Translation in GF

Aarne Ranta

European Masters, University of Malta, 18-22 March 2013

Plan

GF's formal potential as a translation system

Domain-specific vs. open-domain translation

Open-domain problems: interlingual lexicon, robustness, disambiguation

Probabilistic GF grammars

Learning GF grammars from data

GF

Multilingual grammar formalism based on type theory and functional programming

Multilingual grammar = abstract syntax + concrete syntaxes

Parsing: from string to abstract syntax

Linearization: from abstract syntax to string

Translation = parsing followed by linearization

Abstract syntax is interlingua

Potential

GF uses PMCFG = Parallel Multiple Context-Free Grammar

- between context-free and context-sensitive; slightly stronger than TAG (Tree-Adjoining Grammar)

Efficient runtime (empirically linear parsing)

Probabilistic GF grammars (abstract syntax probabilities)

Robust parsing: recovery from out-of-grammar parts of input

Synchronous grammars

Synchronous CFG: two rhs's (e.g. English and Latin)

```
S  -> NP VP   | NP VP
VP -> V2 NP   | NP V2
V2 -> "loves" | "amat"
```

Multilingual GF grammars: a generalization of synchronous CFG (and TAG)

- different lincat's, discontinuous constituents

- this enables a common abstract syntax in ”almost all cases”
- works for all languages so far (26 in the Resource Grammar Library)

RGL, the Resource Grammar Library

Implemented for 26 languages

Afrikaans	Bulgarian	Catalan	Chinese	Danish
Dutch	English	Finnish	French	German
Hindi	Italian	Japanese	Latvian	Nepali
Norwegian	Persian	Punjabi	Polish	Romanian
Russian	Sindhi	Spanish	Swedish	Thai
Urdu				

In progress: Arabic, Estonian, Greek (Ancient and Modern), Hebrew, Latin, Maltese, Turkish

Some RGL statistics

40+ contributors 2001-

3-6 months for a new language

3000-5000 lines of GF code per language

Complete morphology engine, comprehensive syntax, test lexicon (500 lemmas)

Larger dictionaries (10k - 100k lemmas) for 10 languages

Lexicon availability

#lines $\sim$ #lemmas

```
GF/lib/src$ wc -l */Dict???.gf

   53209 bulgarian/DictBul.gf
   64782 english/DictEng.gf
   42057 finnish/DictFin.gf
   92655 french/DictFre.gf
   44242 german/DictGer.gf
   25845 hindi/DictHin.gf
    4147 maltese/DictMlt.gf
  135830 russian/DictRus.gf
  102884 swedish/DictSwe.gf
   23867 turkish/DictTur.gf
   15250 urdu/DictUrd.gf
```

Translation systems of different types

Application grammars

- interlingua based on domain semantics: `Like x y`
- RGL used as library: `Like x y = mkCl x like_V2 y`
- compile-time transfer: `Like x y = mkCl y piacere_V2 x`
- limited but high quality

Resource grammars

- interlingua based on syntactic structures in the RGL
- structure-to-structure translation
- open-ended, but not full quality

Problems solved in application grammars

Translation via semantics (the top of the Vauquois triangle)

Choice of proper idiom: *please* to *s'il vous plaît*, *bitte*,...

Ambiguity reduced: *bank* may only mean the financial institution

Transfer of syntactic structure: *I like this* to *questo mi piace*

Current status in GF-based translation

Application grammars dominate: mathematics, painting descriptions, tourist phrasebook, Attempto controlled language, dialogue systems, pharmaceutical patents, software specifications, contracts, ...

RGL is used as a library, to make application grammar building easy

- less effort than manual coding (by orders of magnitude)
- no linguistic knowledge required from domain experts

A typical application has 15 languages and 200-500 concepts (i.e. abstract syntax functions)

Use cases

Production/publication/dissemination quality can be reached by automatic translation

Broadcasting to many languages

Web interfaces and mobile device app's (Android, iPhone)

Predictive parsing and syntax editing guide the user to enter translatable input

But this is not what mainstream machine translation does!

What about the other direction

Can we translate uncontrolled input?

Simple idea: resource grammar syntax + large dictionary

Possible refinements: statistical disambiguation, robust parsing, back-up strategies,...

Let’s try!

Multilingual lexicon

The first problem to solve: what is the abstract syntax

Words don’t match one-to-one between languages

So, what is the abstract syntax?

Thinking of semantics: it is **word senses**

Thus different fun’s for letter (character) and letter (document)

The Princeton WordNet

A lexical database for English words: http://wordnet.princeton.edu/

Words may have different senses

The senses are organized in hierarchies: **synonyms**, **hypernyms**, etc

Synset: set of synonymous words, i.e. a word sense

The 3.0 database contains 155,287 words organized in 117,659 synsets
for a total of 206,941 word-sense pairs (http://en.wikipedia.org/wiki/WordNe

Linked WordNets

WordNets for other languages, mapping their words to Princeton senses

Rather complete ones:

- Finnish http://www.ling.helsinki.fi/en/lt/research/finnwordnet/
- Hindi http://www.cfilt.iitb.ac.in/wordnet/webhwn/

Many incomplete, automatically extracted ones: Universal Wordnet http://www.mpi-inf.mpg.de/yago-naga/uwn/

General observation: 80% of mappings are unproblematic

Using GF with WordNet

Abstract syntax: synsets (word sense id's)

Concrete syntax: lemma from WordNet + inflection by GF morphology

Choose the most frequent synonym (if you have data for this)

If a synonym doesn't exist, use a hypernym

- *octopus, squid, cuttlefish* -> *bläckfisk* ("cephalopod")

Robustness by metavariables

If parsing fails, e.g. with unknown words, the parser tries to fill in a **metavariable** (placeholder, unknown subtree)

```
  p "he ate a ftira"
  UseCl Past (Pred he_NP (Compl eat_V2 ?))
```

The easiest way to solve this is to return the original word in translation

```
  er ass ein ftira
```

(Related case: if there's no German linearization, return the English word)

Metavariables can also occur in nodes that the construction doesn't parse:

```
  p "her he loves"
  UseCl Present (? she_NP he_NP love_V2)
```

There's no complete theory about how to handle these yet.

Disambiguation, the problem

Different senses of a word may translate to different words

- *this number is even -> diese Zahl ist gerade*
- *this surface is even -> diese Fläche ist eben*

Different syntactic structures may have different linearizations

- *I ate (a pizza with shrimps) -> j'ai mange une pizza aux crevettes*
- *I (ate a pizza with shrimps) -> j'ai mange une pizza avec des amis*

Word-sense disambiguation, grammar-based

A simple solution, using fine categorization

```
cat
  Number ;
  Surface ;
  Proposition ;
fun
  EvenNum : Number -> Proposition ;
  EvenSur : Surface -> Proposition ;
```

A more powerful solution, using **dependent types** to express **selectional restrictions**:

```
cat
  Class ;
  Term (c : Class) ;
  Property (c : Class) ;
  Proposition ;
fun
  Number, Surface : Class ;
  Pred : (c : Class) -> Term c -> Property c -> Proposition ;
  EvenNum : Property Number ;
  EvenSur : Property Surface ;
```

Word-sense disambiguation, statistical

Target language n-grams with wrong senses of words are rare.

Test this in Google translate!

Also, what happens when the distance gets larger.

(Also test syntactic disambiguation with prepositions.)

Syntax disambiguation

Syntactic parsing easily gives thousands of trees

Statistical disambiguation: rank with **tree probabilities**

Estimate probabilities from **treebanks**

Penn Treebank: http://www.cis.upenn.edu/~treebank/

- a set of 40k manually parsed sentences from Wall Street Journal
- converted to GF RGL abstract trees (Angelov 2012)

Tree probability in CFG

Probabilistic CFG: each rule has a probability

```
S  -> NP VP   -- 0.9
VP -> V2 NP   -- 0.3
NP -> "John"  -- 0.1
NP -> "beer"  -- 0.1
V2 -> "likes" -- 0.1
```

Sentence probability = tree probability = product of rule probabilities

```
p(John likes beer) = p((s (NP john) (VP (V2 likes) (NP beer))))
                   = 0.9 * 0.1 * 0.3 * 0.1 * 0.1 = 0.00027
```

Tree probability in GF

Probabilistic GF: each abstract syntax function has a probability

```
Pred  : NP -> VP -> S   -- 0.9
Compl : V2 -> NP -> VP  -- 0.3
John  : NP              -- 0.1
Beer  : NP              -- 0.1
Like  : V2              -- 0.1
```

Works the same way as probabilistic CFG:

```
p(John likes beer) = p(Pred John (Compl Like Beer))
                   = 0.9 * 0.1 * 0.3 * 0.1 * 0.1 = 0.00027
```

How to estimate probabilities

Frequencies of nodes in treebanks (sum to 1 per category)

But: there are no treebanks for many languages

In GF, one can port tree probabilities from one language to another, if the abstract trees are shared! (This is of course just an approximation. It should work better for semantics than for fluency.)

Advantage over PCFG: more abstract trees -> less sparse data

A problem

p(*John likes beer*) = p(*beer likes John*)

This is because the probabilities are context-free.

Could be improved by

- using "n-grams of tree nodes"
- dependent type probabilities (a research topic)

Learning GF grammars from data

Idea:

1. parse with resource grammar
2. recognize a **construction** (a frequent abstract tree pattern)
3. introduce a special rule for it
4. recognize the same construction in parallel data

This generalizes the recognition of phrases in phrase-based SMT

Abstracting out a construction

Data:

John is five years old	`Pred John (ComplAP (Mod (Num 5 Year) Old))`
they are seventy years old	`Pred They (ComplAP (Mod (Num 70 Year) Old))`
I am fifteen years old	`Pred I (ComplAP (Mod (Num 15 Year) Old))`
repeated pattern:	`Pred x (ComplAP (Mod (Num y Year) Old))`

Construction

```
fun YearsOld : NP -> Numeral -> Cl
lin YearsOld x y = Pred x (ComplAP (Mod (Num y Year) Old))
```

Translating a construction

English-French data:

John is five years old	*John a cinq ans*
they are seventy years old	*ils ont soixante-dix ans*
I am fifteen years old	*j'ai quinze ans*
repeated pattern:	`Pred x (ComplV2 Avoir (Num y An))`

Construction

```
lin YearsOld x y = Pred x (ComplV2 Avoir (Num y An))
```

Translating with a construction

Add the new rules to the RGL

More ambiguity in parsing:

```
  she is twenty years old ->
    Pred She  (ComplAP (Mod (Num 20 Year) Old))
    YearsOld She 20
```

However, the special construction gets a higher probability.

```
  * elle est vieille de vingt ans
  elle a vingt ans
```