Machine Translation with Type Theory and Functional Programming

Aarne Ranta
Ramona Enache

Aspects of machine translation

Lexical:

I → jag
am → är
here → här

Aspects of machine translation

Syntactic:

du → you
är → are
här → here

Aspects of machine translation

Syntactic:

du → you
är → are
här → here

Aspects of machine translation

Semantic:

how
are
you

hur
står
det
till

Aspects of machine translation

It is not possible to make a perfect machine translator!

Most tools deal with lexical and partial syntactic correctness when doing machine translation !

Approaches to machine translation

- Statistical machine translation

 - mathematical models inspired by information theory
 - rely on large corpora of aligned data
 - achieve
   - good lexical quality, depending on the choice of corpora
   - n-gram model for syntactic and semantic correctness - works for short phrases

Approaches to machine translation

- Statistical machine translation

- most popular nowadays
- state-of-the-art : Google translate

Pros	Cons
-model applies for all languages	-often not syntactically correct
-fully automatic	-dependent on the corpora
-model applies for all kinds of text	-not customised to a given language

Approaches to machine translation

- Rule-based machine translation

- inspired by formal languages
- relies on building a grammar for the language
- usually domain-specific
- achieve
    - good syntactic and semantic correctness

Approaches to machine translation

- Rule-based machine translation

Pros	Cons
-customised to given language	-more manual work involved
-syntactically correct translations	-little coverage
	-only work for a given domain

Approaches to machine translation

- Rule-based machine translation

Approaches to machine translation

- Rule-based machine translation
  + functional programming

Approaches to machine translation

- Rule-based machine translation
  + functional programming
    + type theory

Approaches to machine translation

- Rule-based machine translation
  + functional programming
    + type theory

= GF

GF

- grammar formalism for describing natural languages
- functional language with support for advanced features of type theory
- approaches machine translation from a programming languages view
- relies on
    - an abstract syntax - interlingua
    - many concrete syntaxes - target languages(16 currently)

GF

Abstract syntax - first-order type theory

- parameters - data types

```
Gender = Masc | Fem ;
```

- lexical categories - data structures

```
Noun = {s : Number => Str; g : Gender}
```

GF

Abstract syntax - advanced features of type theory

- dependent types - semantic constraints

```
isCapitalOf : El City -> El Country -> Formula ;
```

- higher-order syntax

```
reflexiveRelation : (c : Class) ->
            (El c -> El c -> Formula) -> Formula ;
```

GF

Abstract syntax - advanced features of type theory

- semantic definitions

```
data zero : Nat ;
data succ : Nat -> Nat ;

fun plus : Nat -> Nat -> Nat ;
def plus zero n = n ;
def plus (succ m) n = succ (plus m n) ;
```

GF

Concrete syntax

- support for regular expressions, for complex pattern matching
- functional programming without recursion
- function overloading
- allows code sharing through interfaces
- allows code reuse - functional core

GF - solutions

- translations are syntactically correct, due to the specific treatment of each language in its concrete syntax module
- translations are semantically correct for a given domain, due to the use of the abstract syntax as semantic interlingua
- incremental parsing - for word completion and authoring of constructions
- user interaction - demo

GF - solutions

- grammars are portable and usable as software libraries
  - PGF - runtime binary format, encoding of the abstract syntax + concrete syntaxes
  - interpreters for PGF - Haskell, JavaScript, Java
  - uses - web applications, Android applications, ...

GF - solutions

- less manual effort :
  - use of general purpose existing libraries to build new application grammars
  - easier to test and debug
  - functional programming - less code, more readable, easier to write and maintain
  - learning grammars from examples - for non-programmers

GF - future

- European project MOLTO, FP7-ICT-247914 :
  - combine GF with statistical methods - increase robustness
  - large coverage for given domains - mathematics exercises, patents, art and museums to write
  - make GF programming accessible to all categories of users for writing their own grammars

GF - demo

- Tourist Phrasebook for 14 languages
  - high lexical, syntactic and semantic quality
  - automatic treatment of ambiguities
  - user interface - incremental parsing