Noname manuscript No.
(will be inserted by the editor)

Enriching a Massively Multilingual Database of Interlinear Glossed Text

Fei Xia · William D. Lewis · Michael Wayne Goodman · Glenn Slayden · Ryan Georgi · Joshua Crowgey · Emily M. Bender

Received: date / Accepted: date

Abstract The majority of the world's languages have little to no NLP resources or tools. This is due to a lack of training data ("resources") over which tools, such as taggers or parsers, can be trained. In recent years, there have been increasing efforts to apply NLP methods to a much broader swath of the world's languages. In many cases this involves bootstrapping the learning process with enriched or partially enriched resources. We propose that Interlinear Glossed Text (IGT), a very common form of annotated data used in the field of linguistics, has great potential for bootstrapping NLP tools for resource-poor languages. Although IGT is generally very richly annotated, and can be enriched even further (e.g., through structural projection), much of the content is not easily consumable by machines since it remains "trapped" in linguistic

This material is partly supported by the National Science Foundation under Grant No. BCS-1160274 and BCS-0748919, and Singapore Ministry of Education under Tier 2 Grant No. MOE2013-T2-1-016. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation. We would like to thank Sebastian Nordhoff for discussion on the Xigt format and issues with the original IGT data, and anonymous reviewers for helpful comments. We would also like to thank the Linguist List (`http://linguistlist.org/`) for hosting the ODIN database.

Fei Xia
University of Washington
PO Box 352425, Seattle, WA 98195, USA
Tel: 206-543-9764
Fax: 206-685-7978
E-mail: fxia@uw.edu

William D. Lewis
Microsoft Research
One Microsoft Way, Redmond, WA 98052, USA

Michael Wayne Goodman, Glenn Slayden, Ryan Georgi, Joshua Crowgey, and Emily M. Bender
University of Washington
PO Box 352425, Seattle, WA 98195, USA

scholarly documents and in human readable form. In this paper, we describe the expansion of the ODIN resource—a database containing many thousands of instances of IGT for over a thousand languages. We enrich the original IGT data by adding word alignment and syntactic structure. To make the data in ODIN more readily consumable by tool developers and NLP researchers, we adopt and extend a new XML format for IGT, called Xigt. We also develop two packages for manipulating IGT data: one, *INTENT*, enriches raw IGT automatically, and the other, *XigtEdit*, is a graphical IGT editor.

Keywords Resource-poor languages · Interlinear glossed text · ODIN

1 Introduction

Of the world's 7,000+ spoken languages, only a very small fraction have text resources substantial enough to allow for the training of NLP tools, such as part-of-speech (POS) taggers and parsers. Developing enriched resources, e.g., treebanks and POS-tagged corpora, which allow supervised training of such tools, is expensive and time-consuming. In recent years, work has been done to bootstrap the development of such resources for resource-poor languages by tapping the enriched content of a better resourced language, and, through some form of alignment, "projecting" annotations onto data for the resource-poor language. Some studies have focused on the typological similarity of languages, using cognates and similar word forms in typologically similar languages, to bridge between languages and to build tools and resources (Hana et al, 2006; Feldman et al, 2006). Other work has relied on parallel corpora and bitexts, where one language of a corpus is highly resourced, and annotations are projected onto the lesser resourced language(s) (Yarowsky and Ngai, 2001; Hwa et al, 2005). A challenge to this approach is that there might not be a large enough supply of bitexts for both languages to allow for the training of a high-quality statistical word aligner.

Recent work has been done on transferring dependency structures (Täckström et al, 2013; McDonald et al, 2013; Xiao and Guo, 2015) with promising results; however, these methods rely on delexicalized approaches where POS tags are known for both languages. Other studies, such as Das and Petrov (2011); Täckström et al (2012); Ma and Xia (2014), do not rely upon POS tags being available to perform transfer between languages, but still require a large amount of parallel data to achieve good results.

In our previous studies (Georgi et al, 2013, 2014), we proposed to use linguistically annotated data, specifically Interlinear Glossed Text (IGT), to project annotations from a highly resourced language to one or more under-resourced languages, potentially hundreds at a time. Since IGT is a data format commonly used in the field of linguistics, and because linguists study thousands of the world's languages, the possibility exists to build resources for a sizable percentage of the world's languages. The problem is that IGT is typically locked away in scholarly linguistic papers, and not easily accessible to NLP researchers who might otherwise want access to the data. The Online

Database of Interlinear text (ODIN) (Lewis and Xia, 2010), a database of over 200,000 instances of IGT for more than 1,500 languages, tackles the issue of extracting IGT from scholarly resources, but focuses more on presenting the captured content for human consumption and query, rather than for subsequent automated consumption. By taking the content of ODIN, enriching it (e.g., through projected annotations), and reformatting it into a machine readable form, enriched IGT becomes a much more useful resource for bootstrapping NLP tools. IGT is particularly of interest for projection in that it is a resource that is readily available for languages without comprehensive parallel corpora, and the f-scores for word alignment on IGT instances can be as high as 94.0% without requiring large parallel corpora (Lewis and Xia, 2010).

In this paper, we begin with an overview of ODIN in Section 2. Next, we describe, in Section 3, the process of enriching the IGT data in ODIN and demonstrate that the enriched IGT data can be used to bootstrap NLP tools such as parsers. In Section 4, we review Xigt, a new XML format for representing enriched IGT data, and discuss the extensions we have made to Xigt to facilitate the manipulation of the enriched IGT data. Finally, in Sections 5 and 6, we introduce two packages that we have developed for processing IGT: the first one, *INTENT*, enriches raw IGT automatically, and the second one, *XigtEdit*, is a graphic editor that the annotators can use to edit enrich IGT in the Xigt format. The ODIN data (including IGT in both its original and enriched forms) and all the packages for processing IGT data are available to the public.[1] By making these tools, and the resulting data, available to the NLP community, we open the door to a much wider panoply of the world's languages for NLP research.

2 Building ODIN

IGT is a common format that linguists use to present language data relevant to a particular analysis. It is most commonly presented in a three-line form, a sample of which is shown in Ex. (1). The first line, the *language line*, presents data from the language in question, and is either phonetically encoded or transcribed in the language's native orthography or a transliteration thereof. The second line, the *gloss line*, contains a morpheme-by-morpheme or word-by-word gloss for the data on the language line. The third line, the *translation line*, contains a translation of the first line, often into a resource-rich language such as English. There could be additional information in IGT such as citations and language names. In Ex. (1), Bailyn (2001) is the source of the IGT instance, and *cym* is the ISO 639-3 language code for Welsh.

(1)

Rhoddodd	yr	athro	lyfr	i'r	bachgen	ddoe
gave-3sg	the	teacher	book	to-the	boy	yesterday

"The teacher gave a book to the boy yesterday" (Bailyn, 2001) [cym]

[1] `http://depts.washington.edu/uwcl/packages/`

The mechanical processes for constructing a database of interlinear text such as ODIN were first described in (Lewis, 2003). Lewis (2003) observed that IGT is a rich source of linguistic markup, and a collection of harvested IGT could be treated as a gateway to the construction of a resource representing the conceptual space of the field of linguistics, such as an ontology of linguistic concepts (Lewis et al, 2001; Farrar and Langendoen, 2003). It quickly became clear that the database of IGT itself was directly of use to the field of linguistics, in addition to secondary resources like the ontologies derived from it (Xia and Lewis, 2008; Lewis and Xia, 2008a). The ODIN database was created in two stages: automatic construction, followed by manual correction.

2.1 Automatic construction

The automatic construction stage has three steps. First, we crawl the Web for linguistic documents and collect those documents that most likely contain IGT. This is done by throwing queries against an existing search engine, extracting the relevant URLs from the results of the queries, crawling the pages returned (i.e., searching returned pages for relevant URLs), and downloading the pages and documents that contain IGT. Good queries usually use strings contained within IGT itself. For instance, the gloss line in IGT often contains *grams*[2] ("grammatical morphemes", e.g., NOM, ACC, ERG, etc.), and the most successful strategy involves using the highest frequency grams as search terms.

Second, IGT within the discovered documents is detected and extracted. We treat IGT detection as a sequence labeling problem, and apply machine learning methods to the task: first, we train a learner and use it to tag each line in a document with a BIO tag, and then we convert the best tag sequence into a span sequence. The feature set includes word n-grams, shapes of a line (e.g., whether the line starts with an example number), and other cues for the presence of IGT. When trained on 41 documents containing 1573 IGT instances, which we subsequently tested on 10 documents, the f-scores for exact and partial span match on the test data are 81.7% and 96.8% respectively (Xia and Lewis, 2008).

Third, each extracted IGT instance is assigned a language name and a language code. While existing methods for language ID perform very well in a typical language ID setting (e.g., ID across a small set of languages), they all require training data in the relevant languages in order to build a language model or a character n-gram list. They do not work well in our setting because the number of languages represented by IGT on the Web is in the thousands and, likewise, for many of these languages we have no training data. To address this challenge, we proposed to treat language identification as a coreference

[2] The first attested use of the word *gram* in the context of IGT that we are aware of is in (Bybee and Dahl, 1989). Bybee and Dahl used the term *gram* to refer to morphemes that have grammatical functions in a language. Here, and elsewhere, we use the term *gram* to refer to the annotation used by linguists to refer to these morphemes.

Range of IGT instances	# of languages		# of IGT instances		% of IGT instances	
> 10000	3	(1)	36,691	(10,814)	19.39	(6.88)
1000-9999	37	(31)	97,158	(81,218)	51.34	(51.69)
100-999	122	(139)	40,260	(46,420)	21.27	(29.55)
10-99	326	(460)	12,822	(15,650)	6.78	(9.96)
1-9	838	(862)	2,313	(3,012)	1.22	(1.92)
total	1326	(1,493)	189,244	(157,114)	100	(100)

Table 1 The language distribution of IGT instances in ODIN after stage 1 and (stage 2). Stage 1 is the automatic construction of IGT instances extracted from 2868 documents, with language IDs assigned by the language ID system. Stage 2 is the manual correction of IGT instances extracted from 2025 documents, with language IDs selected by human correction of the language ID system output.

resolution task, where an IGT instance is linked to a language name that appears in the same document. When trained on 1372 IGT instances from 125 languages and tested on 1516 instances (only 55.5% of which belong to a language that appears in the training set), the accuracy is 83.1%, much higher than the 55.5%, the upper bound of any language ID algorithm (including the standard n-gram based algorithm) that relies on having training data for the languages that the test data belong to (Xia et al, 2009).

We ran the IGT detection and language ID systems on three thousand IGT-bearing documents crawled from the Web and the extracted IGTs were stored in the ODIN database. Table 1 shows the language distribution of the IGT instances in the database according to the output of the language ID system. For instance, the third row says that 122 languages each have 100 to 999 IGT instances, and the 40,260 instances in this bin account for 21.3% of all IGT in the ODIN database.

2.2 Manual correction

To ensure the high quality of the ODIN database, we manually corrected the output of Steps 2 and 3 of the automatic construction stage. This was done in three steps.

First, the annotators corrected the boundary of IGT instances found by the IGT detection module. In addition, they labeled each line in each IGT instance with a *xx-yy* tag. The *xx* part is the main tag, indicating whether the line is a language line (*L*), a gloss line (*G*), a translation line (*T*), a blank line (*B*), or a line with other information such as the citation or linguistic construction name (*M*). The *yy* part is called the secondary tag; it provides additional properties of the line; for instance, *CR* means that the current line was corrupted when the document retrieved by the crawler was converted from a PDF file to a text file by an off-the-shelf PDF-to-text converter. The tags can be used for automatic enrichment of IGT instances, as discussed in the next section.

Our automatic language ID module labeled each IGT instance with a language name and an ISO 639-3 language code. The annotators corrected the language name in a second pass through the data and the language code in a third pass. The reason that we separated the two processes is that the mapping from language names to language codes is many-to-many and must be done by a linguist who can choose the correct language code for an ambiguous language name. See (Xia et al, 2010) for more information about these correction processes.

We have finished manual correction of more than 83% of the IGT instances in ODIN. The language distribution of this subset of the data is in parentheses in Table 1. Notice that the number of languages in this subset is higher than the number of languages in stage 1. That is because our automatic language ID module maps an ambiguous language name to the most common language code associated with the name. This process is error-prone; manual correction reveals that the ODIN data actually covers more languages than indicated by the automatic construction stage.

Although the canonical form of an IGT instance includes a language line (L), a gloss line (G), and a translation line (T), as noted earlier, linguists often do not follow this canonical form, especially if they show multiple IGT instances in a group. For instance, an IGT instance might include only the L line, because the line has slightly different word order from the language line in a previous instance, and readers could infer what the gloss and translation lines should be from the previous IGT instance. Other non-canonical instances might be due to the extraction process failing to capture all lines; for instance, if a language line contains non-ASCII characters, the line may be jumbled up by the off-the-shelf PDF-to-text converter and consequently is not included as part of IGT. Table 2 gives a breakdown of the number of IGTs by the presence of L, G, T lines. The *Other types* category includes cases such as lines tagged as L-G, which means L and G are displayed side-by-side on the same line. The table shows that only 74.92% of IGT are in the canonical form. For the rest, additional work is required to recover the *"missing"* lines from the context.

Lines in an IGT	# of IGT instances	% of IGT instances
L, G, and T	117,717	74.92
L and G	19,750	12.57
L and T	7,912	5.04
G and T	469	0.30
L only	749	0.48
G only	611	0.39
T only	155	0.10
Other types	9,751	6.21
Total	157,114	100

Table 2 The IGT type distribution in ODIN after stage 2 (manual correction).

2.3 The Initial release of the ODIN Database

The initial release of the ODIN database constructed using the steps described in this section was made available to the public in 2010 (Lewis and Xia, 2010);[3] this subset contains 130,351 instances of IGT across 1,274 languages. The release includes the original IGT data in a plain text format, as shown in Fig. 1, reflecting the information as extracted from the source documents and the language that the IGT belongs to. The first line shows the document ID, the position of the IGT in the document (in this example, in lines 959-961), and the type of each line in the IGT. The second line gives the language name and language code. The next three lines are the original text from the document, annotated with the original line number and assigned tag.

```
doc_id=397 959 961 L G T
language: Korean (kor)
line=959 tag=L: (1) Nay-ka ai-eykey pap-ul mek-i-ess-ta
line=960 tag=G:     I-Nom child-Dat rice-Acc eat-Caus-Pst-Dec
line=961 tag=T:     ‘I made the child eat rice.’
```

Fig. 1 An IGT example in plain text format

The work presented in the following sections of this paper expands on the initial corpus by analyzing and enriching the text data and encoding it in a more structured format.

3 Enriching IGT Data

The unique structure of IGT makes it an extremely rich source of information for resource-poor languages: Implicit in an IGT instance is not only a short bitext between that language and a language of wider communication (almost universally English, but instances of Spanish and German have been discovered as well), but also information encoded in the gloss line about the grammatical morphemes in the source language and word-by-word translations to lemmas of the translation language. Thus even small quantities of IGT could be used to bootstrap tools for resource-poor languages through structural projection (Yarowsky and Ngai, 2001; Xia and Lewis, 2007). However, bootstrapping tools often require the original IGT to be enriched, as explained in this section.

3.1 Cleaning and normalizing IGT instances

The process of collecting IGT from linguistic documents may introduce noise. For instance, ODIN uses an off-the-shelf converter to convert PDF documents into text format and the converter sometimes wrongly splits a language line

[3] http://odin.linguistlist.org

into two lines. One such an example is Fig. 2, where the language line is incorrectly split into two lines by the converter, as indicated by the `CR` (for "corruption") tag for lines 875 and 876.

```
doc_id=1482 874 878 M+AC+LN L+CR L+SY+CR G T+DB
language: Haitian (hat)
line=874 tag=M+AC+LN:  (25) Haitian CF (Lefebvre      1998:165)
line=875 tag=L+CR   :                           ak
line=876 tag=L+SY+CR:         Jani     pale            lii/j
line=877 tag=G      :        (John speak with          he)
line=878 tag=T+DB   :        (a) 'John speaks with   him', (b) 'John
      speaks with   himself'
```

Fig. 2 An IGT isntance with corrupted language line and embedded meta-information.

Furthermore, the raw IGT is often not in the three-line canonical form. For instance, an IGT often contains other information (indicated by the `M` primary tag for "miscellaneous") such as a language name, a citation, and so on. In Fig. 2, the first line contains the language name[4] (indicated by the `LN` secondary tag) and citation (the `AC` tag), the third line includes the coindexation symbols *i* and *i/j* (the `SY` tag stands for general syntactic markup), and the last line shows two possible translations of the sentence (the `DB` tag means "double" annotation).

The cleaning and normalization step aims at fixing errors that were introduced when IGT was extracted from the linguistic documents, separating out various fields in an IGT, normalizing each field, and storing the results in a uniform data structure. Fig. 3 shows the ideal resulting IGT after this step. Our current rule-based system is able to fix simple cases of line corruption where tokens align to whitespace, but the goal is to also cleanly separate the various kinds of meta-information. For example, in Fig. (3) the coindexation symbols *i* and *j* are removed from the language line and stored in a separate field, the wrongly split language lines are merged back together, and the author citation and language name metadata are split into separate fields. Some of these goals are relatively easy to achieve (e.g., detecting and separating out citations), but others are more difficult. For instance, determining whether *i* in *Jani* is a co-index or part of the word is not trivial. For cases that cannot be done automatically with the rule-based cleaner and normalizer, they can be handled manually by human annotators using *XigtEdit* (the editor described in Section 6).

3.2 Adding word alignment and syntactic structure

After we have cleaned the IGT, the next step is to add word alignment and syntactic structure. In our previous work (Xia and Lewis, 2007), we proposed an algorithm to leverage the structure of IGT to enrich it further. We do so

[4] *CF* in the language name stands for French-lexified creole.

```
doc_id=1482 874 878 M+LN M+AC L M+SY G T T+AL
language: Haitian (hat)
line=874 tag=M+LN : Haitian CF
line=874 tag=M+AC : (Lefebvre 1998:165)
line=875-876 tag=L: Jan     pale   ak        li
line=876 tag=M+SY : i                        i/j
line=877 tag=G    : John   speak  with      he
line=878 tag=T    : 'John speaks with him'
line=878 tag=T+AL : 'John speaks with himself'
```

Fig. 3 A recovery of the corrupted language line in Fig. 2 and ideal interpretation of embedded meta-information.

in three steps: (1) parse the English translation with an English parser, (2) align the language line and the English translation via the gloss line, and (3) project syntactic structures from English onto the language line. Given the IGT in Ex. (1), the algorithm will produce the word alignment in Fig. 4, the dependency structures in Fig. 5, and the phrase structures in Fig. 6.

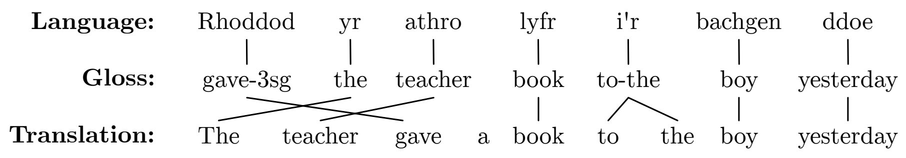


Fig. 4 Aligning the language line and the English translation with the help of the gloss line

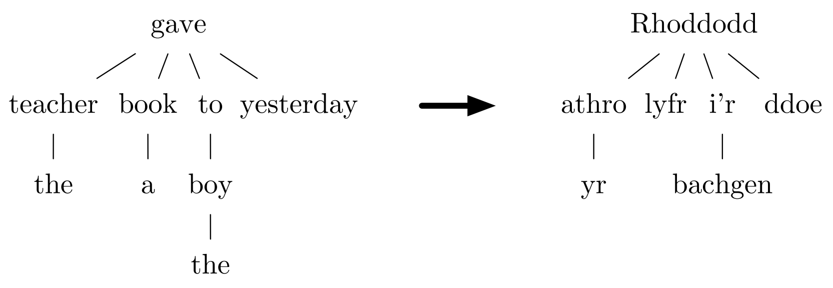


Fig. 5 Projecting dependency structure from the translation line to the language line

The structures produced by syntactic projection often are not perfect because the structures in the translation line and the language line may differ significantly. Dorr (1994) provides a detailed analysis of divergence in languages. In a previous study (Georgi et al, 2014), we investigated the use of small, parallel, annotated corpora to automatically detect divergent structural patterns between two languages. While this detection process was not exhaustive, we demonstrated that common patterns of divergence can be identified

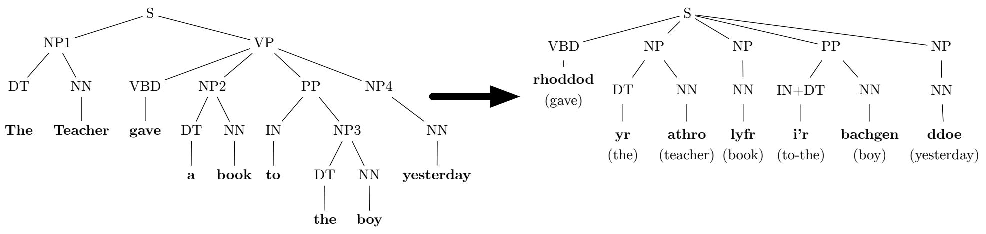


Fig. 6 Projecting phrase structure from the translation line to the language line

automatically without prior knowledge of a given language pair, and the patterns can be used to improve performance of syntactic projection and parsing. When tested on IGT instances from eight languages, applying automatically learned patterns improved projection accuracy from 83.2% to 89.0%.

3.3 Usability of enriched IGT

As a language sample, IGT from linguistics papers is subject to what Lewis and Xia (2008b) term 'IGT bias', overrepresenting structures of interest to linguists. Furthermore, the projected structures are subject to 'English bias' (*Ibid.*), stemming from the fact that they originate as structures for English. Errors made by the English parser and the word aligner add additional noise to the data. So it is reasonable to ask whether the enriched IGT is too noisy to be useful.

In our experience so far, the enriched IGT has been quite useful, across a variety of tasks: It allows linguists to search ODIN for linguistic constructions (e.g., passives, conditionals, double object structures) contained in the IGT instances. Enriched IGT also allows discovery of computationally relevant typological features, such as word or constituent order, or the presence or absence of particular grammatical features, and does so with high accuracy (Lewis and Xia, 2008a; Bender et al, 2013). Furthermore, enriched IGT can also be used to bootstrap NLP tools; for instance, adding features extracted from projected syntactic structures to a statistical parser provided a significant boost to parsing performance (Georgi et al, 2013).

4 Representing Enriched IGT in Xigt

The enriched ODIN database contains much useful information, including the original interlinear glossing, the translations, the added phrase structure annotations, and so on, but the data is not easily accessible in the plain-text format described in Section 2.3. The plain-text format is sufficient for representing the original IGT for reading by humans, but our goal is to enable automatic bulk processing of IGT, and for this we need a structured way to query the data and relationships present in each IGT. For instance, if we have a stored parse tree over the source language words and we want to know what gloss grams annotate the words covered by a node in the parse tree, with the plain-text format we would have to split up the strings and count tokens.[5] Instead of placing ourselves in this situation, we store the structural relationships we have analyzed into a structured data format. For this purpose, we adopted and extended the data model and XML format called Xigt (Goodman et al, 2014).[6] In this section, we explain the basic Xigt format and the extensions we have developed in order to encode enriched IGT in ODIN.

A basic Xigt representation of the IGT in Fig. 1 is given in Fig. 7.

```xml
<?xml version="1.0" encoding="utf-8"?>
<xigt-corpus alignment-method="auto" xml:lang="en">
  <igt id="i1">
    <tier type="phrases" id="p" xml:lang="ko">
      <item id="p1">Nay-ka ai-eykey pap-ul mek-i-ess-ta</item>
    </tier>
    <tier type="words" id="w" segmentation="p" xml:lang="ko">
      <item id="w1" segmentation="p1[0:6]"/>
      <item id="w2" segmentation="p1[7:15]"/>
      <item id="w3" segmentation="p1[16:22]"/>
      <item id="w4" segmentation="p1[23:35]"/>
    </tier>
    <tier type="glosses" id="g" alignment="w">
      <item id="g1" alignment="w1">I-Nom</item>
      <item id="g2" alignment="w2">child-Dat</item>
      <item id="g3" alignment="w3">rice-Acc</item>
      <item id="g4" alignment="w4">eat-Caus-Pst-Dec</item>
    </tier>
    <tier type="translations" id="t" alignment="p">
      <item id="t1" alignment="p1">I made the child eat rice.</item>
    </tier>
  </igt>
</xigt-corpus>
```

Fig. 7 The basic Xigt representation of the IGT in Fig. 1

The format will be explained below, but there are some differences from the plain-text format to note here. The language line has been split into two tiers: the `phrases` tier stores the entire line as a single item, and the `words` tier encodes the tokenization of the line. This dyadic representation of the language line is useful, in part, because the other tiers (glosses and translations) are not annotations of the same thing; the glosses are annotations of the tokens, while the translation is an annotation of the whole language line. Depending on the desired granularity of tokenization, it is also possible to split words into morphemes, or to not tokenize the language line at all. For these cases, the gloss items (tokenized accordingly) would align to the morphemes or to the whole phrase, respectively. Finally, Fig. 7 is a direct translation of the plain-text

[5] Similarly, existing formats (e.g., the CoNLL format for representing dependency structure) aren't designed to handle IGT, including the relationships between the lines.

[6] Goodman et al chose to develop a new format after surveying many existing formats for encoding IGT and finding that none of them fully supported their requirements. The details of this process, including the desiderata they identified, are in (Goodman et al, 2014).

form into Xigt, but for the ODIN data we will encode structural relationships as standoff annotations, as in Fig. 10. The rest of this section explains the Xigt format and the extensions we've made to accommodate enriched IGT in ODIN.

4.1 The Xigt Data Model and Format

Xigt was designed to accommodate the analysis and processing of large corpora of IGT instances, to be readily extensible for custom tier types, and to handle complex annotation alignments. To satisfy these criteria, Goodman et al made the following design decisions: (1) There are only four levels of nesting of the primary (non-metadata) elements: the corpus (`<xigt-corpus>`), the IGT instance (`<igt>`), the tier (`<tier>`), and the tier datum (`<item>`); (2) All data for an IGT instance goes within its own `<igt>` element, and, for each IGT, all data for a vector of annotations of the same type (words, glosses, etc.) goes within its own `<tier>` element;[7] (3) Structural relationships (i.e., annotation alignments) are represented with an ID-reference scheme. Decision (1) ensures that, even with custom tier extensions, processors of Xigt corpora can reliably read the data, even if they choose to do nothing with it. Decision (2) helps when dealing with large corpora, because the processors can work with an IGT instance as soon as it is read, rather than having to first read an entire corpus into memory. Decision (3) allows us to encode annotations in the relatively flat structure, which is more scalable than using element nesting (e.g., where a word element *contains* its morpheme elements, rather than being referenced by them).

The ID-reference scheme is crucial to the interpretation of Xigt data. In fact, Xigt makes no distinction between original data and annotations; "annotations" are simply data that reference other data in some way. Thus, morphemes are annotations of words just as glosses are annotations of morphemes. Goodman et al created a referencing system that goes beyond simple IDs, allowing multiple IDs and sub-selections of content from the ID-bearing element. This referencing system, called "alignment expressions", allows for complex alignments while being concise in form.[8] By default, Xigt has two modes of referencing among three reference attributes: the `alignment` attribute selects the target of annotation; the `content` attribute selects the content, or label, of annotation (e.g., when the annotation label exists elsewhere in the document, as in stand-off annotation); and the `segmentation` attribute selects both

[7] However, it is possible to have multiple tiers of the same type, such as a `words` tier for the tokenization of the language line and another for the tokenization of the translation. These represent two distinct vectors of annotations of the same type.

[8] Alignment expressions are a deviation from common practice for making references in XML documents. A more standard solution might have separate attributes for the referred ID and the start and end positions of substring selections, where necessary. Considering that there can be multiple IDs, each potentially using substring selections, in a single reference, and that there can be more than one kind of reference for a single item, such a solution could quickly become unwieldy and unnecessarily inflate the file size of a document. For these reasons, Goodman et al found alignment expressions to be a more elegant solution.

the target and the content at the same time. The final attribute is named "segmentation" as it is used to specify that an item is a segment of another. Looking back at Fig. 7, the items on the `words` tier are segments of the `phrases` tier, and therefore use substring selections in the alignment expressions. The `glosses` tier annotates words one-to-one, so its items are aligned to IDs without substring selections. The `content` attribute is generally used for stand-off annotation, described below. In some cases these three reference attributes are insufficient, so Xigt extensions may define new reference attributes. For example, bilingual alignments from source language words to translation words need two annotation targets.

Lastly, Xigt supports `<metadata>` elements at the corpus, IGT, and tier levels. Metadata can be used to specify the languages present in a corpus, track the provenance of source documents, add comments, and so on. Xigt provides some basic metadata types, but since metadata is meant to provide extra information and not change the interpretation of the primary data, it can be open-ended. For example, by including the appropriate namespaces, we can encode OLAC metadata elements,[9] as is done in Fig. 10 below.

4.2 Extensions and Methods for Representing the Original ODIN Data

The features of Xigt provide a useful base for the ODIN corpus. In this section, we describe the extensions we defined on top of the base provided by Xigt to aid in accurately representing the original ODIN data. There are extensions for minor things like element attributes and also for major things like new tier types. We also briefly outline methods for extracting and encoding structural information about IGT tiers.

Provenance and metadata: Each IGT instance in ODIN contains metadata to identify the original PDF it came from, the source language it covers, and the tag types (covered in Section 2.2) of each line. In order to capture this information, we add `doc-id`, `line-range`, and `tag-types` as possible attributes on `<igt>` elements. The `doc-id` attribute value can be looked up in an external citation list to find the document title and author, but we can also encode this information directly into the corpus as an OLAC metadata element. Language information is encoded in the `xml:lang` attribute,[10] but to aid in IGT discoverability we also encode the source language and annotation language in OLAC metadata items.

Stand-off annotation: Because we are automatically inferring the structural annotations from the plain-text formatted IGT, there is always the possibility

[9] `http://www.language-archives.org/OLAC/metadata.html`

[10] The `xml:lang` attribute is a standard part of the XML specification: `http://www.w3.org/TR/xml/#sec-lang-tag`. Its semantics state that it is inherited by descendant elements, so the default language may be specified at the corpus level and tier-specific languages may then override the default.

that the structure we infer is incorrect. We do not want our encoding process to discard potentially useful information, and therefore choose to keep the original ODIN plain-text lines, as extracted from the source PDF documents, in a tier and encode the structural relationships over them as stand-off annotation. This tier type, called `odin`, is the first major Xigt extension for the ODIN data. The items on this tier contain a line of plain text and each item is given an ID for later reference. We encode the data in Unicode by default, so nearly any Unicode character is acceptable.[11] The tier is given an attribute `state="raw"` to show that the contained text hasn't yet gone through any cleaning or normalization steps.

Storing cleaned and normalized IGT: Because IGT instances are extracted automatically from Web documents and linguists do not follow a consistent protocol for creating IGT,[12] the original IGT instances can be noisy. Following the cleaning steps discussed in Section 3.1, we will apply any necessary transformations to the raw text so the structural annotations have a good base to build on. The transformations themselves become new `odin` tiers, aligned to the tier they came from. Each new `odin` tier will set the value of the `state` attribute to describe the kind of transformation done, such as "cleaned" or "normalized".

Automatic segmentation and alignments: So far we have described how we transform and encode the ODIN plain-text IGT into Xigt. Here we explain how structural annotations are created on top of them. The first step is to establish structural tiers that correspond to tagged lines from the ODIN data. If there is a line tagged "L", we create a `phrases` tier with an item selecting the line's content (with the `content` reference attribute). Similarly, a line tagged with "G" is selected by a `glosses` tier item, and a line tagged with "T" is selected by a `translations` tier item. Each of these latter two are also aligned (with the `alignment` reference attribute) to the item on the `phrases` tier to show the annotation structure. At this point, all structural annotations (phrases, glosses, and translations) have a single item representing the whole ODIN plain-text line, as the fragment in Fig. 8 shows.

If the `phrases` tier and `glosses` tier can be word-segmented (e.g., by whitespace) yielding the same number of tokens on each, we refine the annotations by creating a `words` tier from the `phrases` tier (using the `segmentation` reference attribute to select each word), split the single gloss item so there is one for each segment, then realign the glosses one-to-one to the words. The fragment in Fig. 9 shows the results of this step.

Now, if each word and its corresponding gloss can be segmented into morphemes (e.g., by common delimiters such as hyphens) yielding the same num-

[11] Unacceptable characters are those illegal in XML documents, such as the form feed character (`0x000C`) and other Unicode control characters. If the original IGT data contain any unacceptable characters, we replace them with the Unicode replacement character (`0xFFFD`).

[12] The problem persists despite efforts to promote consistency, such as the Leipzig Glossing Rules (Bickel et al, 2004).

```
<tier type="phrases" id="p" content="n" xml:lang="ko">
  <item id="p1" content="n1"/>
</tier>
<tier type="glosses" id="g" content="n">
  <item id="g1" alignment="p1" content="n2"/>
</tier>
<tier type="translations" id="t" content="n">
  <item id="t1" alignment="p1" content="n3"/>
</tier>
```

Fig. 8 Automatically creating structural annotations from the ODIN plain-text lines with the initial full-line alignments.

```
<tier type="phrases" id="p" content="n" xml:lang="ko">
  <item id="p1" content="n1"/>
</tier>
<tier type="words" id="w" segmentation="p" xml:lang="ko">
  <item id="w1" segmentation="p1[0:6]"/>
  <item id="w2" segmentation="p1[7:15]"/>
  <item id="w3" segmentation="p1[16:22]"/>
  <item id="w4" segmentation="p1[23:35]"/>
</tier>
<tier type="glosses" id="g" alignment="w" content="n">
  <item id="g1" alignment="w1" content="n2[0:5]"/>
  <item id="g2" alignment="w2" content="n2[6:15]"/>
  <item id="g3" alignment="w3" content="n2[16:24]"/>
  <item id="g4" alignment="w4" content="n2[25:41]"/>
</tier>
<tier type="translations" id="t" content="n">
  <item id="t1" alignment="p1" content="n3"/>
</tier>
```

Fig. 9 Automatic segmentation and realignment of words and glosses.

ber of tokens, we create a `morphemes` tier from the `words` tier, further split the glosses, and realign. This iterative refinement process allows us to get structural annotations that are as precise as possible given only automatic processing of the data, while still giving us useful structure in the case of noisy or simple data. It is important to note that incorrect alignments can still be ascribed even when the segmented token counts are the same, but we will assume this kind of error is rare and ignore it during the construction of the corpus. Future work may target and correct this kind of error and others caused by PDF extraction errors.

Fig. 10 showcases how our Korean example from Fig. 1 is encoded into Xigt with the ODIN extensions for metadata and stand-off annotation and with automatic segmentation to the morpheme level.

Partial representations: Finally, note that the Xigt encoding of IGT allows for partial representations; that is, because of noise in the original IGT, we cannot always produce all levels of enrichment. A minimal Xigt ODIN entry will have the `odin` tiers encoding the information as extracted directly from the PDF, a citation for the original source, and a language ID. If additional information can be added to the IGT, such as the segmentation and alignment steps above, or with the additional tiers described below, this information can be added on

```
<?xml version="1.0" encoding="utf-8"?>
<xigt-corpus alignment-method="auto" xml:lang="en">
  <metadata xmlns:olac="http://www.language-archives.org/OLAC/1.1/"
            xmlns:dc="http://purl.org/dc/elements/1.1/"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:schemaLocation="http://www.language-archives.org/OLAC/1.1/
              http://www.language-archives.org/OLAC/1.1/olac.xsd">
    <meta id="md1.1">
      <dc:subject xsi:type="olac:language" olac:code="ko"/>
      <dc:language xsi:type="olac:language" olac:code="en"/>
    </meta>
    <meta id="md1.2">
      <dc:source>
        Bratt, Elizabeth Owen. ARGUMENT COMPOSITION AND THE LEXICON:
        LEXICAL AND PERIPHRASTIC CAUSATIVES IN KOREAN. 1996
      </dc:source>
    </meta>
  </metadata>
  <igt id="i1" doc-id="397" line-range="959 961" tag-types="L G T">
    <tier type="odin" state="raw" id="r">
      <item id="r1" line="959" tag="L"
        >(1) Nay-ka ai-eykey pap-ul mek-i-ess-ta</item>
      <item id="r2" line="960" tag="G"
        >    I-Nom child-Dat rice-Acc eat-Caus-Pst-Dec</item>
      <item id="r3" line="961" tag="T"
        >    ‘I made the child eat rice.’</item>
    </tier>
    <tier type="odin" state="normalized" id="n" alignment="r">
      <item id="n1" alignment="r1" line="959" tag="L"
        >Nay-ka ai-eykey pap-ul mek-i-ess-ta</item>
      <item id="n2" alignment="r2" line="960" tag="G"
        >I-Nom child-Dat rice-Acc eat-Caus-Pst-Dec</item>
      <item id="n3" alignment="r3" line="961" tag="T"
        >I made the child eat rice.</item>
    </tier>
    <tier type="phrases" id="p" content="n" xml:lang="ko">
      <item id="p1" content="n1"/>
    </tier>
    <tier type="words" id="w" segmentation="p" xml:lang="ko">
      <item id="w1" segmentation="p1[0:6]"/>
      <item id="w2" segmentation="p1[7:15]"/>
      <item id="w3" segmentation="p1[16:22]"/>
      <item id="w4" segmentation="p1[23:35]"/>
    </tier>
    <tier type="morphemes" id="m" segmentation="w" xml:lang="ko">
      <item id="m1.1" segmentation="w1[0:3]"/>
      <item id="m1.2" segmentation="w1[4:6]"/>
      <item id="m2.1" segmentation="w2[0:2]"/>
      <item id="m2.2" segmentation="w2[3:8]"/>
      <item id="m3.1" segmentation="w3[0:3]"/>
      <item id="m3.2" segmentation="w3[4:6]"/>
      <item id="m4.1" segmentation="w4[0:3]"/>
      <item id="m4.2" segmentation="w4[4:5]"/>
      <item id="m4.3" segmentation="w4[6:9]"/>
      <item id="m4.4" segmentation="w4[10:12]"/>
    </tier>
    <tier type="glosses" id="g" alignment="m" content="n">
      <item id="g1.1" alignment="m1.1" content="n2[0:1]"/>
      <item id="g1.2" alignment="m1.2" content="n2[2:5]"/>
      <item id="g2.1" alignment="m2.1" content="n2[6:11]"/>
      <item id="g2.2" alignment="m2.2" content="n2[12:15]"/>
      <item id="g3.1" alignment="m3.1" content="n2[16:20]"/>
      <item id="g3.2" alignment="m3.2" content="n2[21:24]"/>
      <item id="g4.1" alignment="m4.1" content="n2[25:28]"/>
      <item id="g4.2" alignment="m4.2" content="n2[29:33]"/>
      <item id="g4.3" alignment="m4.3" content="n2[34:37]"/>
      <item id="g4.4" alignment="m4.4" content="n2[38:41]"/>
    </tier>
    <tier type="translations" id="t" alignment="p" content="n">
      <item id="t1" alignment="p1" content="n3"/>
    </tier>
  </igt>
</xigt-corpus>
```

Fig. 10 The Xigt representation of the IGT in Fig. 1 with basic ODIN extensions for metadata attributes and stand-off annotation. Compared to Fig. 9, the words and glosses are further automatically segmented and realigned at the morpheme level.

top of—without altering—the existing data. The ability to add information by referencing and not changing or restructuring the underlying data allows users of the ODIN data to add their own tiers of annotations and make them available to the public. Thus we see another benefit of encoding ODIN data in Xigt: Xigt allows easy encoding of information in enriched IGT and helps facilitate the exchange of IGT data with various levels of annotation.

4.3 Extensions for Enriched ODIN Data

Beyond encoding the original ODIN IGT into Xigt, we also want to encode information from enriched IGT. For these, we define additional tier types, as described below.

Syntactic structures: An important source of information in IGT is the implicit structure of the string in the translation line—this is equally true for human readers of IGT and for automatic processors of IGT. Humans use their knowledge as speakers of resource-rich language (the language translated into) to gain an understanding of the language line, whereas automatic processors take advantage of the greater resources available for English (the most common language for IGT translation lines) to do the same. We defined a Xigt extension to allow for the encoding of syntactic structures (both dependency structures and parse trees). Our method of encoding these structures in XML is similar to some other formats, such as the TIGER-XML treebanking format (Brants et al, 2002) for parse trees, or Malt-XML (Nivre et al, 2006) for syntactic dependencies, although there are differences to ensure conformance with the Xigt data model.

For parse trees, we create a new tier type `phrase-structure`, and we treat each `<item>` element as a node in the tree. The `alignment` reference attribute aligns leaf nodes with the words (e.g., words from the translation line) they label. We define a second reference attribute, `children`, so non-terminal nodes can select their child nodes (terminal or non-terminal). The value of the `children` reference attribute is not an alignment expression, but a reference list, so it can take a space-separated list[13] of one or more IDs. The content of each item (whether explicitly given or selected via the `content` attribute) is the node label. Fig. 11 shows how the `phrase-structure` tier, supported by a `words` tier segmenting the translation line,[14] would be encoded to annotate the example in Fig. 10.

[13] Following the specification of the IDREFS attribute type: `http://www.w3.org/TR/REC-xml/#idref`. We choose not to use, e.g., a comma-separated alignment expression because the children of a node are more intuitively a list rather than a string concatenation. Also, we want to disallow sub-selections on children.

[14] It is not strictly necessary to first segment the translation line; the items on the `phrase-structure` tier could use alignment expressions to select the word spans directly from the item on the `translations` tier. However, we find it prudent to segment the words separately in case more than one tier annotates the same segments.

```
<tier type="words" id="tw" segmentation="t">
  <item id="tw1" segmentation="t1[0:1]"/>
  <item id="tw2" segmentation="t1[2:6]"/>
  <item id="tw3" segmentation="t1[7:10]"/>
  <item id="tw4" segmentation="t1[11:16]"/>
  <item id="tw5" segmentation="t1[17:20]"/>
  <item id="tw6" segmentation="t1[21:25]"/>
  <item id="tw7" segmentation="t1[25:26]"/>
</tier>
<tier type="phrase-structure" id="ps" alignment="tw" children="ps">
  <item id="ps1" alignment="tw1">PRP</item>
  <item id="ps2" alignment="tw2">VBD</item>
  <item id="ps3" alignment="tw3">DT</item>
  <item id="ps4" alignment="tw4">NN</item>
  <item id="ps5" alignment="tw5">VBP</item>
  <item id="ps6" alignment="tw6">NN</item>
  <item id="ps7" alignment="tw7">.</item>
  <item id="ps8" children="ps1">NP</item>
  <item id="ps9" children="ps3 ps4">NP</item>
  <item id="ps10" children="ps6">NP</item>
  <item id="ps11" children="ps5 ps10">VP</item>
  <item id="ps12" children="ps9 ps11">S</item>
  <item id="ps13" children="ps12">SBAR</item>
  <item id="ps14" children="ps2 ps13">VP</item>
  <item id="ps15" children="ps8 ps14 ps7">S</item>
</tier>
```

Fig. 11 XML fragment showing the `phrase-structure` tier annotating translation words from the example in Fig. 10.

Syntactic dependency structures are represented with a `dependencies` tier. As the nodes of a dependency graph are the words themselves, there is little to be gained by specifying dependency nodes in the tier. Instead, items in the tier represent the dependency relations between two words. The dependent word is selected by the reference attribute `dep`, while the head is selected with the reference attribute `head`. An item with a dependent and no head is interpreted as the root. For labeled dependencies, the item contents (whether explicitly given or selected via the `content` attribute) encode the label. Fig. 12 shows the encoding of the `dependencies` tier over the translation words given in Fig. 11.

```
<tier type="dependencies" id="dt" dep="tw" head="tw">
  <item id="dt1" dep="tw1" head="tw2">nsubj</item>
  <item id="dt2" dep="tw2">root</item>
  <item id="dt3" dep="tw3" head="tw4">det</item>
  <item id="dt4" dep="tw4" head="tw2">nsubj</item>
  <item id="dt5" dep="tw5" head="tw2">ccomp</item>
  <item id="dt6" dep="tw6" head="tw5">dobj</item>
</tier>
```

Fig. 12 XML fragment showing the `dependencies` tier annotating translation words (listed in Fig. 11) from the example in Fig. 10.

In this example, both the parse tree if Fig. 11 and the syntactic dependencies in Fig. 12 were obtained by the Stanford Parser (Klein and Manning, 2003; de Marneffe et al, 2006), but the users can choose their preferred parser to generate alternative analyses and add them to the Xigt representation.

Parts of Speech: A tier to label the parts of speech of the language line is relatively common in IGT, but it is not included in the standard tier set of Xigt. For ODIN, we also want to label the parts of speech of the translation words. For these cases, we add a `pos` tier. This tier does not require any special attributes. It may be useful to restrict the allowable parts of speech, but we do not place any restrictions for this version of ODIN. Fig. 13 shows the `pos` tier annotating the translation words.

```
<tier type="pos" id="tw-pos" alignment="tw">
  <item id="tw-pos1" alignment="tw1">PRP</item>
  <item id="tw-pos2" alignment="tw2">VBD</item>
  <item id="tw-pos3" alignment="tw3">DT</item>
  <item id="tw-pos4" alignment="tw4">NN</item>
  <item id="tw-pos5" alignment="tw5">VBP</item>
  <item id="tw-pos6" alignment="tw6">NN</item>
  <item id="tw-pos7" alignment="tw7">.</item>
</tier>
```

Fig. 13 XML fragment showing the `pos` tier annotating parts of speech over the translation words (listed in Fig. 11) from the example in Fig. 10.

Note here that the parts of speech are the same as the terminal node labels of the `phrase-structure` tier shown earlier. If this is guaranteed to be the case (e.g., if the parts of speech were taken directly from the parse tree instead of being produced by a separate process), it's possible to select them from the `phrase-structure` tier via the `content` reference attribute instead of spelling them out again. Similarly it is possible to spell out the parts of speech in the `pos` tier and have the `phrase-structure` tier select them for its terminal node labels (the non-terminal node labels would still need to be explicitly given). Fig. 14 is a fragment of what the `pos` tier would look like if its content were selected from a `phrase-structure` tier.

```
<tier type="pos" id="tw-pos" alignment="tw" content="ps">
  <item id="tw-pos1" alignment="tw1" content="ps1"/>
  ...
</tier>
```

Fig. 14 XML fragment showing the `pos` tier with labels selected from a `phrase-structure` tier.

This kind of cross-reference would yield data that is more linked than otherwise, so that updates to the labels on the parse tree would also update the parts of speech, but it also increases dependencies among the tiers. For instance, if the `phrase-structure` tier were removed, the content of each item on the `pos` tier would need to be reified (i.e., the resolution of the `content` reference attribute would be set as the value directly) or else the `content` reference attributes would become invalid. For the ODIN data, the part of speech content links to the items on the phrase-structure tier.

Bilingual Word Alignments In order to project the syntactic structure from the translation line to the language line, as is done in ODIN, we need word alignments from one side to the other, as might be produced by a statistical word aligner. In ODIN we actually align the translation words to the gloss tokens (as the gloss tokens should align one-to-one with the language line words), but the effect is the same. To represent this in Xigt, we need a tier that aligns to two other tiers, therefore we define a `bilingual-alignments` tier with both `source` and `target` reference attributes.[15] Note that, unlike the `phrase-structure` tier above (see Fig. 11), bilingual alignments may have a need for sub-selections when only part of a word aligns, so we use alignment expressions instead of a simple reference list. Fig. 15 shows how we encode these alignments between the translation words and the glosses.

```
<tier type="bilingual-alignments" id="a" source="tw" target="g">
  <item id="a1" source="tw1" target="g1.1"/>
  <item id="a2" source="tw2" target="g4.2,g4.3"/>
  <item id="a3" source="tw3" />
  <item id="a4" source="tw4" target="g2.1"/>
  <item id="a5" source="tw5" target="g4.1"/>
  <item id="a6" source="tw6" target="g3.1"/>
</tier>
```

Fig. 15 XML fragment showing the `bilingual-alignments` tier aligning translation words (listed in Fig. 11) to glosses for the IGT in Fig. 10.

4.4 Dependencies between tiers

So far, we have described the tiers used to represent enriched IGT data. The tiers can be divided into three groups according to the source of information in the tiers:

– Group 1 includes only one tier type, `odin`, and it stores the original IGT text and the text after cleaning and normalization.
– Group 2 (including `phrases`, `words`, `morphemes`, `glosses`, `translations` tiers) encodes the annotations that are implicit in the textual IGT. For instance, segmenting words in the `words` tier by hyphens will yield the `morphemes` tier, and the i^{th} token in the `morphemes` tier should—in a clean, conventional IGT—align to the i^{th} token in a similarly segmented `glosses` tier.

[15] We could have repurposed the `alignment` reference attribute for one of these, perhaps `source`, but we felt that defining two new reference attributes made their purpose clearer.

- Group 3 encodes information that is not present in the original IGT, but is obtained through manual annotation with tools such as *XigtEdit*, or by running the IGT through some NLP systems such as *INTENT*. This group currently includes `bilingual-alignments`, `pos`, `phrase-structure`, and `dependencies`, and it can be extended further if new tier types are needed to present new types of information.

In Fig. 16, the three big boxes correspond to the three groups. The arcs between the tiers show the dependency relations between those tiers, and they correspond to the reference attributes (e.g., `content` and `alignment`). A tier can depend on more than one tier; for instance, the `bilingual-alignments` tier depends on both `glosses` and `words`. In addition to the arcs displayed in the figure, there could be other dependencies between Group 3 tiers based on how they are created. For instance, if the `dependencies` tier is created by syntactic projection, the tier will also depend on the `bilingual-alignments` tier. These dependency relations define a partial order between tiers so that a tier can only refer to itself or tiers that precede it in the dependency chain. This partial ordering is crucial when *XigtEdit* determines how editing in one tier affects other tiers (see Section 6.2).

4.5 Processing documents with the Xigt API

To make it easier for the researchers to access IGT data in the Xigt format, we provide an application-program interface (API), with a reference implementation in Python, for interacting with Xigt-encoded corpora computationally. The API provides the following functionalities:

- Serialize/deserialize Xigt documents to in-memory data structures
- Iterate over data collections (corpora, IGT, tiers)
- Retrieve object attributes, metadata, and content
- Retrieve the parent (i.e., container) of some object, such as a tier from an item
- Resolve the content, or the targeted items/tiers, of alignment expressions
- Construct new in-memory data structures

These functions allow users to easily build more complicated functions for their data, such as for counting statistics (e.g., finding the most frequent word), forming complex queries over data (e.g., "what are all the morphemes appearing on words marked as verbs?"), or augmenting a corpus with new analyses (e.g., creating a word-sense tier by looking up each word and its context in an external ontology and aligning the result to the word it came from). The API also enables users to construct new corpora in-memory (e.g., by converting or analyzing some other data) which can then be serialized to disk. The Python implementation of the Xigt API is available as part of the Xigt project: `http://depts.washington.edu/uwcl/xigt`.

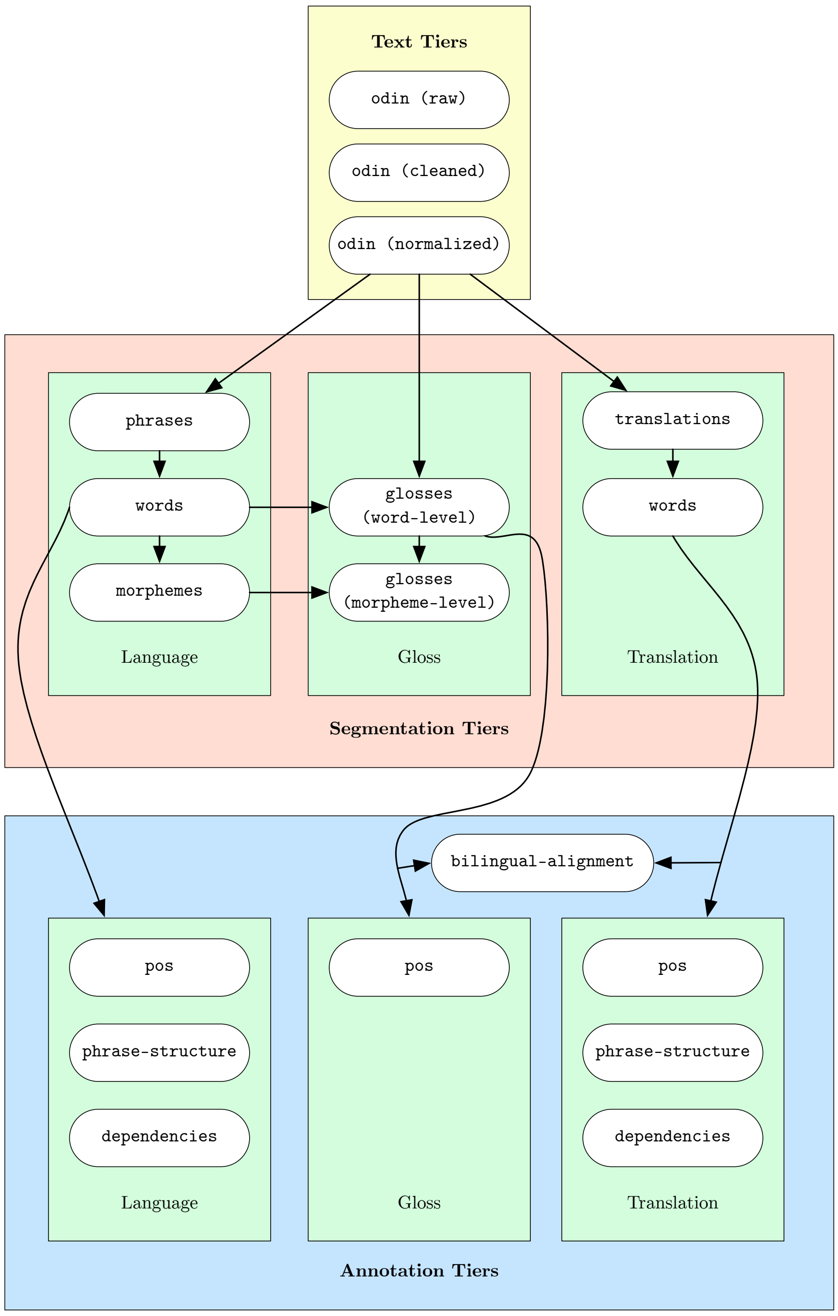


Fig. 16 Dependencies between tiers in enriched IGT

4.6 Summary: Encoding ODIN in Xigt

Here we have described the Xigt data model and XML format and two sets of extensions for encoding the ODIN data: the first set helps us accurately encode the existing data, while the second allows us to encode new information obtained by analyzing the existing data. The minimal IGT will contain an `odin` tier with the text extracted from the PDF files. Where it is possible to infer the structure of the existing data, additional tiers will provide stand-off annotations with the structure. As we analyze and process the existing data, the results will be added as further annotation referencing the existing tiers.

5 ***INTENT***: a package for creating enriched IGTs

In the previous sections, we described what type of information is in enriched IGTs and how it is represented in Xigt. Because manually creating enriched IGTs is time consuming and error-prone, we have developed a package, the INterlinear Text ENrichment Toolkit (*INTENT*), which takes an original IGT file as the input, and produces the enriched IGT in the Xigt format as the output. This output can then be corrected by a human annotator using *XigtEdit*, or be used to train a POS tagger or a parser.

5.1 Toolkit components

Fig. 17 shows a typical enrichment workflow in *INTENT*. The input to *INTENT* is a file with the original IGT in either plain text format or in Xigt. *INTENT* first cleans and normalizes the IGT by some simple heuristic rules. It then generates the second group of tiers including `words`, `morphemes` (if the morpheme boundary is present in the IGT), `glosses`, and the like. After that, the third group of tiers are created by running the word alignment, part-of-speech tagging, and syntactic parsing modules described below.

Word Alignment: In previous work (Xia and Lewis, 2007), we proposed two methods for aligning the gloss line and the translation line. The first method ran a morphological analyzer on the translation line, and then aligned the words in the two lines if they had the same stems. The second method used GIZA++ (Och and Ney, 2003), a statistical word aligner. Experimental results showed that the performances of the two methods were similar and combining them yielded a small boost. *INTENT* has re-implemented those methods, and we plan to enhance the heuristic method by taking advantage of the POS tags in the enriched IGT.

Part-of-Speech Tagging: *INTENT* tags the translation line by running Stanford's English POS tagger (Toutanova et al, 2003)[16] trained on the English Penn Treebank (Marcus et al, 1993). As for the language line, while one

[16] `http://nlp.stanford.edu/software/tagger.shtml`

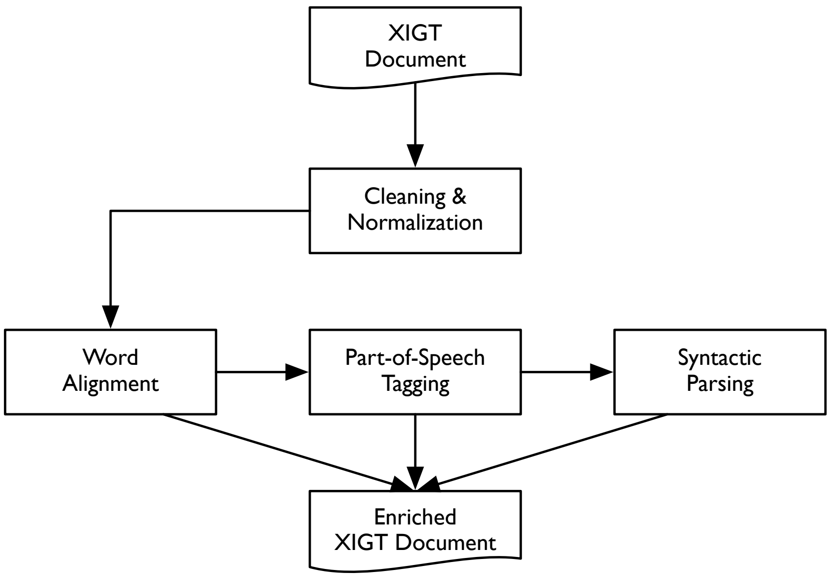


Fig. 17 A typical enrichment workflow in *INTENT*

can simply project the POS tags from the translation line, the quality of the resulting tags is often low due to word alignment errors and translation divergence (Dorr, 1994). Instead, *INTENT* takes advantage of the annotation on the gloss line; for instance, grammatical markers such as *-Nom* (nominative case marker) and *-Dec* (declarative marker) are good cues for predicting the POS tags of the corresponding words in the language line. We can also find the POS tags of most morphemes in the gloss line using an English dictionary even if those morphemes are not aligned to the words in the translation line. We built a classifier using those features and trained it with a small amount of labeled gloss line data from multiple languages.[17] *INTENT* runs that classifier on new IGT data. Although the new IGT in the test data may be in a language that does not appear in the training data for the classifier, the meanings of grammatical markers and the POS tags of the glosses are largely language-independent. Experimental results show that this classifier outperforms the method that projects the POS tags directly (Georgi et al, 2015).

Syntactic Parsing: The syntactic structure (phrase structure or dependency tree) for the translation line is produced by running the Stanford Parser (de Marneffe et al, 2006).[18] *INTENT* then projects the syntactic structure to the language line following the heuristic algorithm in (Xia and Lewis, 2007).

[17] We use a classifier, not a sequence labeler, because the word order in the gloss line will be language-dependent, and the training and test data of our POS tagger can come from different languages.

[18] `http://nlp.stanford.edu/software/lex-parser.shtml`

While the workflow in Fig. 17 shows a pipeline approach, we are expanding the package to allow feedback loops among the modules. For instance, *INTENT* runs the word aligner first to get the initial alignment, which will be used to project high-precision POS tags. The output of that POS tagger can then be fed back to the word aligner to improve word alignment; for instance, two words unaligned during the first pass of word alignment are more likely to be linked together in the second pass if they have the same POS tags after POS tagging. The improved word alignment can in turn improve the next round of POS tagging.

5.2 Implementation of *INTENT*

INTENT is written in Python 3 and uses the Xigt API to interface with the serialized documents. *INTENT* also supplements the Xigt API's internal representations with a number of convenience subclasses for performing tasks such as tokenization and word alignment. Each type of enrichment can be run individually or in sequence. We are in the final stage of packaging *INTENT*, and the enrichment provided by the methods above is included in the IGT instances that are part of the version 2.1 release of the ODIN data.

Because *INTENT* requires configuration and installation of several other packages (e.g., the Stanford English parser), we have created a web server which allows users to upload their whole IGT data, get it enriched, download the results in the Xigt format, and share the data with others (if they desire), all without the need of downloading the packages to their local machines.

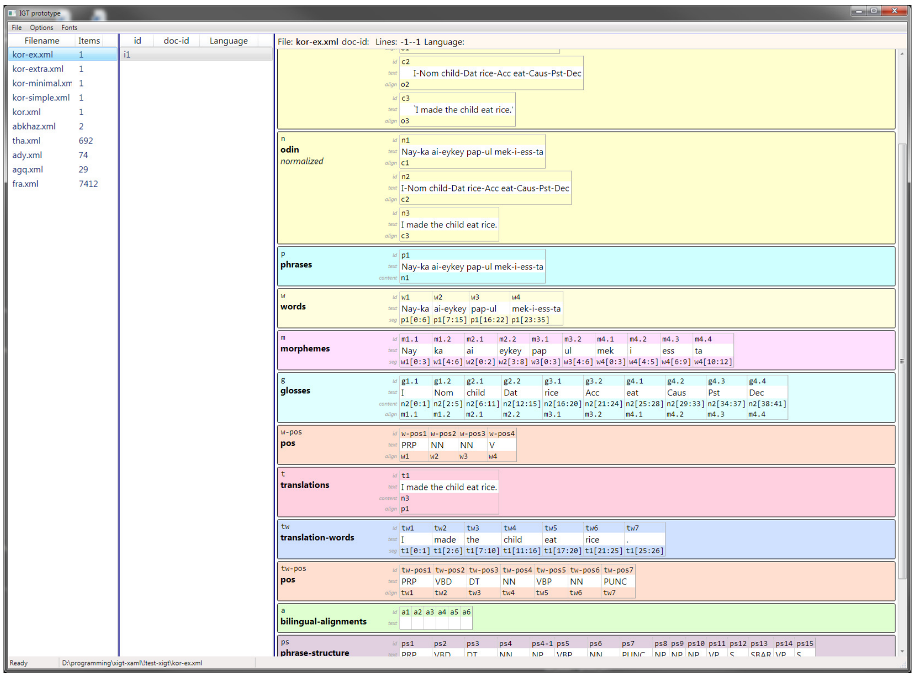
6 ***XigtEdit*****: a GUI editor for enriched IGT**

As Xigt is an XML-based format, it is nominally human-readable and thus editable with any text editor which is compatible with the desired or appropriate text encodings. However, using existing text editors to edit enriched IGT in the Xigt format is not convenient due to the special properties of Xigt:

– Xigt standoff annotation requires each tier and each tier item to have a unique ID, which is used for cross-reference within an IGT instance. Assigning unique IDs manually is tedious and error-prone.
– Some alignment expressions (e.g., `segmentation` and `alignment` fields in many tiers) require precise computation of string offsets, which are tedious to manually derive.
– Phrase-structure and dependency-structure views are inherently graphical views that do not lend themselves to convenient text-based editing.
– Because tiers can refer to one another, editing one tier could affect the validity of annotation in its cross-referenced tiers. Manually keeping track of the ripple effect of such editing is challenging.

In order to address these issues, and because we desire to enable manual correction of enriched IGT, we developed a graphical Xigt editor, *XigtEdit*, which facilitates the creation, editing and manipulation of Xigt files.

6.1 Main functionality of *XigtEdit*



Filename	Items
kor-ex.xml	1
kor-extra.xml	1
kor-minimal.xm	1
kor-simple.xml	1
kor.xml	1
abkhaz.xml	2
tha.xml	692
ady.xml	74
agq.xml	29
fra.xml	7412

id	doc-id	Language
i1		

File: **kor-ex.xml** doc-id: Lines: **-1--1** Language:

id	c2
text	I-Nom child-Dat rice-Acc eat-Caus-Pst-Dec
align	o2
id	c3
text	`I made the child eat rice.'
align	o3

n
odin
normalized

id	n1
text	Nay-ka ai-eykey pap-ul mek-i-ess-ta
align	c1
id	n2
text	I-Nom child-Dat rice-Acc eat-Caus-Pst-Dec
align	c2
id	n3
text	I made the child eat rice.
align	c3

p
phrases

id	p1
text	Nay-ka ai-eykey pap-ul mek-i-ess-ta
content	n1

w
words

id	w1	w2	w3	w4
text	Nay-ka	ai-eykey	pap-ul	mek-i-ess-ta
seg	p1[0:6]	p1[7:15]	p1[16:22]	p1[23:35]

m
morphemes

id	m1.1	m1.2	m2.1	m2.2	m3.1	m3.2	m4.1	m4.2	m4.3	m4.4
text	Nay	ka	ai	eykey	pap	ul	mek	i	ess	ta
seg	w1[0:3]	w1[4:6]	w2[0:2]	w2[3:8]	w3[0:3]	w3[4:6]	w4[0:3]	w4[4:5]	w4[6:9]	w4[10:12]

g
glosses

id	g1.1	g1.2	g2.1	g2.2	g3.1	g3.2	g4.1	g4.2	g4.3	g4.4
text	I	Nom	child	Dat	rice	Acc	eat	Caus	Pst	Dec
content	n2[0:1]	n2[2:5]	n2[6:11]	n2[12:15]	n2[16:20]	n2[21:24]	n2[25:28]	n2[29:33]	n2[34:37]	n2[38:41]
align	m1.1	m1.2	m2.1	m2.2	m3.1	m3.2	m4.1	m4.2	m4.3	m4.4

w-pos
pos

id	w-pos1	w-pos2	w-pos3	w-pos4
text	PRP	NN	NN	V
align	w1	w2	w3	w4

t
translations

id	t1
text	I made the child eat rice.
content	n3
align	p1

tw
translation-words

id	tw1	tw2	tw3	tw4	tw5	tw6	tw7
text	I	made	the	child	eat	rice	.
seg	t1[0:1]	t1[2:6]	t1[7:10]	t1[11:16]	t1[17:20]	t1[21:25]	t1[25:26]

tw-pos
pos

id	tw-pos1	tw-pos2	tw-pos3	tw-pos4	tw-pos5	tw-pos6	tw-pos7
text	PRP	VBD	DT	NN	VBP	NN	PUNC
align	tw1	tw2	tw3	tw4	tw5	tw6	tw7

a
bilingual-alignments

id	a1	a2	a3	a4	a5	a6
text						

ps
phrase-structure

id	ps1	ps2	ps3	ps4	ps4-1	ps5	ps6	ps7	ps8	ps9	ps10	ps11	ps12	ps13	ps14	ps15
text	PRP	VBD	DT	NN	NP	VBP	NN	PUNC	NP	NP	NP	VP	S	SBAR	VP	S

Ready D:\programming\xigt-xaml\\test-xigt\kor-ex.xml

Fig. 18 Main editing interface screen from the *XigtEdit* application

The fundamental user interface of *XigtEdit* is a hierarchical structure which closely follows the Xigt abstract data model. Fig. 18 shows a screen capture of the *XigtEdit* application. There are three resizable panels:

– The leftmost panel is a list of the Xigt files that have been loaded. If there is more than one file, exactly one file is *currently selected*.
– The next panel, in the second column, lists the IGT instances which are in the selected file. Again, if the file contains multiple IGT instances, exactly one instance is *currently selected*.
– The rightmost panel is the editing area for the currently selected IGT instance. Tiers are arranged vertically in this area. For some tiers, items in the tiers are arranged vertically (line-oriented data such as in the `odin` tier) while others have items displayed horizontally (word-oriented data such as in the `words`, `morphemes`, and `glosses` tiers).

At any time during editing, individual Xigt files can be opened, edited, saved, closed (added or removed from the files list), or reverted. Files are read and saved directly in the Xigt format. To enhance annotator productivity, *XigtEdit* assigns unique IDs to new tiers and tier items based on predefined naming conventions.

To address the inconvenience of computing and maintaining Xigt alignment expressions (e.g., the `segmentation` field in the `words` tier), *XigtEdit* allows the text spans for dependent items to be defined automatically. This can be achieved either through automatic tools for segmenting text based on whitespace or other criteria, or manually via intuitive user interfaces for manipulating text ranges. Furthermore, *XigtEdit* displays dependency or phrase structure tiers as graphical trees which the user can edit with mouse clicks. To support efficient annotation, *XigtEdit* provides keyboard alternatives to the use of the mouse for most application navigation and editing operations.

6.2 Editing parent tiers

As mentioned in Section 4.4, tiers can refer to other tiers and we define a partial order among tiers such that any tier can only refer to preceding tiers or itself. If a tier C refers to another tier P, we call P a parent of C. A tier can have multiple parents (e.g., a `bilingual-alignments` tier refers to a `words` tier and a `glosses` tier). Thus, this parent relation among tiers can be represented as a directed acyclic graph, where each node represents a tier, and each arc goes from the parent tier to its child tier, as illustrated in Fig. 16.

XigtEdit supports the propagation of editing changes from parent to child tiers in an IGT. Consider how editing tier P would affect another tier C that refers to it. The behavior depends on the relation between P and C, and whether other tiers refer to the text region in P that was edited. *XigtEdit* keeps track of these relationships and analyzes whether the change would invalidate other tiers. In cases where *XigtEdit* determines that a change has a deterministic effect on its dependent tiers (and where the *edit-propagation* feature has been enabled in the software), the change can be propagated from the parent tier to its child tiers automatically. Alternatively, if the change has ambiguous effects on other tiers, *XigtEdit* will prompt the user to choose what action should be taken for dependent tiers.

Currently, *XigtEdit* is implemented as a Windows application within the WPF graphical environment[19]. We are also in the process of porting the interface to be web-browser based, for cross-platform compatibility. All implementations of *XigtEdit* are open source and licensed under the MIT license.

[19] Windows Presentation Foundation (WPF) is a graphical framework for Microsoft Windows. See `http://msdn.microsoft.com/en-us/library/aa970268(v=vs.110).aspx`

7 Conclusion

The majority of the world's languages lack large-scale annotated resources over which NLP tools such as POS taggers or parsers can be trained. In recent years, there have been increasing efforts in bootstrapping NLP systems for resource-poor languages. In the past decade, we have built ODIN, a collection of IGT data extracted from the linguistic documents posted to the Web, and used the IGT data to project annotations from resource-rich languages to resource-poor ones in order to bootstrap NLP tools for the resource-poor languages.

The ODIN resource has already proved valuable in a number of research projects (Bender et al, 2013; Georgi et al, 2013, 2015). Our goal in the present work is to make it more accessible to the community on several levels. First, we released the original ODIN data both in bulk and on a per-language basis. Second, we adopted and extended Xigt, an XML representation for enriched IGT, and provided an API for it. Users now have the option of accessing the ODIN data in plain text format or in the Xigt-encoded version. IGT encoded in the Xigt format can be easily extended to add more tiers or alternative annotations. Xigt thus serves as a vehicle for users of ODIN to improve annotations on the ODIN data, which in turn can provide for iterative improvements to the resource. The successive versions of ODIN can therefore have broader utility across the community.

Third, and most importantly, much previous work on ODIN has built on the enrichment steps described in Section 3, but the enriched data have not previously been available for use by the broader research community. We have developed a package, *INTENT*, which enriches raw IGT automatically by adding word alignment, POS tags, and syntactic structures to IGT. We also built *XigtEdit*, a graphical editor for annotating IGT, which overcomes limitations of existing, general-purpose text editors. Making those data and tools freely available to the public allows NLP researchers to have easy access to the enriched IGT data without having to re-implement the cleaning, normalization, and enriching steps. They can then focus on exploring new methods for bootstrapping NLP tools for thousands of resource-poor languages by taking advantage of rich annotation in IGT. The ODIN data, along with the enrichments and packages described in this paper, are made available at `http://depts.washington.edu/uwcl/packages/`.

In future work, we plan to use *XigtEdit* to manually correct the output of *INTENT* for a dozen resource-poor languages, and this data set can serve as training and test data for NLP tools for those languages. The data sets will be released to the public, along with *INTENT*, *XigtEdit*, and other IGT processing toolkits.

References

Bailyn JF (2001) Inversion, Dislocation and Optionality in Russian. In: Zybatow G (ed) Current Issues in Formal Slavic Linguistics, Frankfurt: Peter

Lang AG

Bender EM, Goodman MW, Crowgey J, Xia F (2013) Towards creating precision grammars from interlinear glossed text: Inferring large-scale typological properties. In: Proceedings of the 7th Workshop on Language Technology for Cultural Heritage, Social Sciences, and Humanities, Sofia, Bulgaria, pp 74–83

Bickel B, Comrie B, Haspelmath M (2004) The Leipzig Glossing Rules: Conventions for interlinear morpheme-by-morpheme glosses (revised version). Tech. rep., Department of Linguistics of the Max Planck Institute for Evolutionary Anthropology and the Department of Linguistics of the University of Leipzig, http://www.eva.mpg.de/lingua/files/morpheme.html (2006-May-17)

Brants S, Dipper S, Hansen S, Lezius W, Smith G (2002) The TIGER treebank. In: Proceedings of the Workshop on Treebanks and Linguistic Theories, pp 24–41

Bybee JL, Dahl Ö (1989) The creation of tense and aspect systems in the languages of the world. John Benjamins

Das D, Petrov S (2011) Unsupervised part-of-speech tagging with bilingual graph-based projections. In: HLT '11: Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, Association for Computational Linguistics

Dorr BJ (1994) Machine translation divergences: a formal description and proposed solution. Computational Linguistics 20(4):597–635

Farrar S, Langendoen DT (2003) A linguistic ontology for the Semantic Web. GLOT International 7(3):97–100

Feldman A, Hana J, Brew C (2006) A cross-language approach to rapid creation of new morpho-syntactically annotated resources. In: Proc. of the 5th international conference on Language Resources and Evaluation (LREC 2006), Genoa, Italy

Georgi R, Xia F, Lewis WD (2013) Enhanced and portable dependency projection algorithms using interlinear glossed text. In: Proceedings of ACL 2013 (Volume 2: Short Papers), Sofia, Bulgaria, pp 306–311

Georgi R, Xia F, Lewis WD (2014) Capturing divergence in dependency trees to improve syntactic projection. Language Resources and Evaluation 48(4):709–739

Georgi R, Xia F, Lewis WD (2015) Enriching interlinear text using automatically constructed annotators. In: Proceedings of the 9th Workshop on Language Technology for Cultural Heritage, Social Sciences, and Humanities (LaTeCH 2015), Beijing, China

Goodman MW, Crowgey J, Xia F, Bender EM (2014) Xigt: extensible interlinear glossed text for natural language processing. Language Resources and Evaluation pp 1–31, DOI 10.1007/s10579-014-9276-1, URL `http://dx.doi.org/10.1007/s10579-014-9276-1`

Hana J, Feldman A, Amaral L, Brew C (2006) Tagging Portuguese with a Spanish Tagger Using Cognates. In: Proc. of the Workshop on Cross-language Knowledge Induction, in conjunction with the 11th Conference of the European Chapter of the Association for Computational Linguistics (EACL-2006), Trento, Italy

Hwa R, Resnik P, Weinberg A, Cabezas C, Kolak O (2005) Bootstrapping Parsers via Syntactic Projection across Parallel Texts. Special Issue of the Journal of Natural Language Engineering on Parallel Texts 11(3):311–325

Klein D, Manning CD (2003) Accurate unlexicalized parsing. In: Proceedings of the 41st Meeting of the Association for Computational Linguistics, pp 423–430

Lewis W (2003) Mining and migrating interlinear text. In: Proceedings of EMELD 2003 Workshop on Digitizing and Annotating Texts and Field Recordings, East Lansing, Michigan, URL `http://www.emeld.net/workshop/2003/Lewis-paper.pdf`

Lewis W, Xia F (2010) Developing odin: A multilingual repository of annotated language data for hundreds of the world's languages. Journal of Literary and Linguistic Computing (LLC) 25(3):303–319

Lewis WD, Xia F (2008a) Automatically Identifying Computationally Relevant Typological Features. In: Proc. of the Third International Joint Conference on Natural Language Processing (IJCNLP-2008), Hyderabad, India

Lewis WD, Xia F (2008b) Automatically identifying computationally relevant typological features. In: Proceedings of the Third International Joint Conference on Natural Language Processing, Hyderabad, India, pp 685–690

Lewis WD, Farrar S, Langendoen DT (2001) Building a knowledge base of morphosyntactic terminology. In: Proceedings of the IRCS Workshop on Linguistic Databases, University of Pennsylvania, pp 150–156, URL `www.u.arizona.edu/$\sim$farrar/papers/LewFarLang01.pdf`

Ma X, Xia F (2014) Unsupervised dependency parsing with transferring distribution via parallel guidance and entropy regularization. In: Proceedings of ACL-2014, Baltimore, MD

Marcus M, Marcinkiewicz MA, Santorini B (1993) Building a large annotated corpus of English: the Penn Treebank. Computational Linguistics 19(2):313–330

de Marneffe MC, MacCartney B, Manning CD (2006) Generating typed dependency parses from phrase structure parses. In: Proceedings of LREC 2006

McDonald R, Nivre J, Quirmbach-Brundage Y, Goldberg Y, Das D, Ganchev K, Hall K, Petrov S, Zhang H, Täckström O, Bedini C, Castelló NB, Lee J (2013) Universal Dependency Annotation for Multilingual Parsing. In: Proceedings of ACL

Nivre J, Hall J, Nilsson J (2006) Maltparser: A data-driven parser-generator for dependency parsing. In: Proceedings of LREC, vol 6, pp 2216–2219

Och FJ, Ney H (2003) A systematic comparison of various statistical alignment models. Computational Linguistics 29(1):19–51

Täckström O, McDonald R, Uszkoreit J (2012) Cross-lingual word clusters for direct transfer of linguistic structure. In: Proceedings of NAACL/HLT 2012

Täckström O, McDonald R, Nivre J (2013) Target language adaptation of discriminative transfer parsers. In: Proceedings of NAACL 2013

Toutanova K, Klein D, Manning C, Singer Y (2003) Feature-rich part-of-speech tagging with a cyclic dependency network. In: Proceedings of HLT-NAACL 2003, pp 252–259

Xia F, Lewis WD (2007) Multilingual structural projection across interlinear text. In: Proc. of the Conference on Human Language Technologies (HLT/-NAACL 2007), Rochester, New York, pp 452–459

Xia F, Lewis WD (2008) Repurposing Theoretical Linguistic Data for Tool Development and Search. In: Proc. of the Third International Joint Conference on Natural Language Processing (IJCNLP-2008), Hyderabad, India

Xia F, Lewis WD, Poon H (2009) Language ID in the Context of Harvesting Language Data off the Web. In: Proceedings of The 12th Conference of the European Chapter of the Association of Computational Linguistics (EACL 2009), Athens, Greece

Xia F, Lewis C, Lewis WD (2010) The problems of language identification within hugely multilingual data sets. In: Proceedings of the 7th International Conference on Language Resources and Evaluation (LREC 2010), Valletta, Malta, pp 2790–2797

Xiao M, Guo Y (2015) Annotation Projection-based Representation Learning for Cross-lingual Dependency Parsing. CoNLL 2015

Yarowsky D, Ngai G (2001) Inducing Multilingual POS Taggers and NP Bracketers via Robust Projection across Aligned Corpora. In: Proc. of the 2001 Meeting of the North American chapter of the Association for Computational Linguistics (NAACL-2001), pp 200–207