Computational Morphology: Introduction

Aarne Ranta

European Masters Course, Malta, March 2011

Objective

Implement a **morphology module** for some language, comprising

- an inflection engine
- a morphological lexicon

Make this into a **reusable resource**, i.e.

- usable for various linguistic processing tasks
- available free and open-source

What is needed

Theoretical knowledge of morphology

- speaker's intuition
- grammar book

Programming skills

- mastery of appropriate tools
- design and problem solving

What languages will be addressed

Any languages of your choice; you can work in groups, too.

Addressed in the lectures (in more detail): English, Italian, Finnish, Arabic.

What tools will be used

Principal tool: GF, Grammatical Framework.

Also introduced: XFST, Xerox Finite State tool.

These tools can co-operate!

The GF Resource Grammar Project

Morphology and syntax for natural languages. Currently covering

`Afrikaans`	`Amharic`	`Arabic`	`Bulgarian+`	`Catalan`	`Danish`
`Dutch`	`English+`	`Finnish+`	`French`	`German`	`Hindi`
`Italian`	`Latin`	`Norwegian`	`Polish`	`Punjabi`	`Romanian`
`Russian`	`Spanish`	`Swedish+`	`Turkish+`	`Urdu`	

where $+ =$ with large lexicon.

We mainly expect lexica for the other languages, and inflection engines for languages outside the list.

How much work it is

Basic inflection engine: 1 week

Complete inflection engine: up to 8 weeks

Lexicon: 1 to 8 weeks.

All this depends on language and on available resources

Contents of these lectures

Overview of concepts and tools

Getting started with GF

Designing a simple inflection engine: English

Morphology-syntax interface

Richer inflection engine with traditional paradigms: Latin

Complex morphology with phonological processes: Finnish

Nonconcatenative morphology: Arabic

Building a morphological lexicon

Algorithms and tools: analysis vs. synthesis, GF vs. XFST

Overview of concepts and tools

Plan

What morphology is

Morphological processing tasks

Finite state transducers and other formats

Hockett's three models

Not morphology: POS tagging, tokenization, stemming

Morphology

Theory of **forms** (Gr. *morphe*)

- of plants and animals (biology)
- of words (linguistics)

In linguistics, "between phonology and syntax".

Examples of morphological questions:

- What is the past tense of English *drink*?
- What word form in Latin is *amavissent*?
- How are past tenses of verbs formed in Swedish?
- Do Greek nouns have dual forms?
- In what ways can causative verbs be formed in Finnish?

Morphological processing

Analysis: given a word (string), find its form description.

Synthesis: given a form description, find the resulting string.

Example of words and form descriptions in English

```
play  - play +N +Sg  +Nom
        play +V +Inf
plays - play +N +Pl  +Nom
        play +V +IndPres3sg
```

Description = **lemma** followed by **tags**

Both analysis and synthesis can give many results.

Morphology, mathematically

Between words W and their form descriptions D in a language, the morphology is defined by a relation M,

```
M : P(W x D)
```

A morphological analyser is a function

```
f : W -> P(D) such that d : f(w) iff (w,d) : M
```

A morphological synthesizer is a function

```
g : D -> P(W) such that w : g(d) iff (w,d) : M
```

Finite-state morphology

A common assumption in computational morphology: M is a **regular relation**.

This implies:

- M can be defined using a **regular expression**
- word-description pairs in M can be be recognized by a **finite-state automaton**, a **transducer**

In most system of computational morphology, M is moreover finite:

- the language has a finite number of words
- each word has a finite number of forms

A finite morphology M is trivially a regular relation.

We'll return to finite-state descriptions later.

Other formats for a finite morphology

Full-form lexicon: list of all words with their descriptions

```
play   - play +N +Sg  +Nom
         play +V +Inf
plays  - play +N +Pl  +Nom
         play +V +IndPres3sg
player - player +N +Sg +Nom
```

Morpological lexicon: list of all lemmas and all their forms

```
play N: play, plays, play's, plays'
play V: play, plays, played, played, playing
```

```
player N: player, players, player's, players'
```

The forms come in a canonical order, so that it is easy to restore the full description attached to each form.

It is easy to transform a morphological lexicon to a full-form lexicon.

Analysing with a full-form lexicon

It is easy to compile a full-form lexicon into a **trie** - a **prefix tree**.

A trie has transitions for each symbol, and it can return a value (or several values) at any point:

```
                  ' - s(3)         ' - s(12)
                /                /
  p - l - a - y(1,5) ---- e - r(10) - s(11) - '(13)
                \
                 s(2,6) - '(4)
```

N.B. a trie is also a special case of a finite automaton - an **acyclic deterministic finite automaton**.

Three models of morphological description

From Hockett, "Two models of grammatical description" (*Word*, 1954):

- **item and arrangement**: inflection is concatenation of morphemes (stem + affixes).
  - `dog +Pl --> dog s --> dogs`
- **item and process**: inflection is application of rules to the stem (one rule per feature)
  - `baby +Pl --> baby(y -> ie / _ s) s --> babie s --> babies`

- **word and paradigm**: inflection is association of a model inflection table to a stem
  - `{Sg:fly, Pl:flies}(fly := baby) --> {Sg:baby, Pl:babies}`

The word and paradigm model

The traditional model (Greek and Latin grammar).

The most general and powerful: "anything goes".

The other models can be used as auxiliaries when defining a paradigm.

But: there is no precise definition of a paradigm and its application.

Paradigms, mathematically

For each part of speech C ("word class"), associate a finite set $F(C)$ of inflectional features.

An **inflection table** for C is a function of type $F(C)$ -> `Str`.

Type `Str`: lists of strings (which list may be empty).

A **paradigm** for C is a function of type `String` -> $F(C)$ -> `Str`.

Thus there are different paradigms for nouns, adjectives, verbs,...

Example: English nouns

$F(\mathsf{N}) =$ `Number x Case`, where `Number = {Sg,Pl}`, `Case = {Nom,Gen}`

The word *dog* has the inflection table (using GF notation)

```
table {
  <Sg,Nom> => "dog" ;
  <Sg,Gen> => "dog's" ;
  <Pl,Nom> => "dogs" ;
  <Pl,Gen> => "dogs'"
}
```

`regN`, the regular noun paradigm, is the function (of variable `x`)

```
\x -> table {
  <Sg,Nom> => x ;
  <Sg,Gen> => x + "'s" ;
  <Pl,Nom> => x + "s" ;
  <Pl,Gen> => x + "s'"
}
```

Two more paradigms for English nouns

`esN`, nouns with plural ending *es*

```
\x -> table {
  <Sg,Nom> => x ;
  <Sg,Gen> => x + "'s" ;
  <Pl,Nom> => x + "es" ;
  <Pl,Gen> => x + "es'"
}
```

`iesN`, nouns with plural ending *ies*, dropping last character

```
\x -> table {
```

```
  <Sg,Nom> => x ;
  <Sg,Gen> => x + "'s" ;
  <Pl,Nom> => init x + "ies" ;    -- init drops the last char
  <Pl,Gen> => init x + "ies'"
}
```

Building a lexicon with paradigms

For a new entry: just give a stem and a paradigm,

```
dog regN
baby iesN
coach esN
boy sN
hero esN
```

This can be compiled into a morphological lexicon by applying the paradigms.

Analysis can be performed by compiling the lexicon into a trie.

But how do we select the right paradigm for each word?

And how to do with irregular words (such as *man* - *men*)?

Multiargument paradigms

To inflect highly irregular words, one can quite as well use several arguments:

```
irregN = \x,y -> table {
  <Sg,Nom> => x ;
  <Sg,Gen> => x + "'s" ;
  <Pl,Nom> => y ;
  <Pl,Gen> => y + "'s"
}
```

Similarly: irregular verb paradigms taking three forms.

```
man men irregN
mouse mice irregN
house regN
drink drank drunk irregV
```

Arabic verb inflection: the problem

form	**perfect**	**imperfect**
P3 Sg Masc	**k***a***t***a***b***a*	*ya***kt***u***b***u*
P3 Sg Fem	**k***a***t***a***b***at*	*ta***kt***u***b***u*
P3 Dl Masc	**k***a***t***a***b***aA*	*ya***kt***u***b***aAni*
P3 Dl Fem	**k***a***t***a***b***ataA*	*ta***kt***u***b***aAni*
P3 Pl Masc	**k***a***t***a***b***uwA*	*ya***kt***u***b***uwna*
P3 Pl Fem	**k***a***t***a***b***na*	*ya***kt***u***b***na*
P2 Sg Masc	**k***a***t***a***b***ta*	*ta***kt***u***b***u*
P2 Sg Fem	**k***a***t***a***b***ti*	*ta***kt***u***b***iyna*
P2 Dl	**k***a***t***a***b***tumaA*	*ta***kt***u***b***aAni*
P2 Pl Masc	**k***a***t***a***b***tum*	*ta***kt***u***b***uwna*
P2 Pl Fem	**k***a***t***a***b***tunv2a*	*ta***kt***u***b***na*
P1 Sg	**k***a***t***a***b***tu*	*A?a***kt***u***b***u*
P1 Pl	**k***a***t***a***b***naA*	*na***kt***u***b***u*

This is not morphology

Tokenization: split up the input into words, punctuation marks, digit groups, etc. *before* morphological analysis.

Part-of-speech tagging: resolve ambiguities *after* morphological analysis.

Stemming, also known as **lemmatization**: find out the ground form of a word, but ignore the morphological tags. This is sometimes done *instead of* proper morphological analysis, usually in quick-and-dirty ways.

All these techniques can be implemented using finite-state methods, e.g. XFST.

Part-of-speech tagging (= POS tagging)

Task: among the many possible morphological analyses, find the one that is correct in the given context.

```
She plays the guitar.    play +V
She likes your plays.    play +N
```

Statistical POS tagging: `(+Pron, +V, +Det)` is a more frequent trigram than `(+Pron, +N, +Det)`

Rule-based POS tagging (**constraint grammar**): after `+Pron`, `+N` is not allowed.

POS tagging is covered in another course - morphology just "feeds" it.

Other material

The LREC-2010 tutorial to GF:

http://www.grammaticalframework.org/doc/gf-lrec-2010.pdf

GF reference manual:

http://www.grammaticalframework.org/doc/gf-refman.html

GF library synopsis:

http://www.grammaticalframework.org/lib/doc/synopsis.html