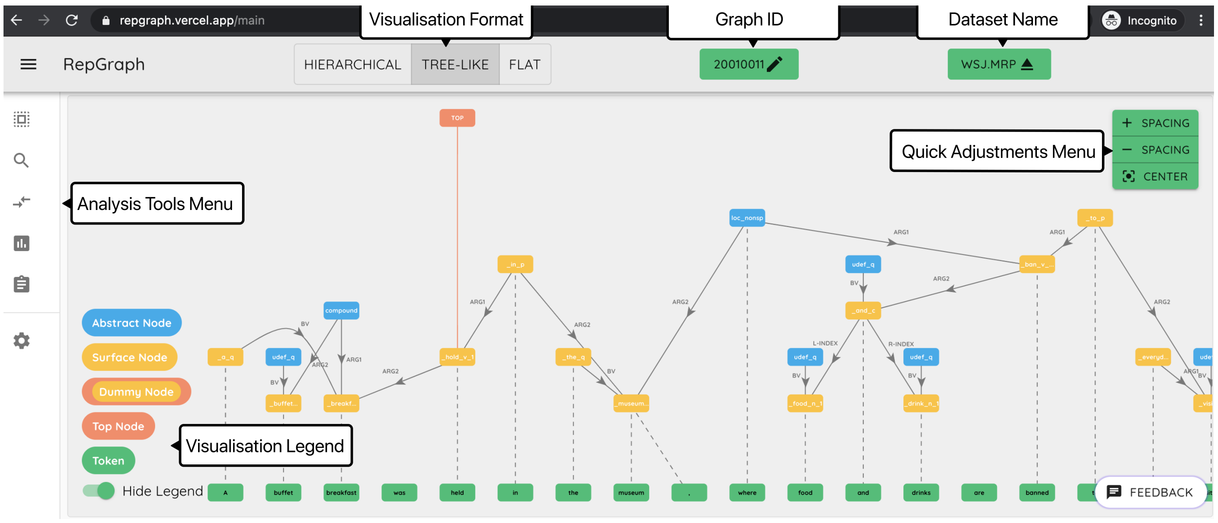


Figure 2: The RepGraph Main Page with an EDS graph visualised in the *Tree-like* format. The main elements on the page are captioned.

to explain the different colours (see Fig. 2).

The user can modify the default colours and spacing of the visualisations from the settings in the panel on the left of the main page. The placement of the edges and edge labels can also be manipulated easily by dragging them with the mouse. The examples in the figures, as well as the demo data provided in application, come from the sample annotations of Wall Street Journal (WSJ) corpus sentences provided for MRP 2020.[7]

We next discuss each of the visualisations, and in particular how node placement is calculated for each of them (§3.1 - 3.3). Then we discuss how edge placement is determined across all the visualisations to minimise collisions (§3.4), and normalisations to make the anchoring more consistent across frameworks (§3.5).

3.1 Hierarchical

The *Hierarchical* layout (Fig. 1) focuses on showing the natural hierarchy that occurs between the anchors of the semantic nodes in the graph. Nodes with larger spans envelop nodes whose spans they subsume in their range of tokens. Horizontal bars (brackets) are placed below the semantic nodes in the graph to represent the alignment with their span of tokens in the input sentence. Tokens are displayed in the bottom row. For example, in Fig. 1, the top-most node with the label `udef_q` has a bracket covering the token span “Champagne and dessert” below it.

[7]`http://svn.nlpl.eu/mrp/2020/public/sample.tgz`

3.2 Tree-like

The *Tree-like* visualisation (Fig. 2), places emphasis on each node’s number of descendants. A tree-like structure is created by having nodes with a larger number of descendants positioned above nodes with a lower number of descendants. The anchoring of nodes that are positioned at the bottom of the graph (directly above the tokens) is represented with a dotted line to the tokens(s) they span. These nodes are positioned vertically in line with the start of their token span. The alignment of other nodes is not indicated visually. The node placement is computed by running a topological sort on each node in order to get a list of its descendants, which is used to determine its level in the tree.

3.3 Flat

The flat visualisation (Fig. 3), orders nodes horizontally based on the first token alignment. Ties between nodes that are aligned to the same first token are broken by prioritising nodes with smaller span lengths. Edges are curved above and below the linear plane according to whether they are directed right or left. No tokens are shown in this format. The span(s) of each node and its corresponding text phrase(s) are present inside a tool-tip that appears when they are hovered upon. This visualisation is similar to the way that dependency graphs are traditionally visualised (albeit using graph nodes rather than tokens as basic elements).

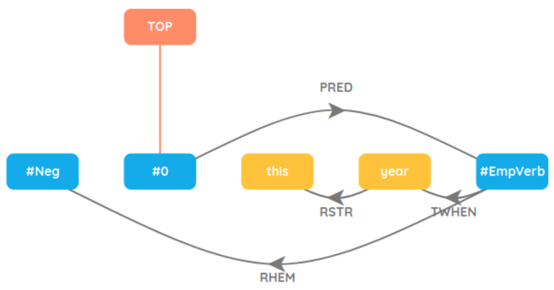


Figure 3: Example of the *flat* visualisation of the PTG graph for the sentence "Champagne and dessert followed."

3.4 Edge Handling

Edges are created as Quadratic Bézier Curves defined through 3 points (P_0, P_1 and P_2):

$$B(t) = (1-t)^2 P_0 + 2(1-t)tP_1 + t^2 P_2, 0 \leq t \leq 1$$

where P_0 is the edge's source node position and P_2 is the edge's target node position.

We designed a set of edge handling heuristics that determine the value of P_1 for each graph edge, with the aim to minimise overlap between edges, edge labels and nodes. The heuristics take into account attributes of edges and their source and target nodes that include their distance, whether they are in the same row or column, and the relative positions of other nodes. Users can also drag edges and edge labels with their mouse pointer, manipulating the value of P_1.

3.5 Framework Normalisation

In order to visualise and analyse graphs across all frameworks in a standardised way, we performed framework-specific normalisations that capture latent information whilst ensuring accuracy of the graph visualisations.

We extract **DMRS** annotations from the Redwoods treebank[8] using PyDelphin.[9] Tokenisation and some additional normalisations were derived from the Redwoods syntactic layer.

For the other frameworks we use the data provided by the MRP 2020 shared task (Oepen et al., 2020). Input sentences were tokenised with Stanford CoreNLP (Manning et al., 2014).[10]

[8]http://svn.delph-in.net/erg/tags/1214/

[9]https://github.com/delph-in/pydelphin

[10]https://stanfordnlp.github.io/CoreNLP/

In **EDS**, nodes with the CARG property (named entities) are treated as surface rather than abstract nodes, and displayed as such in the visualisation.

PTG graphs may contain nodes without token alignments. We align those nodes to their leftmost aligned children, or if that is not possible, their leftmost aligned parents. These induced alignments are use only to determine the node layout, and are not displayed visually.

UCCA nodes do not have explicit labels. We treat nodes with spans as surface nodes; their labels are implicitly derived from their spans, using the corresponding token strings as labels. Abstract nodes (nodes without annotated spans) implicitly derive their spans as the union of the spans of their descendent nodes. We do not assign labels to them.

AMR annotations do not include node spans. As our *hierarchical* and *tree-like* visualisations require node spans, an AMR aligner is used to obtain alignments. The rule-based JAMR Aligner[11] (Flanigan et al., 2014) is integrated into our tool to process only AMR graphs that are uploaded without alignments - this does not impact the language support of the other frameworks. All aligned nodes are designated as surface nodes. Nodes that are left unaligned by the aligner are aligned in the same way as PTG to determine the layout.

Multiple Spans Graphs in the UCCA and PTG frameworks have nodes that are anchored to multiple, potentially discontiguous token spans. These nodes are handled as follows: If the multiple spans are contiguous, node are aligned to the union of their spans. Otherwise, *dummy* nodes are created for each additional non-adjacent span; the regular node corresponds to the left-most span. Each dummy node is placed above its aligned token span and distinguished visually. It takes the label of its original node, with additional information to identify which span it refers to and from which node it was created. When the user hovers over the original node, the dummy nodes are highlighted.

4 Graph Analysis Features

4.1 Subgraph Display

The subgraph display tool allows users to examine specific subgraphs of the currently visualised graph. Users can choose between two types of subgraphs: *adjacent* and *descendent*. Upon selection of a subgraph type, the user selects a node on the graph

[11]https://github.com/jflanigan/jamr

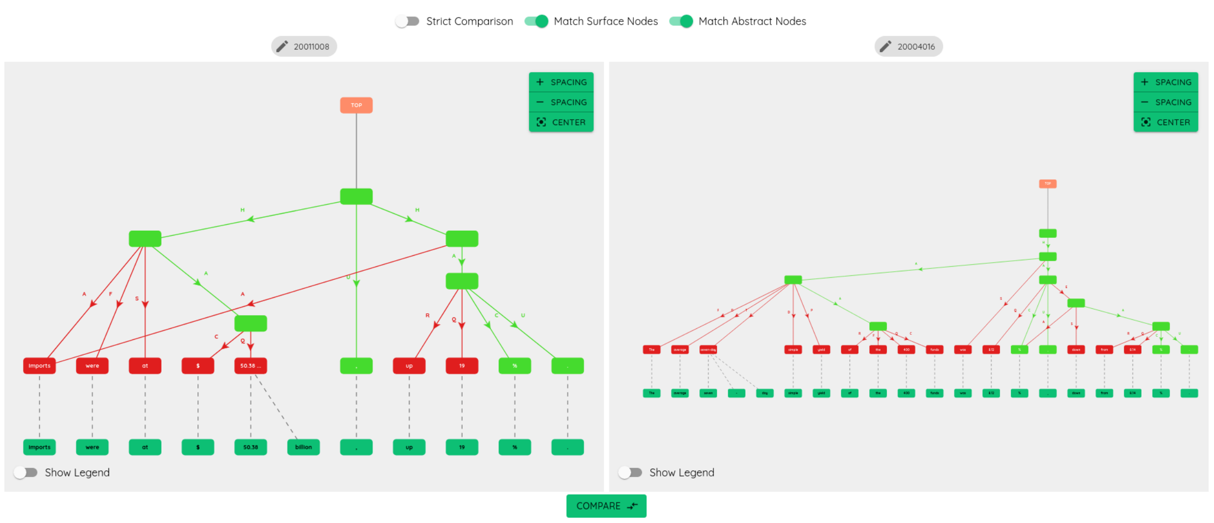


Figure 4: The Graph Comparison Tool showing which nodes and edges in the two UCCA graphs are similar (green) or dissimilar (red).

visualisation and clicks the display button to view the resulting subgraph.

Adjacent subgraphs are created from the nodes directly adjacent to the selected node. This allows users to focus in on the immediate neighbourhood of a node.

Descendent subgraphs are created from the selected node's descendants in the graph. This allows users to focus on a single node and the propagation of its descendants through the graph.

4.2 Subgraph Search

We provide two ways for users to find graphs matching specific patterns in large datasets:

Search by Node Label Set The user can enter one or more node labels as a query to search for all graphs in the dataset which contain all of these node labels. This gives the user the ability to quickly find sentences and graphs containing nodes of interest and see how they are used in different graphs.

Search by Subgraph Pattern The user can also visually select a connected subgraph of the current graph by clicking to select the desired nodes and edges. After selection, the search will return all other graphs that contain this subgraph, and the user can visualise any of these graphs.

4.3 Graph Comparison

The graph comparison interface, seen in Fig. 4, is broken up into two vertically-separated side-by-side panels that enable direct comparison between two graphs. The user selects a graph from the dataset for each side, toggles the comparison settings, and clicks `compare`. The similarities and differences between the graphs are displayed visually through colouring the nodes and edges. If the same node label appears in both graphs, all nodes with that label are deemed as similar, even when the number of nodes with that label differ between the two graphs.[12] The tool also provides the option to only compare surface or abstract nodes. Two modes of graph comparison are provided:

Standard Comparison Nodes are compared using only their labels, whereas edges are matched if both the edge labels and the labels of the nodes they are incident to are equal.

Strict Comparison This mode additionally requires that the concatenation of the token spans corresponding to each node has to be the same (as strings) in order for the nodes to match.

4.4 Graph Properties

A number of graph properties can be evaluated on the current graph.

Planarity An important property of a semantic graph is whether it is planar, also referred to as non-crossing (Kuhlmann and Oepen, 2016). To determine planarity, the nodes are ordered in a similar manner as for the *flat* visualisation (§3.3), except that nodes with the exact same span are placed below one another in the same horizontal position, and have their edges transferred to the node representing their position in the linear ordering. All

[12]This contrasts to Smatch (Cai and Knight, 2013) which finds a 1-to-1 alignment between the nodes to measure graph overlap, while for visualisation purposes we rather show all possible correspondences.

edges are represented as arcs above the nodes.

Nodes with no token alignment are excluded from the construction as it is assumed they can be placed anywhere to avoid causing conflicts. Edges between nodes in the same linear ordering position are also excluded as they have no impact on potential crossing edges.

The check for planarity, once the linear ordering has been established, follows the definition of planarity outlined in (Gómez-Rodríguez and Nivre, 2010). After running the planarity test, the modified graph can be visualised. Crossing edges are highlighted in red.

Graph Connectivity A graph is connected if an undirected path exists between every pair of nodes (Kuhlmann and Oepen, 2016). AMR graphs are always connected, but this is not guaranteed for the other frameworks. Graph connectivity is determined using a variation of Breadth First Search.

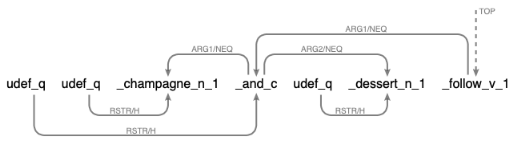
Longest Directed and Undirected Path(s) Our tool also has functionality to find the longest directed or undirected paths in any graph. The longest path is highlighted on the graph. If there are multiple longest paths, the user can cycle through them. The paths are found using derivations of Topological Sort and Breadth First Search. Before the longest path is derived, the graph is checked for cycles using a derivation of Depth First Search.

4.5 Dataset Statistics

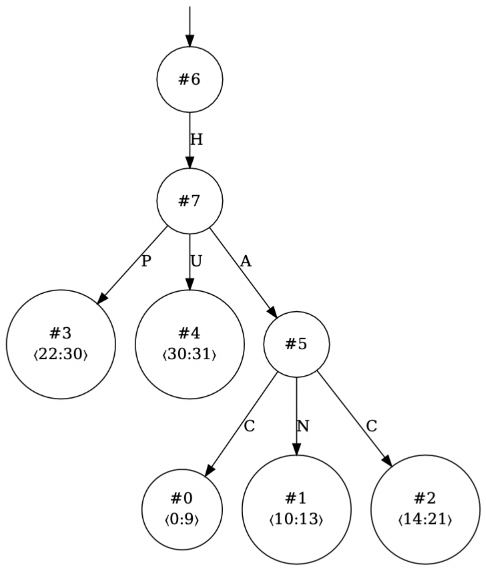
The dataset statistics tool can produce a number of global statistics on the current dataset, which may be useful for comparing datasets. The statistics included are a subset of the ones used by Kuhlmann and Oepen (2016) to compare semantic graph frameworks, and include average number of nodes, average span of node, and percentage of cyclic graphs, amongst others.

5 Related Work

The plethora of tools and visualisation options currently available are fragmented and often built without semantic graphs in mind. In the context of the MRP Shared Tasks, *mtool*[13] was developed to provide various functionality including format conversion, graph analysis, and evaluation. However, *mtool* is a command-line only tool and does not provide a simple and intuitive way to interact with large datasets of semantic graphs. In terms of graph visualisation, *mtool* supports the output of graphs in DOT language that is compiled with *Graphviz*.[14] Unfortunately, since *Graphviz* is a general-purpose visualization tool, the visualisations produced are often complex and inadequate at capturing framework-specific assumptions. As an example, the UCCA visualisation in Fig. 5b lacks lexical information.



(a) DMRS graph visualised with *delphin-viz*



(b) UCCA graph visualised with *mtool*

Figure 5: Visualisations provided by previous work

A number of other framework-specific visualisation tools exist, including for DMRS[15] and AMR (Saphra and Lopez, 2015), but are limited in terms of their functionality and framework support. The delphin-viz DMRS visualisation (Fig. 5a) is similar to our *flat* visualisation, but lacks the structure conveyed by our other visualisations.

6 Conclusion

The proliferation of graph-based and span-based meaning representations has created the need for a general platform for analysing these representations. **RepGraph** provides such a unified platform for the visualisation and analysis of semantic graphs in multiple frameworks. The tool allows users to explore and analyse these representations

[13] `https://github.com/cfmrp/mtool`

[14] `https://www.graphviz.org/`

[15] `http://delph-in.github.io/delphin-viz/demo/`

in an interactive and intuitive manner that is not possible with existing tools. Our work aims to make meaning representations accessible to a broader audience than researchers invested in a particular framework, while providing new insights into these representations through novel visualisations.

For future work, we plan to include support for additional semantic frameworks, including span-based representations such as semantic role labelling. We also intend to integrate parsers for all supported frameworks. This will provide significant utility by allowing users to upload text files or enter sentences directly, and use RepGraph straight away rather than having to run a parser separately.

References

Omri Abend and Ari Rappoport. 2013. Universal Conceptual Cognitive Annotation (UCCA). In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 228–238, Sofia, Bulgaria. Association for Computational Linguistics.

Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, and Nathan Schneider. 2013. Abstract Meaning Representation for sembanking. In *Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse*, pages 178–186, Sofia, Bulgaria. Association for Computational Linguistics.

Jan Buys and Phil Blunsom. 2017. Robust incremental neural semantic graph parsing. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1215–1226, Vancouver, Canada. Association for Computational Linguistics.

Shu Cai and Kevin Knight. 2013. Smatch: an evaluation metric for semantic feature structures. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 748–752, Sofia, Bulgaria. Association for Computational Linguistics.

Ann Copestake. 2009. **Invited Talk:** slacker semantics: Why superficiality, dependency and avoidance of commitment can be the right way to go. In *Proceedings of the 12th Conference of the European Chapter of the ACL (EACL 2009)*, pages 1–9, Athens, Greece. Association for Computational Linguistics.

Jeffrey Flanigan, Sam Thomson, Jaime Carbonell, Chris Dyer, and Noah A. Smith. 2014. A discriminative graph-based parser for the Abstract Meaning Representation. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1426–1436, Baltimore, Maryland. Association for Computational Linguistics.

Carlos Gómez-Rodríguez and Joakim Nivre. 2010. A transition-based parser for 2-planar dependency structures. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1492–1501, Uppsala, Sweden. Association for Computational Linguistics.

Jan Hajič, Eva Hajičová, Jarmila Panevová, Petr Sgall, Ondřej Bojar, Silvie Cinková, Eva Fučíková, Marie Mikulová, Petr Pajas, Jan Popelka, Jiří Semecký, Jana Šindlerová, Jan Štěpánek, Josef Toman, Zdeňka Urešová, and Zdeněk Žabokrtský. 2012. Announcing Prague Czech-English Dependency Treebank 2.0. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC'12)*, pages 3153–3160, Istanbul, Turkey. European Language Resources Association (ELRA).

Daniel Hershcovich, Omri Abend, and Ari Rappoport. 2017. A transition-based directed acyclic graph parser for UCCA. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1127–1138, Vancouver, Canada. Association for Computational Linguistics.

Zhengbao Jiang, Wei Xu, Jun Araki, and Graham Neubig. 2020. Generalizing natural language analysis through span-relation representations. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2120–2133, Online. Association for Computational Linguistics.

Alexander Koller, Stephan Oepen, and Weiwei Sun. 2019. Graph-based meaning representations: Design and processing. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics: Tutorial Abstracts*, pages 6–11, Florence, Italy. Association for Computational Linguistics.

Marco Kuhlmann and Stephan Oepen. 2016. Towards a catalogue of linguistic graph Banks. *Computational Linguistics*, 42(4):819–827.

Christopher Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven Bethard, and David McClosky. 2014. The Stanford CoreNLP natural language processing toolkit. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 55–60, Baltimore, Maryland. Association for Computational Linguistics.

Stephan Oepen, Omri Abend, Lasha Abzianidze, Johan Bos, Jan Hajic, Daniel Hershcovich, Bin Li, Tim O'Gorman, Nianwen Xue, and Daniel Zeman. 2020. MRP 2020: The second shared task on cross-framework and cross-lingual meaning representation parsing. In *Proceedings of the CoNLL*

2020 Shared Task: Cross-Framework Meaning Representation Parsing, pages 1–22, Online. Association for Computational Linguistics.

Stephan Oepen, Omri Abend, Jan Hajic, Daniel Hershcovich, Marco Kuhlmann, Tim O'Gorman, Nianwen Xue, Jayeol Chun, Milan Straka, and Zdenka Uresova. 2019. MRP 2019: Cross-framework meaning representation parsing. In *Proceedings of the Shared Task on Cross-Framework Meaning Representation Parsing at the 2019 Conference on Natural Language Learning*, pages 1–27, Hong Kong. Association for Computational Linguistics.

Stephan Oepen and Jan Tore Lønning. 2006. Discriminant-based MRS banking. In *Proceedings of the Fifth International Conference on Language Resources and Evaluation (LREC'06)*, Genoa, Italy. European Language Resources Association (ELRA).

Naomi Saphra and Adam Lopez. 2015. AMRICA: an AMR inspector for cross-language alignments. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations*, pages 36–40, Denver, Colorado. Association for Computational Linguistics.

Linfeng Song, Yue Zhang, Zhiguo Wang, and Daniel Gildea. 2018. A graph-to-sequence model for AMR-to-text generation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1616–1626, Melbourne, Australia. Association for Computational Linguistics.

Sheng Zhang, Xutai Ma, Kevin Duh, and Benjamin Van Durme. 2019. AMR parsing as sequence-to-graph transduction. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 80–94, Florence, Italy. Association for Computational Linguistics.