Typeful Ontologies with Direct Multilingual Verbalization

Ramona Enache and Krasimir Angelov

Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

September 14, 2010

The Ontological Aspects of GF

The grammars in GF have abstract and concrete syntax:

- **abstract syntax** - the ontological description of the domain
- **concrete syntax** - the verbalization

The combination of this two in one framework gives an unique opportunity for close interaction between logic and syntax.

The Experiment

The logical framework of GF as ontology description language:

- The abstract syntax in GF is a strongly-typed logical framework based on Martin Löf's Type Theory.
- In principle, we could use it to encode various domain models.
- We explored how to model typical ontological relations such as 'instance-of' and 'sub-class-of' in GF.
- As a use case we choose **SUMO** - the biggest open-source ontology

Reasoning

The built in reasoning capabilities of GF (*in development*):

- allows inference as a mean for **knowledge retrieval**
- provides a powerful tool for **disambiguation** based not on syntax but on semantic and knowledge based factors

The Ontology as a Controlled Language

GF is a **Grammatical Framework** after all:

- By adding concrete syntax to the ontological abstract syntax we get **controlled language** that can be used to express axioms in natural language.
- We will give example of how the type information in the ontology helps to produce correct language.

Some Statistics - Success rate

Abstract Syntax

(17 modules from the SUMO distribution)

- 100% declarations
- 90% of simple axioms
- 64% of quantified axioms

Concrete Syntax

- 90% of the two top-level modules - Merge & Mid Level Ontology (English)
- Only few examples for French and Romanian due to the lack for large coverage lexicon.

Classes in the Abstract Syntax

The classes are in the core of every ontology:

```
cat Class;

fun Entity : Class;
    Agent : Class;
    Human : Class;
    ⋮
```

Classes in the Abstract Syntax

Two basic operations from **description logic**:

$$\begin{aligned}
&\textbf{data}\ \textit{both} : \textit{Class} \rightarrow \textit{Class} \rightarrow \textit{Class}; && \text{-- intersection or conjunction}\\
&\phantom{\textbf{data}\ }\textit{either} : \textit{Class} \rightarrow \textit{Class} \rightarrow \textit{Class}; && \text{-- union or disjunction}
\end{aligned}$$

Other operations such as complement could be added as well

Instances

We need another category for the instances of some class:

cat *Ind Class*

Note: *Ind* is a **dependent** category. It has a parameter of type *Class* which tells us the class of the instance.

Instances

Now if we have the SUMO axiom:

$$(instance\ \ Pi\ \ PositiveRealNumber)$$

then we will generate the following abstract syntax definition in GF:

$$\textbf{fun}\ Pi\ \ :\ \ Ind\ \ PositiveRealNumber;$$

i.e. we define *Pi* as a variable of type *Ind* and add the class as type parameter. *PositiveRealNumber* is the principal class of *Pi*.

Class Hierarchy

We define yet another category which is now parameterized by two classes:

$$\textbf{cat } \mathit{SubClass}\ (c_1, c_2 : \mathit{Class});$$

then the definition:

$$\textbf{fun } \mathit{Human\_Class}\ :\ \mathit{SubClass\ Human\ Agent};$$

asserts that *Human* is a sub class of *Agent*.

Note: We have a unique id for the assertion!
(Named Graphs in RDF?)

Inheritance - reflexive transitive closure

We need two *"inference rules"*:

$$\textbf{cat}\ \mathit{Inherits}\ (c_1, c_2 : \mathit{Class});$$

$$\begin{aligned}\textbf{fun}\ & \mathit{inhz} : (c : \mathit{Class}) \rightarrow \mathit{Inherits}\ c\ c;\\ & \mathit{inhs} : (c_1, c_2, c_3 : \mathit{Class}) \rightarrow \mathit{SubClass}\ c_1\ c_2\\ & \qquad \rightarrow \mathit{Inherits}\ c_2\ c_3 \rightarrow \mathit{Inherits}\ c_1\ c_3;\end{aligned}$$

Generalized Instances

If class c_1 is a sub-class of c_2 then every instance of c_1 is also an instance of c_2:

cat *El* *Class*;

fun *el* : $(c_1, c_2 : \mathit{Class}) \rightarrow \mathit{Inherits}\ c_1\ c_2 \rightarrow \mathit{Ind}\ c_1 \rightarrow \mathit{El}\ c_2$;

Functions

In SUMO the functions are just instances of class *Function*:

$$(instance\ RadiusFn\ Function)$$
$$(domain\ RadiusFn\ 1\ Circle)$$

but in GF they have different types:

$$\textbf{fun}\ RadiusFn\ :\ El\ Circle \rightarrow Ind\ LengthMeasure;$$

Predicates

Similarly the predicates are instances of class *Predicate*:

$$(instance\ address\ BinaryPredicate)$$
$$(domain\ address\ 1\ Agent)$$
$$(domain\ address\ 2\ Address)$$

Just like with the function but now the return type is different.

cat *Formula*

fun *address* : *El Agent* $\rightarrow$ *El Address* $\rightarrow$ *Formula*;

Some logical operators

Quantifiers:

$$\begin{aligned}
&\textbf{cat}\ \mathit{Var}\ \ \mathit{Class};\\
&\textbf{fun}\ \mathit{exists} : (c : \mathit{Class}) \to (\mathit{Var}\ \ c \to \mathit{Formula}) \to \mathit{Formula};\\
&\qquad\ \ \mathit{forall} : (c : \mathit{Class}) \to (\mathit{Var}\ \ c \to \mathit{Formula}) \to \mathit{Formula};
\end{aligned}$$

Connectives:

$$\begin{aligned}
&\textbf{fun}\ \mathit{not} : \mathit{Formula} \to \mathit{Formula};\\
&\qquad\ \ \mathit{and}, \mathit{or}, \mathit{impl}, \mathit{equiv} : \mathit{Formula} \to \mathit{Formula} \to \mathit{Formula};
\end{aligned}$$

Axioms

The axioms in SUMO are some logical formulae:

$$\begin{array}{l}(=>(\mathit{instance}\ ?P\ \ \mathit{Wading})\\ \qquad(\mathit{exists}\ (?W)\\ \qquad\qquad(\mathit{and}\ (\mathit{instance}\ \ ?W\ \ \mathit{BodyOfWater})\\ \qquad\qquad\qquad(\mathit{located}\ \ ?P\ \ ?W))))\end{array}$$

which we turn into abstract syntax trees in GF:

$$\begin{array}{l}\mathit{forall}\ \ \mathit{Wading}\ (\backslash P \rightarrow\\ \qquad \mathit{exists}\ \ \mathit{BodyOfWater}\ (\backslash W \rightarrow \mathit{located}\ (\mathit{var}\ \ P)\ (\mathit{var}\ \ W)))\end{array}$$

Note how the type information is handled in GF and in SUMO!

Semantic Disambiguation and Nonsense Filtering

The abstract syntax for sentence like this:

for every human X . . . X has more than 2 Mb of memory

is

$$\textit{forall}\ \textit{Human}\ (\backslash X \rightarrow \ldots\ (\textit{el}\ \textit{Human}\ \textit{Computer}\ ?\ X)\ \ldots)$$

But this will be rejected by the type checker because the meta variable cannot be resolved:

$$?:\textit{Inherits}\ \textit{Human}\ \textit{Computer}$$

SUMO approach

- natural language generation through combination of string templates
- covers most of **Merge** - the main SUMO ontology
- available in 10 languages
- hand-written

SUMO approach - Problems

- cannot model phonetic mutations

```
 (format fr origin "%1 %nne a %npas pour &%origine
%2")
```

  generates "*X ne a pas pour origine Y*"
  instead of "*X n'a pas pour origine Y*" in French.

SUMO approach - Problems (continued)

- cannot model gender agreement

```
 (format ro SquareRootFn "radacina & %square%t
patrata a lui %%1")
 (format ro TangentFn "&%tangent%ttangenta lui %1")
```

generates “*tangenta lui radacina patrata a lui X*”
instead of “*tangenta radacinii patrate a lui X*” in Romanian.

(the tangent of the square root of X)

SUMO approach - Problems(cont'd)

- cannot assign gender to variables (solved with GF)

```
forall Animal (\A → exists Animal (\B → smaller B
A))
forall House (\A → exists House (\B → smaller B A))
```

should generate
“*pour chaque animal A il existe* **un** *animal B* **tel** *que B est plus* **petit** *que A*”
and
“*pour chaque maison A il existe* **une** *maison B* **telle** *que B est plus* **petite** *que A*”
in French.

GF approach

- automatic for concepts - from their SUMO name (English)
- semi-automatic for relations
- optimizations for elegant rendering of formulas
- verbalizations for higher-order functions (not in the original SUMO NLG)
- verbalizations for instance and subclass declarations (not in the original SUMO NLG)
- reusable when adding new languages
- syntactically correct and more readable

GF approach - Examples

- higher-order functions

Abs	$EquivalenceRelation\ Entity\ (\backslash x, y \rightarrow equal\ x\ y)$
Eng	"x is equal to y" is an equivalence relation

- instance declarations

Abs	*instStm PrimaryColor Blue*
Eng	blue is an instance of colour

- subclass declarations

Abs	*subClassStm Beverage Food Beverage_Class*
Eng	beverage is a subclass of food

GF approach - Examples

- **SUMO**

  “for all unique list ?LIST holds for all ?NUMBER1, ?NUMBER2 holds If ?NUMBER1th element of ?LIST” is equal to ”?NUMBER2th element of ?LIST”, then ?NUMBER1 is equal to ?NUMBER2 “

- **GF**

  “ for every unique list LIST, every positive integer NUMBER2 and every positive integer NUMBER1 we have that if the element with number NUMBER1 in LIST is equal to the element with number NUMBER2 in LIST, then NUMBER1 is equal to NUMBER2”

Demo

There is a new user interface for GF where the user could write new axioms and explore the ontology. If the axioms are not type correct the error is reported to the user.

Directions

- The current language follows too strictly the abstract syntax of the underlying logical formulae. It would be nice to make the language more natural.
- Some transfer would be needed from the user language to the core logical language.
- A rendering to the logical language is still a useful tool for the user.