Japanese FrameNet Data Quality Overview: Toward Full Data Integrity

Alexandre Kabbach
Keio University
akabbach@keio.jp

Kyoko Hirose Ohara
Keio University
ohara@hc.st.keio.ac.jp

Abstract

This paper discusses how to maintain Japanese FrameNet (JFN) data integrity and selective consistency with English FrameNet (FN) data. We propose a generic protocol to perform extensive data quality checks and to automatically remove all incomplete, redundant or inconsistent data from the JFN database. We also present a method to perform semi-automatic FN and JFN databases comparison in order to synchronize the language-independent parts of the FN data. Additionally, we suggest several database improvements to enable upstream data corruption prevention and preserve data integrity regardless of database usage. Our approach is language-independent and thus applicable to all FrameNet-style databases following the latest FrameNet data model.

1 Introduction

1.1 JFN project goals

The Japanese FrameNet (hereafter JFN) project (Ohara et al., 2003; Ohara et al., 2004) aims at creating a human and machine-readable lexical database of Japanese, supported by corpus evidence and annotated in accordance with the Frame Semantics framework (Fillmore, 1982). *Frames* - schematic representations of events, relations or entities - provide semantic background for interpreting the meaning of words and illustrating their syntactic valence. The output of the Japanese FrameNet project takes the form of a database of corpus-extracted and annotated sentences, frame-evoking words called *lexical units* (LU)[1] and frame definitions including frame-dependent semantic roles called *frame elements*. The following example illustrates the `Arriving` frame, evoked by the lexical unit *tsuku*.v ("to arrive"), and the corresponding frame elements TIME, THEME and GOAL:

$[_{\text{TIME}}$ mamonaku]	$[_{\text{THEME}}$ watashitachi	wa]		
soon	we	TOP[2]		
$[_{\text{GOAL}}$ meripitto	sō	e]	^{Target}**tsuita**.	
Merripit	House	to	arrived	

"Soon we arrived at Merripit House"

1.2 Historical overview and current status

The JFN project started in July 2002, following the approach of the Berkeley FrameNet (hereafter FN) project (Baker et al., 1998), whose latest data release was in September 2010[3]. An official JFN data release is scheduled for March 2017 and sample data have been made available through FrameSQL (Sato, 2003). Table 1 shows some important figures of both FN and JFN projects.

	FN 1.5	**JFN**
Frames	1019	990
(non-lexical)	111	385
Frame Elements	8,884	8,583
Frame Relations	1,507	980
FE Relations	8,252	4,024
Lexical Units	11,829	3,401
Annotation Sets	173,018	73,372

Table 1: FN data-release 1.5 and JFN data approximation as of March 2014

1.3 Importance of data quality checks

Maintaining good JFN data quality is of crucial importance to both JFN potential users and JFN

[1] Or words paired with meaning, one specific meaning corresponding to a given frame in Frame Semantics.

[2] Topic marker.

[3] `https://framenet.icsi.berkeley.edu/fndrupal/framenet_data`

lexicographers. As detailed in Section 2.1, the JFN annotation process is based on and begins with examining the FN data, data which may be modified anytime, partially or fully.

Also, the diversity of annotation tools used by the JFN project (Saito et al., 2008) and the complexity of the JFN annotation process itself make it somewhat difficult to systematically and fully prevent unintentional entries of incomplete, inconsistent or redundant data (hereafter referred to as 'corrupted[4]') into the JFN database.

In this paper, we detail a protocol to automatically detect, analyze, and deal with such incomplete, inconsistent and redundant data entries in order to maintain high-quality data for both end-users and JFN lexicographers. The protocol defined here is generic and not language-specific, and can therefore be used by any FN-related project relying on the latest FN data model[5].

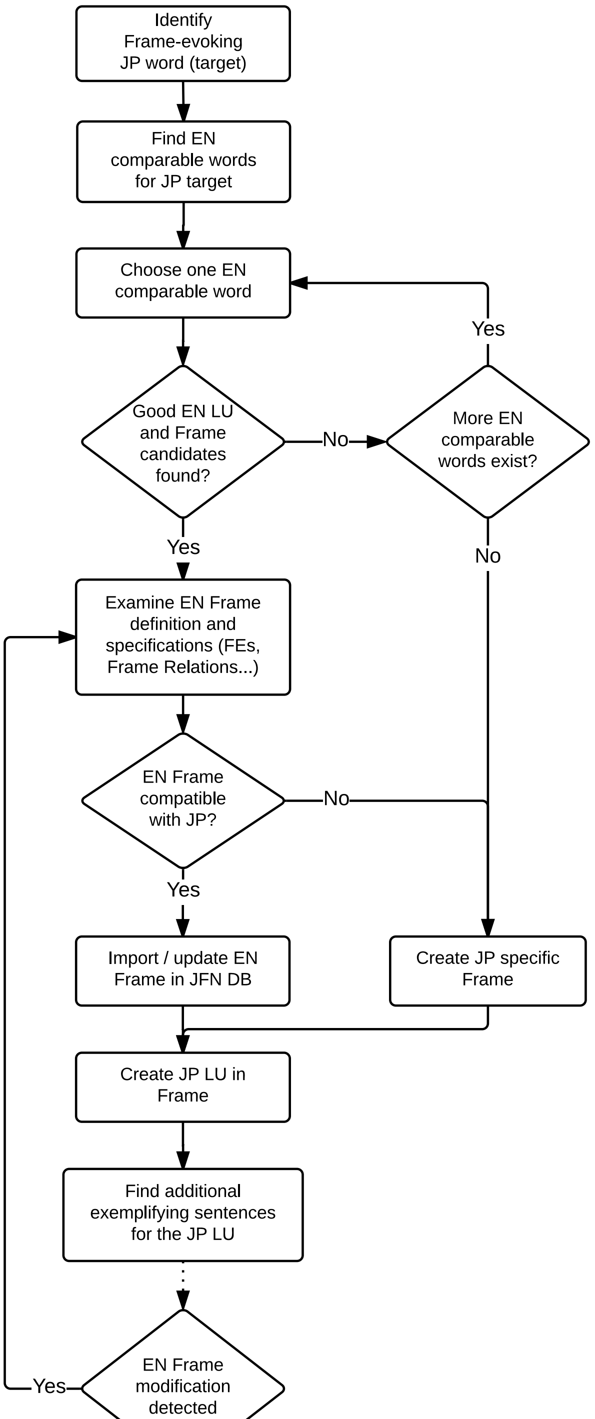
This paper is structured as follows: in Section 2, we will describe the JFN annotation process and the JFN database structure. In Section 3, we will define the concept of data quality applied to JFN. In Section 4, we will detail our methodology for performing data quality checks on the JFN database. In Section 5 we will give the results of our data quality checks. In Section 6, we will detail comprehensive solutions to enforce JFN data integrity. In Section 7 we will discuss future work and finally in Section 8 we will conclude the paper.

2 Overview

2.1 JFN annotation process

The Japanese FrameNet annotation process relies on the "Expand" approach for multilingual resource creation (Lönneker-Rodman and Baker, 2009): frames for ordinary English[6] and frame specifications (frame elements, frame relations, frame element relations), which may be taken as candidates for language-independent frames to a fair degree, are imported in the JFN database and populated with Japanese LUs.

Figure 1 illustrates the JFN LU definition process, focusing on frame identification. When creating a new Japanese LU, JFN annotators first look for comparable English words in order to assign the Japanese LU to an existing frame in the FN database which seems to evoke the Japanese LU. If no good candidates for an appropriate frame are found, JFN annotators may decide to create Japanese-specific frames or to add Japanese-specific frame specifications, such as frame elements (Ohara et al., 2004).



Identify Frame-evoking JP word (target)
Find EN comparable words for JP target
Choose one EN comparable word
Good EN LU and Frame candidates found?
No
More EN comparable words exist?
Yes
Yes
No
Examine EN Frame definition and specifications (FEs, Frame Relations...)
EN Frame compatible with JP?
No
Yes
Import / update EN Frame in JFN DB
Create JP specific Frame
Create JP LU in Frame
Find additional exemplifying sentences for the JP LU
EN Frame modification detected
Yes

Figure 1: JFN Lexical Unit definition process

[4]Data corruption is used as a standard database management terminology and is to be understood as the loss of integrity, such as consistency, completeness or uniqueness, as a result of some external agent.

[5]As of the FrameNetDesktop distribution - 2012/7/31.

[6]To be understood as frames defined by the English FrameNet team, not as English-language-specific frames.

As already documented by Boas (2009), the FN annotation process is an iterative process, where lexicographers constantly question previously annotated data with newly encountered corpus data. Frame compatibility between English and Japanese, defined for language-independent frames, is assumed at first at the time of Japanese LU creation, but any *a posteriori* modification to frame boundaries or specifications in FN or in JFN may "break" compatibility.

2.2 JFN database structure

2.2.1 Latest structure updates

The JFN database structure and data model are identical to those in FN (Baker et al., 2003). The latest FN database structure, released with the 2012 FrameNetDesktop annotation software distribution, implemented several modifications to the 2003 version: new tables were added for construction annotation (Fillmore et al., 2012), and some modifications were made to several tables structures and to the database table engine.

2.2.2 Structure overview

The JFN database is conceptually divided into two sub-databases: the *lexical* database, covering all components relevant to Frame-Semantic annotation, and the *annotation* database, covering corpus-extracted sentences annotated with the lexical database components. All upper-case words in the following subsection paragraphs refer to database tables.

The lexical database The core table of the lexical database is the Lexical Unit, defined as the association between a Lemma and a Frame. A Lemma is a set of one or more Lexeme references, and a Lexeme itself is the set of all WordForm references of a given word, i.e. of all its inflections. Each Lexeme and each Lemma refer to a specific Part of Speech, and a Lexeme Entry table offers the possibility to connect Lexeme to Lemma in a many-to-many relationship, covering thereby multi-word expressions, designed as multi-lexeme lemmas in the FrameNet database. Annotators can check the Status of a given Lexical Unit, referring to a list of pre-defined Status Type references which identify stages of the Lexical Unit annotation process. A Frame may be conceptually related to another Frame, which is formalized in the database through Frame Relation references, connecting two Frame references by specifying their Relation Type. In a similar fashion, a Frame Element may be conceptually related to another Frame Element, and a Frame Element Relation is formalized via Frame Element Relation Type references.

The annotation database The AnnotationSet Table formally connects both lexical and annotation databases. Each AnnotationSet links a Lexical Unit and/or a Construction to a given Sentence, which belongs to a Paragraph, itself referring to a Document included in a Corpus. Lexical Unit and Construction exemplifying annotated sentences are grouped into SubCorpus entities. Each AnnotationSet is composed of a set of Layer references, each referring to a unique Layer Type, such as grammatical function, frame element, phrase type, and postposition. For each Layer, a set of Label references of specific Label Type will indicate for each Sentence chunk relevant syntactic and semantic features.

3 Defining JFN data quality

The specific goals of quality checks are to verify that JFN data:

- comply with the specifications of the underlying relational data model;
- are usable with the JFNDesktop annotation software.

We do not address the question of annotation consistency between annotators, as annotation decisions in JFN are always reviewed and approved collectively. Neither do we address the question of "data accuracy", with respect to mis-specifications of lexical units, frames, or frame elements in Japanese. Our JFN-FN database comparisons are performed at the frame, frame relation, frame element, and frame element relation levels. We do not question the relevance and possible matching of language-specific frames in Japanese and English.

Quality checks are divided into two categories: "JFN-internal database checks" (verifications at the JFN database level only); and "JFN-FN cross database checks" (verifications comparing specific tables of the JFN and FN databases). Additionally, the "JFN-internal database checks" category is divided into two sub-categories, namely 'low-level checks" and "high-level checks".

3.1 JFN-internal database checks

3.1.1 Low-level checks

Low-level JFN-internal database checks are data checks at the table level, which involves checking for data *completeness*, *redundancy* and *consistency*.

Completeness Incomplete data are mostly data containing problematic NULL column values which render them unusable in the JFNDesktop annotation software.

Redundancy Redundant data are data which are treated as redundant by the JFNDesktop annotation software.

Consistency Inconsistent data are data with broken references. For instance, a frame element referring to the ID of a deleted frame.

3.1.2 High-level checks

High-level JFN-internal database checks rely extensively on the work of Scheffczyk and Ellsworth (2006). They involve multi-table complex constraint checks and checks for data which:

- comply with the data model;
- may be usable by the JFNDesktop annotation software.

(but) should nonetheless be excluded from public data releases.

Multi-table Complex multi-table checks involve checking constraints that cover multiple tables of the database. They include:

- conflicting part of speech specifications between lemmas and corresponding lexemes;
- discrepancies between parent and child frame elements.

Data-release exclusion Data which should not be included in public releases such as:

- unused frames (frames in which no Japanese LU is defined);
- non-sensical frames (frames without frame elements);
- un-annotated corpus / documents.

3.2 JFN-FN cross database checks

JFN-FN cross database checks are limited to (potentially) shared parts of the FrameNet database, i.e. frames, frame relations, frame elements and frame element relations. They consist of:

- Frame-to-Frame comparisons (checking for JFN and FN frames having the same Name but conflicting Definition column values and checking for unique[7] FN frames and unique JFN frames);
- FrameElement-to-FrameElement comparisons (checking for JFN and FN frames elements having the same Name but conflicting specifications, such as Definition, Abbreviation, SemanticRoleRank, Type and Core column values, and checking for unique FN and JFN frame elements);
- FrameRelation-to-FrameRelation comparisons (checking for unique FN and JFN frame relations).

4 Methodology

4.1 The JFN CheckUp API

The JFN CheckUp API is designed as a set of java methods used to perform all data integrity checking and fixing operations on the JFN database. It follows the Data Access Object design pattern and relies on the Spring JDBC framework for manipulating SQL queries on the JFN MySQL database.

4.2 JFN-internal database checks details

126 verification operations were carried out on 35 tables of the JFN database. Functional operations do not necessarily correspond to single java methods, for either performance or modularity reasons. Table 2 shows the number of JFN-internal database checks we performed for each data quality check category.

JFN-internal database checks		Nb.
Low-level	Completeness	47
	Redundancy	35
	Consistency	34
High-level	Complex multi-table	5
	Data-release exclusion	5

Table 2: JFN-internal database checks count

[7]A unique frame is a frame which exists in one database but not in the other.

4.3 JFN-FN cross database checks details

For all JFN-FN cross database comparisons, we assume that Frame and FrameElement table rows which share identical Name column values across FN and JFN databases should share the same characteristics. The JFN CheckUp API is designed to operate on two distinct MySQL databases at most:

- the *native* database, which is the focus of the data integrity checks, in our case the JFN database;
- the *foreign* database, which the *native* database is being compared to, in our case the FN database.

4.4 JFN-internal database check example

In order to better explain what kind of internal database checks were performed on the JFN database, we illustrate in Table 3 the 10 separate operations that were carried on the AnnotationSet table in order to fully check its integrity. The structure of the AnnotationSet table is given below:

```
CREATE TABLE AnnotationSet(
  ID mediumint(8) unsigned NOT NULL
      AUTO_INCREMENT,
  LexUnit_Ref mediumint(8),
  SubCorpus_Ref mediumint(8),
  Sentence_Ref mediumint(8),
  CreatedDate datetime,
  CreatedBy varchar(40),
  ModifiedDate timestamp NOT NULL,
  CurrentAnnoStatus_Ref tinyint(3),
  Construction_Ref mediumint(5),
  PRIMARY KEY (ID));
```

JFN-internal database AnnotationSet table checks	
Completeness	Null `Sentence_Ref`
	Null `CurrentAnnoStatus_Ref`
Redundancy	Duplicate AnnotationSets[8]
Consistency	N.E.[9] `LexUnit_Ref`
	N.E. `SubCorpus_Ref`
	N.E. `Sentence_Ref`
	N.E. `Construction_Ref`
	N.E. `CurrentAnnoStatus_Ref`
Complex multi-table	Null `LexUnit_Ref` and Non Null `SubCorpus_Ref`
	Null `SubCorpus_Ref` and Non Null `LexUnit_Ref`

Table 3: Full AnnotationSet table integrity checks

[8]Same LexUnit_Ref, SubCorpus_Ref, Sentence_Ref, Construction_Ref and CurrentAnnoStatus_Ref AnnotationSets.

[9]Non Existing.

5 Results

In this section we provide a broad quantification of JFN ‘corrupted” data, which were either incomplete, inconsistent or redundant, as well as a qualification of these data based on their potential impact on the annotation process and presence in future public xml data releases. Besides “corrupted” data *quantification* and “corrupted”/clean data ratio, “corruption” *spread* (the number of tables affected and the number of categories of data “corruption” affected among those tables) can be considered a pertinent indicator of the quality of the data. Each “corrupted” entity indicates a *breach* in the upstream data quality prevention process, breaches that are nowadays preventable directly through database constraints in MySQL (see Section 6.2). Not only does a “corrupted” entity indicate that a specific type of data “corruption” is *possible*, it also implies that data “corruption” *actually occurs* during a specific step of the annotation process, which is valuable information to JFN system administrators for preventing such data “corruption” from happening.

5.1 Incomplete, redundant or inconsistent data quantification

5.1.1 JFN-internal database checks

Table 4 shows, for each JFN database table containing “corrupted” entities, the number of “corrupted” rows and the number of total rows in the given table.

JFN tables	**Corrupted**	**Total**	**%**
AnnotationSet	30,848	73,372	42.0
Corpus	4	37	10.8
Document	283	3,085	9.2
Frame	7	990	0.7
FrameElement	160	8,583	1.9
Label	105	1,603,999	ε
LabelType	21	9,014	0.2
Layer	148	200,909	0.1
Lemma	53	6,499	0.8
Lexeme	133,214	157,996	84.3
LexUnit	2	3,401	0.1
LexemeEntry	1	9,025	ε
Paragraph	54	19,319	0.3
Sentence	12	34,356	ε

Table 4: JFN-internal database checks results

5.1.2 JFN-FN cross database checks

Here we limit our discussion of the results to those at the frame level, given in Table 5:

JFN-FN cross database Frame table checks	
Unique FN frames	253
Unique JFN frames	61
Conflicting definition frames	569

Table 5: JFN-FN cross database checks results

It is important to emphasize that, as we rely on strict string comparison across FN and JFN frames definitions, even minor string differences would result in conflicting frame definitions. It turned out, however, that the vast majority of identified conflicting frame definitions did not call into question the existing JFN annotated data.

5.2 Incomplete, redundant and inconsistent data qualification

Incomplete, redundant or inconsistent data are qualified in terms of *criticality*:

- **highly critical:** incomplete, redundant, or inconsistent entities may potentially be passed on to future publicly-released data;
- **critical:** incomplete, redundant, or inconsistent entities may lead to bugs in annotation tools, or may be unusable for annotation purposes;
- **neutral:** incomplete, redundant, or inconsistent entities have no impact on the annotated data or annotation process and remain in the database as "ghost" entities, neither visible via annotation tools nor released, mostly taking up space in the database.

5.3 Data "corruption" spread

Table 6 shows the number of tables of the JFN database containing highly-critical, critical or neutral "corrupted" data.

Criticality level	**Nb. of tables**
Highly-critical	1
Critical	3
Neutral	10
Total corrupted tables	14
Total tables	35

Table 6: JFN data corruption criticality

Table 7 shows, for each criticality level, the data "corruption" spread, defined as the NCA/$_{total}DCC$ ratio. **DCC** are data "corruption" categories, to be understood as the number of JFN-internal database checks performed on a given table. **NCA** is the number of DCC containing "corrupted" data. The **total DCC**, i.e. the number of JFN-internal database checks performed on the 35 tables of the JFN database, is 126.

Criticality	**Tables**	**NCA**	**DCC**	**Spread**
HC	1	1	10	0.8
HC+C	4	7	25	5.6
HC+C+N	14	21	69	16.7

Table 7: JFN data corruption spread

Of the 35 tables that compose the JFN database, only 4 contained critical or highly-critical "corrupted" data. Those 4 tables cover 7 data "corruption" categories out of 126. Our data "corruption" spread is about **5.6%**. In other words, critically "corrupted" data are very much localized: they cover only few tables and few "corruption" categories within the JFN database.

6 Solutions to deal with the affected data

In this section we report on two solutions we applied to deal with the JFN data found by the checkups to be either incomplete, redundant or inconsistent. First, we present a straightforward solution used to remove all such entities from the database, without modifying the JFN data model and therefore minimizing necessary rework of the JFN annotation tools. Second, we provide a more long term solution by listing recommendations for designing a more constrained data model, thereby aiming at performing upstream data "corruption" prevention rather than *a posteriori* quality checks. By placing additional constraints directly onto the data model, data robustness will be enforced regardless of database usage.

6.1 Fixing affected data

The following sub-sections discuss semi-automated tasks we used to deal with affected data, relying on the JFN CheckUp API.

6.1.1 JFN-internal database fixes

Incomplete and inconsistent data were removed from the database. Redundant data were merged, i.e. references to duplicates were updated to point at the original rows, after which duplicates were removed. More complex constraint-breaking entities were dealt with on a case-by-case basis.

6.1.2 JFN-FN cross database fixes

Table 8 presents processes to be applied to the JFN database in order to fix cross-database inconsistencies as defined in Section 3.2. For dealing with JFN-FN cross database fixes, a distinction was made between used and unused JFN frames[10], as the JFN CheckUp API is not yet able to perform automated processing of used *native* (here JFN) frames (see Section 7).

JFN-FN data sync case	Process
Unique FN frames	add to JFN
Unique used JFN frames	case-by-case
Unique unused JFN frames	remove from JFN
Conflicting definition frames	case-by-case
Unique FN frame elements	add to JFN
Unique used JFN frame elements	case-by-case
Unique unused JFN frame elements	remove from JFN
Unique FN frame relations	add to JFN
Unique JFN frame relations for used JFN frames	case-by-case
Unique JFN frame relations for unused JFN frames	remove from JFN

Table 8: JFN-FN data synchronization process

6.2 Toward a new JFN data model

The FrameNetDesktop distribution of 2012/7/31, on which the latest version of the JFNDesktop annotation software is based, introduced several changes to the standard FrameNet database structure. The database engine was upgraded from MyISAM to InnoDB, making thereby possible the implementation of several additional MySQL features[11]. Relying on these modifications, we propose a slightly enhanced data model whose specifications prevent incomplete, redundant and inconsistent data from being inserted into the database.

[10]"Unused frame" refers to a frame in which no Japanese LU is defined, while the term "used frame" refers to a frame in which one or more Japanese LU is defined.

[11]Such as row-level locking and foreign-key constraints.

6.2.1 Completeness and data constraints

Incomplete data prevention can be performed by disallowing NULL values to specific table columns. For more complex constraints, such as NULL values tolerated only under specific conditions, see Section 6.2.4.

6.2.2 Redundancy and unique indexes

Redundant data prevention can be performed by adding unique indexes on sub-sets of table columns.

6.2.3 Consistency and foreign keys

Inconsistent data prevention can be performed by implementing identifying relationships via foreign keys on all cross table references. This also makes possible cascade deletes, avoiding thereby additional work at the software level to take care of complete deletions of deleted references.

6.2.4 Complex constraints and triggers

More complex constraints on specific multi-table and multi-column combinations can be enforced through the use of triggers on insert made available since MySQL 5.5.

7 Discussion

As we described in section 6.1.2, frames previously imported from the FN database, augmented with LUs in the JFN database and updated afterwards in the FN database have to be examined carefully and on a case-by-case basis. Since such operations are only necessary when an official FN data release becomes available, which has happened only four times since 1997, we believe that manual case-by-case processing of the JFN-FN cross database checks is acceptable, at least for the time being. This process, however, is time-consuming. Therefore, work is in progress to automatically suggest replacement candidates for deleted or updated[12] FN frames, which could speed up the JFN-FN synchronization process.

8 Conclusion

Although maintaining high data quality in the Japanese FrameNet is a complex task, we were able to design and to perform extensive data quality checks and removed all incomplete, redundant

[12]With modification making them incompatible for Japanese annotation.

and inconsistent data from the JFN database. Additionally, we were able to fully automate JFN-internal database quality checks and fixes. JFN-FN cross database quality checks can also be performed automatically, but cross database fixes still have to be processed manually and on a case-by-case basis. Since JFN-FN cross database checks are used only when an official FN data release becomes available, we believe that manual case-by-case processing is acceptable, at least for the time being. The JFN CheckUp API we implemented to perform data quality checks and fixes on the JFN database is generic, and should be applicable to other FN-related projects relying on the same data model. On more long-term perspectives, we are updating the JFN data model and all JFN annotation tools, in order to enforce upstream data quality constraints and to reduce the number of internal database quality checks.

Acknowledgments

The Japanese FrameNet project is supported in part by Grant-in-Aid for Scientific Research (c) #24520437, Grant-in-Aid for Publication of Scientific Research Results (Databases), and Grant from the National Institute of Japanese Language and Linguistics.

References

Collin F. Baker, Charles J. Fillmore, and John B. Lowe. 1998. The Berkeley FrameNet Project. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics, Volume 1*, pages 86–90, Montreal, Quebec, Canada, August. Association for Computational Linguistics.

Collin F. Baker, Charles J. Fillmore, and Beau Cronin. 2003. The Structure of the Framenet Database. *International Journal of Lexicography*, 16(3):281–296.

Hans C Boas. 2009. *Multilingual FrameNets in computational lexicography: Methods and applications*, volume 200. Walter de Gruyter.

Charles J Fillmore, Russell Lee-Goldman, and Russell Rhodes. 2012. The FrameNet Constructicon. *Sign-based Construction Grammar. CSLI, Stanford, CA*.

Charles J. Fillmore, 1982. *Frame semantics*, pages 111–137. Hanshin Publishing Co., Seoul, South Korea.

Birte Lönneker-Rodman and Collin F Baker. 2009. The FrameNet model and its applications. *Natural Language Engineering*, 15:415–453, 7.

Kyoko Hirose Ohara, Seiko Fujii, Hiroaki Saito, Shun Ishizaki, Toshio Ohori, and Ryoko Suzuki. 2003. The Japanese FrameNet project: A preliminary report. In *Proceedings of PACLING03*, pages 249–254.

Kyoko Hirose Ohara, Seiko Fujii, Toshio Ohori, Ryoko Suzuki, Hiroaki Saito, and Shun Ishizaki. 2004. The Japanese Framenet project: An introduction. In *LREC 2004. Proceedings of the Satellite Workshop“Building Lexical Resources from Semantically Annotated Corpora*, pages 9–11, Lisbon, Portugal.

Hiroaki Saito, Shunta Kuboya, Takaaki Sone, Hayato Tagami, and Kyoko Ohara. 2008. The Japanese FrameNet Software Tools. In *LREC2008. Proceedings of the 6th International Conference on Language Resources and Evaluation*.

Hiroaki Sato. 2003. FrameSQL: A software tool for FrameNet. *ASIALEX03 Tokyo Proceedings*, 25:1–25.

Jan Scheffczyk and Michael Ellsworth. 2006. Improving the Quality of FrameNet. In *LREC2006. Proceedings of the Workshop on Quality assurance and quality measurement for language and speech resources*, pages 8–13, Genoa, Italy.