When life gives you lemons, make GF!

Inducing grammars from the lexicon-ontology interface

Christina Unger
Semantic Computing Group
CITEC, Bielefeld University

In collaboration with:

Jeroen van Grondelle, Frank Smit, Jouri Fledderman
(Be Informed, The Netherlands)

Motivation

Conceptually scoped language technology

Today

Natural language plays an increasingly important role as interface to existing services and data.

Requirements

Requirements

1. Alignment of natural language expressions and domain concepts, data or services

   Would I get housing benefits?

   ```
   ASK WHERE { :user :eligible "true". }
   ```

Requirements

1. Alignment of natural language expressions and domain concepts, data or services

   Would I get housing benefits?

   ```
   ASK WHERE { :user :eligible "true". }
   ```

2. High precision (reliability and predictability)

Requirements

1. Alignment of natural language expressions and domain concepts, data or services

   Would I get housing benefits?

   ```
   ASK WHERE { :user :eligible "true". }
   ```

2. High precision (reliability and predictability)
3. Expertise and time for creating and maintaining grammars (and for porting it across languages or switching domains)

Requirements

1. Alignment of natural language expressions and domain concepts, data or services

   Would I get housing benefits?

   ```
   ASK WHERE { :user :eligible "true". }
   ```

2. High precision (reliability and predictability)
3. Expertise and time for creating and maintaining grammars (and for porting it across languages or switching domains)
4. ~~Unrestricted coverage~~

Conceptually scoped language technology

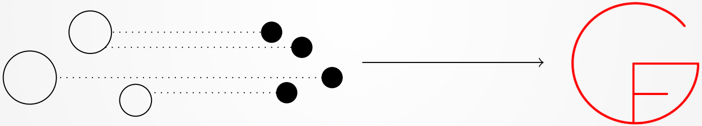
The underlying application introduces a conceptual scope that determines the language fragment that is relevant and meaningful.

Goal

CONCEPTUALIZATION (ONTOLOGY)

LEXICAL INFORMATION (ONTOLOGY LEXICON)

GRAMMAR

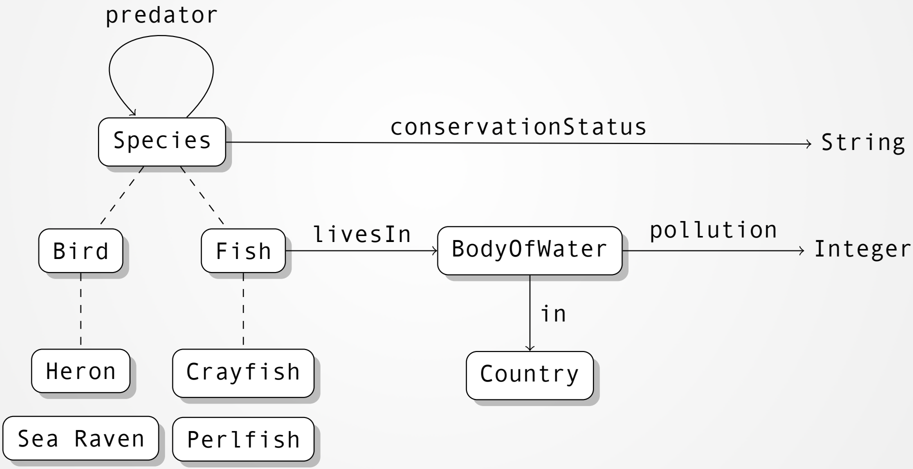


If life gives you lemons...

The lexicon-ontology interface

Ontology

Example: Fresh water animals



Modelling data w.r.t. an ontology

Example: Chiemsee fish

```
:Germany  rdf:type :Country .
:Chiemsee rdf:type :BodyOfWater ;
          :in :Germany ;
          :pollution 2 .

:ChiemseeCrayfish rdf:type :Crayfish ;
          :livesIn :Chiemsee ;
          :conservationStatus "EX" .

:ChiemseePerlfish rdf:type :Perlfish ;
          :livesIn  :Chiemsee ;
          :predator :Heron, :SeaRaven ;
          :conservationStatus "EN" .
```

Ontology lexica

Aim: capture rich and structured linguistic information about how ontology elements are lexicalized in a particular language

Why simple terminological knowledge is not enough

The conceptual granularity of language often does not coincide with that of the schema underlying a particular dataset...

:team → to play for if the subject is any kind of player
→ to race for if the subject is a race driver

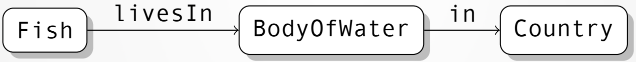
...and can also vary across languages.

:eat $\rightarrow_{en}$ eat
$\rightarrow_{de}$ essen if the subject is a human
$\rightarrow_{de}$ fressen if the subject is an animal

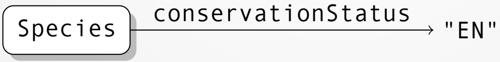
Why simple terminological knowledge is not enough

Not only lexicalizations of single classes or properties are relevant, but also lexicalizations of complex constructions.

- Which fish live in Germany?

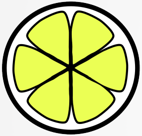


- Which fish are endangered?



lemon (Lexicon Model for Ontologies)

http://lemon-model.net

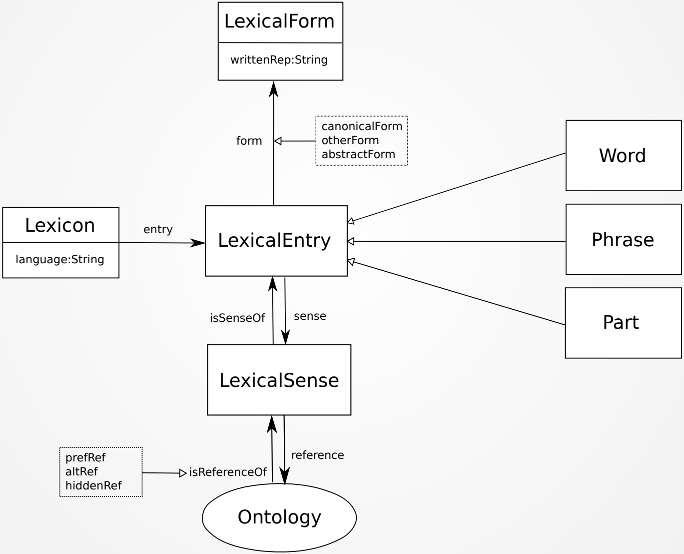


- meta-model for describing ontology lexica with RDF
- declarative, thus abstracting from specific syntactic and semantic theories
- separation of lexicon and ontology

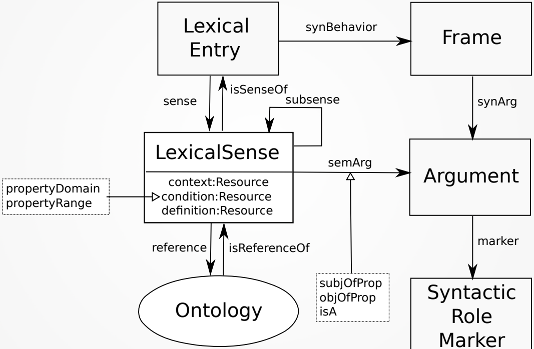
Semantics by reference

The meaning of lexical entries is specified by pointing to elements in the ontology.

The lemon model (core)



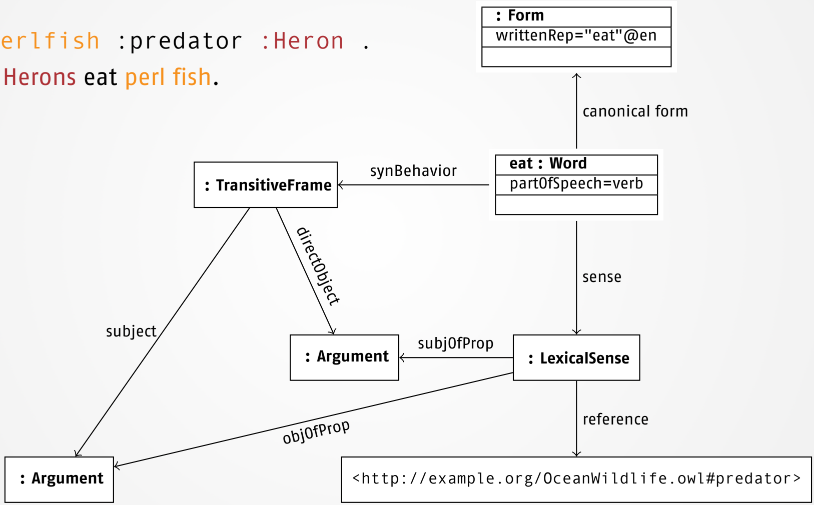
The lemon model (argument mapping)



Example

```
:Perlfish :predator :Heron .
```

→ Herons eat perl fish.



...make GF!

Mapping ontology lexica to grammars

Roadmap

Mapping ontology lexica to GF requires to capture

- the ontological (semantic) level
- the lexical (morpho-syntactic) level

General method:

1. ontology $\rightarrow$ abstract syntax
2. lexical entries $\rightarrow$ concrete syntax

From an ontology to abstract syntax [1]

[1] K. Angelov: The abstract syntax as ontology. GFSS 2009.
K. Angelov & R. Enache: Typeful Ontologies with Direct Multilingual Verbalization. CNL 2010.

Ontology to abstract syntax

```
cat

    Class;
    Individual Class;

    Datatype;
    Literal Datatype;

    Statement;
```

Example


```
fun

    Species, Fish, Bird : Class;
    String              : Datatype;

    ChiemseePerlfish    : Individual Fish;

   conservationStatus   : Individual Species
                       -> Literal    String
                       -> Statement;

 coerce_Fish_to_Species : Individual Fish
                       -> Individual Species;
```

OWL constructs

Add functions for complex

- classes
  (union, intersection, complement, restriction classes)
- properties
  (inverse properties, property chains)

Example:

```
:Endangered rdf:type owl:Restriction;
            owl:onProperty onto:conservationStatus ;
            owl:hasValue   "EN" .

▷ Things_with_conservationStatus_EN : Class;
```

From a lexicon to concrete syntax

Example

Lexicon:

```
:ocean_N a lemon:Word ;
lexinfo:partOfSpeech  lexinfo:commonNoun;
lemon:canonicalForm [ lemon:writtenRep "ocean"@en ];
lemon:otherForm     [ lemon:writtenRep "oceans"@en;
                      lexinfo:number   lexinfo:plural ];
lemon:sense         [ lemon:reference  onto:Ocean ] .
```

Concrete syntax:

```
lin Ocean = mkCN (mkN "ocean" "oceans");
```

Example

Lexicon:

```
:sea_N a lemon:Word ;
lexinfo:partOfSpeech  lexinfo:commonNoun;
lemon:canonicalForm [ lemon:writtenRep "sea"@en ];
lemon:otherForm     [ lemon:writtenRep "seas"@en;
                      lexinfo:number   lexinfo:plural ];
lemon:sense         [ lemon:reference  onto:Ocean ] .
```

Concrete syntax:

```
lin Ocean = variants {
            mkCN (mkN "ocean" "oceans");
            mkCN (mkN "sea" "seas")
            };
```

Linearization categories

```
lincat
```

Linearization categories

`lincat`

- `Individual = NP;`
  - the Pacific Ocean

Linearization categories

`lincat`

- `Individual = NP;`
  - the Pacific Ocean
- `Class = { cn:CN; ap:AP };`
  - whale
  - endangered

Linearization categories

`lincat`

- `Individual = NP;`
  - the Pacific Ocean
- `Class = { cn:CN; ap:AP };`
  - whale
  - endangered
- `Statement = { np:NP; vp:VP; vpSlash:VPSlash };`
  - $[_{NP}$ The finback $]$ $[_{VP}$ lives in the Pacific Ocean $]$.
    Which ocean does $[_{NP}$ the finback $]$ $[_{VPSlash}$ live in _ $]$?

Mapping lexical entries to linearizations

Starting point:

- Input lexicon (centered around entries)

```
:ocean_N lemon:sense [lemon:reference onto:Ocean].
:sea_N   lemon:sense [lemon:reference onto:Ocean].

:eat_V lemon:sense [lemon:reference onto:predator],
                   [lemon:reference onto:prey].
```

- Target grammar (centered around senses)

```
lin Ocean    = variants { ocean_N; sea_N };
lin predator = eat_V;
lin prey     = eat_V;
```

Mapping lexical entries to linearizations

1. Collect all senses that occur in the lexicon (simple or compound), together with all entries that denote this sense.

Mapping lexical entries to linearizations

1. Collect all senses that occur in the lexicon (simple or compound), together with all entries that denote this sense.

Example:

```
:ocean_N lemon:sense [lemon:reference onto:Ocean].
:sea_N   lemon:sense [lemon:reference onto:Ocean].
```

▷ { reference: Ocean, entries: [ocean_N,sea_N] }

Mapping lexical entries to linearizations

2. For each such entry, extract all relevant lexical information:
    - canonical form
    - part of speech (e.g. noun, verb)
    - syntactic frames (with arguments and argument-specific information, such as markers and optionality)
    - POS-specific information
        - noun: gender, singular and plural forms
        - verb: present, past, participle, gerund forms
        - adjective: positive, comparative, superlative forms

Mapping lexical entries to linearizations

3. Based on the collected information, for every sense construct a list of `lin` variants by instantiating a GF template for each frame of each entry lexicalizing that sense.

Mapping lexical entries to linearizations

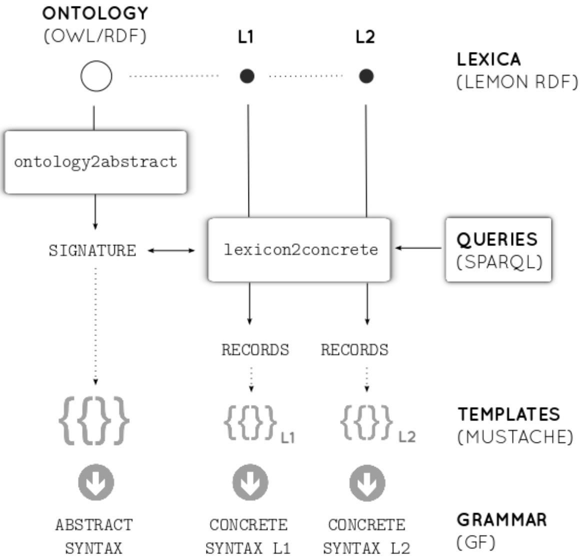
3. Based on the collected information, for every sense construct a list of `lin` variants by instantiating a GF template for each frame of each entry lexicalizing that sense.

Example:

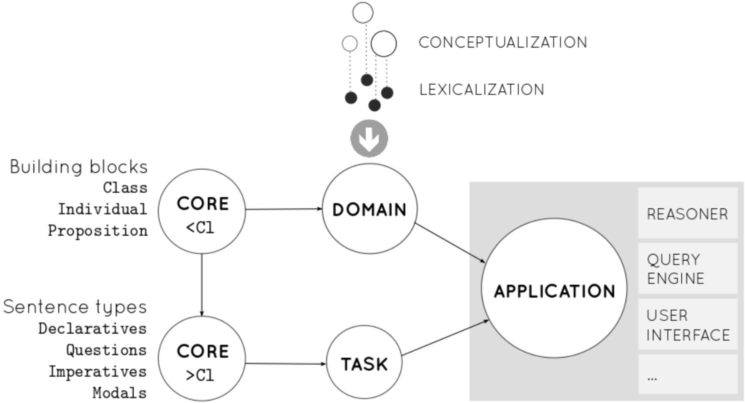
```
lin predator o s = variants {
        { np      = s;
          vp      = mkVP (mkV2 eat_V) o;
          vpSlash = mkVPSlash (mkV2 eat_V) };
        ... };

oper eat_V = mkV "eat" ...;
```

Architecture



Domain grammars as grammar modules

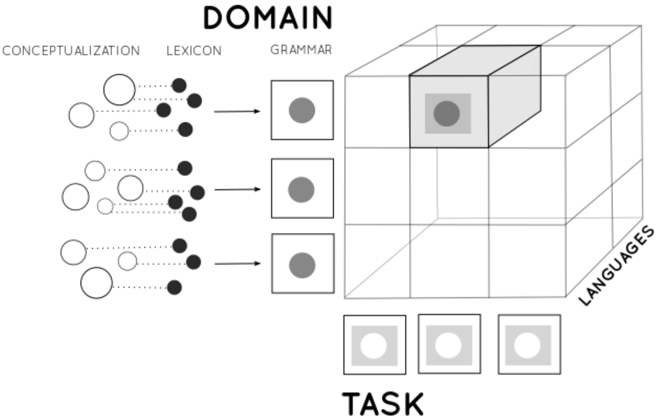


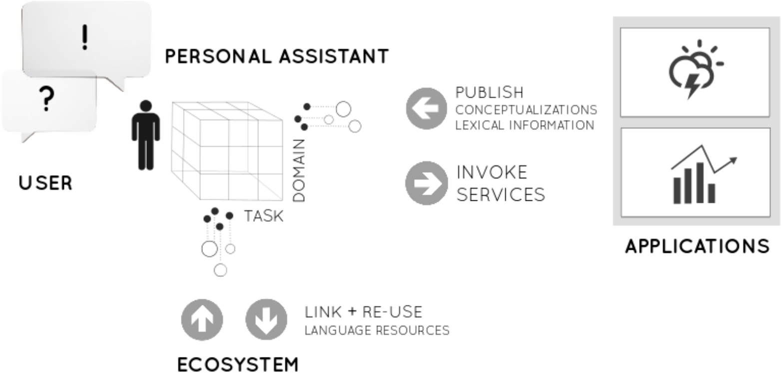
Examples

- liveIn (coerce_Perlfish_to_Fish ChiemseePerlfish) (coerce_Lake_to_BodyOfWater Chiemsee)
  the Chiemsee perlfish lives in Lake Chiemsee
- liveIn (Most Fish) (Generic BodyOfWater)
  most fish live in bodies of water
- neg (liveIn_o_in (Most Fish) Sweden)
  most fish don't live in Sweden
- mod Can (predator (All Species) (Generic Species))
  every species can be eaten
- predator (That Species) (This Species)
  this species is a predator of that species

Outlook

An ecosystem for language technology


DOMAIN
CONCEPTUALIZATION
LEXICON
GRAMMAR
LANGUAGES
TASK


!
?
PERSONAL ASSISTANT
USER
DOMAIN
TASK
PUBLISH
CONCEPTUALIZATIONS
LEXICAL INFORMATION
INVOKE
SERVICES
APPLICATIONS
LINK + RE-USE
LANGUAGE RESOURCES
ECOSYSTEM

Appendix

Code and resources

- lemon2gf (code and documentation)
  `https://github.com/cunger/lemon2gf`
- Grammar modules
  `https://github.com/cunger/grammars`