

Creating Morphological Descriptions with Alchemist

Colin Sprague, University of Chicago

Alchemist is an open-source program that facilitates the creation of morphological descriptions from text. The history of Alchemist, and part of its current functionality, are built into the name. Much like alchemists of old tried to create gold from other metals, field and computational linguists can use Alchemist to create morphological gold standards from corpora. A gold standard in morphology is a human-created morphological description of a language. In the hands of language experts and novices alike, Alchemist provides efficient methods for parsing words into morphemes and for describing the morphemes with features and glosses.

Alchemist supports community standards. There is a recent push in the Linguistics community to standardize language description and documentation. E-MELD¹ and LINGUIST List are front-runners in the creation of best practices and supporting tools. The goal is to increase usability and availability of language descriptions and documentation by educating creators and users about valued methods of creating and archiving data. E-MELD and LINGUIST List's methods are designed to maximize the access and portability of language descriptions. Alchemist helps the field linguist follow these practices by providing accepted descriptors and by outputting data in an open format.

This document is an exposition on how Alchemist addresses the community need for tools that create morphological descriptions as well as a description of the challenges and considerations that must be tackled by such an application. The document is laid out in the following format:

Section 1: Motivations for Alchemist

Section 2: Challenges of Making Morphological Resources

Section 3: Community Considerations

Section 4: Existing Software

Section 5: Alchemist

Section 6: Conclusion

Section 1 describes the original intended purpose of Alchemist and how it has evolved into an effective tool for creating rich morphological resources. Section 2 discusses the minimum requirements for creating morphological descriptions and explains some methods that can be used to make these processes more efficient. Section 3 includes a summary of best practices suggested by Bird and Simons (2003) and a parallel list of responsibilities of community engineers and their products. Section 4 is a quick overview of different freeware applications whose scope of users intersects the scope of

¹ Electronic Metastructure for Endangered Languages Data

Alchemist's. Section 5 describes the implementation of Alchemist, including how its functions increase the efficiency of parsing morphemes from words.

Readers interested in using Alchemist as a tool for creating morphological descriptions should read Sections 2 and 5. They should also read the user documentation found on the project website: <http://linguistica.uchicago.edu/alchemyist.php>.

1 Motivations for Alchemist

Alchemist was designed to create gold standards. The need was originally specific to machine learning, but we soon discovered the broad usefulness of these morphological descriptions. Because the user can so easily parse words into morphemes, it was obvious that Alchemist would also be a great tool for morphological description in general. The original functionality of Alchemist remains intact, but new functions have been added to accommodate more kinds of morphological description. The scope of descriptions Alchemist can help create now includes: grammars, interlinear texts, and dictionaries. Linguists may also find uses for Alchemist as a tool for teaching morphology and field methods. The visual “cut and label” interface of Alchemist is excellent for demonstrations of morphology and morphotactics. An explanation of Alchemist would not be complete, however, without exposing the roots of the project. This section gives a background into the need for gold standards in machine learning.

Gold Standards for Machine Learning

The community of linguists and computer scientists who are interested in unsupervised learning of morphology has been growing in the past few years. In 2005, an unsupervised learning of morphology competition was staged²; it was the first gathering of researchers studying this problem. A few years before, there were only a handful of people who were researching in this area. The problem of automatically learning morphology is especially difficult given the numerous different ways of composing words across the world's languages.

There are two main options for evaluating the results of unsupervised learning: *expert inspection* and *gold standard comparison*. *Expert inspection* involves human evaluation of a tiny subset of learned data and extrapolation of this evaluation to create a score for the entire set. In the case of morphology, an expert would check the correctness of a few hundred learned morphological generalizations; the score for the entire analysis would be based on these. An expert cannot be expected to evaluate the entire output of an unsupervised learning application because the data set is required to be large in order to get a good analysis. *Gold standard comparison* is done when a subset of the unsupervised learning generalizations is compared against a human-created gold standard. Evaluation is accomplished by comparing the output of the learning algorithm with data in the gold standard wherever the input data and the gold standard intersect.

² No publication available. See <http://www.cis.hut.fi/morphochallenge2005/> for more information.

There are many advantages of gold standard comparison over expert inspection and there is one major disadvantage. The first advantage is that results of the test are instantaneous relative to the results of expert inspection. The comparison is automatic, and therefore the evaluation results can be viewed almost immediately after finishing a learning cycle. The second advantage is that comparisons can be performed on a larger subset of the data relative to expert inspection. Thus, the evaluation is more precise. The third advantage is that human work needed to create a standard is only required once and can be used repeatedly. With expert inspection, a human is required every time.

The disadvantage of using gold standards for measuring performance is that they are difficult to make and they are only available for a few languages (e.g. English, Finnish). It is important that unsupervised learning applications be tested on more than just a few sets of data (in this case, languages). If algorithms are tuned to the performance on just a few languages, they are likely to find, and be stuck in, a local maximum. When the application is then run on a different language, the performance will be much worse than with the test languages. Without a larger set of gold standards to work with, the researcher may only effectively evaluate his software on just a few languages.

The first version of Alchemist was implemented in October 2004 to facilitate the creation of gold standards. The first implementation was just a tool for cutting words into morphemes and giving each piece a type (i.e. root, affix) and a certainty score. Before Alchemist, the only available software for morphological gold standard creation and measurement was the Helsinki University of Technology Morphological Evaluation Gold Standard or Hutmegs (Creutz and Linden 2004), which was created only a few months before, and for the same reasons. The reader is referred to Section 4 for a short review of Hutmegs.

Subsequent versions of Alchemist included options for marking up words and morphemes with features, glosses, and phonetic representations. A *morpheme explorer* was introduced in version 2.0. This explorer provides a view of all morpheme instances marked in the word list. Allomorphs and morpheme instances can be merged into morphemes and the morphemes can be described. This new interface made Alchemist powerful enough to create rich morphological descriptions. After version 2.0, many additions were made in order to overcome challenges specific to creating morphological resources and to improve Alchemist as a tool for describing morphology. These challenges are described in the following section.

2 Challenges of Making Morphological Resources

One thing that is true across all languages is the existence of constituent classes and sub-classes. This is especially clear at the word level and below, with the exception of only a few languages (e.g. Chinese). Words can be classified by the positions they occur in and by the derivations and inflections they contain. Roots can be classified by the derivations

and inflections they take. Affixes can be classified by the roots they combine with and by their position relative to word edges. Programs that use unsupervised methods of learning morphology take advantage of these generalizations in language to learn how to parse the words. Surprisingly, many language description tools do not take advantage of these generalizations to speed up the process.

In this section I will list the minimum functions a program must perform in order to be useful for creating morphological descriptions. Following those, I will list the minimum functions a program must perform to create gold standards to be used as a metric for computer learning. Finally, I will discuss how these functions can be made more efficient and remain human-supervised by taking advantage of generalizations in language.

2.1 Requirements for Creating Morphological Descriptions

The minimum requirements of a morphological description creation program are:

- Must be able to parse words into roots and affixes
- Must be able to make multiple parses of the same word string
- Must be able to associate instances of the same morpheme from different words
- Must be able to associate allomorphs as members of the same morpheme
- Must be able to describe morphemes with semantic content

Parsing words into roots and affixes

In the morphological description creation program, there must be a way to parse words into roots and affixes. This may seem like a simple task, at first. Minimally, this is a matter of parsing words into morphemes that have only one piece and do not overlap with adjacent morphemes. However, even languages with relatively simple morphologies have overlapping morphemes, and many languages have multi-piece morphemes. In order for a tool to be useful for parsing the words of most languages, it must be able to parse morphemes that overlap orthographically. It must also be able to parse morphemes that have more than one non-contiguous piece.

Overlapping of morphemes in the orthography can be the result of phonotactics, can come from multi-tiered phonology, and may also just be a result of orthography. Because of the phonotactics of a language, the adjacent edge segments of contiguous morphemes may become one segment in speech. This may then be written as one segment. In Somali, for example, when a morpheme that ends in /l/ is adjacent to a morpheme that begins with /t/ the segments combine into [ʃ] (orthographically: 'sh'). The feminine determiner is the suffix /-ta/. In normal cases, both the stem final segment and the suffix initial segment are pronounced as expected. However, in the case of roots which end in /l/, neither [l] nor [t] is pronounced. These are likewise visible in the orthography:

- (1) a. xéeb + ta → xéebta 'the shore'
 hées + ta → héesta 'the song'

- b. $\acute{u}l + ta \rightarrow \acute{u}sha$ ‘the stick’
 $m\acute{e}el + ta \rightarrow m\acute{e}esha$ ‘the place’
 (Saeed 1999)

Also, languages may have more than one tier of phonology that are written separately. There are languages whose stems may be inflected by tonal changes alone. In this case, the morphemes would be overlapping in speech, which also becomes visible in some orthographies. The third person singular object pronoun in Yoruba is a morpheme with only tonal features. It is just a high tone that will co-occur with the final vowel of the inflected stem.

- (2) $mo\ r\acute{i} + H \rightarrow mo\ ri^3$ ‘I saw him/her/it’
 $mo\ je + H \rightarrow mo\ j\acute{e}$ ‘I ate it’
 $mo\ r\grave{a} + H \rightarrow mo\ r\check{a}$ ‘I bought it’
 (Welmers 1974)

The third way morphemes can overlap in the orthography is a result of the orthography itself. Creutz and Lagus (2004) refer to these as “fuzzy morpheme boundaries.” They occur when there is debate on which side of a segment an orthographic morpheme boundary should appear. Verbal morphology in English displays has “fuzzy” boundaries on verb roots that end in ‘e’. For example, the word *change* might be parsed in any of the following ways: *change -d*, *chang -ed*, or *change -ed*. A parsing tool should be able represent these cases in the output.

In addition to overlapping morphemes, there are also many languages that have morphemes with more than one non-contiguous piece. The most obvious are languages that use templatic morphologies, such as Arabic and others of the Semitic family. Templatic roots have more than one segment that may be separated by other morphemes, either templatic affixes or infixes. Another, less obvious, example of non-contiguous roots and stems are those that are split due to infixation. In some languages, affixes may be positioned in a place other than at the edge of a root or stem. In these cases, the root is broken into two separate pieces. An example from Toba Batak is given in (3). The *um-* infix splits the root into two parts.

- (3) $um + deŋg\acute{a}n \rightarrow dum\acute{e}ŋgan$ ‘better’
 $um + t\acute{i}bbo \rightarrow tum\acute{i}bbo$ ‘taller’
 $um + gok\acute{a}n \rightarrow gumok\acute{a}n$ ‘hotter’
 (Crowhurst 1998)

³ The 3S morpheme occurs as a mid tone when combined with a high toned root.

In each of these cases, the morphemes must be encoded as separate and non-contiguous pieces of the word. The output must represent the non-contiguous morphemes in a way that can be understood by other programs.

Multiple parses of the same word-string

Some word-strings have more than one morphological parse. In English, the orthographic word *polish* can be a verb, as in “I polish my brass knuckles every morning.” It can also be an adjective, as in “Chicago has the best Polish dogs.” A precise morphological description will account for both parses of this word. Therefore, the output of a morphological description tool must encode them both, either as separate parsed words or as separate parses of the same word-string.

Associate instances as the same morpheme

In the descriptive program there must be a way to associate morpheme instances from different words as the same morpheme. The program must not do this automatically. Sometimes, different morphemes have the same orthographic stream (e.g. *-s* in English is both the present tense verbal inflection and the plural noun inflection). If these decisions were made automatically, many morphemes would be misanalyzed. The program may make suggestions, but a human must make the ultimate decision.

Associate allomorphs as one morpheme

In the descriptive program there must be a way to associate allomorphs as one morpheme. Often the same morpheme can occur with different string instances. This can be due to a number of different factors. Mostly the phonotactics of the language drives change on the edges of the morpheme, but this is not the only reason. Because they are instances of the same morpheme, they should be associated together for a more complete description.

Describe morphemes with semantic content

In the descriptive program, there must be a way to describe the morphemes using features. Other than the phonetics and orthography, what sets morphemes apart from each other are their semantic content. A full description of a language’s morphology must include the semantics of each morpheme. Some morphemes are best described with semantic features and others are best described with glossing. Both options should be available.

2.2 Requirements for Creating Gold Standards

The minimum requirements of a general use gold standard creation program are these:

- Must be able to parse words into roots and affixes
- Must be able to make multiple parses of the same word string
- Must be able to give each word a certainty score
- Must be able to output the data in a format that is readable by other programs

The first two requirements are also requirements of morphological descriptions. They are explained in §2.1.

Certainty scores

Even an expert in the morphology of a language may run across words that are difficult to parse. In these cases, the certainty level of the word parse should be lower than normal. Three levels of certainty (CERTAIN, SOMEWHAT CERTAIN, UNCERTAIN) and a NOT SCORED setting are sufficient for comparing a gold standard to unsupervised learning output.

Output in a non-proprietary format

One advantage of using a gold standard as a metric is that it can be used by a community as a way to compare results. In order to do this, the gold standard must be readable by anyone who wants to use it. Therefore, the program output must be fully accessible to whoever might use it.

2.3 Using Language Generalizations to Speed Up the Process

Programs for creating gold standards and for describing morphology must at least meet the minimum requirements listed previously. There are many ways to exceed these requirements. Using the generalizations found in language can yield large time saving advantages without sacrificing human supervision. Listed below are five ways to shorten the time spent by program users that harness the power of language and text generalizations.

- Scrubbing out punctuation and other “garbage” from the text
- Stacking segments of words in columns
- Forward and backward alphabetical sorting
- Filtering words by regular expression
- Filtering by morphemes

Scrubbing

When gathering words to be used in a gold standard or description, the source is often a corpus containing punctuation, numbers, and markup. These are usually unwanted features in the words. If they are not explicitly removed, they will remain in the description or gold standard. These can be deleted as each word is cut into morphemes, but there is a better way. *Scrubbing* is the use of replacement rules to change and/or remove segments of a string. Each side of a rule uses a regular expression⁴. On the left side is a regular expression for identifying certain sub-strings of words. On the right side is a regular expression that may contain references to portions of the expression on the left hand side. This type of rule allows replacement or removal of any regular sub-string in any location relative to any other regular sub-string or relative to either edge of the

⁴ Regular expressions describe complex patterns in text in a concise way. They can be used to search for and modify patterns. For a good resource on regular expressions, visit <http://www.regularexpressions.info/>.

string. It also allows replacement or removal of any regular sub-string regardless of its location in the string.

Stacking segments in columns

Taking advantage of many of the patterns in morphology is as simple as lining them up vertically. Rather than work on one word at a time, the morphemes of large stacks of words can be marked with one click+drag+mark sequence. Different ways of lining the words up give quick access to different morphemes. These will be discussed in the next few parts of this section.

Forward and backward alphabetical sorting

What better place to look for morphemes than at the edges? There will always be a morpheme at each edge of a word. Stacking the words in columns lined up from an edge puts edge morphemes together if the words are sorted from the lined-up edge to the opposite edge. If words are sorted forward alphabetically (A-Za-z, lower indexed segments are given precedence) and are lined up from the first segment on, then all matching word-initial morphemes will be together in a vertical group. This gives one-mouse-sequence access to prefixes and non-prefixed roots. If words are sorted backward alphabetically (A-Za-z, higher indexed segments are given precedence) and are lined up from the last segment, then all matching word-final morphemes will be together in a vertical group. This gives one-mouse-sequence access to suffixes and non-suffixed roots.

Filtering words by regular expression

Not all morphemes occur at the edges of words but it is even possible to stack word internal patterns with filtering. To align word-internal morphemes, words that do not contain a target morpheme must be filtered out. The remaining words should then be lined up vertically by one edge of the target morpheme. This works well with a simple string match, but can also be made more powerful with a regular expression match. With regular expression filtering, the filter is not restricted to exact matches. This even makes it possible to filter for reduplications. Again, the alignment of patterns makes them more visible and easier to parse.

Filtering by morphemes

After some morphemes have already been marked, filtering by morphemes not only lines up the marked morphemes, it also lines up their neighboring morphemes. When these morphemes are lined up vertically, their neighbors will also be lined up. It is then easy to see and parse other patterns.

3 Community Considerations

A community without values and best practices for engineering solutions to their problems could actually be generating more problems with each new solution. Solutions to specific problems tend to introduce practices into the community. These may not be the best practice for all cases. When the power goes out in the Dominican Republic,

controlled intersections become a slow moving blob of cars and motorcycles trying to get through. There is no standard behavior for crossing an intersection without the traffic signal. Each driver tries to create a solution that is best for him, and everyone suffers the effects.

As Alchemist is a tool for language description, I will discuss the best practices of the Linguistics community in regard to language description. At this point, there is no widely accepted set of best practices. Some practices are in development and are encouraged by community projects, including Electronic Metastructure for Endangered Languages Data (E-MELD) and LINGUIST List. These are categorized into dimensions introduced by Bird and Simons' (2003) *Seven Dimensions of Portability for Language Documentation and Description*, which I will discuss in the first part of this section.

The difficulty facing our community comes because there is no central body regulating our practices, and for good reason. Because of this, however, it will take work and consensus among the community to adopt practices as "best." This puts the responsibility on the individual members of the community. In the second part of this section I will lay out the responsibilities of different classes of community members. I will focus especially on those who are creating tools for community use: *engineers*.

3.1 Seven Dimensions of Portability

Steven Bird and Gary Simons' (2003) *Seven Dimensions of Portability for Language Documentation and Description* set the groundwork for the E-MELD project's best practice suggestions. It expresses well-founded concerns over the *portability*⁵ of digital language documentation and description. The article identifies seven classes of problems that affect portability, and describes twenty-three specific problems under those headings. It also lists value statements for each problem and recommends best practices based on those statements. Following is a reproduction of Bird and Simons' best practice suggestions. These statements will be useful to establish the responsibilities of community members as well as the strengths and weaknesses of software tools created for use by the language description community. For a complete description of the problems, values, and practices see Bird and Simons' article in Language 79.

Content

COVERAGE: [T]he best practice is one that establishes a record that is sufficiently broad in scope, rich in detail, and authentic in portrayal that future generations can experience and study the language, even when no speakers remain.

ACCOUNTABILITY: [T]he best practice is one that provides the documentation that lies behind the description.

⁵ The re-usability of resources across time, platform (both software and hardware), scholarly community, and purpose (Bird & Simons 2003).

TERMINOLOGY: [T]he best practice is one that makes it easy to identify the comparable aspects of unrelated resources.

Format

OPENNESS: [T]he best practice is one that puts data into a format that is not proprietary.

ENCODING: [T]he best practice is one that fully documents what the character codes in the resource represent.

MARKUP: [T]he best practice is one that represents all of the information using a transparent descriptive markup, rather than in procedural code or in presentational markup.

RENDERING: [T]he best practice is one that supplements the information resource with all the auxiliary software resources that are needed to render it for display.

Discovery

EXISTENCE: [T]he best practice is one that makes it easy for anyone to discover that a resource exists.

RELEVANCE: [T]he best practice is one that makes it easy for anyone to judge the relevance of a resource based on its description.

Access

SCOPE OF ACCESS: [T]he best practice is one that makes it easy for users to obtain a complete copy of the resource.

PROCESS FOR ACCESS: [T]he best practice is one in which there is a clearly documented procedure by which users may obtain a copy of the resource.

EASE OF ACCESS: [T]he best practice is one that makes it possible for users to access some version of the resource regardless of physical location and access to computational infrastructure.

Citation

BIBLIOGRAPHY: [T]he best practice is one that makes it easy for electronic language documentation and description to be cited.

PERSISTENCE: [T]he best practice is one that archives resources with identifiers that are independent of location or file name.

IMMUTABILITY: [T]he best practice is one that makes it possible for users to cite particular versions that never change.

GRANULARITY: [T]he best practice is one that ensures each sub-item of a resource has a durable identifier.

Preservation

LONGEVITY: [T]he best practice is one that stores resources in formats that are likely to remain usable for generations to come.

SAFETY: [T]he best practice is one that stores copies of resources in multiple locations so as to ensure against the catastrophic loss of copies in just one location.

MEDIA: [T]he best practice is one that migrates resources to new physical and digital media before the ones they are stored in become unusable.

Rights

TERMS OF USE: [T]he best practice is one that clearly states the terms of use as part of the resource package.

BENEFIT: [T]he best practice is one that does not hinder the fair use of a language resource for scientific, educational, humanitarian, or other non-commercial uses.

SENSITIVITY: [T]he best practice is one that protects any sensitivities stipulated by the contributors.

BALANCE: [T]he best practice is one that clearly identifies the nature of a sensitivity and associates it with an explicit time frame.

(Bird & Simons 2003)

3.2 Community Member Responsibilities

The Linguistics community has three classes of member as they relate to language description. I will refer to them as creators, users, and engineers. *Creators* document and/or describe languages. *Users* use the documentation and descriptions. *Engineers* implement tools to facilitate the creation and use of resources. These three classes have different responsibilities to encourage and enforce the best practices of language description. In the following discussion, I will focus on the responsibilities of language description tools, like Alchemist, and refer to them as the responsibilities of engineers. Table (4) shows the responsibilities of engineers with regard to each of the seven dimensions of portability and their specific problem areas. Following the table, each is detailed and stated in regard to Bird and Simons’ suggested practices.

(4) Responsibilities of Community Engineers

Dimension	Problem	Level of Responsibility
-----------	---------	-------------------------

		Enforce	Encourage	Allow	Not Inhibit
Content	COVERAGE				X
	ACCOUNTABILITY			X	
	TERMINOLOGY		X		
Format	OPENNESS	X			
	ENCODING		X ⁶		
	MARKUP	X			
	RENDERING		X		
Discovery	EXISTENCE		X		
	RELEVANCE		X		
Access	SCOPE OF ACCESS				X
	PROCESS FOR ACCESS				X
	EASE OF ACCESS				X
Citation	BIBLIOGRAPHY		X		
	PERSISTENCE				X
	IMMUTABILITY				X
	GRANULARITY				X
Preservation	LONGEVITY	X			
	SAFETY				X
	MEDIA				X
Rights	TERMS OF USE			X	
	BENEFIT				X
	SENSITIVITY			X	
	BALANCE			X	

Content

COVERAGE: It is the responsibility of the engineer to *not inhibit* creators from “establishing a record that is sufficiently broad in scope, rich in detail, and authentic in portrayal that future generations can experience and study the language, even when no speakers remain.” There is no way for the engineer to ensure scope, detail, or authenticity of the data. This responsibility falls solely on the creator. The engineer should do no more than avoid inhibiting it.

ACCOUNTABILITY: It is the responsibility of the engineer to *allow* creators to “provide the documentation that lies behind the description.” A creator cannot be compelled to document his description, just as a programmer cannot be compelled to write comments in his code. As long as the means of doing so are readily available, the engineer has complied with this responsibility.

TERMINOLOGY: It is the responsibility of the engineer to *encourage* creators to “make it easy to identify the comparable aspects of unrelated resources.” If standard terminology is available, then the engineer should make it available to the creator. However, the use of terminology should not be enforced. Creators should be allowed to create their own

⁶ Best practices in ENCODING should be enforced when able. It is either not possible or not feasible yet, given current technology limitations.

terminology if needed. This can be achieved by supplying standard terminology as a default and allowing new terms to be added.

Format

OPENNESS: It is the responsibility of the engineer to *enforce* “putting data into a format that is not proprietary.” As a member of the community, the engineer should attempt to follow the same values as the creator and user, wherever possible. An engineer who allows output to be saved in any proprietary format risks propagating the problems the best practices are meant to address.

ENCODING: It is the responsibility of the engineer to *encourage* (and *enforce* when able) creators to “fully document what the character codes in the resource represent.” With current encoding technology, it is only possible to encourage using documented character codes. Even Unicode does not cover every character used. Instead, it supplies “private use” codes. When it is possible to use fully documented encoding, it should also be the responsibility of the engineer to enforce its use.

MARKUP: It is the responsibility of the engineer to *enforce* the “representation of all of the information using a transparent descriptive markup, rather than in procedural code or in presentational markup.” As with issues of openness, an engineer who allows output to be marked up in a non-transparent non-descriptive markup risks propagating the problems the best practices are meant to address. The engineer is best placed to use and promote this best practice. It is his duty, as a community member, to do so.

RENDERING: It is the responsibility of the engineer to *encourage* creators to “supplement the information resource with all the auxiliary software resources that are needed to render it for display.” It is especially important to encourage the creator to supply information about fonts when using special encoding. This can be done by asking about font information during metadata gathering steps.

Discovery

EXISTENCE: It is the responsibility of the engineer to *encourage* the creator to “make it easy for anyone to discover that a resource exists.” Of course, the engineer cannot always provide a way for making data public. He can make it easy to tag the description with metadata that would be helpful for discovery.

RELEVANCE: It is the responsibility of the engineer to *encourage* the creator to “make it easy for anyone to judge the relevance of a resource based on its description.” The engineer can facilitate the addition of metadata to a language description. Metadata (data about data) is helpful for judging relevance.

Access

SCOPE OF ACCESS: It is the responsibility of the engineer to *not inhibit* the creator from “making it easy for users to obtain a complete copy of the resource.” There is no way to enforce the total availability of all language descriptions without treading on the rights of

some informants and language communities. Given the sensitivities creators must take into account, this should be their sole responsibility.

PROCESS FOR ACCESS: It is the responsibility of the engineer to *not inhibit* the creator from “clearly documenting [the] procedure by which users may obtain a copy of the resource.” The engineer will never be required to store the data of every creator who uses his tool, even though it may be possible in some cases (e.g. web services). Providing public access to descriptions is the responsibility of the creator and will be dependent on terms of use and sensitivities that must be adhered to.

EASE OF ACCESS: It is the responsibility of the engineer to *not inhibit* the creator from “making it possible for users to access some version of the resource regardless of physical location and access to computational infrastructure.” Again, the engineer cannot be held responsible for the distribution of the resource. This is the responsibility of the creator.

Citation

BIBLIOGRAPHY: It is the responsibility of the engineer to *encourage* the creator to “make it easy for electronic language documentation and description to be cited.” This should also be a best practice for software engineering in general. Data should bear the mark of its creator in case questions need to be raised or in case there is a need to cite the work. The engineer can help with this by requesting bibliographic information in a metadata collection stage.

PERSISTENCE: It is the responsibility of the engineer to *not inhibit* the creator from “archiving resources with identifiers that are independent of location or file name.” This is the sole responsibility of the creator of the archived resource. Engineers cannot ensure the persistence of data without storing it themselves.

IMMUTABILITY: It is the responsibility of the engineer to *not inhibit* creators from “making it possible for users to cite particular versions that never change.” Again, this is the sole responsibility of the creator of the resource. The same reason applies.

GRANULARITY: It is the responsibility of the engineer to *not inhibit* creators from “ensuring each sub-item of a resource has a durable identifier.” Here, there are three cases. In the case where the resource spans multiple files created separately, the engineer has no power to help provide durable identifiers within the set. In the case where the resource spans multiple files created together, the engineer does have the ability to provide durable identifiers within the set if, and only if, the resource is only edited in the software created by the engineer. Otherwise, there is no way to ensure all links are sound. The third case is where the resource is completely contained on one file. In this case, the engineer can ensure that all internal links are sound only if the file is edited in software created by the engineer. Again, if it is not, there is no assurance.

Preservation

LONGEVITY: It is the responsibility of the engineer to *enforce* “storing resources in formats that are likely to remain usable for generations to come.” If the engineer enforces the best practices for formatting (i.e. openness and markup), then longevity should automatically follow. The suggested best practices of openness and markup are meant to increase the longevity of resources.

SAFETY: It is the responsibility of the engineer to *not inhibit* creators from “storing copies of resources in multiple locations so as to ensure against the catastrophic loss of copies in just one location.” The engineer cannot be expected to store even one copy of the data. This is the sole responsibility of the creator.

MEDIA: It is the responsibility of the engineer to *not inhibit* creators from “migrating resources to new physical and digital media before the ones they are stored in become unusable.” Other than backward compatibility issues with new software versions, the engineer is not responsible for data that has already been processed/created by the tools he creates. Migrating data is the responsibility of the creator.

Rights

TERMS OF USE: It is the responsibility of the engineer to *allow* creators to “clearly state the terms of use as part of the resource package.” A creator cannot be compelled to give the terms of use or any other rights information. Nor is it always the case that the creator knows the rights. The engineer can only provide a way to include this information with the data.

BENEFIT: It is the responsibility of the engineer to “not hinder the fair use of a language resource for scientific, educational, humanitarian, or other non-commercial uses.” This should never be a concern. Engineers (and users) will err on the side of over-usage rather than fall short of the fair use of resources. Creators are usually the guilty party in hindering the fair use of language resources.

SENSITIVITY: It is the responsibility of the engineer to *allow* creators to “protect any sensitivities stipulated by the contributors.” All possible sensitivities cannot be accounted for in software implementation. The engineer can only provide notes/remarks space for the creator to include this information.

BALANCE: It is the responsibility of the engineer to *allow* creators to “clearly identify the nature of a sensitivity and associate it with an explicit time frame.” Again, the engineer can only provide space for this information to be input by the creator.

4 Existing software

The following software titles are all free, or otherwise easily accessible. With the exception of Hutmegs, they can all be used to create morphological descriptions of language. This is not meant to be a complete survey or review of these titles but is only a

short summary of titles similar to Alchemist. I will only outline the scope of intended use of the applications and list their strengths and weaknesses as they apply to morphological description or gold standard creation.

4.1 Spreadsheet Applications (e.g. Microsoft Excel⁷, OpenOffice.org Calc⁸)

Spreadsheet applications are made to perform calculations and create useful graphs from data. They can also be used for creating and manipulating tables of any data that can be encoded as text. Spreadsheet applications are probably the most widely used tool for language description because of familiarity and availability.

Strengths

- Familiar
- Data possibilities are less limited
- Handles all major encodings
- Table layout allows the creator to view many (possibly) related words at once

Weaknesses

- Not linguistically intelligent
- Sorting is difficult
- Filtering is very difficult
- Difficult to create relations

4.2 The Field Linguist's Toolbox (Toolbox)⁹

Toolbox is a field linguistics tool that lays out texts and the working lexicon to make linguistic patterns more visible. The intended audience is field linguists who do not already have a command of the language they are studying. Toolbox enables the linguist to create a corpus, grammar, and lexicon simultaneously from small amounts of text. It interlinearizes the texts with glosses and translations automatically.

Strengths

- Search function matches any sub-string in the lexicon
- Allows definition of morphophonemics
- Word formulas (grammar rules) are a handy way to disambiguate words
- Automatically interlinearizes the glosses and translations with the text

Weaknesses

- Word types are analyzed one at a time.

⁷ <http://office.microsoft.com/>

⁸ <http://www.openoffice.org/>

⁹ http://www.sil.org/computing/catalog/show_software.asp?id=79

- Toolbox makes some unsupervised guesses during interlinearization. These must be corrected by the user, if he notices them in the first place. These can be minimized by using word formulas.

4.3 Helsinki University of Technology Morphological Evaluation Gold Standard (Hutmegs)¹⁰

Hutmegs was created as a tool for measuring the performance of unsupervised morphological analyses. It is not a tool for facilitating the description of language. Hutmegs is a computer generated gold standard of English and Finnish that uses human-created descriptive data from other sources.

Strengths

- Evaluates 1.4 million Finnish words and 120,000 English words
- Handles overlapping morphemes (“fuzzy morpheme boundaries”)
- Handles allomorph and morpheme segmentations of words

Weaknesses

- Limited to English and Finnish
- Gold standards are created by a computer, using human-created descriptions
- Not completely free, must purchase licenses

4.4 Kura¹¹

Kura is a field linguistics tool designed to allow multiple users working on multiple languages to collaborate in a collective space. Its description capabilities include creation of interlinear texts, glossing of lexemes, phonetic transcription of lexemes, and creation of inter- and intra-lingual relations between lexemes.

Strengths

- Extensible, both for the CS trained user and the non-CS trained user
- Unicode ready
- One database can store data from multiple languages and accepts input from multiple users while modularizing the work into different projects.
- Imports intelligently from interlinear texts
- Can search for similarity between words in different languages

Weaknesses

- Does not handle overlapping and non-contiguous morphemes
- Word types are parsed into morphemes one at a time
- Unicode only

¹⁰ <http://www.cis.hut.fi/projects/morpho/>

¹¹ <http://www.ats.lmu.de/kura/>

4.5 Interlinear Text Editor (ITE)¹²

ITE's main functionalities are transcription and interlinearization from recordings. The target audience is field linguists, though the incorporation of audio files makes it useful for non-field language learning.

Strengths

- Built in concordance viewer
- Suggests analyses of new texts based on previous supervised learning
- No download necessary (except Java Virtual Machine), can be run from the internet
- Sorting by transcription and gloss
- User defined sort-orders for transcription and gloss

Weaknesses

- Word types can only be analyzed one at a time

5 Alchemist

Alchemist is designed to facilitate the process of making gold standards and morphological descriptions of language. Alchemist harnesses the power of language generalizations through the use of word tables that can be sorted and filtered in ways that present reoccurring patterns to the user. Other utilities for making the description process more efficient and for making the data useful have been built into Alchemist. These include functions for pre-processing the data to functions for exporting the data differently for different uses. The most useful functions of Alchemist are described in this section.

5.1 Marking roots and affixes

The main purposes of Alchemist are to create gold standards of morphology and to create morphological descriptions of language. Both of these require that words be parsed into morphemes. This is the most basic functionality of Alchemist, and it does it very well. Care has been taken to ensure that these analytical steps can be done as quickly and efficiently as possible. The user interface of Alchemist exploits language generalizations, allowing the user to see them and analyze groups of words in one mouse stroke. This is done without sacrificing the security of human supervision or creating a reliance on the software to find the generalizations.

The strength of tables

¹² http://michel.jacobson.free.fr/ITE/index_en.html

Tables provide a view on sorted word lists that makes it easy to see reoccurring patterns. In Alchemist, the word collection is laid out in a table with one word per row. The first cell of the row contains the whole word for increased readability. The content of remaining cells of a row depends on the sort order and filtering of the word collection. Each cell may possibly be empty or contain only one character from the word. The characters of the word will appear in order with no blank cells in between. The specifics of their starting and ending cells will be discussed later with each sorting and filtering option.

Putting the characters of each word in rows and stacking the rows in tables has the effect of aligning reoccurring sub-strings on top of each other, in columns. Columns of matching patterns have a way of attracting the eye, thus pointing the user of Alchemist toward language generalizations. Because they are laid out together in a table, patterns that are morphemes can be marked at the same time by selecting the cells that contain the pattern and demarking them with a key stroke or by clicking a button. In most cases, a large portion of all instances of a morpheme can be parsed with just one mouse stroke (click-drag-release) and one key stroke (*Ctrl-A* for affixes or *Ctrl-R* for roots). In some cases, the entire set of a morpheme's instances can be parsed. The sorting and filtering of the word collection determine which sub-strings of a word can participate in the pattern stacking.

Forward alphabetical sorting

Forward alphabetical sorting is what is found in the phone book. It is what is referred to with the term "Alphabetical Order." In this order, priority is given to earlier characters in the word and the order is A-Za-z. When an unfiltered word collection in Alchemist is sorted into forward alphabetical order, each row contains one word. The first cell of a row has the entire word. The second cell of the row contains the first character of the word. The number of columns is based on the length of the longest word, therefore only the longest word will fill every cell in its row. All other words will fill only $L + 1$ rows, where L is the number of characters in the word.

(5) Forward alphabetical sorting

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
128	adjourned	a	d	j	o	u	r	n	e	d							
129	adjourning	a	d	j	o	u	r	n	i	n	g						
130	adjournment	a	d	j	o	u	r	n	m	e	n	t					
131	adjust	a	d	j	u	s	t										
132	adjusted	a	d	j	u	s	t	e	d								
133	adjustment	a	d	j	u	s	t	m	e	n	t						
134	adjustments	a	d	j	u	s	t	m	e	n	t	s					
135	administration	a	d	m	i	n	i	s	t	r	a	t	i	o	n		
136	administration's	a	d	m	i	n	i	s	t	r	a	t	i	o	n	'	s
137	administrative	a	d	m	i	n	i	s	t	r	a	t	i	v	e		
138	administrator	a	d	m	i	n	i	s	t	r	a	t	o	r			
139	administrators	a	d	m	i	n	i	s	t	r	a	t	o	r	s		

The first character of every word is lined up in the second column. Because the words are ordered alphabetically, any sub-string that begins with the first character of a word will

be stacked in columns with its matching word-initial sub-strings. This makes it easy to identify and parse word-initial morphemes, including prefixes and roots.

Backward alphabetical sorting

In backward alphabetical sorting, priority is given to later characters in the word, but the order is still A-Za-z. Thus, the word ‘zebra’ will come before the word ‘baklava’ in a backward alphabetical sorted list. When an unfiltered word collection in Alchemist is sorted into backward alphabetical order, each word occupies one row. The first cell of the row contains the entire word. The last cell of the row contains the last character of the word. Because the number of columns is based on the longest word in the collection, the second cell for all but the longest word(s) will be empty.

(6) Backward alphabetical sorting

	1	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
5034	integration						i	n	t	e	g	r	a	t	i	o	n
5035	decoration						d	e	c	o	r	a	t	i	o	n	
5036	corporation						c	o	r	p	o	r	a	t	i	o	n
5037	concentration				c	o	n	c	e	n	t	r	a	t	i	o	n
5038	registration					r	e	g	i	s	t	r	a	t	i	o	n
5039	administration			a	d	m	i	n	i	s	t	r	a	t	i	o	n
5040	demonstration			d	e	m	o	n	s	t	r	a	t	i	o	n	
5041	saturation						s	a	t	u	r	a	t	i	o	n	
5042	compensation					c	o	m	p	e	n	s	a	t	i	o	n
5043	sensation							s	e	n	s	a	t	i	o	n	
5044	conversation					c	o	n	v	e	r	s	a	t	i	o	n
5045	citation									c	i	t	a	t	i	o	n

The final character of every word is lined up in the final column. Ordering the words backward alphabetically allows any sub-string that ends with the final character of a word to be stacked in columns with its matching word-final sub-strings. When the word list is in backward alphabetical order, it is easy to identify and parse word-final morphemes, including suffixes and roots.

Filtering

When the word collection is filtered in Alchemist, every remaining word contains either a specific sub-string or one of a set of sub-strings. Again, each word occupies only one row. The first cell of each row contains the entire word, for convenience. The number of columns in a filtered table is not based on the longest word. The number of columns is equal to one plus the length of the longest filter string, plus the maximum amount of characters that occur before the filter string in a word, plus the maximum amount of characters that occur after the filter string in a word.

The actual cells that are occupied by the word depend on how the collection is sorted. If the words are filtered and sorted forward alphabetically, sort priority goes to the first letter following the filter pattern and continues to the end of the word. The last character of the filter pattern in every word will be lined up in the same column. This odd way of sorting and stacking the words puts all matching sub-strings that follow the filter pattern in a stack. Thus, the user can see and parse both the patterns that follow the filter pattern

as well as the filter pattern itself. For example, if the filter is the first person plural ergative prefix in K'ichee' verbs 'qa*', the filter will match 'qa' and 'q'. Inflected roots and stems will be sorted in forward alphabetical order:

(7) Filtered by morpheme, forward alphabetical sort

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
15	keqata		k	e	q	a	t	a							
16	xatqata	x	a	t	q	a	t	a							
17	xqatij			x	q	a	t	i	j						
18	kaqatz'ib'aj		k	a	q	a	t	z	'	i	b	'	a	j	
19	keqatz'ib'aj		k	e	q	a	t	z	'	i	b	'	a	j	
20	xqatz'ib'aj			x	q	a	t	z	'	i	b	'	a	j	
21	xqatzalb'a			x	q	a	t	z	a	l	b	'	a		
22	keqatzijob'ej		k	e	q	a	t	z	i	j	o	b	'	e	j
23	xqatzijob'ej			x	q	a	t	z	i	j	o	b	'	e	j
24	kixqatzukuj	k	i	x	q	a	t	z	u	k	u	j			

A more natural way of aligning the characters would be to line them up by the first character of the filter pattern. However, unless all the filter patterns are the same length, there is no guarantee that any other patterns will be found in stacks, either before or following the filter pattern. Therefore, only the filter pattern is guaranteed to be stacked by pattern. Because this is less useful, Alchemist uses the alignment method described previously. The symmetric alignment is used when words are sorted backward alphabetically.

If the words are filtered and sorted backward alphabetically, sort priority goes to the last letter before the filter pattern and continues to the beginning of the word. The first character of the filter pattern in every word will be lined up in the same column. This sorting and stacking method puts all matching sub-strings that precede the filter pattern in a stack, as well as the filter pattern. This enables the user to see and parse both the patterns that precede the filter pattern and the filter pattern. In the same K'ichee' example, sorting backward alphabetically lines up the absolute prefixes, which always precede ergative prefixes:

(8) Filtered by morpheme, backward alphabetical sort

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
7	kaqaraq		k	a	q	a	r	a	q						
8	kaqatz'ib'aj		k	a	q	a	t	z	'	i	b	'	a	j	
9	keqata		k	e	q	a	t	a							
10	keqatz'ib'aj		k	e	q	a	t	z	'	i	b	'	a	j	
11	keqatzijob'ej		k	e	q	a	t	z	i	j	o	b	'	e	j
12	katqaloq'oj	k	a	t	q	a	l	o	q	'	o	j			
13	xatqata	x	a	t	q	a	t	a							
14	kixqatzukuj	k	i	x	q	a	t	z	u	k	u	j			
15	kixqil	k	i	x	q	i	l								
16	xixqariqo	x	i	x	q	a	r	i	q	o					

Overlapping morphemes

In Alchemist, morphemes are not restricted to adjacency. Any two morphemes or substrings of two morphemes may occupy the same word-string positions. This can be done in Alchemist when the word collection is sorted in any order and with any filters applied. Parsing morphemes only requires one more simultaneous step than marking non-overlapping morphemes. Holding down the ‘o’ key when marking roots or affixes will cause new morphemes to overlap any that already occupy the selected cells. This is represented visually in the table by painting the top half of the cell with the corresponding color of one morpheme and the bottom half of the cell with the corresponding color of the other morpheme.

(9) Overlapping morphemes

	1	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
481	changed										c	h	a	n	g	e	d
482	unchanged								u	n	c	h	a	n	g	e	d
483	exchanged								e	x	c	h	a	n	g	e	d
484	ranged										r	a	n	g	e	d	
485	challenged							c	h	a	l	l	e	n	g	e	d
486	fringed										f	r	i	n	g	e	d
487	charged										c	h	a	r	g	e	d

Marking non-contiguous morphemes

Also, morphemes are not restricted to one piece. In Alchemist, a morpheme can be composed of more than one non-contiguous pieces of a word. Like all other morpheme parsing, this is not limited to any specific sort order or filtering of the word collection. Creating a non-contiguous morpheme can be done in two ways. First, if the entire substring span, which will cover the morpheme, is already marked as a root or affix, splitting the existing morpheme with a new morpheme, or clearing one or more cells, will create two pieces that are associated as the same morpheme. An alternate way is to mark two non-contiguous morphemes of the same type in the same word and choosing “Bind Morpheme Pieces” from the “Morphology” menu. Non-contiguous morphemes will be represented visually in the table by matching colors on their pieces.

(10) Non-contiguous morphemes

	1	2	3	4	5	6	7	8	9	10	11
1	دَرَسَ			س	ر	د					
2	دَرَسَتْ	ت		س	ر	د					
3	دَرَسْنَا	ا	ن	س	ر	د					
4	دِرَاسَةٌ	ة		س	/	ر	د				
5	مَدْرَسَةٌ	ة		س	ر	د				م	
6	يُدْرَسُ			س	ر	د					ي

5.2 Word filtering options

Filtering in Alchemist limits the displayed words in the collection table by user specified criteria. All words that do not fit the criteria are hidden from view. They still exist in Alchemist’s memory and can be included in the table by changing the filter or by choosing to display all the words. Limiting the visible collection in this way allows the

user to declutter the list and concentrate on a specific pattern or patterns. Filtering is especially helpful when searching for morphemes that can be found in positions other than the edges of words. When filters are applied in Alchemist, the words are lined up by one of the edges of the pattern, depending on the sort order. This makes the pattern highly visible and can also make adjacent patterns more visible if the sorting is based on the filter. There are two kinds of filters in Alchemist: by regular expression and by morpheme. There are also three filter settings: show analyzed words only, show unanalyzed words only, and show all words.

Filter by regular expression

Regular expressions are a powerful tool for locating patterns in strings. Regular expressions do not only define simple patterns. They may match non-contiguous patterns, patterns in which later characters depend on earlier parts of the pattern, and many other interesting expressions. Most kinds of patterns of interest to linguists can be found with regular expressions, including reduplication, templatic morphemes, and morphemes with assimilating segments. To filter the collection by a regular expression, there is a field in the “Collection Tools” tab called “Filter: (regular expression).” The user should enter the regular expression matching what patterns are of interest and press the “Enter” key or click the “Show Filtered” button. The following are some useful filters, including those mentioned above:

(11) Useful regular expression filters

Regular Expression	Matches
<code>ing</code>	Matches words containing the string ‘ing’ anywhere in the word.
<code>ing\$</code>	Matches words containing the string ‘ing’ at the end of the word only.
<code>^i(n m r)</code>	Matches words containing either ‘in’, ‘im’, or ‘ir’ at the beginning of the word.
<code>k\w*t\w*b</code>	Matches words containing ‘k’, ‘t’, and ‘b’, in that order, possibly with intervening characters.
<code>(\w{2,})\1</code>	Matches reduplications of length 2 or more with no intervening characters. ‘\1’ is a backreference that refers to the string matching the first set of parenthesis.
<code>(\w{2,})\w*\1</code>	Matches reduplications of length 2 or more that may have intervening characters.

(Wiener 2005)

Filter by morpheme

Another generalization of language that can be harnessed to speed up the description process is the order of morphemes. Morphemes tend to occupy the same, or a similar, position across the set of words they occur in. Filtering by morpheme removes all words from view that do not contain the target morpheme or morphemes. The words are ordered according to the sorting options chosen. The characters of the words are lined up based on one of the edges of the filter morpheme. The sort order and alignment of the filtered words lines up other morphemes that are adjacent to the filter morpheme. This makes it

easy to see and parse morphemes that immediately precede or follow the filter. For an example of this, see screenshots (7) and (8).

Filter settings

The filtered word collection can be further pared by whether or not the words already have analyzed morphemes. Cutting out the previously analyzed words lets the user concentrate on what still needs to be done. Removing the unanalyzed words allows the user to verify groups of analyses together. The third setting is to display both analyzed and unanalyzed words—the default setting.

5.3 Analysis certainty

When using a description as a gold standard, it is important to know the analysis as well as the level of certainty of the analysis. It is not always clear what analysis is best, even to a human. A gold standard evaluation method may give partial credit for different levels of certainty. There are four settings for analysis certainty in Alchemist: *Not scored*, *Certain*, *Somewhat certain*, and *Uncertain*. The analysis certainty of each word in the collection is initialized to *Not scored*. When either a root or affix is parsed in a word, the word's certainty will automatically be set to *Certain*. This should be the choice in a majority of cases.

5.4 Word scrubbing

Scrubbing is the use of replacement rules to change and/or remove segments of all the word-strings of the collection. Scrubbing is the quickest way to remove unwanted characters from the words. It is most efficient to scrub the word collection as it is read in from a corpus, but Alchemist allows the user to scrub the word collection at any point in the description process. Scrubbing will only affect words that are currently included in the collection table. Therefore, if the collection is filtered, the words containing the filter pattern are the only words that may be affected by scrubbing.

Alchemist gives the user a few simple scrubbing options in an easy to use dialog. The dialog is also an editor for creating advanced scrubbing rules and for changing the order in which both the simple and advanced rules are applied. Scrub options can be saved to a file and loaded when needed. The simple rules include a rule for making all characters lower-case, a rule for removing all numbers, and rules for removing user defined characters from the beginning, middle and end of words. The advanced rule editor provides two fields for creating a new rule: one is for the “Find” regular expression and the other is for the “Replace with” regular expression. Creating a new rule adds it to a list of rules that may include simple rules. The list may be ordered according to user preference.

5.5 Multiple parses of the same word-string

In Alchemist, any number of duplicates can be made of the same word. These duplicates can then be analyzed differently. Alchemist allows as many different parses of the same

word as a user wants to create (and as memory allows). The process for duplicating a word is simple. The user must select any part of the word to be duplicated then press *Ctrl-D* or choose “Duplicate word” from the “Morphology” menu.

5.6 Encoding and text-directionality

A tool for creating language descriptions that could not read and display most of the world’s written languages would be almost useless. One problem a description tool could encounter is the different encodings of characters. Alchemist supports ASCII encodings as well as 8-bit (UTF-8) and 16-bit (UTF-16) Unicode encodings. Even UTF-16 does not cover every character of every script, but it is the closest thing available right now. Alchemist detects the encoding automatically and translates everything to UTF-16. Another problem is right-to-left scripts. Though the Unicode standard displays right-to-left strings correctly, the table setup of Alchemist will order the characters backwards. Alchemist has a setting that reorders the characters in the cells so that right-to-left scripts appear in the correct order.

5.7 Adding morpheme features

In Alchemist, morphemes can be described with glosses and features. The complete set of default features are drawn from the General Ontology for Linguistic Description (GOLD)¹³ (Farrar, Lewis, and Langendoen 2002). Not every node of GOLD is relevant to morphological description, therefore the complete ontology is not included in Alchemist. The features available in Alchemist are from the *MorphoSyntacticFeatures* and *SyntacticUnit* sub-trees of the ontology. These features are organized in a tree-structured list box. Every leaf feature has a check-box. When the user selects a morpheme in the morpheme explorer, any previously chosen feature for that morpheme will be checked. Checking any other features will associate the checked features to the selected morphemes.

The complete set of features drawn from GOLD is not immediately available to the user. In order to remove clutter on the screen, the feature box is populated with a default set of the features. However, Alchemist provides a simple way to edit the list of available features. Choosing to edit the morpheme features (either in the “Edit” menu or with the “Edit features” button) opens a dialog with a similar feature list box. The dialog will display every feature available from GOLD. Checking the box next to a feature will add it to the available feature list. There are also options to add new features and feature groups. This allows the user to include features that are not available from GOLD due to their specificity to a particular language.

¹³ An ontology is a formal model of basic categories and relationships. GOLD is an ontology for linguistics, primarily useful in linguistic description. For more information, the reader is referred to <http://www.linguistics-ontology.org/>.

5.8 Outputs

Alchemist has two different data export options and also saves the data in an XML format. The first data export option formats the data appropriately for use as a gold standard. The second export option creates an interlinear text from the morphological data and a text file newly provided as input. Finally, all descriptive data can be saved to an XML document that can be read back into Alchemist.

Gold standard export

Alchemist has a gold standard export wizard that exports data to a plain-text file. The wizard helps choose the specific data to export and what the line format should be. Files in which each word is separated by a line break are simpler to parse and use as a gold standard. The exported data is limited to the parsed word and the morpheme features of each parse.

Interlinear text export

Interlinear texts can be generated directly from the descriptive data created in Alchemist given a text file in the same language. The text file will most often be the same file used to generate the word list, but using the same text is not a requirement. Alchemist matches the words from the text file with words in the collection and produces an interlinear text in XML format.

XML output

All descriptive data is saved into an XML file. This file can be read back into Alchemist when the user resumes description of the language. Because the output is in an open format, any other application can be programmed to parse the Alchemist data. The output includes metadata about the author, subject language, and document; a list of all morpheme features; the entire list of morphemes, with references to features; and the complete word collection with references to morphemes.

6 Conclusion

Alchemist is a very efficient tool for parsing words into morphemes and describing the morphemes with glosses and features. Every part of the process is made easier by functions designed to take advantage of generalizations in language and the intended use of the output. Alchemist increases the efficiency of the human researcher and shortens the time of creating grammars, interlinear texts, and dictionaries.

Alchemist does not make guesses about the structure. Instead, the data are organized so like-patterns appear together, making them easier to see and parse from the word list. Doing so ensures the description will be as reliable as the expert who created it. Alchemist does not remove the human element of language description.

Alchemist comes at a time when there is a surge in the number of computational linguists studying morphology and a lack of morphological descriptions. It is critical to have

human-created gold standards with which to test systems on many different languages and types of languages. Alchemist shortens the production time of gold standards.

It is important to consider the impact of following (or not following) community standards when creating software. There are a number of best practices that the engineer is responsible to follow in order to allow other members to also comply with the best practices of the community. Alchemist is true to the values of the community.

There are many software tools created to simplify the gathering and storage of language documentation and for creating language descriptions. The strengths and weaknesses of these programs are related to their scope and intended audience. Alchemist stands alone as an efficient tool for creating morphological descriptions.

In the time it took to read this paper, many thousands of words could have been parsed, and their morphemes described, using Alchemist.

Resources

- Bird, Steven, and Gary Simons. (2003). Seven Dimensions of Portability for Language Documentation and Description. *Language* 79. Issue 3. 557–82.
- Clark, Mary. (1990). *The Tonal System of Igbo*. Publications in African Languages and Linguistics. Foris Publications USA Inc.: Providence, RI.
- Creutz, Mathias, and Krister Lindén. (2004). *Morpheme Segmentation Gold Standards for Finnish and English*. Publications in Computer and Information Science, Report A77, Helsinki University of Technology, October.
- Crowhurst, Megan. (1998). Um Infixation and Prefixation in Toba Batak. *Language* 74. Issue 3. 590–604.
- Edwards, Rodney. Personal Communication. 11 May 2006.
- E-MELD Grant Proposal. (Date Unknown). E-MELD Website. Retrieved April 17, 2006, from <http://emeld.org/documents/E-MELD.html>.
- Farrar, Scott and Terry Langendoen. (2003). A linguistic ontology for the Semantic Web, *GLOT International* 7(3), 97–100.
- Farrar, Scott and William Lewis. (2005). *The GOLD Community of Practice, An Infrastructure for Linguistic Data on the Web*. Summer 2005 E-MELD Workshop on Morphosyntactic Annotation and Terminology: Linguistic Ontologies and Data Categories for Linguistic Resources, July 1–3, Radcliffe Institute, Harvard University.
- Farrar, Scott, William Lewis, and Terry Langendoen. (2002). An ontology for linguistic annotation. *Semantic Web Meets Language Resources: Papers from the AAAI Workshop, Technical Report WS-02-16'*, AAAI Press, Menlo Park, CA, pp. 11–19.
- Par Sapón, María and Can Pixabaj, Telma (2000). *Ujunamaxiik ri k'ichee' ch'ab'al = Variación dialectal en k'ichee'*. Proyecto de Investigación Lingüística de Oxlajuuj Keej Maya' Ajtz'iib'. Guatemala: Cholsamaj.
- Saeed, John. (1999). *Somali*. Vol. 10. London Oriental and African language library. John Benjamins North America: Philadelphia.
- Welmers, William E. (1974). *African language structures*. University of California Press: Berkeley, CA.
- Wiener, Alison. (2005). *Alchemist v2.0: A GUI-based tool for analyzing morphemes and creating morphological gold standards in XML format*. Available online as of May 11, 2006 from http://linguistica.uchicago.edu/alchemistv2_0_final.pdf.