UCCAApp: Web-application for Syntactic and Semantic Phrase-based Annotation

Omri Abend, Shai Yerushlami, Ari Rappoport

Department of Computer Science, The Hebrew University of Jerusalem
{oabend|shaiy|arir}@cs.huji.ac.il

Abstract

We present UCCAApp, an open-source, flexible web-application for syntactic and semantic phrase-based annotation in general, and for UCCA annotation in particular. UCCAApp supports a variety of formal properties that have proven useful for syntactic and semantic representation, such as discontiguous phrases, multiple parents and empty elements, making it useful to a variety of other annotation schemes with similar formal properties. UCCAApp's user interface is intuitive and user friendly, so as to support annotation by users with no background in linguistics or formal representation. Indeed, a pilot version of the application has been successfully used in the compilation of the UCCA Wikipedia treebank by annotators with no previous linguistic training. The application and all accompanying resources are released as open source under the GNU public license, and are available online along with a live demo.[1]

1 Introduction

We present UCCAApp, a web-application for semantic and general phrase strucutre annotation, which has been developed for annotation using the UCCA scheme (Abend and Rappoport, 2013), but can support the annotation of most phrase-structure annotation schemes, including support for discontiguous units, multiple categories for a phrase, and reentrancy (multiple parents for a phrase, resulting in DAG structures). Despite the recent interest in web-based annotation applications, very few open source applications support phrase-structure annotation (see Section 5), a gap we address in this work.

UCCA (Universal Conceptual Cognitive Annotation) is a cross-linguistically applicable semantic representation scheme, building on the Basic Linguistic Theory typological framework (Dixon, 2010a,b, 2012), and Cognitive Linguistics literature (Croft and Cruse, 2004). It has demonstrated applicability to multiple languages, including English, French, German and Czech, support for rapid annotation, accessibility to non-expert annotators and stability under translation (Sulem et al., 2015). The scheme has recently proven useful for machine translation evaluation (Birch et al., 2016).

UCCA emphasizes accessibility and intuitive distinctions in the definition of its categories. UCCAApp complements this effort by offering an intuitive user interface (see Figure 1 and Section 4), which does not require background in formal representation and linguistics, as attested by the steep learning curve of annotators with no background in these fields that used UCCAApp in the annotation of the UCCA Wikipedia corpus (Abend and Rappoport, 2013).[2]

Aside from the annotation interface, the application includes modules for defining annotation schemes (layers), and for project management. Importantly, the system supports a multi-layered architecture, where the same text passage may be annotated by multiple layers. See Section 3.

In order to facilitate the adoption of UCCAApp by other research groups, we built UCCAApp using recent, standard web technology. The server is based on Django, accompanied with a Post-

[1] https://github.com/omriabnd/UCCA-App

[2] UCCA's resources are freely available through http://www.cs.huji.ac.il/~oabend/ucca.html.

greSQL database. The API follows the *apigee* style guide,[3] and the client is based on AngularJS.

2 The UCCA Framework

UCCA graphs are edge-labeled, directed acyclic graphs (DAGs), whose leaves correspond to the tokens of the text. A node (or *unit*) corresponds to a terminal or to several sub-units (not necessarily contiguous) viewed as a single entity according to semantic or cognitive considerations.

Edges bear a category, indicating the role of the sub-unit in the parent relation. Figure 2 presents a few examples. One incoming edge for each non-root node is marked as *primary*, and the rest (mostly used for implicit relations and arguments) as *remote* edges, a distinction made by the annotator (see Section 4). The primary edges thus form a tree structure, whereas the remote edges enable reentrancy, forming a DAG. The primary tree structure can be equivalently viewed as a node-labeled tree, where each node is labeled with the category of the edge leading to its primary parent. We adopt this view in our user interface.

UCCA is a multi-layered framework, where each layer corresponds to a "module" of semantic distinctions. UCCA's *foundational layer*, with which existing UCCA corpora are annotated, covers predicate-argument relations for predicates of all grammatical categories (verbal, nominal, adjectival and others), their inter-relations, and other major linguistic phenomena such as coordination and multi-word expressions. The layer's basic notion is the *scene*, describing a movement, action or state. Each scene contains one main relation (marked as either a Process or a State), as well as one or more Participants. For example, the sentence "After graduation, John moved to Paris" (Figure 2a) contains two scenes, whose main relations are "graduation" and "moved". "John" is a Participant in both scenes, while "Paris" only in the latter. Further categories account for inter-scene relations and the internal structure of complex arguments and relations .

3 System Architecture

The system is built as two completely separate modules: a RESTful web API for monitoring and manipulating the system's resources, and a client which interfaces with the system's resources through http requests to the API. This modular architecture allows to create, read, update and delete the system's resources without using the client, which is useful for the integration of external resources, such as parsers and analysis tools.

The system's resources can largely be categorized into resources that define annotation schemes (layers and categories), and resources for managing the annotation projects and tasks.

Defining Annotation Schemes. The atomic units of annotation schemes are the *categories*. A *layer* is a set of categories, paired with a set of restrictions over the joint occurrence of these categories (see Section 4). The grouping of categories into layers and the definition of restrictions are fully configurable either through chart-based admin interface or through the web API.

UCCAApp supports a multi-layered architecture, which is implemented using three types of inter-layer relations. The first is REFINEMENT, where one layer defines sub-categories for the parent layer (e.g., a Participant category may be refined to Agent and Patient categories). The second is COARSENING, where one layer defines super-categories for sub-sets of categories in the other layer (i.e., the inverse of REFINEMENT). The third is EXTENSION, where two disjoint sets of categories, that may inform the annotation of one another, are annotated one on top of the other (e.g., information structural categories and phrase structure annotation).

Passages and Users. UCCAApp includes standard management tools for text passages and users. There are four types of roles in the system. ADMIN users are super-users who administrate the system, and have complete access to its resources. PROJECT MANAGERS oversee one or more projects, and may create new layers and categories, invite new users and assign annotators with tasks. ANNOTATORS carry out tasks assigned to them, while GUESTS can view all the layers and categories defined in the system, as well as try out the annotation interface, in order to get an impression of the system's capabilities.

Passages are the source texts that the annotation procedure targets. The system allows inserting passages either through a web interface, or by uploading a delimited text file.

Projects. An annotation project is managed by a project manager, and consists of a layer (possibly shared across projects), and a set of tasks. Tasks come in several types, where each is defined according to the type of input it receives and what it

[3] https://apigee.com/

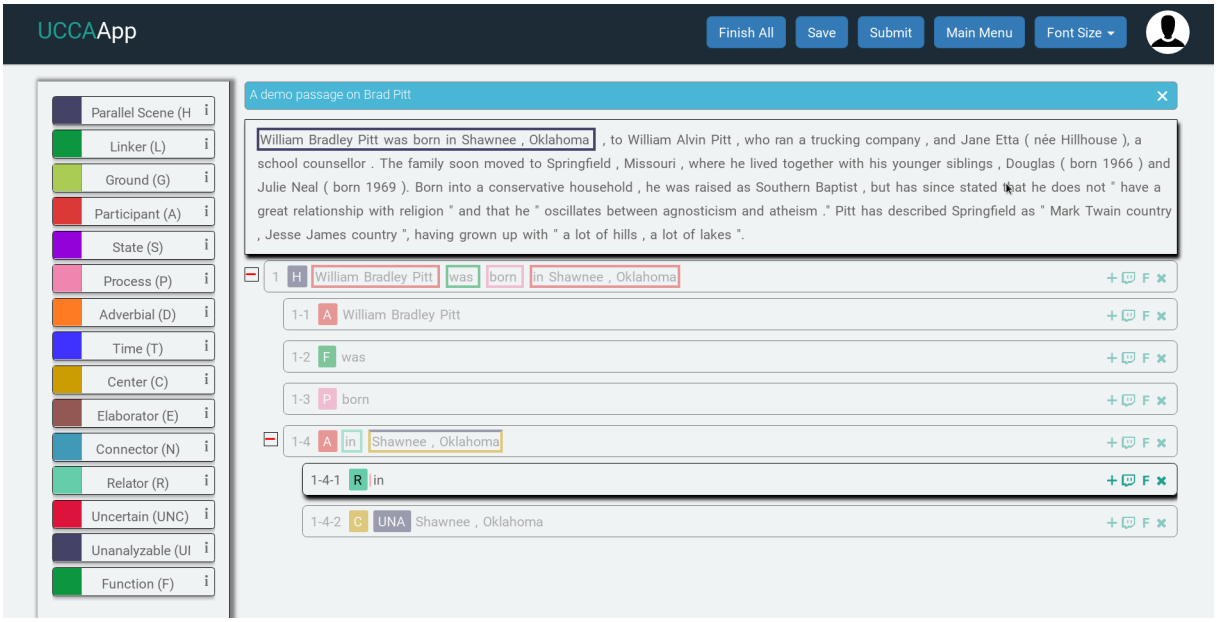

Figure 1: The layout of UCCAApp's annotation interface. See Section 4.

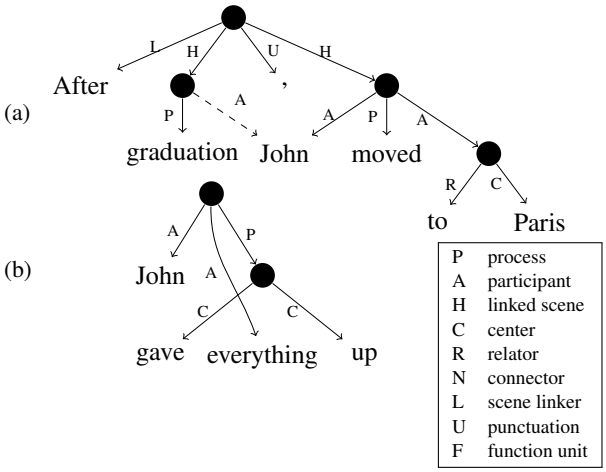

Figure 2: UCCA structures demonstrating two structural properties exhibited by the scheme. (a) includes a remote edge (dashed), resulting in "John" having two parents. (b) includes a discontinuous unit ("gave ... up"). Pre-terminal nodes are omitted for brevity. Right: legend of edge labels.

allows annotators to do.

- **Tokenization tasks**, whose input is a passage of text and whose output is a sequence of tokens. Tokenization tasks are carried out by hand-correcting the output of an automatic tokenizer.
- **Annotation task**, whose input is a set of tokens, and whose output is a DAG structure, whose leaves are the tokens. The task proceeds by iteratively grouping tokens into units.
- **Review task**, which is similar to an annotation task, but whose input is a full annotation of a given passage, and not only its set of tokens.

Annotation tasks in a project that uses a derived layer (i.e., a layer which is either a REFINEMENT, an EXTENSION or a COARSENING of a parent layer), start where a full annotation of a passage using the parent layer is in place. Such tasks are thus created by selecting a completed task that uses the parent layer as a parent task.

A complete documentation of the API is included in the project's repository. Figure 3 presents a screenshot of the admin screens used for managing the different resources. The admin interface is designed to scale to tens of thousands of tasks and passages, and supports advanced pagination, search and filtering capabilities.

4 Annotation Interface

The system's core is its annotation interface. Annotation is carried out over passages, rather than individual sentences. The application thus supports the annotation of inter-sentence relations, which feature in many annotation schemes, including the Penn Discourse Treebank (Miltsakaki et al., 2004), the Groningen Meaning Bank (Basile et al., 2012), and UCCA. UCCAApp fully supports both mouse functionality, for novice annotators, and keyboard functionality, which we have found to be faster, and more suitable for experienced annotators.

Layout. The layout of the web-interface is presented in Figure 1. Its main components are (1) a navigation bar with general functionality such as saving, submitting and returning to the main

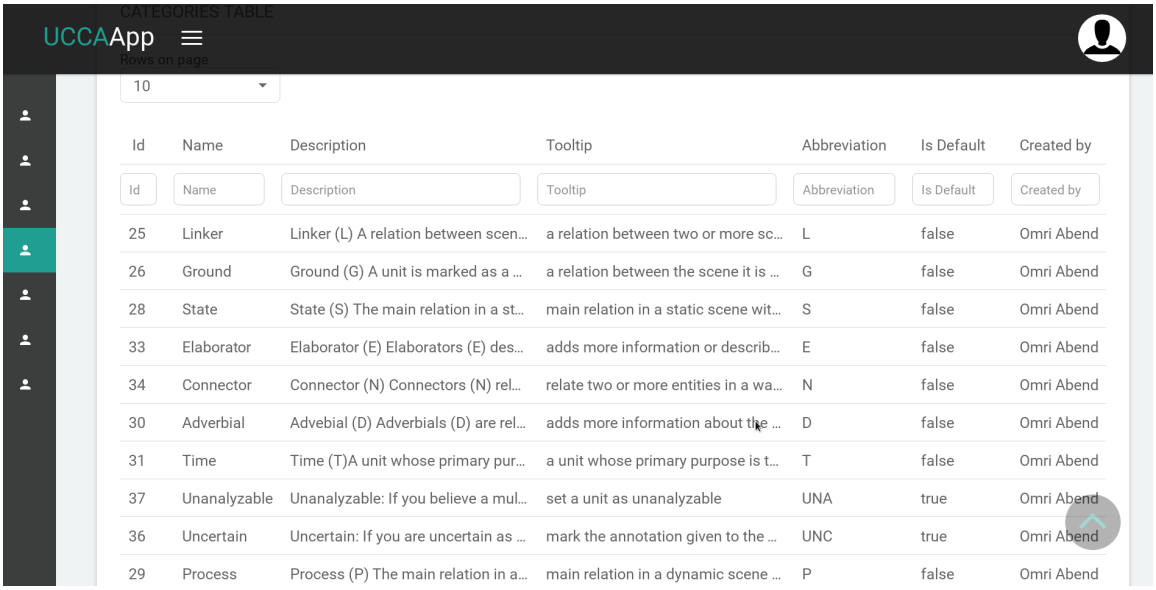

Figure 3: An example admin screen, for accessing and manipulating the system's resources.

menu; (2) a left sidebar that includes buttons for creating new categories, and assigning and removing categories from existing units, and (3) the annotation panel on which the annotation is carried out, and which takes up most of the screen.

Within the annotation panel, the passage box is presented at the top of the annotation panel, and corresponds to the root of the DAG. Upon the creation of a unit, a unit box is created, indented below its parent. Each unit thus appears twice in the annotation panel: as a nested unit box, and as a bordered span in the unit box of its parent. The cursor is always found within one of the unit boxes, which is highlighted (henceforth, the *focus unit*).

Creating and Deleting Units. Annotation is carried out by iteratively grouping tokens and units into larger units. Units are created by marking spans of text (not necessarily contiguous), declaring them as units, and assigning them any number of categories.

Preliminary user studies conducted with a pilot version of the application have shown that most annotators find it most intuitive and efficient to perform the annotation top-down, starting off by creating larger units (e.g., sentences), and iteratively analyzing their sub-parts, until the resulting unit cannot be further decomposed.

We consequently designed the interface to optimally support top-down annotation flow. Once an annotation unit has been created, the focus immediately shifts to the newly formed unit, allowing the user to internally analyze it. Once the annotation of a unit has been completed, the user may mark it as "finished". The system then validates the annotation of the unit and its sub-tree (according to the layer's restrictions), collapses its sub-unit tree and shifts the focus to the parent unit. Such a procedure minimizes the number of keystrokes required for a user who is annotating bottom-up. Still, the user is free to override this action order, and annotate the passage in any order she pleases.

Remote Sub-Units. In order to support the annotation of remote sub-units, we note that most schemes that allow DAG structures, including UCCA, use multiple parents to express shared argumenthood, namely a single argument participating in multiple relations (see Figure 2a). However, supporting such structures does not require general DAG annotation functionality, but only support for drawing additional edges between units that otherwise form a tree. For instance, UCCA annotation does not require a functionality for forming partially overlapping sub-units, such as having both w_1, w_2 and w_2, w_3 be sub-units of a unit whose span is w_1, w_2, w_3.

Remote edges are annotated by allowing the user to set any previously annotated unit (except for its descendant and ancestor units) as a remote sub-unit. For instance, in the sentence "After graduation, John moved to Paris", "John" is an argument of both "graduation" and "moved". UCCAApp can capture this shared argumenthood by

annotating "John" as a sub-unit of "John moved to Paris" and a remote sub-unit of "graduation".[4]

Remote sub-units may not have any children and are displayed in a different color in the unit hierarchy. For example:

Implicit Sub-Units. Many annotation schemes encode units that do not correspond to any span of text. Examples include traces, such as in the Penn Treebank (Marcus et al., 1993), and implicit arguments (Gerber and Chai, 2010). UCCAApp supports this functionality by allowing units to have remote sub-units that do not correspond to any span of text. Implicit sub-units may receive categories just like any other unit.

Restrictions Module. UCCAApp supports the introduction of restrictions over the joint occurrence of different categories within a layer. Restrictions may either forbid a category from having any children, require that two categories appear together as siblings or children of one another, or forbid them from doing so. Restrictions are validated when a unit is declared finished, or when the passage is submitted. If the validation fails, an error message pops up, indicating what the user should fix.

Multi-layered Annotation. In order to keep the user interface user friendly, annotation tasks that use derived layers, i.e., layers that were derived from another layer through extension, refinement or coarsening, are built upon a complete annotation according to the parent layer. The annotation task in a derived layer thus only includes adding units and categories to an existing annotation, which in itself is not editable.

For instance, assume L is a refinement layer of L_{parent}. Then an annotation task t by L begins with the submitted annotation of a parent task t_{parent}, annotated with L_{parent}. Completing t requires traversing the units created in t_{parent}, and for each of the units that is annotated with a category $c \in L_{parent}$, selecting which of the refined categories in L applies to it.

[4]There is some arbitrariness in selecting which unit "John" is a remote sub-unit of. Both assigning "John" to be a remote sub-unit of "graduation" and of "moved to Paris" are valid options, although annotators tend to prefer selecting contiguous units wherever possible.

5 Previous Work

While there are a number of open-source web-applications for corpus annotation, very few of them support phrase-based annotation in general, and the formal properties required by UCCA in particular. Annotald[5] is a web application for phrase structure annotation, originally developed for Icelandic. It is however difficult to use in a web-based setting, as it requires all annotators to be logged in to the same system, which often leads to security issues. Folia FLAT[6] is an open-source web-application with support for distributed collaborative annotation in a variety of annotation formats, including phrase-structure annotation. However, as it is mostly geared towards flat annotations, phrase structure annotation with FLAT is somewhat difficult. SynTree[7] also supports phrase-structure annotation, focusing on Chinese. Its applicability, however, has so far been limited due to its documentation being formulated only in Chinese. To the best of our knowledge, none of these applications support DAG annotation, or full keyboard functionality.

The AMR editor[8] (Banarescu et al., 2013) supports annotation through an interface based on the linearization of the AMR DAGs using the PENMAN notation. The application has a number of dedicated functionalities both for facilitating the annotation process (e.g., nodes and edges can be added either by directly inputting their textual representations, or through more elaborate modals), and for validating that the resulting annotation is indeed well-formed. The editor is also well integrated with the AMR category set, facilitating the navigation through its rich ontology. UCCAApp supports many of the properties required for AMR annotation, including reentrancy, and offers a graphical user interface that does not require annotators to be versed in formal notation. While not all functionalities required for AMR annotation are currently supported in UCCAApp (importantly, the application does not support representations that are not anchored in the words and phrases of the text), future work will address the adaptation of UCCAApp to AMR annotation.

A number of recent annotation web applications support dependency annotation, and provide ef-

[5]https://annotald.github.io/
[6]https://github.com/proycon/flat
[7]http://syntree.github.io/
[8]http://www.isi.edu/~ulf/amr/AMR-editor.html

fective tools for collaborative annotation by geographically dispersed parties. The brat application (Stenetorp et al., 2012) is a popular web-based annotation tool that supports a wide variety of annotation types, including dependency analysis, chunking and named entity recognition. However, annotation in brat is carried out through dialogue boxes, which slows down the annotation process. WebAnno (Eckart de Castilho et al., 2016) is a generic and flexible annotation tool for collaborative annotation, which supports the joint annotation of semantic and syntactic dependencies. One of its major design principles is multi-layered analysis and the effective browsing of rich category sets, which it supports using a suggestion engine, and both manually-configurable and automatically-induced constraints on the joint appearance of categories. It also improves upon brat's (and earlier versions of WebAnno) user interface, allowing improved keyboard functionalities and visualization. Arborator (Gerdes, 2013) is a lightweight, web-based annotation application which focuses on dependency structures. The tool is easily configurable, and has a mouse-based interface for creating annotations. None of these recently proposed tools support phrase-structure grammar annotation.

6 Conclusion

We presented UCCAApp, an open-source web application for phrase-based and UCCA annotation, with an intuitive and accessible design. An earlier version of UCCAApp was developed as part of the compilation of the English UCCA corpus, and was successfully employed by annotators with no background in linguistics or formal representation.

UCCAApp supports annotation with a variety of formal properties, including discontiguous units, inter-sentence annotation, reentrancy and multi-layered annotation, making it suitable for other syntactic and semantic annotation schemes that use these properties. Future extensions of UCCAApp will include further analysis capabilities, such as inter-annotator agreement tools, as well as support for AMR annotation.

Given the scarcity of freely available annotation web applications for phrase structures, UCCAApp's modular and extensible design, and its intuitive user interface, we believe UCCAApp will make a substantial contribution to the field of linguistic annotation.

References

Omri Abend and Ari Rappoport. 2013. Universal Conceptual Cognitive Annotation (UCCA). In *Proc. of ACL*. pages 228–238.

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 *Proc. of LAW*. pages 178–186.

Valerio Basile, Johan Bos, Kilian Evang, and Noortje Venhuizen. 2012. Developing a large semantically annotated corpus. In *Proc. of LREC*. pages 3196–3200.

Alexandra Birch, Omri Abend, Ondřej Bojar, and Barry Haddow. 2016. HUME: Human UCCA-based evaluation of machine translation. In *Proc. of EMNLP*. pages 1264–1274.

William Croft and Alan Cruse. 2004. *Cognitive linguistics*. Cambridge University Press.

R.M.W. Dixon. 2010a. *Basic Linguistic Theory: Grammatical Topics, Vol. 2*. Oxford University Press.

R.M.W. Dixon. 2010b. *Basic Linguistic Theory: Methodology, Vol. 1*. Oxford University Press.

R.M.W. Dixon. 2012. *Basic Linguistic Theory: Further Grammatical Topics, Vol. 3*. Oxford University Press.

Richard Eckart de Castilho, Eva Mujdricza-Maydt, Seid Muhie Yimam, Hartmann Silvana, Iryna Gurevych, Annette Frank, and Chris Biemann. 2016. A web-based tool for the integrated annotation of semantic and syntactic structures. In *Proc. of the LT4DH workshop*. pages 76–84.

Matthew Gerber and Joyce Chai. 2010. Beyond NomBank: A study of implicit arguments for nominal predicates. In *Proc. of ACL*. pages 1583–1592.

Kim Gerdes. 2013. Collaborative dependency annotation. In *Proc. of DepLing*. pages 88–97.

Mitch Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics* 19:313–330.

Eleni Miltsakaki, Rashmi Prasad, Aravind Joshi, and Bonnie Webber. 2004. The penn discourse treebank. In *LREC*. pages 2237–2240.

Pontus Stenetorp, Sampo Pyysalo, Goran Topić, Tomoko Ohta, Sophia Ananiadou, and Jun'ichi Tsujii. 2012. Brat: a web-based tool for nlp-assisted text annotation. In *Proc. of EACL*. pages 102–107.

Elior Sulem, Omri Abend, and Ari Rappoport. 2015. Conceptual annotations preserve structure across translations: A French-English case study. In *Proc. of Semantics-Driven Statistical Machine Translation (S2MT) Workshop*. pages 11–22.