Collaborative Multilingual Knowledge Engineering Based on Controlled Natural Language

Kaarel Kaljurand

Institute of Computational Linguistics, University of Zurich

College of Graduate Studies and the Academy of African Languages and Science, University of South Africa
2013-11-29
About myself

• studied at the University of Tartu, Estonia
• worked 2002-2013 at the University of Zurich, Switzerland
• mainly in projects related to controlled natural languages
  ◦ REWERSE (2004-2008): using Attempto Controlled English (ACE) as a semantic web language
  ◦ MOLTO (2012-2013): building a multilingual semantic wiki based on ACE and Grammatical Framework (GF)
  ◦ joint work with: Norbert E. Fuchs (group leader) and Tobias Kuhn
• other interests:
  ◦ grammar-based speech recognition applications
  ◦ Estonian grammar in GF
  ◦ biomedical text mining
• see also: http://google.com/+KaarelKaljurand
Overview of the tutorial

• semantic web (SW)
  ◦ knowledge representation languages (OWL, SWRL) and reasoning
  ◦ user interfaces for the SW (Protégé, Semantic MediaWiki)

• controlled natural languages (CNLs)
  ◦ Attempto Controlled English (ACE)
  ◦ its construction and interpretation rules, discourse representation

• ACE as a user interface language for the SW
  ◦ translating between ACE and OWL
  ◦ end-user tools: ACE View, AceWiki

• Grammatical Framework (GF)
  ◦ GF as a framework for defining multilingual CNLs
  ◦ resource grammar library (RGL)

• ACE-in-GF: ACE in ~20 languages

• AceWiki-GF: multilingual collaborative knowledge editor
Semantic Web
Semantic Web

• original vision by Berners-Lee, Hendler, Lassila (2001)
  ◦ smart personal agents that book flights and negotiate hotel prices
  ◦ concepts are denoted by URIs
  ◦ ontologies relate concepts to each other

• simple knowledge representation language Resource Description Framework (RDF)
  ◦ knowledge as a set of URI triples `<subject,predicate,object>`, e.g.
    ◦ `<john,like,mary>`, `<john,rdf:type,man>`, ...

• highly-expressive ontology and rule languages
  ◦ fragments of first-order logic, OWL based on description logics (DLs)
  ◦ core reasoning tasks have decidable algorithms
  ◦ focus on reasoning efficiency (=> define efficient OWL profiles: EL, QL, RL)

• ~2008: web of linked open data ("little semantics goes a long way")
  ◦ publish data as RDF and access it using the query language SPARQL
  ◦ use very little OWL reasoning
Semantic Web stack

- User interface and applications
- Trust
- Proof
- Unifying logic
- Querying: SPARQL
  - Ontologies: OWL
  - Rules: RIF/SWRL
  - Taxonomies: RDFS
- Data interchange: RDF
- Syntax: XML
- Identifiers: URI
- Character set: UNICODE
Expressive SW languages: OWL

- formal language for defining terms (URIs) via other terms
- entities: classes (aka concepts), individuals, object and data properties (roles, relations)
- simple and complex classes and properties
  - `uri://.../animal` # denotes the class of animals
  - `uri://.../eats` # denotes the relation of eating
  - `(uri://.../animal AND uri://.../African)` # "African animal"
  - `InverseOf(uri://.../eats)` # the relation of being eaten by something
- complex classes can use: negation, conjunction, disjunction, existential and universal restrictions, cardinality restrictions
  - `(african AND animal) OR (NOT (InverseOf(eat) SOME animal))`
- axioms (ontology is a set of axioms)
  - `lion SubClassOf animal`
  - `(african AND animal) SubClassOf (InverseOf(eat) SOME lion)`
The structure of OWL (2009)
Syntax of OWL

• many different syntaxes
  ◦ RDF/XML
  ◦ OWL/XML
  ◦ Turtle
  ◦ functional syntax
  ◦ Manchester syntax

• some based on the RDF idea of triples (RDF/XML, Turtle)
• some based on description logic style syntax (functional-style, Manchester)
• in this tutorial: functional-style and Manchester syntax, e.g.
  ◦ \texttt{SubClassOf(:country ObjectSomeValuesFrom(:border :country))}
  ◦ \texttt{country SubClassOf border SOME country}

• also in this tutorial: ACE-based syntax, e.g.
  ◦ Every country borders a country.
# Semantics of OWL

The meaning of OWL elements is defined via set theory.

<table>
<thead>
<tr>
<th>OWL</th>
<th>set theory</th>
<th>example</th>
</tr>
</thead>
<tbody>
<tr>
<td>individual</td>
<td>instance of some set</td>
<td>Switzerland</td>
</tr>
<tr>
<td>class</td>
<td>set of instances</td>
<td>country</td>
</tr>
<tr>
<td>object property</td>
<td>set of pairs <code>&lt;instance1, instance2&gt;</code></td>
<td>bordering</td>
</tr>
<tr>
<td>data property</td>
<td>set of pairs <code>&lt;instance, dataitem&gt;</code></td>
<td>population</td>
</tr>
<tr>
<td><code>C1 AND C2</code></td>
<td>intersection of <code>C1</code> and <code>C2</code></td>
<td>EU-country and NATO-country</td>
</tr>
<tr>
<td><code>C1 OR C2</code></td>
<td>union of <code>C1</code> and <code>C2</code></td>
<td>EU-country or NATO-country</td>
</tr>
<tr>
<td><code>NOT C</code></td>
<td>complement set of <code>C</code></td>
<td>not an EU-country</td>
</tr>
<tr>
<td><code>P SOME C</code></td>
<td>set of things that have a <code>P</code>-property with a member of class <code>C</code></td>
<td>something that flows into a lake</td>
</tr>
<tr>
<td>DifferentIndividuals(I1 I2)</td>
<td></td>
<td>Switzerland is not Germany</td>
</tr>
<tr>
<td>ClassAssertion(C I)</td>
<td><code>I</code> is an instance of <code>C</code></td>
<td>Switzerland is a country</td>
</tr>
<tr>
<td>SubClassOf(C1 C2)</td>
<td><code>C1</code> is a subset of <code>C2</code></td>
<td>every EU-country is a country</td>
</tr>
<tr>
<td>Domain(C P)</td>
<td>the subject of <code>P</code> is in <code>C</code></td>
<td>everything that borders something is a country</td>
</tr>
<tr>
<td>SubPropertyOf(P1 P2)</td>
<td><code>P1</code> is a subset of <code>P2</code></td>
<td>if X borders Y then X neighbors Y.</td>
</tr>
</tbody>
</table>

...
Semantics of OWL

- **Individuals** $i^I \in \Delta^I$
  - John
  - Mary

- **Concepts** $C^I \subseteq \Delta^I$
  - Lawyer
  - Doctor
  - Vehicle

- **Roles** $r^I \subseteq \Delta^I \times \Delta^I$
  - hasChild
  - owns

(Lawyer $\cap$ Doctor)
Semantics of OWL: UNA and OWA

**Unique Name Assumption (UNA):** differently named individuals are necessarily different instances of the domain. OWL **does not** make the UNA.

**Open World Assumption (OWA):** unstated facts are not assumed to be false. OWL makes the OWA.

Thus, OWL is different from some database/knowledgebase languages (SQL, Prolog) where a missing assertion means that it is false rather than unknown.
T-Box and A-Box

T-Box

• terminological statements / universal statements (universal quantification in first-order logic), e.g.
  ◦ Every EU-country is a country.
  ◦ If X border Y then Y borders X.
• conceptionally complex (domain, range, reflexivity, inheritance, ...)
• usually small and static

A-Box

• instance data, e.g.
  ◦ Switzerland is not a EU-country.
  ◦ Switzerland borders Germany.
• conceptually much simpler
• usually very large and often changing
Expressive SW languages: SWRL

- rule-like statements with explicit variables
  - OWL: eu-country SubClassOf country
  - SWRL: IF eu-country(?X) THEN country(?X)

- more expressive extension of OWL
  - OWL does not support unrestricted variable sharing
  - English: Every man that owns a car likes the car.
  - OWL approx.: (man AND own SOME car) SubClassOf like SOME car

- still a fragment of first-order logic, but reasoning is not decidable
Automatic reasoning with OWL

• reasoning helps to detect modeling errors during development time
  ◦ e.g. necessarily empty classes should be avoided

• reasoning can be used during runtime to classify new instances

• reasoning tasks
  ◦ is the ontology consistent?
  ◦ which axioms does the ontology entail? (e.g. classification = entailment of simple SubClassOf-axioms)
  ◦ is a class satisfiable (i.e. can it contain individuals?)
  ◦ which (smallest) subsets of the ontology cause a class to be unsatisfiable?
  ◦ which classes and individuals does the given class contain? (DL Query)

• reasoning is decidable, i.e. will always solve the task in finite time (although sometimes very slowly)

• many off-the-shelf reasoners: Pellet, HermiT, FaCT++, ...

• integrated into end-user tools (Protégé, Semantic MediaWiki, AceWiki, ...)
Reasoning example

(written in ACE for easier readability)

Ontology

Every country is a territory.
If X borders Y then Y borders X.
If X borders something then X is a country.
Germany borders Switzerland.

Entailments

Switzerland borders Germany.
Switzerland is a country.
Switzerland is a territory.
Germany is a country.
Germany is a territory.
It is false that Germany does not border Switzerland.
...

Unknown truth value

Switzerland is not Germany. # no UNA in OWL
Switzerland borders exactly 1 territory.
Switzerland borders itself.
...
OWL: Usage examples

• SNOMED Clinical Terms
  ◦ large (300k concepts) collection of medical terms, with NL labels and formal relations
  ◦ multilingual labels on concepts, currently English (~1m labels), Spanish, Danish, Swedish
  ◦ EL fragment of OWL (excludes disjunction, negation, cardinality constraints, ...)
  ◦ Common-cold SubClassOf causative-agent SOME Virus
  ◦ used >50 countries

• content management with semantic wikis
  ◦ e.g. Semantic MediaWiki, OntoWiki
  ◦ small subset of OWL considered for reasoning

• DBpedia
  ◦ structured part of Wikipedia (infoboxes)
  ◦ background ontology (SubClassOf, domain, range)
  ◦ multilingual
User interfaces for the Semantic Web

• semantic web is for both machines (agents) and humans

• humans require a user-friendly language to interact with the SW

• RDF's triple-based data model is very simple
  ◦ does not directly allow for a user-friendly syntax for complex structures
  ◦ originally was written in RDF/XML (unnecessarily verbose)

• OWL (version 2009) syntax is much more elegant (based on Description Logics)
  ◦ still remote from natural language
  ◦ lots of syntactic sugar for various types of SubClassOf-constructs

• separate ontology, rule, and query languages

• little focus in the SW community on UI issues, see e.g. D. Karger ESWC 2013 keynote "A Semantic Web for End Users"
  ◦ http://videolectures.net/eswc2013_karger_semantic/
Two examples of SW tools
Protégé

- open-source and widely used OWL editor (actually predates OWL)
- user interface
  - sub class tree hierarchy with drag-and-drop
  - otherwise reflects the OWL axiom types 1-to-1
  - integrates Manchester OWL Syntax (MOS) for expressing classes and axioms
- focus on T-Box editing
- integrates reasoners
  - query answering
  - entailment explanation
- WebProtégé: simpler web-based editor with focus on collaborative editing (e.g. revision history support)
Protégé
Protégé: DL Query

Query (class expression)

person and (not (drive some van))

Query results

Equivalent classes

Ancestor classes

animal
owl:Thing
person

Super classes

person

Sub classes

kid

Descendant classes

kid
mad-cow
owl:Nothing

Instances
Protégé: Entailment explanation

Smallest subset that explains why `mad-cow` is an empty class, indicating that the ontology contains a modeling error.
Wikis and Semantic wikis

**Wiki**

- user-friendly collaborative environment for knowledge management
- content typically unconstrained natural language, therefore not easily automatically processable
- powered by software, e.g. MediaWiki, best known example: Wikipedia

**Semantic wiki** = wiki + formal semantics

- provides a richer query language + improves consistency by reducing redundancy: the same information stated only once
- content typically in natural language enriched with typed links (i.e. RDF triples)

**User interface**

- wiki-style: plain text with embedded meta information
- forms for filling templates

**Software**: Semantic MediaWiki, Freebase, ...
Semantic MediaWiki

• extension of MediaWiki / Wikipedia
• add: "semantic annotations" structured formats for
  ◦ attaching attributes to concepts
  ◦ representing lists, tables, etc.
  ◦ representing maps, timelines, calendars, etc.
• queries that automatically generate wiki pages
• supports subset of OWL for query answering
  ◦ subclass, subproperty, inverse property, equality
  ◦ excludes e.g.: transitivity, domain/range, cardinality restrictions
• focus on A-Box editing
Semantic MediaWiki

Example: sentence in an article *Berlin* is entered as

| Berlin is the capital of [Is capital of::Germany] |
| and has a population of [Has population::3,292,365|3.3 million]. |

and is rendered as

| Berlin is the capital of Germany |
| and has a population of 3.3 million. |

and adds two triples to the underlying model

| <Berlin, Is_capital_of, Germany> |
| <Berlin, Has_population, 3292365> |
Semantic MediaWiki: Query

Which countries are located in Africa?

```
{{#ask: [[Category:Country]] [[located in::Africa]]
| ?Area#km²
| ?Name
| ?Population
| sort = population
| order = descending
}}
```

Is rendered as a sortable table listing links to the pages of category Country (or some of its subcategories) that contain the property located in (or some of its subproperties) with the value Africa.

<table>
<thead>
<tr>
<th>Country</th>
<th>Area</th>
<th>Name</th>
<th>Population</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nigeria</td>
<td>923,768 km²</td>
<td>131,530,000</td>
<td></td>
</tr>
<tr>
<td>Democratic Republic of the Congo</td>
<td>2,344,858 km²</td>
<td>59,319,660</td>
<td></td>
</tr>
<tr>
<td>South Africa</td>
<td>1,221,037 km²</td>
<td>47,432,000</td>
<td></td>
</tr>
<tr>
<td>Uganda</td>
<td>236,040 km²</td>
<td>38,816,000</td>
<td></td>
</tr>
<tr>
<td>Tanzania</td>
<td>945,090 km²</td>
<td>38,329,000</td>
<td></td>
</tr>
<tr>
<td>Sudan</td>
<td>2,505,813 km²</td>
<td>36,992,490</td>
<td></td>
</tr>
<tr>
<td>Kenya</td>
<td>582,650 km²</td>
<td>34,256,000</td>
<td></td>
</tr>
</tbody>
</table>
Multilingual Semantic Web
Multilingual SW: Ontology verbalization

- RDF, OWL, ... keywords are in English
- most ontologies are written with English entity names
- recommendation (in biomed ontologies) to name entities in a language-neutral way (with ID codes)
- entity and axiom annotations provide a way to include free-form natural language in the ontologies
  - e.g. via `rdf:label` with the ISO language tag
- The Lexicon Model for Ontologies (lemon)
  - new proposal for annotating ontologies with linguistic information
  - each entity is mapped to its corresponding lexical entries specifying the word forms and part of speech categories (for different languages)
Multilingual SW: Semantic wikis

- wikis: multilingual articles but interlinked only at the document level
- WikiMedia: proposal for multilingual Wikipedia
- AceWiki-GF: a prototype for creating ontologies in a multilingual way
Links

• the original semweb paper
• OWL specification: http://www.w3.org/TR/owl2-overview/
• Protégé: http://protege.stanford.edu/
• Semantic MediaWiki: http://semantic-mediawiki.org/
• Dagstuhl Seminar 12362 (2012): The Multilingual Semantic Web
  ◦ http://www.dagstuhl.de/12362
• lemon: http://lemon-model.net/
CNLs and ACE
Controlled Natural Languages

• motivation
  ◦ natural language with tool support, e.g. auto-completion, paraphrasing, consistency checking, question answering, translation
  ◦ wide-coverage parsers achieve only \(\sim90\%\) accuracy, and on simpler tasks (e.g. detecting subjects and objects)

• formal syntax, but fragment of a natural language

• formal semantics
  ◦ ambiguity handling
  ◦ translations to other (natural or formal) languages
  ◦ reasoning, paraphrasing, etc.

• many languages with different goals and formal properties
  ◦ see Tobias Kuhn. A Survey and Classification of Controlled Natural Languages, Computational Linguistics

• well-known example: Attempto Controlled English (ACE)

• frameworks: Grammatical Framework (GF)
Attempto Controlled English (ACE)

• subset of English
• controls ambiguity and synonymy
  ◦ A dog hates a cat. == There is a dog that hates a cat.
  ◦ A dog hates a cat. =/= Every dog hates a cat.
  ◦ Every dog hates a cat. == If there is a dog then the dog hates a cat.
• end-user documentation: construction and interpretation rules, as restrictions of English
  ◦ construction rules, interpretation rules, ...
  ◦ easy to learn
• well-defined translations to first-order logic, OWL, ...
• tool support for end-users
• developed in the Attempto project at the University of Zurich, lead by Norbert E. Fuchs
ACE: Language overview

• subset of natural English
  ◦ conjunction (and), disjunction (or), negation (no, not, it is false that), if-then, ...
  ◦ anaphoric references: pronouns (he, it), definite noun phrases (the man), variables (X, Y123)
  ◦ quantifiers: every, no, at least 3, ...
  ◦ content words: proper names, common nouns, verbs, adjectives, adverbs
• grammar is fixed, but users can change content words
• deterministic ambiguity handling
  ◦ anaphora resolution (France borders Spain and it borders Portugal.)
  ◦ quantifier scope (Every country borders a country.)
  ◦ attachment (Every EU-country borders a country that is a EU-country and is a NATO-country.)
Example: Syntactic restrictions

English: *Koalas eat eucalyptus leaves.*

Violates the ACE construction rules:

- every noun must have a determiner ('every', 'a', 'at least 2', ...)
- multi-word nouns are not allowed

Correct ACE:

- Every koala eats an eucalyptus-leaf.
- If a koala eats something X then X is an eucalyptus-leaf.
- ...
Example: Semantic restrictions

English: *Every EU-country borders a country that is a EU-country and is a NATO-country.*

The attachment differences must be expressed in a lexically different way in ACE:

• Every EU-country {borders a country that is a EU-country} and {is a NATO-country}.
• Every EU-country borders a country {{that is a EU-country} and {that is a NATO-country}}.

(The {curly brackets} are not part of ACE. They are used here to denote the scopes.)
Discourse Representation Structure

*The Mediterranean Sea is a sea. Every country that does not border a sea is a landlocked-country. Switzerland does not border the sea.*

<table>
<thead>
<tr>
<th>[A,B]</th>
<th>[F,G]</th>
</tr>
</thead>
<tbody>
<tr>
<td>object(A,sea,countable,na,eq,1)-1/4</td>
<td>object(F,landlocked-country,countable,na,eq,1)-2/13</td>
</tr>
<tr>
<td>predicate(B,be,named(The Mediterranean Sea),A)-1/2</td>
<td>predicate(G,be,C,F)-2/9</td>
</tr>
</tbody>
</table>

- [C]  
  - object(C,country,countable,na,eq,1)-2/2  
  - [D,E]  
    - object(D,sea,countable,na,eq,1)-2/8  
    - predicate(E,border,C,D)-2/6  

- [H]  
  - predicate(H,border,named(Switzerland),A)-3/4

- **DRS**  
  - interpreted as a first-order logic formula  
  - ACE content words map to atomic conditions  
  - ACE function words (and, or, not, if-then) introduce various complex DRS "boxes"  
  - variables identify content words  
  - variables denote anaphoric references to nouns, the DRS box structure reflects accessibility constraints on references
Syntactic sugar

It is possible to express the same meaning (the same DRS) in syntactically different ways, e.g. every = if-then:

- Every koala eats an eucalyptus-leaf. # M1
- If there is a koala then the koala eats an eucalyptus-leaf. # M1
- Everything that a koala eats is an eucalyptus-leaf. # M2
- If a koala eats something X then X is an eucalyptus-leaf. # M2

relative clause = coordination

- Every mammal that is endemic-to Australia is a marsupial or is a .... # M3
- If a mammal is endemic-to Australia then the mammal is a marsupial or is a .... # M3
- If there is a mammal and the mammal is endemic-to Australia then the mammal is a marsupial or is a .... # M3

(M_{1,2,3} indicate sentences with the same DRS.)
Content word lexicon

Content words are user-defined. They are classified by their word class and have one or more surface forms.

- nouns: 2 forms: singular: man, plural: men
- proper names: 1 form: singular: John
- verbs
  - intransitive (wait), transitive (eat), ditransitive (give)
  - 3 forms: 3rd person finite: eats, infinite: eat, past participle: eaten
- adjectives
  - intransitive (rich), transitive (fond-of)
  - 3 comparison forms (rich, richer, richest)
- adverbs (3 comparison forms)

The lexicon allows the definition of aliases but does not otherwise provide any semantics, e.g. statements like "Every man is a human." can be made only at the level of ACE.
Tools

- ACE parser (APE): maps ACE to DRS
- DRS verbalizer: maps DRS to ACE
  - the roundtrip: ACE -parser-> DRS -verbalizer-> ACE usually results in a syntactically different ACE text thanks to syntactic sugar
- DRS translators: map DRS to
  - OWL/SWRL
  - TPTP
  - ...
- look-ahead editor for a subset of ACE
- native ACE reasoner: RACE
- OWL verbalizer: maps OWL to ACE
- end-user ACE / SW ontology editors:
  - ACE View
  - AceWiki
Attempto Parsing Engine (APE)

- tokenizes and parses the given ACE text
- resolves anaphora
- translates to one or more logical forms (TPTP, OWL/SWRL, ...) via DRS
- paraphrases the input in ACE via DRS
- accessible from Prolog, Java, HTTP, commandline

**Commandline example.** Parse a sentence into the DRS form.

```
$ echo "No dog is a cat." | ape.exe -solo drspp
[]
  [A]
  object(A,dog,countable,na,eq,1)-1/2
=>
[]
  NOT
  [B,C]
  object(B,cat,countable,na,eq,1)-1/5
  predicate(C,be,A,B)-1/3
```
ACE as a semantic web language
Motivation

• user-friendly language because based on English
  ◦ standard English instead of various formal notations
  ◦ easy to read (and read out-loud)
  ◦ easy to write (?)
• single language instead of 3 different languages for ontologies, rules and queries
• different (more natural) motivation for the syntactic sugar
• usage:
  ◦ verbalizing existing ontologies
  ◦ writing new and modifying existing ontologies
  ◦ querying existing ontologies
  ◦ presenting entailments and entailment explanations
## Mapping between ACE and OWL/SWRL

<table>
<thead>
<tr>
<th>ACE</th>
<th>OWL/SWRL</th>
</tr>
</thead>
<tbody>
<tr>
<td>proper name</td>
<td>individual</td>
</tr>
<tr>
<td>noun</td>
<td>named class</td>
</tr>
<tr>
<td>intransitive adjective</td>
<td>named class</td>
</tr>
<tr>
<td>noun phrase (with relative clause)</td>
<td>complex class</td>
</tr>
<tr>
<td>transitive verb with NP object</td>
<td>property</td>
</tr>
<tr>
<td>transitive verb with data object</td>
<td>data property</td>
</tr>
<tr>
<td>transitive adjective</td>
<td>property</td>
</tr>
<tr>
<td>sentence</td>
<td>OWL axiom or SWRL rule (SWRL rule as a fallback)</td>
</tr>
<tr>
<td>question (with 1 query word)</td>
<td>DL-Query</td>
</tr>
<tr>
<td>text</td>
<td>ontology</td>
</tr>
</tbody>
</table>
ACE sentence and question in OWL

Every country that does not border a sea is a landlocked-country.

SubClassOf(
  ObjectIntersectionOf(
    :country
    ObjectComplementOf(
      ObjectSomeValuesFrom(
        :border
        :sea
      )
    )
  )
  :landlocked-country
)

Which country is a landlocked-country?

ObjectIntersectionOf(
  :country
  :landlocked-country
)

Verbalizing OWL ontologies as ACE texts

OWL contains many shorthand constructs (DisjointClasses, ObjectPropertyDomain) with no direct ACE counterpart

```plaintext
DisjointClasses(
    :country
    ObjectHasSelf( :border )
)
```

Directly reflecting the keywords and axiom structure would give something like:

```
Countries and self-borderers are disjoint.
```

ACE-style verbalization gives a more natural:

```
No country borders itself.
```
Rewrite an OWL axiom, e.g.

```
ObjectPropertyDomain( write human )
```

into something with a more suitable structure (but preserving meaning):

```
SubClassOf(
    ObjectIntersectionOf(  
        owl:Thing  
        ObjectSomeValuesFrom(  
            write  
            owl:Thing  
        )  
    )  
    human
)
```

then directly map it to ACE (with a Prolog DCG grammar):

```
Everything  
   that  
    writes something  
  is a human.
```
OWL Verbalizer: Issues

• OWL naming conventions for entities
  ◦ usually: nouns for classes, verbs for properties
  ◦ sometimes contain complex structure: nonNormal, MaleOrFemalePatient

• simple one-to-one mapping of axioms to sentences
  ◦ other approaches also reorder/combine axioms
Evaluation: Comparison to MOS

- Manchester OWL Syntax (MOS)
  - concise notation for OWL
  - part of the OWL (2009) standard
  - used in editors like Protégé

  - compare the ACE and MOS representation of wide variety of OWL axiom types
  - evaluation with 64 users (students, but not of computer science)
  - ask subjects to look at diagrams and evaluate corresponding ACE and MOS statements as true or false
  - result: ACE is easier to learn and understand
  - no evaluation of "writability"
Evaluation: Comparison to MOS

<table>
<thead>
<tr>
<th>OWL (Manchester syntax)</th>
<th>ACE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bob HasType developer</td>
<td>Bob is a developer.</td>
</tr>
<tr>
<td>developer SubTypeOf professional</td>
<td>Every developer is a professional.</td>
</tr>
<tr>
<td>developer SubTypeOf owns some cup</td>
<td>Every developer owns a cup</td>
</tr>
<tr>
<td>Bob HasType owns some (not cup)</td>
<td>Bob owns something that is not a cup.</td>
</tr>
<tr>
<td>loves SubRelationOf likes</td>
<td>If X loves Y then X likes Y.</td>
</tr>
</tbody>
</table>

Which of the statements are true and which are false?

- Every golfer sees a TV.
- Kate sees no officer.
- Every officer loves nothing but golfers.
- Bill does not love Kate.
- Kate sees nothing but officers.
- John loves something that is not a TV.
- Everything that sees nothing but aquariums is an officer.
- Kate loves an officer.
- Kate sees Bill.
- Everything that loves a person is a golfer.
Evaluation: Comparison to MOS

- **Objective Understandability (Classification Score):**
  - ACE: 91.4%
  - Manchester OWL Syntax: 86.3%

- **Subjective Understandability (Questionnaire Score):**
  - ACE: 2.59
  - Manchester OWL Syntax: 1.92

- **Needed Effort (Time in Seconds):**
  - ACE: 13.72
  - Manchester OWL Syntax: 18.42
**ACE as a SW language: Issues**

- need to support word-forms (e.g. in the lexicon editor):
  - 2 for nouns: man, men
  - 3 for verbs: eats, eat, eaten by

- English-specific
  - can be misleading (if interpreted as English, and not ACE)
  - non English speakers might prefer a more language-neutral notation

- harder to implement tool support (parser, verbalizer, etc.) for ACE than for MOS
  - covering a wide variety of programming languages
Two ACE tools for the SW
ACE View

- ACE-based OWL/SWRL editor, viewer, query interface
- plug-in for Protégé 4+
- provides set of ACE "views" to the ontology
- integration with other Protégé views
  - changes done via standard Protégé views are reflected in the ACE views
  - motivation: sometimes graphical UI is simpler to use than plain text
- integrates both ACE->OWL/SWRL and OWL->ACE
- ACE-based entailment explanation
ACE View: Snippets

Presenting the complete ontology as a sortable/searchable set of ACE sentences.
ACE View: Entailments

Explaining an entailment by a set of ACE sentences.
ACE View: Question answering

Counting and presenting DL Query results.

<table>
<thead>
<tr>
<th>Question</th>
<th>Indv.</th>
<th>Sub class</th>
<th>Super class</th>
</tr>
</thead>
<tbody>
<tr>
<td>What is an EU-country?</td>
<td>27</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>What is a NATO-country?</td>
<td>26</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Which EU-country is a NATO-country?</td>
<td>21</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>Which EU-country is not a NATO-country?</td>
<td>2</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>Which NATO-country is not an EU-country?</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>What is an EU-country and is a NATO-country and is a G8-country?</td>
<td>4</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>What is a landlocked-territory?</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Which EU-country borders a NATO-country?</td>
<td>24</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>What does Austria border?</td>
<td>8</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>What does Belgium border?</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>What does Bulgaria border?</td>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>What does Cyprus border?</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

21 named individuals: Belgium, Bulgaria, Czech_Republic, Denmark, Estonia, France, Germany, Greece, Hungary, Italy, Latvia, Lithuania, Luxembourg, Netherlands, Poland, Portugal, Romania, Slovakia, Slovenia, Spain, United_Kingdom.

1 named classes: baltic-state.

Every answer is: country, EU-country, European-country, NATO-country, territory, ... (5)
ACE View vs Protégé

Property axioms presented in ACE vs checkboxes+labels.
AceWiki

• semantic wiki engine
• focus on user-friendliness and high expressivity of formal content
• uses ACE for the content
  ◦ OWL-compatible fragment of ACE (e.g. excludes prepositional phrases and adverbs)
  ◦ user is syntactically assisted by a look-ahead editor
• uses OWL for the reasoning language
  ◦ supports different OWL profiles
  ◦ more expressive than usually in semantic wikis
  ◦ completely hidden from the users
AceWiki: Screenshot

AceWiki article about *landlocked country* and the look-ahead editor.
AceWiki: Main features

- model
  - content word = article (like in traditional wikis)
  - article = collection of statements
  - statement = declarative sentence or query or comment

- declarative sentences and queries are written in an OWL-compatible subset of ACE
  - Codeco grammar which formally describes this subset and
  - ... makes it available via a look-ahead editor
  - UI for avoiding syntactically and semantically not supported inputs

- automatic feedback
  - inconsistent sentences are tagged by red color
  - queries are answered by automatically populating the list of matching entities

- collaborative editing of
  - lexicon
  - articles
AceWiki: Evaluation

Two small usability experiments:

• altogether 26 untrained participants
• task: collaborative creation of a knowledge base
• results:
  ◦ 78%-81% of the sentences were correct and sensible
  ◦ 61%-70% of them were complex (containing negations, implications, disjunctions or number restrictions)
  ◦ creation of a correct sentence every 5-6 minutes
  ◦ definition of a new word every 5-7 minutes

=> Even untrained users can effectively use AceWiki
AceWiki: Other usages

AceWiki (or parts of it) have been used in various research projects, e.g.

- Coral: a CNL-based query interface for annotated text corpora (UZH, Zurich)
  - uses the AceWiki sentence editor
  - does not use ACE, but a new CNL that maps to the ANNIS Query Language
  - no collaborative editing
  - source code: https://github.com/tkuhn/Coral

- AceCAPTCHA (AGH UST, Kraków)
  - CAPTCHA = Completely Automated Public Turing test to tell Computers and Humans Apart
  - AceWiki preloaded with terminological knowledge (entered by experts)
  - presents end-users with ACE questions about the wiki content to (1) distinguish them from computers (CAPTCHA) and to (2) populate the wiki with instance data (collaborative editing)

- ACE for source code documentation (University of Chile)
  - AceWiki populated with ACE statements about software structure (classes, methods, inheritance, ...)
  - users can ask ACE questions about module dependencies etc.
AceWiki: Small case study

Article on *Fynbos* containing the original definition (in full English), its formal content (in ACE), and automatically generated upper concepts.

---

**fynbos**

- *Fynbos or Macchia*
  - *is an arid veld in which the dominant vegetation is non-grass with Sclerophyllous and ericoid or cupressoid leaves.*
- Every fynbos is a macchia.
- Every fynbos is an arid-veld.
- No fynbos is a grassland.
- Every fynbos has-plant-property a sclerophyllous-plant that has-leaf-type Ericoid-leaf or that has-leaf-type Cupressoid-leaf.

---

**fynbos - Hierarchy**

**Upward**
- Every fynbos is an arid-veld.
- Every fynbos is a natural-vegetation.
- Every fynbos is a vegetation.
- Every fynbos is a veld.

**Downward**
(downward hierarchy is empty)
AceWiki: Small case study

Agriculture glossary (= set of term-definition pairs) as a CNL-based semantic wiki

- convert an existing glossary into the wiki format (wiki article for every term, every term considered to be a common noun)
- represent the core information in the definition as a set of ACE sentences
- purpose
  - educational: definition easier to understand for non English speakers
  - sentences are formal: automatic semantic feedback
  - collaborative development of course content
  - sentences are easier to translate (future work)
- results
  - several new terms (e.g. relations/verbs) were added
  - some ambiguity/vagueness was identified and resolved
  - correct and natural formulation is sometimes difficult to find
Links

• some overview papers about CNL for SW
  ◦ Paul Smart. Controlled natural languages and the semantic web. (2008)

• Attempto project: http://attempto.ifi.uzh.ch/
  ◦ documentation, tools, demos, etc.

• ACE-based SW tools
  ◦ ACE View (http://attempto.ifi.uzh.ch/aceview/)
  ◦ AceWiki (http://attempto.ifi.uzh.ch/acewiki/)

• other CNL-based editors for OWL
  ◦ ROO (http://sourceforge.net/projects/confluence/)
  ◦ Fluent Editor (http://www.cognitum.eu/semantics/FluentEditor2/)
  ◦ CLOnE
Grammatical Framework as a CNL framework
Other CNLs (in comparison to ACE)

ACE is general-purpose, English-based (and difficult to port to other NLs), with fixed grammar and largely fixed interpretation.

Other CNLs:

• more domain specific
  ◦ several SW-oriented CNLs: Rabbit, SOS, Clone
  ◦ optimized for querying (not authoring), e.g. Coral, NL-interfaces to databases

• no underlying formal semantics

• relaxed syntactic constraints

• relaxed or different ambiguity handling

• speech-oriented: voice commands for mobile devices

• other goals:
  ◦ automatic translation into other natural languages
Components of a CNL framework

• programming language with built-in support for describing natural language grammars
• CNL design guidelines and best practices
• language-independent parsing engine
• library that contains the linguistic knowledge of several NLs, ideally with a language-independent API
• general enough to support a variety of uses cases
Grammatical Framework (GF)

- framework for multilingual grammar engineering
  - functional programming language optimized to handle natural languages
  - resource grammar library implementing common morphological and syntactic structures
- a GF program (aka grammar) consists of
  - language-neutral abstract syntax
  - multiple concrete syntaxes that implement the abstract functions and categories, specifying words, word order, agreement, etc.
- main operations
  - parsing: map a string in some language to abstract tree(s)
  - linearization: linearize tree(s) as strings in some language
  - translation = parse a string in language A to tree(s) + linearize these tree(s) as strings in language B
- various tools + bindings to Python, Java, Javascript, Prolog, ...
- developed at the University of Gothenburg, lead by Aarne Ranta
that Italian pizza is very boring, delicious, expensive, fresh, Italian, very warm.

that Italian pizza is very expensive.

Pred (That (Mod Italian Pizza)) (Very Expensive)

-daardie Italiaanse pizza is baie duur
-[Pizza] icmpf or g- or .
-onazi italiano a pesa e molto cara
-aquella pizza italiana è molto cara
-那个意大利式的比萨饼是非常昂贵的
-tamta italská pizza je velmi drahá
-die Italiaanse pizza is erg duur
-that Italian pizza is very expensive
GF example: Abstract module

abstract Unitconv = {
  flags startcat = Unitconv;
  cat Unit; Unitconv;
  fun
    unitconv : Unit -> Unit -> Unitconv;
    land_mile, nautical_mile : Unit;
}

• declares the categories (Unit, Unitconv)

• declares the functions (unitconv, land_mile, nautical_mile) and their types (Unit->Unit->Unitconv, Unit)

• trees that can be obtained from the start category:
  ◦ unitconv land_mile nautical_mile
  ◦ unitconv land_mile land_mile
  ◦ ...

GF example: Concrete modules

concrete UnitconvEng of Unitconv = {
  lincat Unit, Unitconv = {s : Str} ;
  lin
    unitconv x y = {s = "how much is" ++ x.s ++ "in" ++ y.s ++ "?"} ;
    land_mile = {s = "mile"} ;
    nautical_mile = {s = "nautical mile" | "mile"} ;
}

concrete UnitconvWolfram of Unitconv = {
  lincat Unit, Unitconv = {s : Str} ;
  lin
    unitconv x y = {s = "convert" ++ x.s ++ "to" ++ y.s} ;
    land_mile = {s = "mile"} ;
    nautical_mile = {s = "nmi"} ;
}

• define how trees are linearized in a concrete language
• assign linearization structures to categories (e.g. "record containing a string": { s : Str } )
• define how functions are linearized, e.g. as strings or concatenation of strings
  ◦ e.g. ( x.s ++ "in" ) concatenates the string of the argument ( x.s ) with the string "in"
GF parsing and linearizing

Parsing i.e. converting a string **how much is nautical mile in mile ?** to tree(s)

```sh
Unitconv> parse -lang=Eng "how much is nautical mile in mile ?"
unitconv nautical_mile land_mile
unitconv nautical_mile nautical_mile
```

Linearization i.e. converting a tree **unitconv nautical_mile land_mile** to string(s)

```sh
Unitconv> linearize -treebank -list (unitconv nautical_mile land_mile)
UnitconvEng: how much is nautical mile in mile ?, , how much is mile in mile ?
UnitconvWolfram: convert nmi to mile
```

Translation i.e. parse + linearize

```sh
Unitconv> parse -lang=Eng "how much is nautical mile in mile ?" | l -lang=Wolfram
convert nmi to mile
convert nmi to nmi
```
GF example: More complex example

abstract Geography = {
  cat
  Country ; Relation ;
  fun
    germany : Country ;
    switzerland : Country ;
    border : Country -> Country -> Relation ;
}

concrete GeographyGer of Geography = {
  param
    CaseGer = Nom | Acc | Dat ;
  lincat
    Country = CaseGer => Str ; Relation = Str ;
  lin
    germany = table { _ => "Deutschland" } ;
    switzerland = table { Dat => "der Schweiz" ; _ => "die Schweiz" } ;
    border c1 c2 = c1 ! Nom ++ "grenzt an" ++ c2 ! Acc ;
}

• German cases require a more flexible linearization structure (table of strings)

• fortunately this (and other types of) complexity can be hidden into a library, allowing to
  simplify the code to something like:

  border c1 c2 = c1 ! Nom ++ (mkVP3rdPers "grenzen" c2)
GF Resource Grammar Library (RGL)

- morphology and syntax for ~30 languages via language-neutral API
- developers do not need detailed knowledge of the languages that they want to support in their application

<table>
<thead>
<tr>
<th>Function</th>
<th>Type</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>genericCl</td>
<td>VP → Cl</td>
<td>one sleeps</td>
</tr>
<tr>
<td>mkCl</td>
<td>NP → V → Cl</td>
<td>she sleeps</td>
</tr>
<tr>
<td>mkCl</td>
<td>NP → V2 → NP → Cl</td>
<td>she loves him</td>
</tr>
<tr>
<td>mkCl</td>
<td>NP → V3 → NP → NP → Cl</td>
<td>she sends it to him</td>
</tr>
<tr>
<td>mkCl</td>
<td>NP → VV → VP → Cl</td>
<td>she wants to sleep</td>
</tr>
<tr>
<td>mkCl</td>
<td>NP → VS → S → Cl</td>
<td>she says that I sleep</td>
</tr>
<tr>
<td>mkCl</td>
<td>NP → VQ → OS → Cl</td>
<td>she works</td>
</tr>
<tr>
<td>mkCl</td>
<td>NP → VA → A → Cl</td>
<td>she becomes</td>
</tr>
<tr>
<td>mkCl</td>
<td>NP → VA → AP → Cl</td>
<td>she becomes</td>
</tr>
<tr>
<td>mkCl</td>
<td>NP → V2A → NP → A → Cl</td>
<td>she paints</td>
</tr>
<tr>
<td>mkCl</td>
<td>NP → V2A → NP → AP → Cl</td>
<td>she paints</td>
</tr>
<tr>
<td>mkCl</td>
<td>NP → V2S → NP → S → Cl</td>
<td>she answers</td>
</tr>
<tr>
<td>mkCl</td>
<td>NP → V2Q → NP → OS → Cl</td>
<td>she asks</td>
</tr>
<tr>
<td>mkCl</td>
<td>NP → V2V → NP → VP → Cl</td>
<td>she begins</td>
</tr>
<tr>
<td>mkCl</td>
<td>NP → VPSlash → NP → Cl</td>
<td>she begins</td>
</tr>
</tbody>
</table>

- API: mkUt (mkCl she_NP say_VS (mkS (mkCl i_NP sleep_V)))
- Afr: sy sê dat ek slaap
- Bul: ma kasa , che a'i cny
- Cat: ella du que jo dormo
- Chin: 她说我睡
- Dan: hun siger at jeg sover
- Dut: ze zegt dat ik slaap
- Eng: she says that I sleep
- Fih: hän sanoo essá miñá nukun
- Fre: elle dit que
- Ger: sie sagt , dass ich schlafe
- Gre: tini léet òti eva koujyta
- Hin: वह कहती है कि मैं सोता हूँ
- Ita: lei dice che io dormo
RGL: Content words

• some languages have many content word forms, e.g. Estonian has ~28 noun forms (2 numbers with 14 cases each)

• smart paradigm
  ◦ construct (all) word forms on the basis of as few base forms (nominative, genitive, ...) as possible
  ◦ other input information: gender (for some languages), verb valency (for verbs, transitive adjectives, ...)

• benefit: keeps the lexicon equally simple across languages

• lexicon editors are likely to be familiar with the base forms (rather than language-specific morph. type systems)

• examples
  ◦ mkN "dog" constructs all the noun forms for the singular nominative "dog": dog, dogs, dog's, dogs'
  ◦ mkN "man" "men" (two forms needed for irregular English words)

• some languages in the RGL have also large (~50k entries) lexicons (with mapping to WordNet senses)
RGL: Syntax

- Language independent API for many syntactic functions
  - `mkNP : Numerl -> N -> NP : five men`
  - `mkCl : NP -> V -> Cl : five men sleep`
- Library handles word order, agreement, choice of function words, etc.
  - "red apple" vs "pomme rouge"
- Language-specific constructions can be included in the Extra-modules
  - E.g. Verb phrase coordination

Example: constructing a clause:

```
$ gf
> i -retain alltenses/TryEng.gfo
> cc -one
    mKS pastTense
       (mkCl (mkNP (mkNumeral "1") (mkN "dog")) (mkV "sleep" "slept" "slept"))
one dog slept
> cc -one
    mKS (mkCl (mkNP (mkNumeral "2") (mkN "dog")) (mkV "sleep" "slept" "slept"))
two dogs slept
```
RGL-based GF programs

```
incomplete concrete FoodsI of Foods = open Syntax, LexFoods in {
  lincat
    Phrase = Cl ;
    Item = NP ;
  ...
  lin
    is x y = mkCl x y ;
    ...
    wine = mkCN wine_N ;
    ...
}
concrete FoodsGer of Foods = FoodsI with
  (Syntax = SyntaxGer),
  (LexFoods = LexFoodsGer) ;
```

- best practice for multilingual grammars: share most of the code via a functor
- functor = "incomplete concrete" module that is parametrized by language-specific resources
- the functor references the RGL via its language-neutral API (Cl, NP, mkCl, mkCN)
- the concrete languages import the functor and plug in language-specific resources
Benefits of the GF framework

• support for multilinguality
• support for both parsing and generation
• grammar engineering tools and guidelines
• parser/translator with look-ahead support
• library of pre-implemented linguistic knowledge for ~30 languages
• large lexicons (for some languages)
ACE-in-GF
Motivation

• implement ACE in a multilingual way
  ◦ ACE-based modeling and ACE tools become available to speakers of other natural languages
  ◦ test how English-specific is ACE
• implement ACE in a different and more general formalism
• gain new tools
  ◦ tree-based sentence editor
  ◦ access to other programming languages (C, Python, Javascript, ...)
Multilingual ACE

An ACE grammar implemented in GF adds multiple natural languages as front-ends to ACE. As a result, these languages can be mapped to and from various formal languages already supported by ACE.
German <-> ACE <-> OWL

German

Jedes Land, das nicht an ein Meer grenzt, ist ein Binnenland.

ACE-in-GF tree

baseText (sText (s (vpS (everyNP (relCN (cn_as_VarCN country_CN))
  (neg_predRS which_RP (v2VP border_V2 (thereNP_as_NP
    (aNP (cn_as_VarCN sea_CN)))))) (npVP (thereNP_as_NP
    (aNP (cn_as_VarCN landlocked_country_CN)))))))

ACE

Every country that does not border a sea is a landlocked-country.

OWL

SubClassOf(
  ObjectIntersectionOf(
    :country
    ObjectComplementOf(
      ObjectSomeValuesFrom( :border :sea )
    )
  )
  :landlocked-country
)
Implementation of ACE-in-GF

• extension of Angelov and Ranta. Implementing Controlled Languages in GF (CNL 2009)

• implementation of the ACE syntax
  ◦ focus on the subset of ACE that can be mapped to OWL
  ◦ about 100 syntactic functions reflecting the reference implementation of ACE (APE)
  ◦ no direct generation of discourse representation structures (DRS)
  ◦ almost 100% coverage at almost 0% ambiguity (formally tested for ACE)

• multilinguality
  ◦ support most RGL languages: Bulgarian, Catalan, Chinese, Danish, Dutch, English, Estonian, Finnish, French, German, Greek, Hindi, Italian, Latvian, Maltese, Norwegian, Polish, Romanian, Russian, Spanish, Swedish, Thai, Urdu
  ◦ RGL-based design provides automatic increase in quality and language-coverage over time

• lexicon
  ◦ user-specified lexicon as needed by ACE applications
  ◦ rely on RGL smart paradigms and large lexicons
ACE: every person that speaks a language X does not forget X.
Bul: всеки човек който говори език X не забравя X.
Cat: cada persona que parla una llengua X no oblida X.
Dan: hver person, som taler et sprog X glemmer ikke X.
Dut: elke persoon, dat een taal X spreekt vergeet niet X.
Est: iga inimene, kes räägib keelt X ei unusta X.
Fin: jokainen henkilö, joka puhuu kieltä X ei unohda X:ää.
Fre: chaque personne qui parle une langue X n’oublie pas X.
Ger: jede Person, die eine Sprache X spricht vergisst X nicht.
Gre: κάθε πρόσωπο που μιλά μία γλώσσα τον X δεν ξεχνά τον X.
Hin: हर [person_CN], जो [language_CN] X बोलता है X नहीं भूलता है.
Ita: ogni persona che parla una lingua X non dimentica X.
Mlt: kull persuna, li jkellem lingwa X ma jinsix X.
Lav: ikviena persona, kas saka valodu X neaizmirst X.
Nor: hver person, som snakker et språk X glemmer ikke X.
Pol: każda osoba, która rozmawia z językiem X nie zapomina X.
Ron: orice persoană care vorbește o limbă X nu îl uită pe X.
Rus: каждое лицо, который говорит на языке X не забывает X.
Spa: cada persona que habla una lengua X no olvida X.
Swe: varje person, som talar ett språk X glömmer inte X.
Tha: ????? ?? ?? ??? ????? ???? ??? ?? X
Urd: نئي؟ بھولتا؟ X بھولتا؟ ر شخص，چو زياز?
RGL-based implementation

Import most of the implementation from the RGL using a functor.

- adding a new language is easy
- profit from continuous improvements in the RGL
ACE-in-GF: Issues

• precision problems (over-generation), which are visible in the look-ahead editor
  ◦ anaphoric references do not obey DRS accessibility constraints (e.g. Every man likes the woman.)
  ◦ some language-independent features (e.g. ACE NP types) are hard to describe in the abstract syntax

• coverage problems in some languages
  ◦ e.g. verb phrase coordination not available in the core RGL
  ◦ can be handled in the tool (e.g. AceWiki-GF) using paraphrasing, etc.

• ambiguity problems in some languages
  ◦ e.g. missing determiners in Finnish
  ◦ can be handled in the tool (e.g. AceWiki-GF) using disambiguation dialogues

• by default, sentences use the same structure in every language
  ◦ extra work and linguistic competence is needed to override this (possibly using Extra-modules)

• lack of smart paradigms and/or large lexicons for some languages
Evaluation of ACE-in-GF: Design

- picked 10 languages which had a complete implementation
- generated ~100 ACE sentences/questions and automatically translated them to all the languages, ensuring
  - full coverage of all the grammar functions
  - large coverage of OWL axiom structures (subclass, range, domain, transitivity, ...)
- measured translation accuracy from ACE to other languages
- used Google Translate as the baseline
- ... and 20 human evaluators (2 per language) as the gold standard
Evaluation of ACE-in-GF: Results

- participants preferred ACE-in-GF translations to Google translations and post-edited them less
- many edits were just stylistic
  - e.g. users preferred elliptical sentences but these are not allowed in ACE
- some languages performed clearly better, e.g. Finnish, German, Dutch
Evaluation of ACE-in-GF: Results
AceWiki-GF
A Multilingual CNL-based Semantic Wiki
Multilingual CNL-based Semantic Wiki

- multiple languages
  - natural: English, German, ...
  - formal: first-order logic, OWL, ...
  - languages for content vs user interface
- CNL-based
  - backed by formal grammar(s)
  - formal languages are hidden
- semantic
  - content automatically kept in sync via precise translation
  - consistency checking, question answering, ... (depending on the domain)
- wiki
  - user-friendly
  - collaborative
Motivation

for adding GF to AceWiki

• increase user-friendliness for non English speakers
• experiment with a more general CNL setting
  ◦ content not necessarily based on ACE and OWL
• experiment with more aspects of collaboration
  ◦ ambiguity handling
  ◦ full grammar editing
Possible use cases

• multilingual ontology editor
  ◦ e.g. environment where users agree on the content and multilingual vocabulary of an OWL-style ontology (for a certain domain, e.g. geography)
  ◦ like AceWiki, but multilingual

• catalog of museum objects
  ◦ each object (e.g. painter, painting) on its own wiki page
  ◦ rich queries (e.g. "which Dutch painter painted which French painter?")
  ◦ like previous, but more focus on instance data and multimedia content

• tourist phrasebook
  ◦ book structure (chapters and sections)
  ◦ multilingual content presented in parallel (at least 2 languages)
  ◦ e.g. based on the MOLTO Phrasebook grammar
  ◦ like AceWiki, but different UI and no reasoning

• other
  ◦ collection of math exercises
  ◦ ...

AceWiki integration with GF

- wiki content is based on a (single) GF grammar
  - provided by GF Webservice / Cloud service
- wiki entry is GF abstract tree set
  - viewed via linearization(s)
  - can represent ambiguity
- multilingual viewing and editing of wiki content
  - grammar-based look-ahead editing that shows next possible tokens
  - ambiguity resolution via another concrete language
- grammar integrated into the wiki
  - GF grammars are very modular
  - grammar modules as wiki articles (wiki-linking of grammar and content)
  - grammar can be changed while editing the wiki
AceWiki-GF user interface

- UI not (yet) customizable for any specific application (e.g. MOLTO Phrasebook)
  - i.e. most suitable for ACE-based wikis
- language-switching menu
  - displays the wiki articles and sentence editor in the given language
  - user interface labels reflect the content language
- grammar editor
  - table-based editor for lexical functions
  - basic full grammar editor
- ambiguity resolution dialog
ACE-based geography wiki

- main use case developed in the MOLTO project
- uses ACE-in-GF as the underlying grammar
ACE-based geography article
Ambiguity resolution

Disambiguation dialog (only) if the entry was added in another language, and the trees of this ambiguity have different linearizations in the viewed language.

Der Rhein fließt durch eine Stadt, die die Schweiz enthält. - Übersetzungen

**ACE**
The Rhine flows through a city that contains Switzerland.  (2 Alternativen)

**ACE+lex**
The Rhine flows through a city that contains Switzerland.

**Deutsch**
Der Rhein fließt durch eine Stadt, die die Schweiz enthält.

**Español**
El Rin fluye a través de una ciudad que contiene la Suiza.

**Alternativen**


- The Rhine flows through a city that contains Switzerland.

- The Rhine flows through a city that Switzerland contains.

Wählen  Schließen
Lexicon modules as a table

<table>
<thead>
<tr>
<th>Léxico</th>
<th>GeographySpa</th>
<th>GeographyGer</th>
<th>GeographyI</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>mkNP el &quot;ríop_Aar&quot;</td>
<td>mkNP die &quot;Aare&quot;</td>
<td>aceNP defsg &quot;Aar&quot;</td>
</tr>
<tr>
<td>Aargau_NP</td>
<td>mkNP &quot;Argovia&quot;</td>
<td>mkNP der &quot;Aargau&quot;</td>
<td>aceNP &quot;Aargau&quot;</td>
</tr>
<tr>
<td>Abrantes_NP</td>
<td>mkNP &quot;Abrantes&quot;</td>
<td>mkNP &quot;Abrantes&quot;</td>
<td>aceNP &quot;Abrantes&quot;</td>
</tr>
<tr>
<td>Achill_island_NP</td>
<td>mkNP la &quot;isla_de_Achill&quot;</td>
<td>mkNP die &quot;Achill-Insel&quot;</td>
<td>aceNP &quot;Achill&quot;</td>
</tr>
<tr>
<td>Adriatic_Sea_NP</td>
<td>mkNP el &quot;mar_Adiático&quot;</td>
<td>mkNP die &quot;Adria&quot;</td>
<td>aceNP defsg &quot;Adriatic_Sea&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Aegian_Sea_NP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Albania_NP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Albanian_lake_NP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Alps_NP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Amsterdam_NP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Andorra_NP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Apennine_Mountains</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
abstract Attempto = Numeral, Symbols, Questions ** {
  flags startcat = ACEText ; -- Use Text to get only single-sentence texts
  fun aNP : VarCN -> ThereNP ;
  fun theNP : VarCN -> NP ;
  fun noNP : VarCN -> NP ;
  fun everyNP : VarCN -> NP ;
  fun pnNP : PN -> NP ;
  fun somebody_IPron : IndefTherePron ;
  fun something_IPron : IndefTherePron ;
  fun everybody_IPron : IndefPron ;
  fun everything_IPron : IndefPron ;
  fun nobody_IPron : IndefPron ;
  fun nothing_IPron : IndefPron
  fun indefTherePronNP : IndefTherePron -> ThereNP ;
  fun indefPronNP : IndefPron -> NP ;
  fun indefTherePronVarNP : IndefTherePron -> Var -> ThereNP ;
  fun indefPronVarNP : IndefPron -> Var -> NP ;
  fun at_leastNP : Card -> VarCN -> ThereNP ;
}
Landlocked country

- Si un país X no limita con un mar y no limita con un océano entonces X es un país interior.
- Ningún país interior limita con un océano.
- Ningún país interior limita con un mar.
- ¿qué país es un país interior?
  - Austria
  - Liechtenstein
  - Suiza
- ¿qué país no es un país interior?
  - Estonia
  - Letonia
  - Lituania
- Liechtenstein limita con un país. El país no es un país interior.
AceWiki-GF: Technologies

- web application written in Java
- Echo Web Framework
- OWL API (managing the ontology and reasoners)
- ACE Parser (conversion to OWL)
- GF Webservice (predictive parsing, translation, grammar compilation)
AceWiki-GF: Dependencies

AceWiki-GF components

- GUI
- Language engines
- Echo Web Framework
- OWL API
- APE Java Interface
- GF-Java
- HermiT
- OWLLink
- External reasoners: FaCT++, Pellet,...
- ACE Parser (APE)
- GF Cloud Service
Evaluation of AceWiki-GF: Design

- developed a 500-word geography lexicon
  - 3 languages: English, German and Spanish
  - 3 authors (incl. native speakers of German and Spanish, and a GF engineer)
  - avoid lexical ambiguity
- asked users of different languages to supply the wiki with sentences and tag each as true or false
- asked them then to evaluate others' sentences as true or false
- measured the user (dis)agreement and how much it is influenced by the automatic translation
  - Hypothesis: A group of users reaches almost the same level of agreement on the content of an article presented to them in different languages as when the article is presented to all of them in the same language.
- asked them for general feedback via a questionnaire
Evaluation of AceWiki-GF: Results

• 30 participants entered 316 sentences
• almost all the syntactic functions were used (i.e. they were discoverable via the look-ahead editor)
• agreement level was ~83% with no significant influence from the translation
• AceWiki-GF user interface was found to be easy to use
• 80% reported the the CNL did not let them express everything that they wanted
  ◦ missing content words
  ◦ some grammar restrictions were confusing or presented confusingly in the look-ahead editor
Future work

• generalize to handle other types of grammars and reasoning
• tree-based (language-neutral) sentence construction
• optimize the user interface to other types of content
• improve collaborative grammar editing features
• improve ambiguity management (e.g. automatic reasoning-based ambiguity resolution)
• use the wiki content to automatically generate documentation, grammar fragments, look-ahead editor customizations, etc. for novice users
• more evaluation needed
Get involved!

- many ways to extend AceWiki-GF, i.e. good for a thesis project
- the source of most projects mentioned today is available on GitHub:
  - APE: http://github.com/Attempto/APE
  - OWL Verbalizer: https://github.com/Kaljurand/owl-verbalizer
  - AceWiki / AceWiki-GF: http://github.com/AceWiki/AceWiki
  - GF: https://github.com/GrammaticalFramework/GF
- ... i.e. easy to fork and/or contribute to
- in case of questions send an email to the Attempto mailing list or kaljurand@gmail.com
Links

• ACE-in-GF
  ◦ project page: http://github.com/Attempto/ACE-in-GF

• AceWiki-GF
  ◦ project page (same as for AceWiki): http://github.com/AceWiki/AceWiki
  ◦ demos, links, etc.: http://attempto.ifi.uzh.ch/acewiki-gf/

• MOLTO project: http://www.molto-project.eu/
  ◦ see the deliverables of Work Package 11

• Grammatical Framework project: http://www.grammaticalframework.org/
Thank You!
Questions?