The MOLTO Phrasebook

An Electronic Phrasebook for 15 Languages

Krasimir Angelov, Olga Caprotti, Ramona Enache, Thomas Hallgren,
Aarne Ranta

Multilingual Online Translation, FP7-ICT-247914

MOLTO's goal

Translate between **multiple languages** in **real time** with **high quality**.

- Multiple languages: up to 15 simultaneously
- Real time: automatic, efficient, constantly updatable
- High quality: usable by producers of information

The phrasebook showcase

The first translation system delivered

- Work Package 10, "Dissemination and Exploitation"
  - Deliverable 10.2, "MOLTO web service", due June 2010
    * showcase for MOLTO translation for the general public

Explore and illustrate the MOLTO technology

Provide something useful and accessible for the general public

MOLTO technology aspects

Debug and document resource grammar library

Examples and best practices for functors vs. transfer

Experimenting with example-based grammars

Lower the requirements of expertise

See how everything is put together

Reach uncompromised quality and idiomaticity

A basic use case

A tourist wanting to translate phrases on the go

- fixed phrases, like *please*
- phrases with variable parts, like *this (pizza|wine|bread) is (delicious|bad)*

The translation should work in a **portable device** (mobile phone without internet connection)

The translator must be usable without previous training

Extended use cases

SMS/email/social forum translation: *see you in the bar tomorrow*

Language training: randomized translation exercises

Extensible vocabulary

The same system should be tailored for

- different platforms: PC, mobile phones
- different subsets of languages

From: Fre Del Clear Random

vingt - quatre bières et dix-huit poissons

anglais belges bon chauds chers délicieux ennuyeux et finlandais frais français froids italiens roumains suspects suédois trop très

Bul:
двадесет и четири бири и осемнадесет риби

Eng:
twenty - four beers and eighteen fish

Fin:
kaksikymmentäneljä olutta ja kahdeksantoista kalaa

Fre:
vingt - quatre bières et dix-huit poissons

Ger:
vier und zwanzig Biere und achzehn Fische

Ita:
ventiquattro birre e diciotto pesci

Ron:
douăzeci şi patru de beri şi optsprezece peşţi

Swe:
tjugofyra öl och arton fiskar

Characteristics of the phrasebook domain

Canned phrases: *please - var så god - s'il vous plaît - bitte*

Idiomatic constructions:

- *how old are you*
- *quel âge avez-vous* "what age do you have"
- *quanti hanni ha* "how many years do you have"

Politeness and gender distinctions

- *are you Swedish* - (*es-tu* | *êtes-vous*) (*suédois* | *suédoise*)

Disambiguation

When the English user types *Are you Swedish?* she gets four French translations, with **disambiguations** in English:

- *Es-tu suédois ?* Are you (familiar, male) Swedish?
- *Es-tu suédoise ?* Are you (familiar, female) Swedish?
- *Êtes-vous suédois ?* Are you (polite, male) Swedish?
- *Êtes-vous suédoise ?* Are you (polite, female) Swedish?

Minimal disambiguation

What is *Are you hungry?* in French?

There are four underlying phrases, but just two distinct ones:

- *As-tu faim ?* Are you (familiar) hungry?

- *Avez-vous faim ?* Are you (polite) hungry?

Principle: **ignore ambiguities that are common to source and target**.

Components of the phrasebook system

Multilingual GF grammar: abstract syntax + concrete syntaxes

GF grammar compiler

GF resource grammar library

User interface built with HTML and JavaScript

GF web server / GF runtime system in Java

JavaScript library for building the UI and calling the server

(*boldface = built in MOLTO ; roman = given in GF*)

The MOLTO vision

Translation System = Multilingual GF Grammar

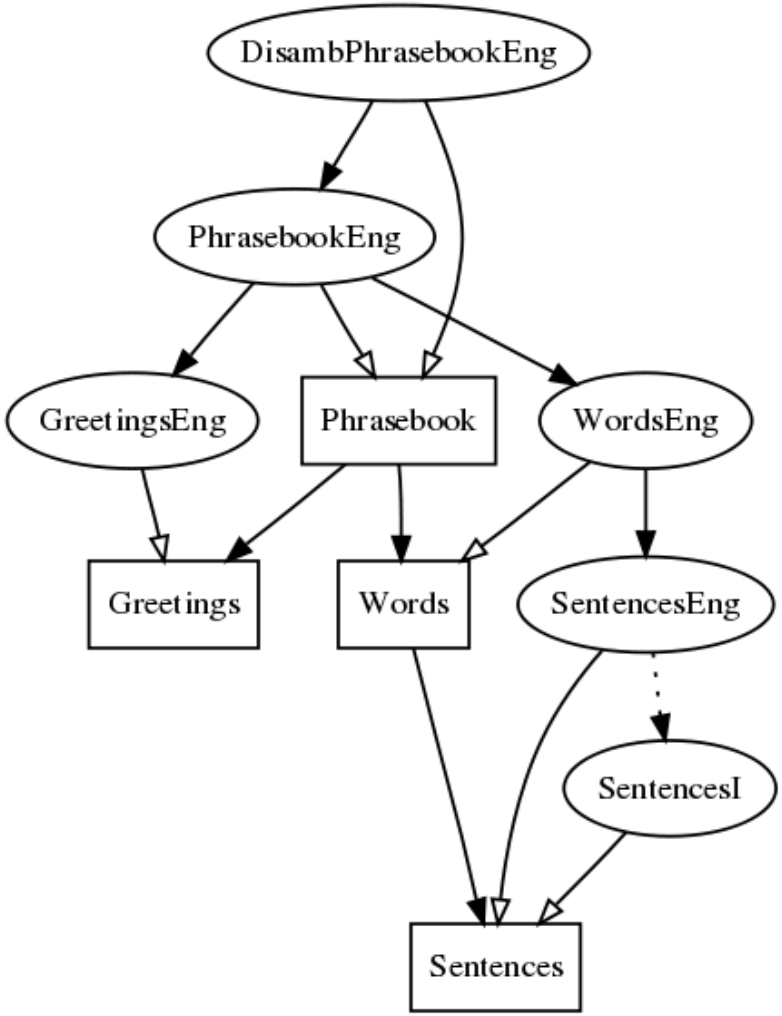
The grammar is specific to some **domain** and a set of **languages**

The rest is given in GF

Therefore, we focus here on how to build the grammar

We also take a look at how the complete system works

The structure of the grammar



Abstract syntax

- Phrasebook: top module
- Sentences: common linguistic structure
- Words: potentially idiomatic phrases
- Greeting: canned phrases

Concrete syntax

- Sentences: by functor SentencesI
- Words: by ParadigmsL, SyntaxL, ExtraL
- Greetings: as strings

Greetings

Originally collected from Wikipedia Phrasebook

```
cat
  Greeting ;
fun
  GBye : Greeting ;
  GCheers : Greeting ;
  GDamn : Greeting ;
  GExcuse, GExcusePol : Greeting ;
  GGoodDay : Greeting ;
```

```
lincat
  Greeting = SS ;
lin
  GBye = ss "pa" ;
  GCheers = ss "noroc" ;
  GDamn = ss "ptiu" ;
  GExcuse, GExcusePol = ss "pardon" ;
  GGoodDay = ss "buna ziua" ;
```

Abstract syntax: **union of distinctions in all languages**. E.g. politeness.

Disambiguation grammars

GreetingsFre: unambiguous

```
GExcuse = ss "excuse-moi" ;
GExcusePol = ss "excusez-moi" ;
```

GreetingsEng: ambiguous

```
GExcuse, GExcusePol = ss "excuse me" ;
```

DisambPhrasebookEng: exceptions to PhrasebookEng, adding explanations

```
concrete DisambPhrasebookEng of Phrasebook = PhrasebookEng -
     [GExcuse, GExcusePol ...] ** {
  GExcuse = ss "excuse me (familiar)" ;
  GExcusePol = ss "excuse me (polite)" ;
```

Alternative treatment of politeness

Parameter governing *all greetings*:

```
lin PGreeting : Politeness -> Greeting -> Phrase

lin PGreeting pol g = g.s ! pol.p        -- Fre, Fin, ...

lin PGreeting pol g = g.s                -- Eng

lin PGreeting pol g = g.s ++ pol.s       -- DisambEng
```

Overkill: more ambiguities than really needed

Tedious to thread through all phrases

Other sources of ambiguity

Gender of hearer

- *are you Finnish*
- *vous êtes finlandais - vous êtes finlandaise*

Gender of speaker

- *I am Finnish*
- *je suis finlandais - je suis finlandaise*

Miscellaneuous phrases

- *anteeksi*

- *sorry - excuse me*

Semantic fields

here we are	var så god	bitte	GHereWeAre
please	snälla	"	GPleaseDo
"	tack	"	GPleaseGive
thanks	"	Danke	GThanks

The principal set of ambiguities

```
IMale         = mkP i_Pron      "(male)" ;
IFemale       = mkP i_Pron      "(female)" ;
YouFamMale    = mkP youSg_Pron  "(familiar,male)" ;
YouFamFemale  = mkP youSg_Pron  "(familiar,female)" ;
YouPolMale    = mkP youPol_Pron "(polite,male)" ;
YouPolFemale  = mkP youPol_Pron "(polite,female)" ;
```

Sentences: the functorizable module

Definitions of `cats`

```
cat Proposition ; Sentence ; Item ; Quality ; Person ; Place
```

Syntactic combinations of general kind

```
Is          : Item -> Quality -> Proposition ;  -- this pizza is good
SProp       : Proposition -> Sentence ;         -- this pizza is good
SPropNot    : Proposition -> Sentence ;         -- this pizza isn't good
QProp       : Proposition -> Question ;         -- is this pizza good

WherePlace  : Place  -> Question ;              -- where is the bar
WherePerson : Person -> Question ;              -- where are you

IMale, YouFamFemale,... : Person
```

Linearization types

Easy cases: ready-made resource categories

```
Sentence = S ;
Question = QS ;
Proposition = Cl ;
Item = NP ;
Kind = CN ;
```

More complex cases: records of resource categories

```
Place = {name : NP ; at : Adv ; to : Adv}
```

Using records

In all languages: place-dependent cases and prepositions

```
ABePlace p place = mkCl p.name place.at ;
AWantGo p place = mkCl p.name want_VV (mkVP (mkVP L.go_V) place.to) ;

Airport = mkPlace "airport" "at" ;
Bank = mkPlace "bank" "at" ;
Bar = mkPlace "bar" "in" ;

Airport = mkPlace (mkN "lento" (mkN "kenttä")) lla ;
Bank = mkPlace (mkN "pankki") ssa ;
Bar = mkPlace (mkN "baari") ssa ;
```

How this is modularized (Finnish)

```
mkPlace : N -> Bool -> {name : CN ; at : Prep ; to : Prep} = \p,e -> {
  name = mkCN p ;
  at = casePrep (if_then_else Case e adessive inessive) ;  -- True: exter
  to = casePrep (if_then_else Case e allative illative) ;
  } ;

ssa = False ;
lla = True ;
```

Some functorial linearizations

```
Is = mkCl ;                      -- this pizza is good

SProp = mkS ;                    -- positive declarative
SPropNot = mkS negativePol ;     -- negative declarative
QProp p = mkQS (mkQCl p) ;       -- sentential question

WherePlace place = mkQS (mkQCl where_IAdv place.name) ;    -- where is place
WherePerson person = mkQS (mkQCl where_IAdv person.name) ; -- where is person

ObjNumber n k = mkNP n k ;       -- five pizzas
ObjIndef k = mkNP a_Quant k ;    -- a pizza
ObjAndObj = mkNP and_Conj ;      -- a pizza and two beers

This kind = mkNP this_Quant kind ;  -- this pizza
The kind = mkNP the_Quant kind ;    -- the pizza

ThePlace kind = placeNP the_Det kind ;  -- the hospital
APlace kind = placeNP a_Det kind ;      -- a hospital

AHave person kind = mkCl p.name have_V2 (mkNP kind) ;  -- person has kind
```

Minimal use of functors

No domain lexicon interface: just the RGL ones

```
incomplete concrete SentencesI of Sentences = Numeral **
  open Syntax, Lexicon, Symbolic, Prelude in ...
```

Constructs that might require parameters are put into `Words` (which therefore has some repetition)

Exceptions to the functor take care of random deviations:

```
  concrete SentencesFre of Sentences = NumeralFre ** SentencesI - [
   QProp,IFemale, YouFamFemale, YouPolFemale] with ...
```

Actions of persons: typically non-functorial

```
ALike : Person -> Item -> Action ;  -- I like this pizza
  mkCl p.name (mkV2 (mkV "like")) item ;                              -- Eng
  mkCl p.name (mkV2 (mkV "pitää") elative) item ;                     -- Fin
  mkCl item (mkV2 (mkV (piacere_64 "piacere")) dative) p.name ;       -- Ita

QWhatAge : Person -> Question ;     -- how old are you
  mkQS (mkQCl (ICompAP (mkAP L.old_A)) p.name) ;                      -- Eng
  mkQS (mkQCl
    (mkIP whichSg_IDet (mkN "âge" masculine)) p.name have_V2) ;       -- Fre
  mkQS (mkQCl (mkIP how8many_IDet L.year_N) p.name have_V2) ;         -- Ita
```

The ontology

Take a look at `Sentences` and `Words`

Rough semantic distinctions via simple types (Place, Person, Transport...)

No domain distunctions by modules.