

Jubilee: Propbank Instance Editor Guideline (Version 2.1)

Jinho D. Choi
choijd@colorado.edu

Claire Bonial
bonial@colorado.edu

Martha Palmer
mpalmer@colorado.edu

Center for Computational Language and Education Research
Institute of Cognitive Science
University of Colorado at Boulder

Institute of Cognitive Science
Technical Report 02-09

October 2, 2009

Abstract

This report gives a guideline of how to use a Propbank instance editor, Jubilee. Propbank is a corpus in which the arguments of each predicate are annotated with their semantic roles in relation to the predicate. In addition to semantic role annotation, Propbank annotation requires the choice of a sense ID (also known as a roleset or frameset ID) for each predicate. Thus, for each predicate in the Propbank, there exists a corresponding frameset file that shows the predicate argument structure of all senses related to the predicate. Jubilee facilitates the annotation process by displaying to the annotator several resources of syntactic and semantic information simultaneously: firstly, the syntactic structure of a sentence is displayed in the main frame using the Penn Treebank style of phrase structure; secondly, the available senses with their corresponding argument structures are displayed in another frame; thirdly, all available Propbank arguments including modifiers (e.g., **ARGO**, **ARGM-TMP**) are displayed for the annotator's choice; finally, example annotations of each sense of the predicate are available to the annotator for viewing whenever desired. The easy use of each resource allows the annotator to quickly absorb and apply the necessary syntactic and semantic information pertinent to each predicate for consistent and efficient annotation. Jubilee has been successfully adopted to Propbank projects in several organizations such as the University of Colorado at Boulder, the University of Illinois at Urbana-Champaign, and Brandeis University. The tool runs platform independently, is light enough to run as an X11 application and supports multiple languages such as Arabic, Chinese, English, Hindi and Korean.

Contents

1	Introduction	3
2	Getting started	4
2.1	Install JDK	4
2.2	Download and install Jubilee	4
2.3	Run Jubilee	4
3	Jubilee in normal mode	6
3.1	Launch Jubilee for annotators	6
3.2	Treebank view	7
3.3	Frameset view	7
3.4	Argument view	7
3.5	Annotate traces, concatenations and links	9
3.6	Annotate verb particle constructions	11
3.7	Save annotations	11
3.8	Open a new task	11
4	Jubilee in gold mode	12
5	Conclusion	13

1 Introduction

Jubilee is a Propbank instance editor developed at the University of Colorado at Boulder. Propbank is a corpus in which the arguments of each verb predicate are annotated with their semantic roles in relation to the predicate¹ (Palmer et al., 2005). In addition, each predicate is annotated with its sense ID, also known as a roleset or frameset ID. Currently, all the Propbank annotations are done on top of the phrase structure such as Penn Treebank (Marcus et al., 1993). For each verb predicate in every tree (representing the phrase structure of the corresponding sentence), we create a Propbank instance that consists of the predicate’s sense ID (e.g., `run.02`) and its arguments labeled with the semantic roles (e.g., `ARG0`, `ARG1`). The argument structure of the predicate is outlined in the corresponding frameset file (`run.xml`), which is an XML file defining the argument structure and the exact meaning of the numbered arguments for each sense of the predicate along with some additional semantic and syntactic information (Choi et al., 2009). If more than one verb predicate occurs in a tree, several Propbank instances can be generated from the tree. Table 1 shows an example of a Propbank instance associated with a verb, ‘run’.

E.g.	John <i>ran</i> the Boston Marathon.
Sense ID	<code>run.02</code> (‘walk quickly, a course or contest’)
ARG0	John (‘runner’)
ARG1	the Boston Marathon (‘course, race, distance’)

Table 1: An example of Propbank instance associated with ‘run’

Prior to the annotation, instances that are neither sense-tagged nor annotated yet are grouped by different predicates and formed into tasks such that each task consists of Propbank instances organized by the same predicate lemma. If a task includes too many instances, it can be broken down into smaller ones (usually 100 instances per task). Each task is double-annotated and subsequently adjudicated; during the adjudication process, the annotation that is the most appropriate is selected for the gold standard. If neither annotation is appropriate, adjudicators may correct the annotation for the instance.

In the past, this procedure was done using three different tools. First, an annotator would claim a task to work with using a command-line tool, so other annotators would know which tasks had been already taken. As is the case today, each task could only be claimed by a certain number of annotators, generally two, in order to prevent annotation of the same task more than twice. Once the claiming was done, the annotator would then start annotating the task using a different tool with graphical user interface. Using this tool, the annotator was able to view the tree structure of each instance in parenthetical notation (examples of trees in parenthetical notation are shown in Section 3.5). However, visualizing the constituent boundaries and selecting the appropriate node for an argument was often very difficult for annotators with less expertise in this format. Once both annotators finished annotating the task, an adjudicator would adjudicate the annotations using the same annotation tool; however, the adjudicator would subsequently have to use another tool to denote the verb sense of each instance. Thus, three different tools were required to create one Propbank instance in the task. Although each tool worked effectively on its own, using several tools for one task not only decreased the efficiency of the project, but also required another process of checking if all tools were properly used. This motivated the creation of a new annotation tool, Jubilee, which can do all of above and more.

With Jubilee, the entire annotation procedure can be done using one tool that simultaneously provides rich, graphical syntactic information as well as comprehensive semantic information from the frameset files. Perhaps most importantly, the use of Jubilee ensures that all the annotations can be saved in one place using one unified format, the lack of which had been a delaying factor when we used the other tools. Moreover, this makes the data maintenance much easier and more consistent. The tool is highly adaptable such that creating a new Propbank project from a different corpus is as easy as editing a simple text file.

Jubilee is developed in Java, J2SE Development Kit (JDK) 6.0, so it runs on all kinds of platforms (Microsoft Windows, Mac OS, and Linux), as long as the right version of JDK is installed.² It is light enough to run as an X11 application. This aspect is important because both Treebank and Propbank files are usually stored in a server, and annotators and adjudicators are to work on the tasks remotely (via SSH) using their local machines. One of the biggest advantages of using Jubilee is that it runs on many different languages; in fact, it has been used for Propbank projects in Arabic (Diab et al., 2008), Chinese (Xue

¹There exists another corpus called Nombank (Meyers et al., 2004) that takes nouns as the predicates. Recently, Propbank started including eventive nouns as the predicates as well; however the corpus is yet available.

²The current version of Jubilee also runs on JDK 5.0 but running on JDK 6.0 is preferable for the future updates.

and Palmer, 2009), English (Palmer et al., 2005) and also been tested on Hindi and Korean (Han et al., 2002).

This report details how to setup and run Jubilee in normal and gold modes. The normal (annotation) and gold (adjudication) mode is meant to be run by annotators and adjudicators, respectively. In normal mode, annotators are allowed to see and open only tasks that have been claimed by themselves or those that have been claimed by one other annotator (when the maximum number of annotators allowed for each task is two). In gold mode, adjudicators are allowed to see and open all tasks that have undergone at least single-annotation. Adjudicators can then either choose appropriate annotations or modify annotations if necessary to set the gold standard annotation. Although there are two different modes, the interfaces are very similar, so learning one mode effectively teaches the other.

2 Getting started

2.1 Install JDK

You first need to install JDK 6.0 or higher on your machine. To install JDK, you need to download the installation file from <http://java.sun.com/javase/downloads/> and follow the guideline provided by the webpage.

2.2 Download and install Jubilee

You can download Jubilee from <http://code.google.com/p/propbank/downloads/>. From the list, download both `jubilee-system.tar.gz` and `jubilee-version.jar` in the same directory. Move to the directory and unarchive `jubilee-system.tar.gz` by typing the following command on the terminal:

```
tar -zxvf jubilee-system.tar.gz
```

This command will create two folders: `resource` and `system`. The `resource` folder contains resources required for the Propbank annotation. There are four sub-folders under `resource` folder: `annotation`, `frameset`, `task` and `treebank`. The `annotation` folder is initially empty, but will be filled with annotation files. The `task` folder contains a sample task file, `AFGP.task`, including four unannotated instances. The `frameset` and `treebank` folders contain frameset and Treebank files used for the task.

The `system` folder contains configuration files required to run Jubilee. There are three kinds of configuration files in the folder: a language-argument file, a function-argument file and a project-path file. The language-argument file contains the language-specific semantic role labels and their keyboard shortcuts used in Jubilee (see Section 3.4 for more details). The naming convention of the language-argument file is as follows:

```
LANG      ::= arabic | chinese | english
FILENAME ::= LANG.args (e.g., english.args)
```

The function-argument file (`function.args`) contains operators and their keyboard shortcuts used for Propbank annotations (see Section 3.5 for more details). Finally the project-path file contains directory paths of all resource folders (`annotation`, `frameset`, `task` and `treebank`). When you start a new Propbank project, you need to create a new project-path file that contains directory paths of appropriate resource folders used for the project. The resource folders can be under any location that the project-path file points to; they do not have to be under `resource` folder. The naming convention of the project-path file is as follows:

```
LANG      ::= arabic | chinese | english
PROJECT   ::= name of the project
FILENAME  ::= LANG.PROJECT.path (e.g., english.sample.path)
```

2.3 Run Jubilee

To run Jubilee, type the following command on the terminal:

```
Usage: java -jar jubilee-version.jar <user ID> [max annotators = 2] [skip = 0]
```

`<user ID>` is a required parameter that indicates the user ID. Jubilee uses this user ID to distinguish tasks annotated by different users. If you use ‘gold’ as the user ID, Jubilee will run in gold mode (Section 4). The `[max annotators]` option indicates the maximum number of annotators allowed to annotate each task (the default value is 2). The `[skip]` option is used only for gold mode. If `skip` is equal to 1, Jubilee skips instances whose annotations (done by different annotators) are the same. This option is useful when annotators are well-trained and seasoned so that annotations agreed upon by the annotators are considered correct, and therefore do not need to be checked by the adjudicator. If `skip` is equal to 0, Jubilee does not skip such instances. This option is useful when many annotators are less skilled or annotation procedures have undergone changes; thus, annotations agreed upon by two annotators may still be incorrect and require modification by the adjudicator. The default value for the `[skip]` flag is 0.

3 Jubilee in normal mode

3.1 Launch Jubilee for annotators

Annotators are expected to run Jubilee in normal mode. In normal mode, annotators are allowed to view and edit only annotations either done by themselves or claimed by one other annotator (when the maximum number of annotators allowed for each task is two). To run Jubilee in normal mode, type the following command in the terminal:

```
java -jar jubilee-version.jar choijd
```

This will launch Jubilee for a user `choijd` in normal mode, allowing two annotators for each task. If you want to allow three annotators for each task, type the following command instead:

```
java -jar jubilee-version.jar choijd 3
```

When you launch Jubilee, you will see an open-dialog (Fig. 1). There are three components in the dialog. The combo-box at the top shows a list of project-path files in the `system` folder (Section 2.2). Once you select a project (e.g., `english.sample`), both `New Tasks` and `My Tasks` lists will be updated. `New Task` shows a list of task files that have either not been claimed, or claimed by only one other annotator. `My Tasks` shows a list of task files that have been claimed by the current user (e.g., `choijd`). In any case, you can choose only one task at a time.

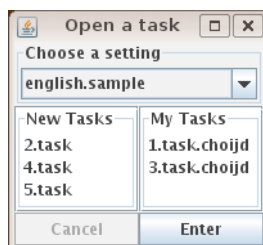


Figure 1: Open-dialog

Once you choose a task and click `Enter` button, the Jubilee main window will be prompted (Fig 2).

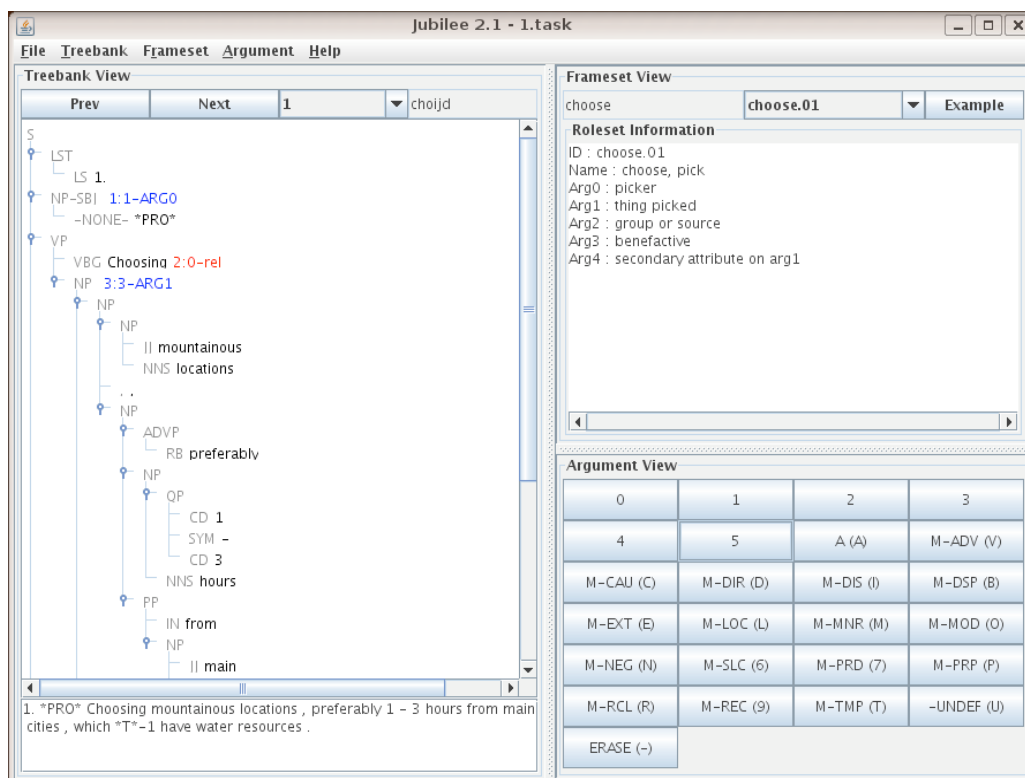


Figure 2: Jubilee main window

There are three kinds of views available in the main window: treebank view, frameset view and argument view. The following sections explain how each view is used for Propbank annotation.

3.2 Treebank view

By default, the treebank view shows the first tree in the selected task file. Each node in the tree consists of two elements: POS-tag and word-token. Upon annotation, a Propbank argument label is added to each node where appropriate. The verb predicate is marked with a special tag ‘rel’. To navigate through different trees in the task, click [Prev] and [Next] buttons (shortcut: , and .) or [Jump] combo-box for quick jump (Ctrl+J).

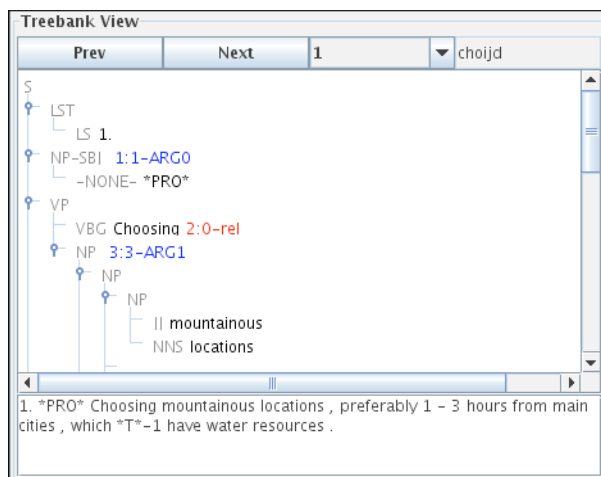


Figure 3: Treebank view

The annotator text-box on the right shows the user ID (e.g., **choijd**) indicating the user annotating this instance. All Propbank instances include such user IDs so that later the adjudicators know which instances are annotated by whom. The sentence text-box at the bottom shows the raw sentence of the tree. To view the tree in parenthetical format, click [Treebank - View Tree in Text] on the menu (Ctrl+T).

3.3 Frameset view

The frameset view displays and allows the annotator to choose the sense of the predicate with respect to the current tree. The predicate text-box on the left shows the lemma of the predicate associated with the selected roleset³ in the roleset combo-box. The roleset combo-box at the middle gives the full list of roleset IDs associated with the predicate lemma. Initially, the predicate lemma followed by .XX is displayed in the roleset combo-box to indicate that no sense has been chosen yet (e.g., **choose.XX**). Annotators can peruse the roleset IDs by clicking on the roleset combo-box, or using keyboard shortcuts, [and], to move back and forth through the list. As a roleset is selected, the argument structure and a short definition of the roleset, extracted from the corresponding frameset file (e.g., **choose.xml**), appear in the roleset information pane. To view annotation examples of the currently selected roleset, click [Example] button (Ctrl+E). To view the arguments annotated in the current tree, click [Frameset - View Arguments] on the menu (Ctrl+W).

3.4 Argument view

The argument view contains buttons representing each of the Propbank argument labels (Fig. 5). To annotate an argument (e.g., **ARG0**), select the node to be annotated in the treebank view, then click the corresponding argument button (e.g., [0]). Alternatively, you could use a keyboard shortcut to insert the argument (e.g., 0). All keyboard shortcuts can be found from the menu, [Argument - Arguments]. When you click the button (or use the shortcut), an argument label (e.g., **ARG0**) is annotated on the selected node, along with its relative location in the tree (e.g., 1:1). In the location designation, the first number indicates the index of the word-token closest to the selected node and the second number indicates the phrase level of the node from the closest word-token. For example, NP-SBJ node in Fig. 3 is annotated as 1:1-ARG0. This indicates that the node has semantic role of **ARG0** and its location in the

³The term ‘roleset’ is interchangeable with a term ‘frameset’ depending on languages.

Frameset View

choose choose.01 Example

Roleset Information

ID : choose.01
 Name : choose, pick
 Arg0 : picker
 Arg1 : thing picked
 Arg2 : group or source
 Arg3 : benefactive
 Arg4 : secondary attribute on arg1

Figure 4: Frameset view

tree is 1:1 such that the 1st word-token in the tree, ***PRO***, is the closest token-level node from **NP-SBJ**, and the phrasal node **NP-SBJ** is 1 level higher than ***PRO***. Note that numbered arguments reflect either the arguments that are obligatory given the valency of the predicate (e.g., agent, patient, benefactive), or if not obligatory, those that occur with high-frequency in actual usage.

0	1	2	3
4	5	A (A)	M-ADV (V)
M-CAU (C)	M-DIR (D)	M-DIS (I)	M-DSP (B)
M-EXT (E)	M-LOC (L)	M-MNR (M)	M-MOD (O)
M-NEG (N)	M-SLC (6)	M-PRD (7)	M-PRP (P)
M-RCL (R)	M-REC (9)	M-TMP (T)	-UNDEF (U)
ERASE (-)			

Figure 5: Argument view

The **[Erase]** button is used to remove the annotation assigned to the selected node. **[-UNDEF]** button is used to assign an argument that is not defined in the predicate-argument structure of the currently selected roleset. For example, if the selected roleset has only **ARG0** and **ARG1** in its argument structure but the annotator found a node in the tree that should be **ARG2**, **ARG-UNDEF** tag is used to indicate potential changes in the corresponding frameset file.

3.5 Annotate traces, concatenations and links

Several operators are used to annotate certain traces, and to perform concatenations and coreference links. In the absence of Treebank co-indexing between a null element and its overt referent, annotators can provide semantic information about the null element serving as an argument by manually linking it to an overt referent in the instance by using ‘*’. For example:

Treebank annotation:

```
. . .
(NP-SBJ (NP (NNP China) (POS 's))
  (NN income)
  (NNS taxes))
(VP
  (VBD amounted)
  (PP-CLR (TO to)
    (NP (QP (RB approximately) (CD 180) (CD billion))))
  (, ,)
  (S-ADV
    (NP-SBJ (-NONE- *PRO*))
    (VP
      (VBG accounting)
      (PP-CLR (IN for)
        (NP (QP (RB about) (CD 6) (SYM \%))
          (PP (IN of)
            (NP (NP (DT the) (NN state) (POS 's))
              (JJ financial)
              (NNS revenues))))))))))
```

Propbank annotation:

```
REL : accounting
ARG0: [NP-SBJ *PRO*] * [NP-SBJ China's income taxes]
ARG1: [PP-CLR for about 6% of the state's financial revenues]
```

This operator is also used to create a semantic link between relative pronouns and their referents, which are never co-indexed in the Treebank:

Treebank annotation:

```
. . .
(NP (NNS answers))
(SBAR
  (WHNP-6 (WDT that))
  (S (NP-SBJ-3 (PRP we))
    (VP (MD 'd)
      (VP (VB like)
        (S (NP-SBJ (-NONE- *-3))
          (VP (TO to)
            (VP (VB have)
              (NP (-NONE- *T*-6))))))))))
```

Propbank annotation:

```
REL : have
ARG0: [NP-SBJ *-3]
ARG1: [NP *T*-6]
ARGM-LINK-SLC: [WHNP-6 that] * [NP answers]
```

Similarly, the ‘&’ operator is used to link the object trace after a passive verb to its referent in the subject position in reduced relative constructions:

Treebank Annotation:

```

. . .
(NP (DT the) (JJ second) (NN batch))
(PP (IN of)
  (NP
    (NP (NNS countries))
    (VP (VBN covered)
      (NP (-NONE- *))
      (PP-LOC (IN in)
        (NP (DT the) (JJ eastward) (NN expansion))))))

```

PropBank Annotation:

```

REL : covered
ARG1: [NP *] & [NP countries]
ARGM-LOC: [PP-LOC in the eastward expansion]

```

Additionally, in the cases where an argument is discontinuous such that it cannot be captured in the annotation of one node, ‘,’ can be used to concatenate more than one node into a single argument:

Treebank Annotation:

```

(NP-SBJ
  (DT The)
  (ADJP (RB recently)
    (VBN concluded))
  (NML
    (NNP China)
    (HYPH -)
    (NNP Africa))
  (NNP Forum))

```

PropBank Annotation:

```

REL : concluded
ARG1: [NML China - Africa] , [NNP Forum]
ARGM-TMP: [RB recently]

```

The following shows how to provide the semantic * link between a null element and its referent. Other operators work in an analogous manner.

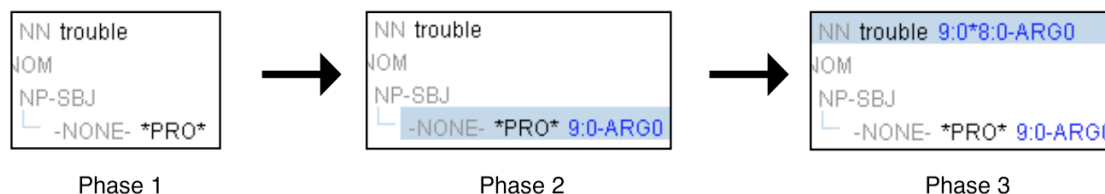


Figure 6: Argument view

1. Assume that *PRO* is ARG0 and trouble is the node that *PRO* needs to be linked to.
2. We first need to annotate *PRO* as ARG0. Select the *PRO* node in the treebank view and click either [0] button in the argument view or ‘0’ on the keyboard. This inserts the argument label (ARG0) with its relative location in the tree (9:0) to the selected node (*PRO*).
3. Next, select the trouble node and type Ctrl+Shift+8. Jubilee remembers the last annotation you inserted (9:0-ARG0) and adds the annotation with a * link to the currently selected node. Hence, the final annotation of trouble becomes 9:0*8:0-ARG0 where 8:0 is the location of the trouble node.

3.6 Annotate verb particle constructions

Some English verbs use particles to form their predicate lemmas (e.g., ‘run up,’ as in ‘run up a bill’). For such verb particle constructions, the particle is concatenated with the verb to form a single predicate lemma, annotated with the ‘rel’ tag. To concatenate the particle, choose the particle node and type **Ctrl+Shift+/,** The resulting **rel** annotation will reflect the locations of both the original predicate and the concatenated particle; for example, **8:0,9:0-rel** where **8:0** indicates the location of the verb and **9:0** indicates the location of the particle. Note that if for some reason the annotator accidentally erases the **rel** tag on any verb, it can be recovered using the same command, **Ctrl+Shift+/,** on the verb.

3.7 Save annotations

All annotations are saved automatically as you move on to a different tree. Notice that Jubilee may not save the very last tree you worked on; in this case, you can manually save the annotations by clicking [**File - Save**] on the menu (**Ctrl+S**). The annotation file is saved with a filename ***.task.userID** and can be found in the annotation folder (Section 2.2). For example, if the currently selected task is **1.task** and the user ID is **choijd**, the annotations will be saved to a file **1.task.choijd**. If you want to save the annotations to some place other than the default location, click [**File - Save As**] on the menu.

3.8 Open a new task

You can open a new task without relaunching Jubilee. Click [**File - Open**] on the menu (**Ctrl+O**) then the open-dialog will be prompted (Fig. 1). On the dialog, choose a new task either from the same project you previously selected or from a different project.

4 Jubilee in gold mode

Adjudicators are expected to run Jubilee in gold mode. In gold mode, adjudicators are allowed to view and edit all annotations. To run Jubilee in gold mode, type the following command in the terminal:

```
java -jar jubilee-version.jar gold
```

This will launch Jubilee in the default gold mode, which accommodates and displays two annotations for each instance. Notice that the only difference between running in normal and gold mode is the use of the user ID, `gold`. In Jubilee, `gold` is a reserved ID only for adjudicators. If you want to view three annotators' annotations for each instance, type the following command instead:

```
java -jar jubilee-version.jar gold 3
```

This will launch Jubilee in gold mode, viewing three annotations for each instance. If you want to skip the instances that all annotators agreed on, type the following command:

```
java -jar jubilee-version.jar gold 2 1
```

This will launch Jubilee in gold mode, viewing two annotations for each instance and skipping instances for which all annotations are the same.

When you launch Jubilee, you will see the same open-dialog as you saw in Fig. 1. The [New Tasks] shows a list of task files that have not been adjudicated and the [My Tasks] shows a list of task files have been adjudicated. In gold mode, however, Jubilee does not allow opening tasks in [New Tasks] unless there already exists one or more annotation files created for the task. If you try to open a task that has not been annotated by any annotator, it gives you an error message (Fig. 7). This prevents adjudicators from claiming tasks that have not yet been annotated.

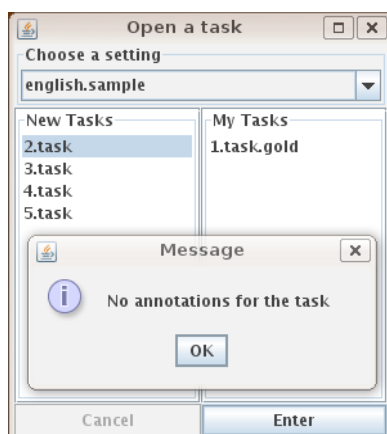


Figure 7: Open-dialog in gold mode

Once you launch Jubilee in gold mode, you will see almost the same main window as in Fig. 2 (page 6), except now you have a list showing all annotations for the current instance at the top of the treebank view (Fig 8). The first line shows the annotation done by adjudicator, and the following lines show annotations done by different annotators. By default, the annotation from the first annotator is chosen as gold, but it can be changed by clicking the other annotation. To modify the annotation in cases where neither is perfectly correct, the adjudicator can simply manipulate the arguments as described for the normal annotation mode.

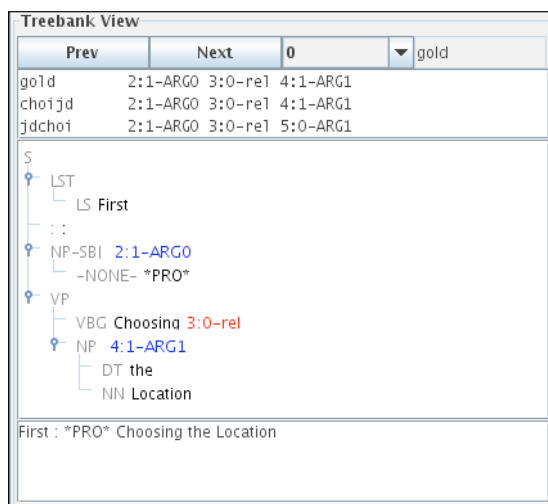


Figure 8: The treebank view in gold mode

When you save the adjudications, Jubilee creates an adjudication file called `*.task.gold` (e.g., `1.task.gold`) in the annotation folder.

5 Conclusion

Jubilee has drastically simplified and streamlined the Propbank annotation process because what was once done with three separate tools can now be completed with one. This has several distinct advantages. First, it speeds up the entire process of annotation and adjudication because argument annotation and the choice of a sense ID (roleset or frameset) for the verb can be completed simultaneously. This also ensures that annotators have a greater awareness of the argument structure and semantics of the predicate as defined in the frameset file. Thus, annotators can quickly and efficiently familiarize themselves with a predicate and apply that knowledge consistently to each instance. Additionally, Jubilee’s treebank view gives the annotator a more effective visual representation of the syntax of each instance than the previous parenthetical notation, clarifying the relationship between the argument structure found in the frame and that of the individual instance. Finally, the use of one tool simplifies data maintenance because one unified file format can be used throughout the process.

Jubilee has been successfully adopted to Propbank projects in several organizations such as the University of Colorado at Boulder, the University of Illinois at Urbana-Champaign, and Brandeis University. We will continuously develop the tool by improving its functionalities through user-testing and feedback, and also by applying it to more languages.

Acknowledgments

Special thanks to Professor Nianwen Xue at Brandeis University. Without his insight, the tool would not be as complete as it is now.

References

- Jinho D. Choi, Claire Bonial, and Martha Palmer. 2009. Cornerstone: Propbank frameset editor guideline (version 1.3). Technical Report 01-09, Institute of Cognitive Science, the University of Colorado at Boulder.
- Mona Diab, Aous Mansouri, Martha Palmer, Olga Babko-Malaya, Wajdi Zaghoulani, Ann Bies, and Mohammed Maamouri. 2008. A pilot arabic propbank. In *Proceedings of the 7th International Conference on Language Resources and Evaluation (LREC'08)*.
- Chunghye Han, Narae Han, Eonsuk Ko, and Martha Palmer. 2002. Korean treebank: Development and evaluation. In *Proceedings of the 3rd International Conference on Language Resources and Evaluation (LREC'02)*.
- Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. 1993. Building a large annotated corpus of english: the penn treebank. *Computational Linguistics*, 19(2):313–330.
- A. Meyers, R. Reeves, C. Macleod, R. Szekely, V. Zielinska, B. Young, and R. Grishman. 2004. The nombank project: An interim report. In *HLT-NAACL 2004 Workshop: Frontiers in Corpus Annotation*, pages 24–31.
- Martha Palmer, Daniel Gildea, and Paul Kingsbury. 2005. The proposition bank: An annotated corpus of semantic roles. *Computational Linguistics*, 31(1):71–106.
- Nianwen Xue and Martha Palmer. 2009. Adding semantic roles to the chinese treebank. *Natural Language Engineering*, 15(1):143–172.