Mrph: a Morphological Analyzer

Noah Evans
noah-e@is.naist.jp

ABSTRACT

Developing tools for Natural Language Processing is hard, requiring careful tuning of statistical models and data processing optimization. It's even harder given the many competing and incompatible tools, encodings and data sets in use in modern NLP research.

To implement new tools researchers have to reimplement or port the previous tools, using time for development that could be better spent doing productive research. There have been attempts to make portable, flexible low level analysis systems, notably Freeling, that incorporate a flexible NLP tool chain in a language independent way that can be easily incorporated into other tools and workflows.

We present a new morphological analyzer, mrph, which attempts to implement a language independent morphological analyzer both a viable vehicle for research and the day to day use. The system is divided into modules, written with native support for utf8 , and uses a shell and pipeline syntax that is semantically identical across systems. It is also written in a statically typed language with module support, allowing it to dynamically load and discard language resources at will. This allows mrph to change the processed language on dynamically, giving it potential for irregular data sets like the web.

Introduction

Mrph is a morphological analyzer written in Limbo for inferno.

A morphological analyzer is a tools for taking a sentence and breaking it up into its component morphology, a set of terms describing the implicit structure of the sentence.

For instance the sentence:

日本語を教える人と日本語を学習する人がともに楽しめるポータルサイトです。

when tokenized and classified by a morphological analyzer becomes:

```
日本語    ニホンゴ    日本語    名詞-一般
を        ヲ          を        助詞-格助詞-一般
教える    オシエル    教える    動詞-自立    一段          基本形
人        ヒト        人        名詞-一般
と        ト          と        助詞-並立助詞
日本語    ニホンゴ    日本語    名詞-一般
を        ヲ          を        助詞-格助詞-一般
学習      ガクシュウ  学習      名詞-サ変接続
する      スル        する      動詞-自立    サ変・スル    基本形
人        ヒト        人        名詞-一般
が        ガ          が        助詞-格助詞-一般
ともに    トモニ      ともに    副詞-一般
楽しめる  タノシメル  楽しめる  動詞-自立    一段          基本形
ポー      ポー        ポー      名詞-固有名詞-地域-一般
タル      タル        タル      名詞-接尾-助数詞
サイト    サイト      サイト    名詞-一般
です      デス        です      助動詞        特殊・デス    基本形
。        。          。        記号-句点
```

This provides an annotation that allows the sentence to be dealt with by the user and do what the user wants when they want to do greater amounts of research. In the case of this Japanese sentence is provides the pronounciation, the uninflected form of the verb, and its inflection type in each column respectively.

This information and annotation provided by the analysis forms the basis for creating solutions to larger problems in natural language problems, including syntax tree parsing and anaphora resolution.

Given their importance to other tasks, good morphological analyzers are a foundational tool for NLP researcher, much other research depends on it. This means that a lot of effort goes into optimizing the performance and accuracy of different analyzers.

The streaming nature of the morphological analyzer's task(i.e. a stream of input sentences each transformed into a set of moprhological tokens annotated with linguistic information) coincides nicely with the unix piped workflow, which connects small tools using pipelines provided by the operating system.

However, despite this natural affinity morphological analyzers are rarely inplemented as software tools. This happens for a few reasons, primarily one of portability. Given the importance of morphological analyzers to linguistic analysis and the popularity developers and researchers make special effort to implement the system as a library that can be used by a larger application or by providing bindings to other scripting languages like perl and python. This also encourages a style of programming where many types of functionality independent of the analyzer, like formatting systems and character set handling functions are implemented into the analyzer directly.

By trying to shoehorn analyzers into a variety of different operating systems with functionality trying to be all things to all users, morphological analyzers typically become arcane and verbose, making it difficult to add new functionality and change the system without major changes to the underlying analyzer itself. This makes it very difficult to support new languages or implement improved analysis systems in preexisting systems, typically they are reimplemented from scratch.

There have been attempts to deal with this problem, Freeling[cite] but they fall back on the method of using libraries and overly complicated interprocess communication protocols like corba to implement.

Goal: A software tool that can be used for research

With these problems we designed mrph with the goal of providing a tool that can reliably used for both day to day use as a morphological analyzer for higher level tasks and be easy to use to advance the state of the art in morphological analyzer research. With these motivations in mind we set the following goals:

1. develop a well engineered modular analyzer suitable to generalizing its methods. especially one with a set interface. Morphological analyzers typically use "one-shot" methods[cite], so the ideal way to deal with the system is to generalize one shots and allow *any* method to deal with it.

   Make it possible so that any developer as well as user can add the various parts of it. A tool for research.

2. engineer a system that would work as part of the inferno/plan 9 "software tools" ecosystem. giving data in a form that could easily be reparsed using stream transform tools similar to awk or sed.

3. choose an interface that allows the user to use unix style goodies, but, at the same provides sane defaults without configuarability. keep the interface the same across systems.

Mrph: a software tool for morphological analysis

Mrph was implemented with these goals in mind.

It is structured as a set of modules that works to compose. Mrph takes a different approach. it is a set of modules the goal is to be able to swap languages and data on the fly. it uses a system based on tokenization of asian languages. sacrifices efficincy for that ability to handle words as prefixes. it implements caching manually to allow itself to handle ranges that are much larger.

unlike many morphological analyzers only analyzes unicode.

it also attempts to be multilingual by ignoring traditional language tokenization, using the approach of asian language analyzers of deciding on possible morphs by doing prefix searches. this does stemming and lemmitization and multi word expression validation essentially for free. by ignoring

Interface

Unlike many morphological analyzers mrph is implemented as a "software tool" in the unix tradition.

linguistic researchers can be traditional researchers, but they typically have a variety of systems to work on. Given the idiosyncrises of these systems, it is impossible to assume support for things. to support all possible users people use approaches like Chasen[cite] or Freeling[cite] developing systems as tools and libraries, allowing the system to be used as part of a greater monolithic system.

Implementing the analyzer in limbo obviates many of the problems. both in terms of interface and implementation,

Input

Mrph expects plain utf8 text data as input, now, currently limited to the format of one sentence per line. It takes Japanese text input and gives you the value of their analysis.

Given that its expected language is utf8 by having native support for the system. Since inferno supports utf natively both in the programming language and the system interface level it makes it possible for mrph to handle any language naturally(except right to left languages like Arabic and Hebrew which still confound the construction of a simple

interface).

This allows mrph to handle any language automatically(potentially, right now it only supports Japanese). interspersed english and Japanese are handled in the same way provided that the input is utf8.

Output

The system outputs data in tree paths[cite]. This may seem unnecessary but in the future the system will support the input of values already in treepath format, allowing the system to potentially take advantage of higher levels of morphological data when doing analysis, allowing potential positive feedback loops where annotation is fed back to the analyzer allowing each level of the linguistic analysis process to have positive feedback with each other.

```
/N/日本語  ニホンゴ    日本語
/PG/を      ヲ          を
/VBT/教える             オシエル    教える                  一段        基本形
/NG/人      ヒト        人
/APC/と     ト          と
/N/日本語  ニホンゴ    日本語
を          ヲ          を          助詞-格助詞-一般
学習        ガクシュウ  学習        名詞-サ変接続
する        スル        する        動詞-自立   サ変・スル  基本形
人          ヒト        人          名詞-一般
が          ガ          が          助詞-格助詞-一般
ともに      トモニ      ともに      副詞-一般
楽しめる    タノシメル  楽しめる    動詞-自立   一段        基本形
ポー        ポー        ポー        名詞-固有名詞-地域-一般
タル        タル        タル        名詞-接尾-助数詞
サイト      サイト      サイト      名詞-一般
です        デス        です        助動詞      特殊・デス  基本形
。          。          。          記号-句点
```

This sacrifices. some of the readability of the original format, the abbrevation of the more descriptive version. Maintaining the original format added to much visual clutter with unicode fonts,

This is not especially pleasing with other fonts.

Pipelines

utf8 also allows for pipeline streaming covered in Thompos et al[cite]. which makes it possible to use the system with data that is potentially cut up, making it a better software tool candidate.

because the text that mrph supports can be broken up easily it can support pipelining in a very natural way. This allows it to be used as the part of a toolchain which builds up to the solution of a more general problem. In fact the system is meant to be used in Cocytus[cite] which discuss pipelines and the viability of Inferno as a NLP environment in greater detail.

However, as mentioned early pipelines are impossible to implement using an operating systems inherent primitives, which makes it impossible to move analyzers like mrph between systems because the operating system cannot reasonably be expected to handle everything natively.

A great advantage of programming a tool in limbo is that it comes with emu, which allows the system to be semantically identical across platforms. While other bytecompiled languages like Java allow programs to behave similarly across platforms they don't ensure the same *platform* between different systems, breaking one of the primary advantages of a portable language. The quirks of each system forces the user to

abandon.

Because mrph is a simple software tool running in inferno, it interacts with other tools(in inferno) using pipelines, allowing it to be without extra features or extraneous interface. Any text after processing, any character set conversions can be do as pre and post processing over pipelines, allowing mrph to concentrate on its purpose Morphological analysis.

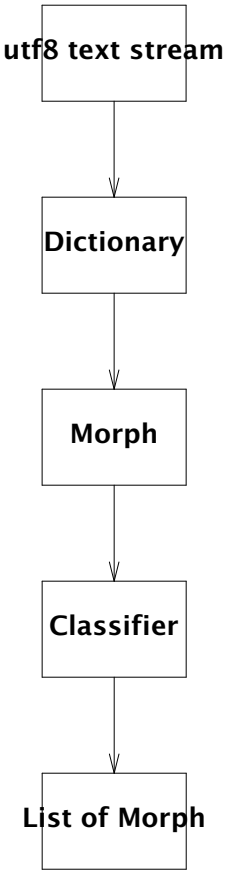
For example if you wanted to analyze a webpage and wanted all examples of consecutive noun phrases.

```
hget http://www.asahi.com | htmfmt | mrph |  readline
```

Implementation

This section describes the implementation of mrph. It begins by describing the modular structure of the implementation and the behavior of the main analyzer. It then goes on to describe the implementation of the original modules for the system and the practical considerations that went into their construction.

Modular construction



The system is implemented as a set of modules which implement the various struc tures of mrph:

takes a text stream and views the input as a set of prefixes. These prefixes are then fed to a dictionary which returns the valid morphs. which is then done with that.

Fundamentally the system is a mapping from text stream -> a set of morphs.

the system does so by taking the prefixes and giving them to a dictionary. it then gives a constant weight to the undefined and uses that to establish the valid path.

Once the valid path is established then the system is given the proper value. and the sequence of morphs is printed to standard output.

Separate system into modules

Since the system is taking streams of utf8 text and converting them into morphs.

that is why different morph modules need to have a way of dealing with this.

This allows the system to be easily modified. Your dictionary data structure as long as it supports prefix searches

the path module just takes a list of possible morphs and works that out itself.

the goal is the keep the modules as separate as possible, so that when any one of the system is altered it doesn't change the system itself.

Role of Modules

This separation of the system into modules is not novel, information hiding is part of any modular system, but this is especially important regarding parts of language.

programs like freeling use a set of files to determine the structure of the system. This works to a point, but at the same time it is limited to the scope of initial programmer and adds another level of indirection and complexity which precludes the understanding of the system itself.

when the user want to change language it's as simple as compiling a dictionary, set of paths and morphs for the system.

You want the user to be able to take advantage of the system and put everything together.

Tokenization

In order to deal with the largest amount of languages with no change in the processing of the underlying system the system is dealt with in a fashion that maximizes the way that language is dealt with. Language is viewed as a series o characters rather than words. word boundaries are determined entirely by the dictionary which determines all the possible prefixes of the sentence stream that can provide valid words.

This may seem like a waste in whitespace separated languages, where hash based methods on individual words may be more efficient like english but has the advantage of catching simple multiword expressions like "hard drive" automatically. This does not catch all the possible multiword expressions possible see Bond et al. [cite] for a list of the problems of multiword expressions and their detrimental effect for NLP.

This has the effect of punting the tokenization from the hardcoded analyzer itself to the dictionary and its contents, which are replacable.

Using the modules for research

By separating out the modules the system can be optimized in various ways.

The linguistic structure of the values being read is entirely up to the Morph module. The dictionary and the analyzer itself have no idea of the internal structure of a mrph. The morph module also includes a string function which allows it to print itself as well. obviating any need for the morph to know the state.

The dictionary

Actual implementation of the modules

All of this work to modularize the system is means nothing if the system is not efficient. However practical concerns are important as well. The data dealt with the system is large enough that the system is kept in a large enough size by putting the stuff together.

the goal should be to solve a large enough subsection of the modules and so on.

The dictionary module

The central problem of implementing a dictionary module is that there can be a huge amount of words that a system needs to understand. Unkown words are disasterous for the accuracy of morphological analysis, so any effective morphological analyzer will require a large dictionary(ours, IPAdic[cite] is 3910000 words) to effectively deal with language parsed and implemented.

So to do this we need a dictionary structure that is large enough to work with but at the same time.

There is a huge number of words that work well together.

these are well understood problems of morphological analysis, the traditional method of solving this is for the system is to simply write the dictionary structure to an mmap'ed file and let the operating system page in.

however this is inefficient[cite] and non portable, saving processor and memory specific information to the data file. current morphological analyzers get around this problem by compiling the structure for an original dictionary file when they are first set up on a system, but this a suboptimal solution and one that is traditional solved in other ways in the bell labs style.

We solved the problem by using a layered dictionary.

the main problem is that by not using a hash based data structure we are stuck with a trie based structure which is not space efficient to begin with. A hash based structure such as dbm(1) would be better but that would force the system to infer the tokens. by adopting a prefix based approach the dictionary can assign possible tokens as it goes, allowing the natural processing of asian languages as well as English.

Does this by implementing a patricia trie with nodes that are kept in various levels.

the central problem is that since everything is prefix based you eventually end up have to search the entire data structure. if you decompose the dictionary. you can potentially miss morphs. so the system just uses a hash table where the function is computed for each word.

this means that the system potentially does lookups in much slower time than a trie. however because it does this by going through the system. this has potentially poor behavior and may cause many seeks.

you still fail because if you get to the end you never know where the real end is.

So we go about this by going through the system by

Morphs

Likewise by having to store 390,000 keys in a database the system needs to hold the values stored in those keys. Each morph consists of a part of speech id and a various values. this means that a morph includes on average about 32 bytes of data(x words + x values + y somethings).

Also various morphs are much more popular than others. This means that the system needs to cache them in order to get a very good benefit. The caching for the morph module is modeled on the subfonts in Plan 9.

The problem is that the dictionary doesn't know the location of either. The system finds the Morphs by checking the cache value first. The cache is implemented as a hash table. anything that exists in the hash are then available. the system keeps an age for each of the morphs and little used morphs are purged from the system during periodic garbage collections.

need to keep a list of morphs. that is the problem.

The Paths module

The paths module is relatively simple. it uses a hardcoded limbo array, compiled from an external matrix file. this establishes the possible transitions from one combination to another.

The path module is similar to regular morphological analyzers, it contains a conjuction table which figures out how the paths are connected, it also contains a connection matrix which keeps track of how the values are connected in the system.

Finally it contains a Lattice which preserves the state of the analyzer, where the value is in the string, which paths exist and how the paths should be classified.

An easily modifiable tool for research

Data formats

Traditionally morphological analyzers, like Chasen[cite] and Mecab[cite], take the modern unix approach, mmap'ing its data structures to external files and then treating those files as part of the executable itself.

Data Organization

Have three different types.

```
Morphs, the data.
Dictionary
the matrix.
```

Morphs

Have to manually figure out the cache types. Morphs typically have a great deal of locality. Sam, acme

Morph handling

Plan 9 offers a rich model for caching.

```
mrphs sets of files
        sucks in files based on their values.
        files ordered according to their commonality.
        works in a way similar to
```

the goal is to avoid mmap

a caching system similar to fonts

Disadvantages

Inferno doesn't really have the set of tools that it needs to be productive for more tools. Tools like awk are very useful for language processing because they take textual input divided into fields by white space and allow their easily accumulation and editing. Inferno's shell while power is still too verbose for quick and dirty shell pipeline construction.

It can be pretty ugly. Many of the unix conventions came from strictly ascii text which makes it hard for typographical conventions which are human readable like

```
/1.NP/2.Word
```

Which become much harder when they are put into practice using other languages.

Traditional path formats don't look that bad in greek.

```
/M α/
/μόλις εβδομ δα
```

but characters with large widths and values are much more difficult to visualize easily using an editor like acme. which prompted moving the data formats to ascii.

Conclusion

Eliminates many of the problems and pitfalls that come from trying to implement a tool for a software tools system.

Limbo is a great language for doing multilingual programming. By allow the language itself to use utf8 and integrating it with the system.

Also by working the same way across architectures you don't need to go through the same issues that people normally go through to integrate with other tools, especially languages like Java.

It avoids the problems with C(i.e. people being able to randomly type things) but it lets the user get the way of doing things right.

No mmaping.

Forces many of the system's dark corners into the light.

The act of doing this, and making the various formerly implicit or recondite aspects of the system more accessible to programmers makes the system much more amenable to experimentation.

The module boundaries are clear.

None of this is specific to limbo per se.

but limbo does provide a way of doing things that encourages well engineered programs with less complexity.

Future work

Inferno's shell tools are still insufficient. for instance many tricks that work well in unix i.e. sort | uniq -c don't work in Inferno.

A morphological analyzer is very nice. Want to experiment with a variety of dictionary types.

the inferno shell, while general and powerful doesn't provide a nice environment for dealing with utf8 tabular output. a utf8 aware "little language" similar to or based on awk would be ideal.

Inferno really needs a font with complete coverage of the unicode set.

Make the system fully concurrent.

Work on incorporating polymorphism correctly. the amount of private data, breaks the interface. especially in terms of polymorphism. can include another internal module, but that adds complexity.

Come up with a way of making tree paths look better when used with a tree path.

[Asahara00] Asahara, M. and Matsumoto, Y., ''Extended models and tools for high-performance part-of-speech tagger'', Proc. of COLING Saarbrücken, Germany 2000.

[Carreras04fos] Carreras, X. and Chao, I. and Padro, L. and Padro, M., ''Freeling: An open-source suite of language analyzers'' Proc. of the 4th LREC 2004.

[Sag02] Sag, I.A. and Baldwin, T. and Bond, F. and Copestake, A.A. and Flickinger, D., ''Multiword Expressions: A Pain in the Neck for NLP'' Proc. of the Third International Conference on Computational Linguistics and Intelligent Text Processing},
pages={1--15},
year={2002},
publisher={Springer-Verlag London, UK}