CHALMERS

Reasoning and Language Generation in the SUMO Ontology

Master of Science Thesis in the Programme Foundations of Computing – Algorithms, Languages and Logic

RAMONA ENACHE

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Göteborg, Sweden, February 2010

Reasoning and Language Generation in the SUMO Ontology

Ramona Enache

Examiner: Prof. Aarne Ranta

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden February 2010

Abstract

We describe the representation of SUMO(Suggested Upper-Merged Ontology) in GF(Grammatical Framework). SUMO is the largest open-source ontology, describing over 10,000 concepts and the relations between them. In addition to this, there are axioms that specify the behaviour of relations and the connections between various concepts. The languages that are widely used for encoding ontologies do not have a type system and have mainly descriptive purpose. For checking the consistency of ontologies or generating natural language, other tools are used. GF is a grammar formalism with support for dependent types, and has built-in support for natural language generation and multilingual translation for 16 languages.

The benefits of the translation of SUMO to GF are the possibility to perform type-checking on the content of the ontology, and the generation of syntactically correct natural language. The representation of SUMO uses dependent types for flexibility and better control of semantic actions. The current work provides algorithms for type inference and type checking of the translated axioms. From the concepts, relations and axioms from SUMO, we generate constructions in natural language for English, Romanian and French.

The resulting GF files are further more translated to a first-order logic format, TPTP-FOF and checked for consistency with an automated theorem prover. The resulting set of axioms can be used for making inferences.

The representation of SUMO in GF preserves the expressivity of the original ontology, adding to this the advantages of a type system and built-in support for natural language generation.

Acknowledgements

I would like to thank my supervisor, Krasimir Angelov who introduced me to GF, and guided me when implementing the GF resource grammar for Romanian and the present master thesis. Thanks to his constant help, patience and inspiring advice, I could complete these projects. Also I would like to express my gratitude towards my examiner, Professor Aarne Ranta for his insightful feedback on my work and constructive advice about writing a research article. I would also want to thank Docent Koen Claessen for his help with the automated reasoning part of the thesis and the illuminating discussions we had on first-order logic.

Moreover, I would like to thank the dearest grandmother in the world, Gica for teaching me the ABC of life and being my best friend and role model. My parents and my little brother, Bogdan who permanently encouraged and supported all my plans and dreams have all my gratitude, also.

Last but not least, I would like to thank all the wonderful people that I met in Gothenburg, who gave me a new perspective on life and research and made me enjoy my stay here to the full.

Contents

Chapter 1

Preliminaries

The constantly growing amount of information brought about the necessity of classifying it according to criteria such as meaning or domain of usage. Also, there is need to represent the relations between various pieces of information. An *ontology* is a formalism used for representing the abstract projections of information, named *concepts* and the *relations* between them. The behaviour of relations can further more be expressed by *axioms* from the ontology, which are logic formulas, written in a formal language. A taxonomy on domains would separate ontologies in *domain ontologies*, which describe the concepts and relations from a given area, and *upper ontologies*, which feature more general concepts and the relation between them, and which do not belong to a specific domain.

Ontologies were introduced in the 70's. At the beginning, they were used in Artificial Intelligence. Nowadays, ontologies are are an important field of Semantic Web. Some of the most widely used ontologies are Cyc, DOLCE, SUMO and Gellish. In addition to this, there exists a larger number of formats for encoding ontologies, such as CycL, OWL, RDF, KIF and others. The most widely used languages for formalizing ontologies are based on first-order logic with predicates, allowing implicit or explicit higher-order logic constructions, depending on the structure of the ontology and do not normally have a type system.

The current work represents the content of the SUMO, the largest open-source ontology in GF, showing the benefits of this over the usual languages for encoding ontologies. GF is a grammar formalism based on type theory, and also a domain-specific functional programming language used for building grammars. It has mechanisms for rule-based machine translation and was used in various applications that deal with multilingual translation and natural language generation. An analisys on SUMO is also performed and the work develops specific techniques to represent its contents in GF and the advantages of this representation. Further more from GF, a translation to a first-order logic format is provided, and the GF projection of the ontology is checked for consistency and is used for making various inferences, with the aid of an automated prover.

From the total of 30 domain ontologies that SUMO provides, 17 were translated into GF. These are: Merge and Mid-level-ontology, the most general ontologies, CountriesAndRegions, Communications, Economy, Elements, Engineering, FinancialOntology, Geography, Government, Military, Mondial, QoSOntology, Transportation, WorldAirportsA-K, WorldAirportsL-Z and WMD.

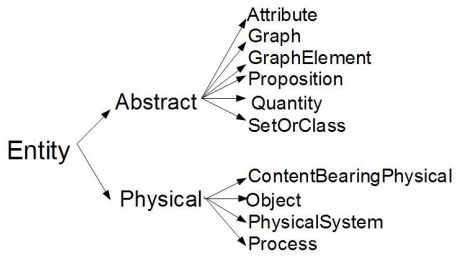
We describe the techniques used for converting definitions of concepts and relations from SUMO to GF and of the axioms specifying properties and behaviours of the concepts and relations. A translation from an untyped system to a strongly typed one is not straighforward, and further processing of the data was needed.

The advantages of representing the SUMO ontology in GF are the possibility to type check the axioms and definitions and also to generate higher-quality natural language. The translation to GF, is also an in-depth analysis on SUMO, showing its structure and the possible problems that the absence of a type system brings about.

1.1 SUMO

SUMO (Suggested Upper-Merged Ontology)[7] is the largest freely-available ontology, describing over 10.000 concepts and the relations between them. The entries in SUMO are grouped in a hierarchy that ranges from the most general concept – `Entity` to very particular instances such as `Atoll`. Functions, predicates and relations are also descendants of `Entity`, since there is just one tree that covers all the entries from the ontology. This will bring about some difficulties when formalizing the relations between the concepts and checking them for consistency in a logic framework, as it allows certain constructions in higher-order logic and also the possibility to express paradoxes in the ontology. The following picture describes the first 3 levels in the SUMO hierarchy, which is the initial part of the taxonomy of concepts. The large number of ramifications shows the variety of the SUMO entries. Although this initial part has just simple dependencies between concepts, further on in the hierarchy, a concept can have multiple inheritance.

Figure 1.1: Hierarchy of concepts in SUMO



SUMO covers areas that range from Graph Theory to Weapons of Mass Destruction and Countries of the World. It features a main file, called Merge which defines the main functions and basic concepts, such as `Entity` and `Object` that represent the basis of the hierarchy. In addition to this, the file Mid-level-ontology, takes one step further in defining basic notions, which represent general concepts from various domains, such as `Soldier`, `Atom`, and `EuropeanNation`, from ontologies such as Military, Elements and Mondial from SUMO. Merge and Mid-level-ontology represent general ontologies, that cover almost 1,500 entries. SUMO also features some more specific domain ontologies. In the current distribution there are 30 other ontologies that cover domains from world airports to military concepts and transnational issues.

The content of the ontologies is expressed in a version of KIF(Knowledge Interchange Format), called $SUO-KIF$, which permits the declaration of concepts in a human-readable form, featuring support for expressing first-order predicate calculus constructions. The axioms of the ontology that express relations between concepts and relations that manipulate them are expressed in this format. Due to the modeling of the hierarchy, considering functions and relations along with all the other concepts, it is possible to express second-order logic construction in SUO-KIF, as quantifications over a function or predicate.

In order to verify the consistency of the ontology, the declarations and axioms from Merge and Mid-level-ontology have been translated to TPTP[8]— FOF (an untyped first-order logic standard), and checked for consistency with various theorem provers. There is an annual competition for finding an inconsistency in the system of axioms. It is held at the CADE conference, which is the premier conference on Automated Deduction.

SUMO has mappings to WordNet [16], which is a large open-source lexical database, containing over 150,000 words. The WordNet entries are grouped in synsets, which are sets of words with equivalent semantic meaning in a given context. For example, the word *school* belongs to the following synsets (among others):

> (n) school, shoal (a large group of fish) "a school of small glittering fish swam by"
> (n) school, schooling (the process of being formally educated at a school) "what will you do when you finish school?"
> (v) educate, school,train,cultivate, civilize, civilise (teach or refine to be discriminative in taste or judgment) "Cultivate your musical taste"; "Train your tastebuds"; "She is well schooled in poetry"

The synsets are classified considering the part of speech that the constituents denote. The parts of speech considered are nouns, adjectives, verbs and adverbs.

The mappings to WordNet [4] was done for the entries from Merge and Mid-level-ontology. In this way, every concept from one of the two general-purpose ontologies corresponds to a number of synsets from WordNet. There are three kinds of such mappings : synonymy, hypernymy and instantiation. Most of the mappings are instantiations of a SUMO concept.

For example: The `BiologicalAttribute` concept from Merge, corresponds to 177 entries from WordNet denoting nouns, most of them being instances of a biological attribute, such as

inanition | exhaustion resulting from lack of food – `BiologicalAttribute`

The mapping to WordNet makes SUMO valuable from linguistical point of view, since it can be used in Natural Language Processing applications, while the translation to TPTP makes it valuable from the point of view of automated reasoning, since it can be used by applications that deal with first-order logic.

The SUMO ontology is intended to be an universal ontology, in the sense that it should reflect universally common concepts and the relations between them, without being biased towards a certain culture or philosophical system. Every concept is followed by its documentation, in order to avoid ambiguities.

The SUMO ontology was originally written in English, but translations are provided for Mandarin Chinese, Czech, Italian, Romanian, German and Tagalog. There is an online browser, named KSMSA Ontology Browser[1], for the SUMO ontology and its connections to WordNet, which features translations of the SUMO concepts in the above mentioned languages. In addition to this, there are translations for French, Hindi and Arabic which are not featured in the KSMSA browser. The translations are based on a set of templates, which are combined by concatenation. They are hand-written, and partially cover the Merge ontology.

Since 2001, when it was created, SUMO has been developed further more every year. Over 70 papers have been written on SUMO and they analyzed several aspects of the ontology.

SUMO has an associated knowledge engineering environment. It can be used for intelligent browsing and developing new ontologies in SUO-KIF. The environment features an older version of the Vampire theorem prover, for checking the first-order logic for consistency.

1.2 Grammatical Framework

GF[1] is a grammar formalism, which uses type theory to express the semantics of natural languages, for multilingual grammar applications. The GF resource grammars are the basic constituents of the GF library, on top of which applications are built. Notable applications that use GF are the verification tool KeY, for the generation of natural language from the formal language OCL, the dialogue systems research project TALK and the educational project WebALT, for generating natural language for mathematical exercises in different languages, and performing multilingual translations.

The two main operations that are regularly performed with resource grammars are the generation of natural language, based on a term in the abstract syntax (linearization) and parsing. Multilingual translation is achieved as a combination of these two processes.

[1] http://virtual.cvut.cz/ksmsaWeb/main

A GF resource grammar basically consists of the abstract syntax, which is a set of rules common to all grammars, and provides the structure of the grammar, and the concrete syntax, which implements the elements of the abstract syntax in the given language, considering its specific features. The abstract syntax provides consistency for the resource library, also ensuring grammatically correct multilingual translations. Resource grammars are general-purpose, as they capture the basic traits of the language. Domain specific applications use a more restricted domain ontology. In this case, there is more emphasis on the semantic aspect, than in the case of general - purpose grammars. In both cases, only syntactically correct constructions can be generated and parsed.

So far the resource library contains 16 languages: English, French, Italian, Spanish, Catalan, Swedish, Norwegian, Danish, Finnish, Russian, Bulgarian, German, Interlingua, Romanian, Polish and Dutch. The last three languages were added in 2009.

The grammar features a complete set of paradigms for the inflectional morphology of the main categories, namely nouns, adjectives, verbs, numerals and pronouns.

In the abstract syntax, lexical entries are represented as nullary functions (constants), which will be given a linearization in the implementation of the concrete syntax, which consists of a table with all the inflection forms. For example: For example:

```
fun airplane_N: N;
```

from the abstract syntax will be linearized in the Romanian resource grammar[9] as:

```
lin airplane_N = mkN "avion";
```

where the function `mkN` will generate all the 12 flexion forms needed for the concrete form of the abstract noun, also inferring the gender.

Special categories are the relational nouns, adjectives and verbs, where we specify the case of the object, and the preposition that binds it with the relational category. For example:

```
fun forget_V2: V2;
```

will be linearized in the English grammar as

```
lin forget_V2 = dirV2
       (irregDuplV "forget" "forgot" "forgotten");
```

where `irregDuplV` indicates that the verb is irregular, so all 3 forms are needed in order to build the whole representation table. The function `dirV2` indicates that the verb is transitive, and the corresponding object will in the Accusative case, with no binding preposition (direct object).

A small lexicon of almost 300 words is also included, for testing purposes. It is based on the Swadesh list, and features the most common notions, which are to be linearized in each language. Specific applications that use GF, usually have their own dictionaries.

1.3 First-Order Logic

The first-order logic is a basic formal logic system, which is widely used for formalizing aspects of fields like computer science, mathematics, philosophy, linguistics and others.

We will proced by shortly defining the two main dimenssions of the first-order logic system - syntax and semantics.

Having $S = (V,C,PSig,FSig)$, where

- V is the set of variables
- C is the set of constants
- $FSig$ is the set of function signatures, denoting the arities of the functions
- $PSig$ is the set of predicate signatures, denoting the arities of the predicates.

We define *terms* as:

- all variables and constants are terms
- if t_i, 1≤i≤n are terms and f is a function of arity n, then f$(t_1, t_2...t_n)$ will be a term, also.

We define *first-order formulas* as:

- if t_i, 1≤i≤n are terms and p is a predicate of arity n, then p$(t_1, t_2...t_n)$ will be a first-order formula.
- if α and β are first-order formulas, then $\alpha \vee \beta$, $\alpha \wedge \beta$, $\alpha \rightarrow \beta$, $\alpha \leftrightarrow \beta$ and $\neg\alpha$ are first-order formulas, also.
- if x is a variable, and α is a first-order formula, then $\forall x\alpha$ and $\exists x\alpha$ are first-order formulas, also.

First-order logic can also have a special predicate for equality =, which is treated differently than the other predicates, on the semantical level.

There exists another variant of first-order logic which is typed, and the inductive process of forming terms and first-order logic formulas ensure that only well-typed constructions can be obtained.

The semantics of first-order logic assumes interpreting the variables and constants with values from a set, like the set of natural number, or the set {0,1}. The first-order connectors are translated into their equivalents from the standard Boole algebra. Regarding the quantifiers, $\forall$ will be interpreted as a conjunction over the elements of the set, and $\exists$ will be interpreted as a disjunction over the elements of the set. Functions are mapped into functions on the set, while predicates are intepreted as predicates taking elements from

the set and returning whether True or False as a result. The equality, if present is always interpreted as the equality from the set.

The interpretation of a first-order formula, typically assumes assigning values to each of its atomic constituents(terms consisting of constants or variables), and evaluating the formula, with the regard to the interpretations of functions and predicates. The result of evaluating a first-order logic formula is either True or False.

In this way, one might want to determine if a first-order formula is *satisfiable* on some set, i.e. there exists an assignment of the variables of the formulas to elements from the set, such that the evaluation of the formula yields to True. The satisfiability of a formula is NP-complete problem for first-order propositional logic and undecidable for first-order predicate logic.

Also, one could examine if a first-order formula is satisfiable in all interpretations. In this case we call the formula, *tautology*. Verifying that an arbitrary first-order formula is a tautology is theoretically undecidable, but in practice, automated theorem provers manage to obtain good results for a fair amount of first-order formulas.

Proving statements in first-order logic is the most mature and well-developed branch of automated theorem proving so far. Various optimization techniques are meant to increase the performance of tools that verify the validity of a specific first-order formula.

TPTP[8](Thousands of Problems for Theorem Provers) is the largest database of problems from various fields, related to Mathematics, which are formulated in first-order logic, and also the name of the format in which the problems are written, so that they can be processed by the various automated provers. There is an annual competition for automated provers, the CADE ATP System Competition, where various automate provers compete on solving TPTP problems. The TPTP standard representation for first-order formulas, is called FOF and works with untyped first-order logic. There is work in progress for defining a standard for typed first-order logic[2].

[2]http://www.cs.miami.edu/ tptp/TPTP/Proposals/TypedFOF.html

Chapter 2

Translating SUMO to GF

The original ontologies are expressed in SUO-KIF, which is completely untyped, and has as primitive operations only the first-order logic operators `and`, `or`, `implies`, `equiv`, `not`, `forall`, `exists`. Since all the SUMO concepts are grouped in one hierarchy, it is possible to express constructions in second-order logic, like quantifying over functions or predicates. This makes it more difficult to automatically check for consistency, since a translation to first-order logic would not be possible in all cases. All the concepts and relations from SUMO are to be defined. This yields to the particular situation where the predicate `instance` is defined in terms of itself:

```
(instance instance BinaryPredicate).
```

Also, there is no separation between types and instances, the `instance` predicate, which is meant to reflect the relation between a variable and its type, is defined as having 2 arguments of type Entity. In these conditions the expression `(instance x x)` can be expressed. This opens a gate to potential self-referencial paradoxes and constructions that might not make sense in a typed context. Another controversial construction in SUMO is the function `KappaFn` that denotes the class of variables satisfying a given property. This approach fits the naive way to define sets in set theory, and permits expressing Russell's paradox in SUMO.

This is one of the most well-known paradoxes of naive set theory and was discovered in 1901 by the Bertrand Russell. It can be expressed as: we name *normal* the sets X with the property that $X \notin X$. In this conditions we can prove that the set of *normal* sets is neither *normal*, nor *non-normal*.

Let $\mathbf{S} = \{ X \mid X \notin X \}$.

- Assuming that $\mathbf{S}$ is *normal*, we have that $\mathbf{S} \notin \mathbf{S}$, so $\mathbf{S}$ is not a member of the set of *normal* sets, which is itself, hence it is *non-normal*.
- Assuming that $\mathbf{S}$ is *non-normal*, we have that then $\mathbf{S} \in \mathbf{S}$, hence $\mathbf{S}$ is a member of the set of *normal* sets, so it is *normal*.

The solutions to this paradox were the axiomatic set theory, introduced by Zermelo and Fraenkel and a typed set theory, proposed by Russell.

In SUO-KIF Russell's paradox can be expressed as:

```
(instance
  (KappaFn "x" (not (instance x x)))
  (KappaFn "x" (not (instance x x))))
```

where `KappaFn "x" (not (instance x x))` is the class of *normal* elements, as defined before. In a similar manner, one can show that the axiom is both True and False.

The type system from GF does not use the predicate `instance`, which was used to formulate Russell's paradox in SUMO due to its peculiar signature, that allows a form of self-reference.

2.1 Basic GF Type System

GF is a functional programming language based on Martin-Löf type theory. It has strong static typing and provides support for dependent types.

Translating from an untyped system to a typed system is not straightforward, and further modeling on the data was needed. An inconvenient in this process is the fact that the SUMO ontologies have been written by hand, and not type-checked in any way, since the language is mostly descriptive. Another difficulty is the fact that the SUO-KIF framework where everything is expressed as a predicate, and the task of checking the consistency is passed to the automated prover. As seen from the definition of first-order terms and formulas, the only type checking that can be done is that functions and predicates are applied to the right number of arguments. Also, the representation of all concepts in one hierarchy gives rise to constructions that belong ultimately to higher-order logic, and cannot be translated and checked by a first-order automated prover.

The first step in the modeling of the data is to split the hierarchy into types and instances of types: `Class` will denote the type of SUMO classes. For instances of a certain type, two dependent types are used:

- `El Class` - for direct or indirect instances of a class used as argument of a function;
- `Ind Class` - for direct instances of a class, defined as such, or obtained from applying a function to its arguments.
- `Var Class` - variables which are direct or indirect instances of a class and are used in the quantified formulae.

In SUMO the predicate `immediateinstance` corresponds to the `Ind` category in GF and represents a direct instance of a particular class. The `instance` predicate corresponds to the categories `El` and `Var` in GF and represents a direct or indirect instance of a class. An indirect instance of a class signifies a direct

instance of a subclass of that class. The different treatment of quantified variables is useful for the natural language generation and the automated reasoning parts.

Examples of using these categories for declaring SUMO relations and concepts:

```
fun RadiusFn: El Circle -> Ind LengthMeasure;
fun YearDuration: Ind UnitOfDuration;
```

A special type is `Formula`, which denotes the result type of a predicate. `Formula` does not have instances or subclasses. For manipulating `Formulas`, the following operators, inspired from first-order logic are provided:

```
fun not: Formula -> Formula;
fun and: Formula -> Formula -> Formula;
fun or: Formula -> Formula -> Formula;
fun impl: Formula -> Formula -> Formula;
fun equiv: Formula -> Formula -> Formula;
```

and for quantified predicates:

```
fun exists: (c: Class) -> (Var c -> Formula) -> Formula;
fun forall: (c: Class) -> (Var c -> Formula) -> Formula;
```

In the original files, these operators are the only predefined relations, while all the other functions and predicates are to be subsequently defined.

In the original SUMO hierarchy `Formula` is a member of the hierarchy, but also the type of the return value of a predicate. The first-order logic operators from SUO-KIF are applied to `Formulas`, and there exist cases of quantification over `Formulas`, also. This is an indirect second-order logic construction, as there are no instances of `Formula` in SUMO. One could only get a value of this type by applying a predicate to its arguments. So, a quantification over `Formula`, assumes quantification over predicates and their possible arguments.

There are 27 functions in the total of 17 SUMO files considered, that take an argument of type `Formula`. An idea to reduce the second-order logic constructions to first-order logic would be to replace a function call of any of these functions, with the axiom specifying their definition. In this way, the resulting axioms would have the same meaning, and they could be used for automated reasoning. Unfortunately, only 3 of the 27 functions have an axiom that specifies their behavior and that could be used as a macro, instead of the function call. They are `decreasesLikelihood`, `increasesLikelihood` and `independentProbability`. The rest of the functions depend on each other, or do not have any axiom at all, as it is the case with `hasPurpose` which is the most widely used in axioms, among the 27 functions.

In the GF type system, for these reasons, we do not provide the possibility to quantify over `Formula`, because it would basically mean that the system will not be first-order logic anymore.

In this typed framework, Russell's paradox can no longer be expressed.

The hierarchy of `Classes` is represented in GF, by the following dependent categories:

```
cat SubClass (c1, c2:  Class)
cat SubClassC (c1, c2:  Class) -> (Var c2 -> Formula)
```

Examples:

```
fun Ice_Water: SubClassC Ice Water (\ICE -> attribute (var
    Water Object ? ICE)(el PhysicalState Attribute ? Solid));

fun April_Class: SubClass April Month;
```

where `el` is a coercion function from `Ind` to `El`, that has the following signature:

```
fun el: (c1, c2: Class) -> Inherits c1 c2 -> Ind c1 -> El c2;
```

The function `var` is similar to `el`, as it ensures coercion from `Var` to `El`, and was declared as following:

```
fun var: (c1, c2: Class) -> Inherits c1 c2 -> Var c1 -> El c2;
```

The semantics of `SubClassC` is that the condition that if an instance of class c2 fulfills the given condition, then it is an instance of c1 - the subclass. `SubClass` specifies a more general inheritance relations, where the above mentioned condition is not specified Most of the relations between classes are of the type `SubClass`, while about 1% are `SubClassC`. When extending the ontology with more concepts and relations, it is possible that the number of `SubClassC` will increase.

The generic inheritance relation between instances is the reflexive-transitive closure of `SubClass` and `SubClassC`, named `Inherits (c1, c2:  Class)`, which can be formed with the following functions:

```
fun inhz: (c: Class) -> Inherits c c;
fun inhs: (c1, c2, c3: Class) -> SubClass c1 c2 ->
    Inherits c2 c3 -> Inherits c1 c3;
fun inhsC: (c1, c2, c3: Class) -> (p: Var c2 -> Formula) ->
    SubClassC c1 c2 p -> Inherits c2 c3 -> Inherits c1 c3;
```

The `Inherits` type corresponds to the SUMO concept of `subclass`, were the `SubClass` and `SubClassC` correspond to the SUMO concept of `immediate-Subclass`. The `Inherits` category is used for defining coercion functions between instances. For example:

```
fun NonnegativeRealNumber_RealNumber : SubClassC
    NonnegativeRealNumber RealNumber (\NUMBER ->
    greaterThanOrEqualTo (var RealNumber Quantity ?
    NUMBER) (el Integer Quantity ? (toInt 0)));
```

where `greaterThanOrEqualTo: El Quantity -> El Quantity -> Formula`. The function `var` casts `NUMBER` from its type - `Var RealNumber` to `El Quantity`, which is the type that `greaterThanOrEqualTo` expects. For double and integer values, the built-in GF types `Float` and `Int` are used, and two function `toRealNum` and `toInt` convert from the GF types to `Ind Double` and `Ind Integer`. Further on, the coercion function `el` casts the resulting value from the resulting type to an `El Quantity`, which is the type of the second argument that the function `greaterThanOrEqualTo` expects.

The first three arguments of `var` and `el` are not necessary for the type checking phase, and they can be replaced with the wild-cart sign `?`. The GF type checker can infer the first two, but not the `Inherits` object, currently. If an Inherits instance is provided, the type checker will determine its correctness, however. The GF translation of SUMO uses the wild-cart sign for the first three arguments of `el`, and the GF type checker infers the type of the first two afterwards.

Because many SUMO concepts develop multiple inheritance, two operators for creating derived types have been defined:

```
fun both: Class -> Class -> Class;
fun either: Class -> Class -> Class;
```

Their behaviour in the type system is described by the following functions:

```
fun bothL: (c1, c2: Class) -> Inherits (both c1 c2) c1;
fun bothR: (c1, c2: Class) -> Inherits (both c1 c2) c2;
fun bothC: (c1, c2, c3: Class) -> Inherits c3 c1 ->
    Inherits c3 c2 -> Inherits c3 (both c1 c2);

fun eitherL: (c1, c2: Class) -> Inherits c1 (either c1 c2);
fun eitherR: (c1, c2: Class) -> Inherits c2 (either c1 c2);
fun eitherC: (c1,c2,c3: Class) -> Inherits c1 c3 ->
    Inherits c2 c3 -> Inherits (either c1 c2) c3;
```

These functions that manipulate the types built with `both` and `either`, are heavily inspired by the definitions of supremum and infimum in a partial ordered set.

Actually the hierarchy of types, with the `both` and `either` operations has the structure of a lattice having a least element - `Entity`, a supremum function - `either`, and an infimum one - `both`, where the partial order relation is `Inherits`. One can prove that the lattice is also distributive.

Although the SUMO ontology does not have an element which is the opposite of `Entity`, signifying the most particular element, that cannot have any instances, in GF one can consider a special class, `Nothing`, which would mean the empty type, in the Curry-Howard tradition, as it would not have any instance, or subclasses. The additional construction makes the distributive lattice a Boole algebra. As Boole algebras also have a negation unary operator, it is also possible to define in GF the complement of a `Class`, but the behaviour of

the complement is harder to model in the current type system. In traditional type systems, the complement is modeled in terms of the implication and the False object, but in the current setting, the only functions provided are the equivalents of conjuction and disjunction.

As in SUMO, functions and predicates do not only take instances as arguments, but also subclasses of a certain class, we use the dependent category `Desc Class` to model this situation. The constructor for this type is:

```
data desc: (c1,c2: Class) -> Inherits c1 c2 -> Desc c2;
```

Hence, c1 belongs to `Desc c2` (descendant of c2), if one can provide an `Inherits` object from c1 to c2. So, only classes that inherit c2, directly or indirectly can be used to build a `Desc c2`.

From a `Desc` object, one might extract the ancestor class and the proof of inheritance.

```
fun descToClass: (c: Class) -> Desc c -> Class;
def descToClass _ (desc c _ _) = c;
fun descInh: (c: Class) -> (p: Desc c) -> Inherits
    (descToClass c p) c;
```

It is possible that functions return a result of type `Desc Class` also. In this case, one needs a coercion function for descendents:

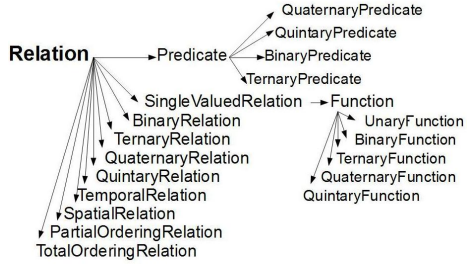
```
fun descToDesc: (c1,c2: Class) -> Inherits c1 c2 -> Desc c1
    -> Desc c2;
```

The function basically states that if c1 inherits c2, and we have a descendent of c1, then the latter is also a descendent of c2.

With the aid of this types and functions that manipulate them, it was possible to translate the concepts and relations from the SUMO sources into GF.

In the original SUMO files, every concept, except for `Entity`, is defined either as an instance of a type, or a subclass of another type. For relations - functions and predicates, the following figure reflects the relations between these special concepts from the ontology.

Figure 2.1: Hierarchy of relations in SUMO



The concept of `Function` represents functions with variable number of arguments, and a similar situation holds for the `Predicate` concept. Instances

of `Relation`, other than `Predicate`, have been interpreted as predicates, also, providing additional axioms, where necessary. The types of the arguments and of the result (for functions) is defined by the relations `range`, `rangeSubclass`, `domain`, `domainSubclass`.

For example, in Merge-

```
(instance address BinaryPredicate)
(domain address 1 Agent)
(domain address 2 Address)
```

wiil be represented in GF as:

```
fun address: El Agent -> El Address -> Formula;
```

There is also a number of functions and predicates that are just defined as subrelations, with or without specifying the types of the arguments. In this case they have the same signature as their direct ancestor. If certain argument have different types, the definition is updated subsequently.

For example, in SUMO -

```
(subrelation earthAltitude distance)
(instance distance TernaryPredicate)
(domain distance 1 Physical)
(domain distance 2 Physical)
(domain distance 3 LengthMeasure)
```

will be represented in GF as:

```
fun distance: El Physical -> El Physical -> El LengthMeasure
    -> Formula;
fun earthAltitude: El Physical -> El Physical -> El
    LengthMeasure -> Formula;
```

The is also the case of sub attributes, which are not defined as instance of a class. In that case, they take the type of their direct ancestor.

For example, in SUMO-

```
(subAttribute Larval NonFullyFormed)
(instance NonFullyFormed DevelopmentalAttribute)
```

will be translated to GF as:

```
fun NonFullyFormed: Ind DevelopmentalAttribute;
fun Larval: Ind DevelopmentalAttribute;
```

It is the case that there are chains of such sub attributes. For example, in FinancialOntology and Merge-

```
(subAttribute FinancialContract Contract)
(subAttribute Contract Promise)
(subAttribute Promise Obligation)
(instance Obligation DeonticAttribute)
```

In this case all concepts will be instances of `DeonticAttribute`. The algorithm that infers the type for sub attributes, takes care of chains of sub attributes, for the whole workspace of 17 files. The same situation holds for the algorithm that infers the right signature of sub relations.

Regarding difficulties of the translation of SUMO definitions to GF, we name the presence of concepts that appear both as subclass and instance, in the same file or in different files. For example, in Mid-level-ontology-

```
(subclass PoliticalFigure Celebrity)
(subclass ReligiousFigure Celebrity)
(instance Celebrity SocialRole)
```

This is an example of bad design of the ontology that should be overcome in the translation to GF, as it is not possible in a type system that something could be both a type and an instance of a type. In this case, we keep the problematic concepts as classes, rather than instances, so that we preserve the inheritance relations with the other concepts.

2.2 Translation of SUMO Axioms

Regarding the SUMO axioms, they are of two kinds: simple and quantified formulas. In the 17 files that we considered there are 3,461 quantified axioms and 37,222 simple ones.

An example of a simple axiom is the following one from CountriesAndRegions:

```
(capitalCity MoscowRussia Russia)
```

where an example of quantified axiom, chosen from Mid-level-ontology is:

```
(<=>
  (subCollection ?COLL1 ?COLL2)
  (forall (?MEMBER)
   (=>
     (member ?MEMBER ?COLL1)
     (member ?MEMBER ?COLL2))))
```

The variables in SUMO are written with ? in front of the name. In the axiom, we have `?COL1`, `?COL2` and `?MEMBER` as variables, while `member` and `subCollection` are predicates. `?MEMBER` is a bound variable, while `?COL1` and `?COL2` are free.

The SUMO axioms are to be translated as GF abstract syntax trees. Since GF is a strongly typed system, all the variables need to be typed also. It also the case of the considered axiom. Also open expressions are not valid GF trees, so the first step is to transform these expressions into closed ones. The abovementioned axiom will be interpreted as:

```
(forall (?COL1 ?COL2)
(<=>
```

```
(subCollection ?COLL1 ?COLL2)
(forall (?MEMBER)
  (=>
    (member ?MEMBER ?COLL1)
    (member ?MEMBER ?COLL2)))))
```

The next step is to infer the type of variables. A predicate of the form `(instance ?X SetOrClass)` indicates that the type of the variable `?X` is an instance of the class `SetOrClass`. For example,in Mid-level-ontology-

```
(=>
 (instance ?P Wading)
 (exists (?W)
   (and
     (instance ?W BodyOfWater)
     (located ?P ?W))))
```

can be translated into GF straightforwardly as:

```
forall Wading (\P -> exists BodyOfWater (\W -> located
(var Wading Physical ? P) (var BodyOfWater Object ? W)))
```

where `var` is the previously defined coercion function for quantified variables. The `instance` predicates have been removed, since they convey information about the types of the variables, and in GF, this is done in a more direct way. The axiom was pruned of this kind of type declarations, and the result was translated using the GF equivalents to the first-order logic operators and quantifiers. The first two arguments of the coercion function `var` were inferred by the GF type checker.

However, type inference is not straightforward in many other cases. There could be two or more type declarations of a variable in the same formula. For example, in Elements-

```
(=>
 (and
   (instance ?ATOM Hydrogen)
   (instance ?ATOM Atom))
 (measure ?ATOM (MeasureFn 1.0079 Amu)))
```

and in Merge-

```
(=>
    (instance ?PROC OrganOrTissueProcess)
    (exists (?THING)
        (and
            (located ?PROC ?THING)
            (or
                (instance ?THING Organ)
                (instance ?THING Tissue)))))
```

In this case the `both` and `either` operators are used to express the derived types of the variables:

```
forall (both Atom Hydrogen) (\ATOM -> measure (var (both Atom
Hydrogen) Object ? ATOM) (el PhysicalQuantity PhysicalQuantity
? (MeasureFn (el RealNumber RealNumber ? (toRealNum 1.0079))
(el UnitOfMass UnitOfMeasure ? Amu))))

forall OrganOrTissueProcess (\PROC -> exists (either Organ
Tissue) (\THING -> located(var OrganOrTissueProcess Physical ?
PROC) (var (either Organ Tissue) Object ? THING)))
```

However, as it is the case of the first quantified axiom from Mid-level-ontology, there can be quantified axioms that do not have type declarations for some of the variables. Since the type coercion function needs a convenient type in order to build the right Inherits object, one should have a mechanism of inferring the type of such variables.

A simple statistic shows that 58% of the quantified axioms from the total of 17 files contain variables with no type declaration. SUMO ontologies like Economy, have a percentage of 81% of quantified axioms with untyped variables.

The current work provides a type inference algorithm for dealing with such cases. It keeps the signatures of the functions, from all files, as axioms do not normally use only functions from the only one module. For every untyped variable that appears in an axiom, the algorithm infers the type that it would require, considering the functions that use it. The final type is the `both` combination of all such types. For example, in Economy-

```
(=>
(attribute ?AREA MajorIndustrialEconomy)
(economyType ?AREA DevelopedCountry))
```

while in GF-

```
forall (both Object Agent) (\AREA -> impl (attribute (var (both
Object Agent) Object ? AREA) (el EconomicDevelopmentLevel
Attribute ? MajorIndustrialEconomy)) (economyType (var (both
Object Agent) Agent ? AREA) (el UNEconomicDevelopmentLevel
EconomicAttribute ? DevelopedCountry)))
```

where the variable `AREA` is used by the function `attribute` as first argument, and by the function `economyType` as first argument. The functions' signatures are:

```
fun attribute: El Object -> El Attribute -> Formula;
fun economyType: El Agent -> El EconomicAttribute -> Formula;
```

Having the definition of `both`, and the relation between it and the `Inherits` type, one could see that the inferred type is enough to make the axiom type

check. Of course the inferred type could be simplified, as (`both Object Agent`) is actually `Agent`, as `Agent` is a direct subclass of `Object`.

The algorithm also rejects some cases of ill-typed axioms, which can be traced when inferring the type of some variables.

Regarding the simple axioms that do not contain quantified variables, their translation to GF was more straightforward and less error-prone. The translation ratio is almost 90% and the main reason for leaving out some axioms is the presence of instance with no previous declaration.

For example, the following axioms from CountriesAndRegions-

```
(capitalCity LondonUnitedKingdom UnitedKingdom)
```

will be translated to GF as:

```
capitalCity (el EuropeanCity City ? LondonUnitedKingdom)
(el (both Country Nation) GeopoliticalArea ? UnitedKingdom)
```

2.3 Translation of SUMO Higher-Order Functions to GF

Special attention was given to the higher-order functions and the axioms that use them. They are problematic, as their correspondent type signature from GF would require more information on the types of the parameters, and also type casting between on functions is not possible.

There is a relatively small number of significant higher-order functions. Some of them just describe a pattern for applying the given function. The significant functions specify a certain behaviour of the function-argument, such as commutativity, associativity, etc.

For example, in GF-

```
fun capability: (c: Desc Process) -> (El Entity -> El
    Entity -> Formula) -> El Object -> Formula;
```

which corresponds in SUMO to:

```
(=>
  (and
    (instance ?ROLE CaseRole)
    (?ROLE ?ARG1 ?ARG2)
    (instance ?ARG1 ?PROC)
    (subclass ?PROC Process))
 (capability ?PROC ?ROLE ?ARG2))
```

where `CaseRole` is a kind of binary predicate, so the meaning of the axioms is applying the function to an instance of the first argument which is a type and the second argument, which is an instance already. A possible interpretation of the `capability` function would be the ability / possibility to perform a certain

action. This interpretation would require a modal logic system, and a specific modality operator.

For example, in Merge-

```
(=>
(instance ?CLOUD Cloud)
(capability Seeing patient ?CLOUD))
```

has been translated to GF as:

```
forall Cloud (\CLOUD ->  forall Seeing (\SEEING -> patient
(var Seeing Process ? SEEING) (var Cloud Entity ? CLOUD)))
```

Approximately 6% of the axioms use `capability`. In the original SUMO files, these axioms could not be processed by an automated reasoner because `capability` is a higher-order function.

Other higher-order functions that express some special property of a function, such as `TransitiveRelation`, have been defined separately in the file HigherOrder.gf . They have been given a more comprehensive signature, and the axioms that define their behaviour have been rephrased.

The file HigherOrder.gf contains a total of 22 higher-order functions which were given type-correct definitions. Axioms that use these functions were translated to GF separately. The higher-order functions considered are: `AntisymmetricRelation`, `IntentionalRelation`, `ReflexiveRelation`, `SymmetricRelation`,`EquivalenceRelation`, `TransitiveRelation`, `IrreflexiveRelation`, `AsymmetricRelation`, `PropositionalAttitude`, `ObjectAttitude`, `IntransitiveRelation`, `PartialOrderingRelation`, `distributes`, `TrichotomizingRelation`, `TotalOrderingRelation`, `inverse`, `OneToOneFunction`, `SequenceFunction`, `AssociativeFunction`, `CommutativeFunction`, `identityElement`, `subRelation2El`.

For example,in Merge-

```
(instance MultiplicationFn CommutativeFunction)
```

was translated into GF as:

```
CommutativeFunction Quantity Quantity (\x,y ->
MultiplicationFn x y)
```

where `CommutativeFunction` has the following definition:

```
fun CommutativeFunction: (c1,c2: Class) -> (El c1 -> El c1
    -> Ind c2) -> Formula;
def CommutativeFunction c1 c2 f = forall c1 (\x -> forall c1
    (\y -> equal (el c2 Entity ? (f (var c1 c1? x)
    (var c1 c1 ? y))) (el c2 Entity ? (f (var c1 c1 ? y)
    (var c1 c1 ? x)))));
```

For the relations from HigherOrder, the proper signature was chosen respecting the mathematical conventions, since the SUMO original functions were very vague about it. There is a flagrant case of improper use of such a higher-order function - `AsymmetricRelation`. An asymmetric relation is a relation which is irreflexive and antisymetric.

- forall x, $\neg$ R(x,x)
- forall x and y, R(x,y) & R(y,x) $\rightarrow$ x = y

It is normal, hence, to assume that the two arguments have the same type, or at least there is an inheritance relation between the two types.

However, in SUMO it seems that the concept of `AsymmetricRelation` signifies the opposite of `SymmetricRelation`, with little concern for the types of the arguments. For example,in Merge-

```
(instance frequency BinaryPredicate)
(instance frequency AsymmetricRelation)
(domainSubclass frequency 1 Process)
(domain frequency 2 TimeDuration)
```

where there is no inheritance relation between `Process` and `TimeDuration`. It would be even awkward to assume that two objects of these types would ever be equal.

The GF version of `AsymmetricRelation` assumes that the two arguments have the same type:

```
fun AsymmetricRelation: (c: Class) -> (El c -> El c ->
    Formula) -> Formula;
def AsymmetricRelation c f = and (AntisymmetricRelation c f)
    (IrreflexiveRelation c f);
```

On translating higher-order axioms, a type checking is performed, to rule out this kind of type mismatches. Because of the wide usage of `AsymmetricRelation`, mainly, the percentage of higher-order axioms that could be translated is just of almost 43%.

Another reason is that the signatures from HigherOrder are adapted for arguments of type `El` c, but not for `Desc` c, `Class` or `Formula`. However, the majority of functions and predicates take arguments of type `El` c. For better coverage, one might define patterns for the other cases, but that would not improve the statistics considerably.

A special case of the higher-order functions is `subrelation`, which takes two other relations as arguments. Since in over 80% of its appearances, it defines a relation between two binary predicates, its signature in GF was chosen as:

```
fun subRelation2El: (c1,c2,c3,c4: Class) -> Inherits c1 c3
    -> Inherits c2 c4 -> (El c1 -> El c2 -> Formula) -> (El c3
    -> El c4 -> Formula) -> Formula;
def subRelation2El c1 c2 c3 c4 i1 i2 f g = forall c1 (\x ->
```

```
forall c2 (\y ->impl (f (var c1 c1 ? x) (var c2 c2 ? y))
(g (var c1 c3 ? x) (var c2 c4 ? y))));
```

The axiom requires that the types of the arguments of the subrelation are subclasses of the classes used by the ancestor relation, in an explicit way. For the other cases, such as ternary predicates or kinds of function, another subrelation can be defined in a similar manner. The `subrelation2El` function will be used for the translation of SUMO subrelations into GF.

For example, in FinancialOntology-

```
(subrelation accountAt agreementMember)
```

where

```
fun accountAt: El FinancialAccount -> El FinancialOrganization
    -> Formula;
fun agreementMember: El Contract -> El CognitiveAgent ->
    Formula;
```

will be translated into GF as:

```
subRelation2El FinancialAccount Contract FinancialOrganization
CognitiveAgent ? ? (\x,y -> accountAt x y) (\x,y ->
agreementMember x y)
```

Other higher-order function were not translated, as they cannot be properly used in a type system. For example `AssignmentFn`, which denotes the application of a function to its arguments. In GF this was replaced by the mere function assignment operation. Another example is `playsRoleInEvent`, the behaviour of which is defined by the following axiom:

```
(=>
 (playsRoleInEvent ?OBJ ?ROLE ?EVENT)
 (?ROLE ?EVENT ?OBJ))
```

which states that the higher-order function simply denotes the application of the second argument, which is a function to the other two arguments. A similar situation occurs for `playsRoleInEventOfType`, the definition of which relies on `playsRoleInEvent`:

```
(=>
 (and
  (playsRoleInEvent ?OBJ ?ROLE ?EVENT)
  (instance ?EVENT ?CLASS)
  (subclass ?CLASS Process)
  (time ?EVENT ?TIME)
  (located ?EVENT ?PLACE))
  (playsRoleInEventOfType ?OBJ ?ROLE ?CLASS ?TIME ?PLACE))
```

Unlike `capability`, which was rather widely used, these function are not used in other axioms, except the ones that specify their behavior. Consequently they were not translated to GF.

A total figure of 78% of the axioms involving higher-order function from the HigherOrder file were translated successfully from SUMO to GF.

2.4 Evaluation of the Translation of SUMO to GF

A total of 17 files, including Merge and Mid-level-ontology were translated to GF, but other SUMO ontologies can be added semi-automatically after the processing of the initial files. There is a number of cases where the translation was not possible due to reasons that we will explain further on.

Regarding the definitions of concepts and relations, a number of them could not be translated to GF, due to insufficient information. It is the case of relations and concepts which are not defined anywhere, but are just used in axioms. For example:

- in Mid-level-ontology- `partlyLocatedAtTime`, `Launcher`, `Tire`
- in Geography: `Swamp`
- in Economy `OtherChemicalAndFertilizerMineralMining`

Also there are relations which are just defined as functions or predicates, but no additional information is given on them. For example, in FinancialOntology-

```
(instance totalBalance BinaryPredicate)
```

In the first case, the unknown information along with the axioms that use it, will not be represented in GF. For the second case, as we cannot infer any information about the predicate, it will not be translated to GF, and the axioms that use it will be ignored, also.

Regarding the translation of SUMO axioms, there are cases when inferring the type declaration of a variable could not be covered by our approach are the situation when an instance ocurrs negated. For example, in Merge-

```
(=>
  (and
     (instance ?LANG AnimalLanguage)
     (agent ?PROC ?AGENT)
     (instrument ?PROC ?LANG))
  (and
     (instance ?AGENT Animal)
     (not (instance ?AGENT Human))))
```

or when an instance declaration appears as argument of another function, like in Merge-

```
(=>
    (and
      (instance ?Closing ClosingAnAccount)
      (patient ?Closing ?Account))
    (and
      (holdsDuring
        (ImmediatePastFn (WhenFn ?Closing))
        (instance ?Account FinancialAccount))
      (holdsDuring
        (ImmediateFutureFn (WhenFn ?Closing))
        (not (instance ?Account FinancialAccount)))))
```

These axioms are less than 0.05% of the total number and were not translated to GF.

The occurences of negated instance declarations can be solved after defining a complement operator for `Classes`. In this way the type of a variable would be an element from the Boolean algebra that we previously described.

Another example of inconsistency in SUMO is the case when a relation is declared with a given signature, but it is mostly used with a different number of arguments. For example, `earthAltitude`, that was discussed previously. Since `earthAltitude` did not have additional new type definitions for any of the arguments, in GF the signature was inferred from the relation with distance. However in all the axioms from Geography and mondial, `earthAltitude` is used with just 2 arguments, instead of 3, as `distance` has.

In this case, we assume that the type definition is correct and reject the axioms as ill-formed in the type checking phase. There are several other reasons for which an axiom could be rejected as ill-typed. One of them is the remodeling of the type system. In SUMO, `Formula` is a subclass of `Sentence`, while `Class` is a subclass of `SetOrClass`. In the GF type system, `Formula` denotes the return type of predicates, while `Class` represents the set of types. Several axioms that use `Class` or `Formula` would not type check anymore, because of type mismatches, like:

- implicit quantification on `Formulas`. For example, in Government-

```
(=>
 (and
   (agreementEffectiveDate ?AGR ?DATE)
   (confersRight ?FORMULA ?AGR ?AGENT)
   (instance ?TIME ?DATE))
 (holdsDuring (ImmediateFutureFn ?TIME)
          (holdsRight ?FORMULA ?AGENT)))
```

- impossibility to use a variable of type `Formula` and as instance of other types such as `Entity`, which in the original SUMO would have been possible. For example, in Merge-

```
(=>
  (and
     (instance ?PREDICT Predicting)
     (patient ?PREDICT ?FORMULA))
  (exists (?TIME)
    (and
       (holdsDuring ?TIME ?FORMULA)
       (or
          (before ?TIME (WhenFn ?PREDICT))
          (earlier ?TIME (WhenFn ?PREDICT))))))
```

- implicit quantification on `Class`

```
(=>
  (exhaustiveDecomposition @ROW)
  (=>
    (inList ?ELEMENT (ListFn @ROW))
    (instance ?ELEMENT Class)))
```

  This axiom can be expressed directly in the type system, by making sure that all elements of the list are instances of `Class`. In GF this is expressed already, from the definition of `exhaustiveDecomposition`.

- mismatches between `Class` and other types

```
(=>
    (instance ?Loan PiggybankLoan)
    (equal (CardinalityFn
      (KappaFn ?Lender (lender ?Loan ?Lender))) 2))
```

  where `KappaFn` is a function returning a `Class`, while `CardinalityFn` expects an argument of type `El` (`either SetOrClass Collection`).

Some of these missmatches caused by cutting the original hierarchy could be solved by providing additional functions, like a kind of `CardinalityFn` that takes an argument of type `Class`. The others are just collateral effects of the redesigning of the ontology, and would need some extensions of the ontology, that would fit the new design better.

There are other type errors of original SUMO axioms which have been discovered at this phase:

- usage of functions with wrong number of arguments

```
(=>
  (instance ?ORG Manufacturer)
  (hasPurpose (exists (?MANUFACTURE)
                (and
```

```
(instance ?MANUFACTURE Manufacture)
(instance ?MANUFACTURE CommercialService)
(agent ?MANUFACTURE ?ORG)))))
```

where `hasPurpose` has the signature:

```
fun hasPurpose: El Physical -> Formula -> Formula;
```

- usage of relations and concepts that have not been defined (as mentioned before)
- mismatches between subclasses and instances
  - using a subclass instead of an instance For example, in Mid-level-ontology-

```
(=>
 (and
   (instance ?C CavalryUnit)
   (instance ?B Battle)
   (agent ?B ?C))
   (exists (?P ?V ?T)
     (and
     (attribute ?P Soldier)
     (member ?P ?C)
     (instance ?T Transportation)
     (agent ?T ?P)
     (instance ?V Vehicle)
     (patient ?T ?V)
     (during ?T ?B))))
```

where `attribute` has the signature:

```
fun attribute: El Object -> El Attribute -> Formula;
```

and `Soldier` is defined as a class, and not an instance.

  - use an instance as a class For example, in Mid-level-ontology-

```
(=>
(instance ?X Theology)
(exists (?Y)
 (and
  (instance ?Y ReligiousProcess)
  (refers ?X ?Y))))
```

where

```
fun Theology: Ind FieldOfStudy;
```

Further axioms that could not be straightforwardly translated into GF are the ones containing the `subclass`, `domain`, `domainSubclass`, `rangeSubclass`, `range`. Some of these axioms can be rephrased in the type system, but since there is not pattern that would fit most of them, they were not translated to GF. For example, in Merge-

```
(=>
(subclass ?X ?Y)
(and
(instance ?X SetOrClass)
(instance ?Y SetOrClass)))
```

The axiom states that the subclass predicate is applied to arguments which are of type `SetOrClass`. This constraint can easily be implemented in the type system, by properly setting the type of the operand to instances of `SetOrClass`. Neither `instance`, nor `subclass` are used in GF, so implementing the constraint in the type system is the only way to capture the meaning of this axiom.

There are axioms, for which the translation is not straightforward. For example, in Merge-

```
(=>
  (and
     (parent ?CHILD ?PARENT)
     (subclass ?CLASS Organism)
     (instance ?PARENT ?CLASS))
    (instance ?CHILD ?CLASS))
```

The axiom basically states that if `?PARENT` is the parent of `?CHILD`, then for any Class `?CLASS`, of which `?PARENT` is an instance, `?CHILD` must be an instance of `?CLASS` also. Shortly, if the `?PARENT` is a direct instance of some class, `?CHILD` should be a direct instance of a subclass of that class. This can be expressed in the type system with the help of the `Desc` category, but this requires human analyzing and cannot be done automatically.

Each axioms could follow a different pattern, that would require different techiques for translating it into GF, and this cannot be done automatically, or even semi-automatically, since one should manipulate directly the type system.

Higher-order quantified axioms, that contain variables of type `Formula`, `Class` or a kind of `Relation` cannot be expressed in GF. Adding new functions would make this possible, at the cost of making the type system more heavy and complicated, and possibly not equivalent to first-order logic anymore.

Another distinctive aspect of SUO-KIF that cannot be translated into GF is the use of the special variable `@ROW`, which denotes variable number of arguments. The functions and predicates which take variable number of arguments, use the GF built-in lists, when translated. However the GF list is not a normal variable, so one cannot quantify over it.

For example:

```
(=>
```

```
 (and
 (subrelation ?REL1 ?REL2)
 (?REL1 @ROW))
(?REL2 @ROW))
```

which is the axiom that defines the behaviour of `subrelation`. It is higher-order, as it implicitely quantifies over 2 relation, and it also uses the `ROW` variable. In GF it has been rephrased as the definition rule of the `subrelation2El` function.

The percentage of these axioms is about 11% of the total number of axioms.

Another special category of axioms that could not be translated into GF is the ones where the type declaration of a variable comes as a consequence of the usage of that variable in a particular situation. For example, in Merge-

```
(=>
  (and
   (instance ?DRINK Drinking)
   (patient ?DRINK ?BEV))
  (instance ?BEV Beverage))
```

By applying the normal transformation the GF correspondent would be:

```
forall Drinking(\DRINK -> forall Beverage (\BEV -> patient
(var Drinking Agent ? DRINK) (var Beverage Object ? BEV)))
```

The semantic difference is visible. This situation occurs when the left side of an implication of equivalence relation contains an usage of a variable, while the right side contains a type definition for the same variable. The conjunction and disjunction are commutative, so they are not problematic. A statistics shows that almost 12% of the axioms have this property, so they cannot be translated to GF.

Other functions that rely on these higher-order definition were not translated to GF, as `reflexiveRelationOn`, defined in SUMO as

```
(instance reflexiveOn BinaryPredicate)
(instance reflexiveOn AsymmetricRelation)
(domain reflexiveOn 1 BinaryPredicate)
(domain reflexiveOn 2 SetOrClass)
(=>
 (reflexiveOn ?RELATION ?CLASS)
 (forall (?INST)
    (=>
      (instance ?INST ?CLASS)
      (?RELATION ?INST ?INST))))
```

As the function quantifies over a type, this feature cannot be expressed in the GF type system. Further more, it has an implicit quantification over a relation, which is a feature of second-order logic.

After the type checking phase an approximate of 64% of the quantified axioms, were accepted. The main classes of axioms that could not be translated to GF are

- axioms where the type declaration of a variable is a consequence of a specific behaviour of the variable — 12%
- axioms that use predicates which were not translated to GF, such as `range`, `domain` and others — 11%
- axioms rejected by the type checker — 12%
- other cases — 1%

The presence of the type system does not affect the expressivity very much. The benefit is that ill-typed axiom can be rejected before interacting with an automated reasoner. The SUMO system appears to be very permissive from the semantic point of view also, at the risk of allowing the construction of ill-typed constructions.

The wide usage of higher-order functions has the disadvantage that these constructions cannot generally be checked by an automated reasoner, since higher-order logic is undecidable. GF has a better control over higher-order functions.

The GF type checker is a very useful tool, as it provides another guarantee on the type correctness of the axioms, inferring some of the types, so that the axioms will be type checked, before their validity is questioned. It is very useful to differentiate between mere type errors and more logical errors, and concerning this aspect, GF makes the task of the automated reasoner a lot easier.

Chapter 3

Natural Language Generation

Apart from the advantages that the GF type system provides, for the natural language generation the benefits of using GF are considerably more substantial.

As mentioned before, SUMO provides natural language generation of the concepts for English, Mandarin Chinese, Czech, Italian, Romanian, German, Tagalog, French, Hindi and Arabic. Also there are applications such as the KSMSA browser that allow intelligent browsing of the SUMO ontology.

The present work deals with the generation of natural language for the two biggest ontologies - Merge and Mid-level-ontology in 3 languages: English, Romanian and French.

The SUMO templates are built manually, while the GF approach is completely automatic for concepts and semi-automatic for relations. The natural language generation for relations relies on the SUMO solutions for relations, which are processed automatically by GF. It is worth mentioning that the GF solution can be more easily extended to other languages, and it guarantees syntactic correctness.

For English, an approximate of 7,000 concepts and relations have been translated to natural language, while for Romanian and French, we show the limitations of the templates and how the GF approach overcomes them.

3.1 Evaluation of NLG in SUMO

SUMO uses a set of templates for natural language. There are templates for concepts and templates for relations that need to be combined with the natural language representation of their arguments, offering the possibility of changing the polarity. For example:

- the function `age` expressed in English-

```
(format en age "the &%age of %1 is %n %2")
```

where %n will be replaced with "not" for the negation of the predicate, and with the empty string for the affirmative form.

- the concept `SetOrClass` expressed in German-

```
(termFormat de SetOrClass "Menge oder Kategorie")
```

The templates cover the largest part of Merge, but the other ontologies do not have templates even for English.

There are some patterns for the logical connectors and, or, not, implies and equivalence, but it appears as they are not used in the KSMSA browser, which uses the English version for all languages. It is also the case that for many languages, except English, the templates are just partly translated, the rest is kept in English. For example for Czech, just approximately 25% of the templates are translated, the rest is in English.

The structure of the templates is rather simple, and works reasonably just for simple languages, such as English. The templates do not take into account the presence of declension forms for nouns, of the gender agreement with verbs and prepositions, the various moods of a sentence, depending on its usage.

We will proceed analyzing the difficulties of the SUMO approach in generating syntactically correct constructions in natural language.

Starting with Romanian, for example when expressing "the tangent of the square root of X" the combination of the templates for Romanian templates would generate *tangenta lui rădăcina pătrată a lui X* from

```
(format ro SquareRootFn "rădăcina &%square%tpătratã a lui
%1")
(format ro TangentFn "&%tangent%ttangenta lui %1")
```

This construction is incorrect for two reasons. The first is that in Romanian nouns have different declension forms and the possessive preposition *lui* (of) requires Genitive case, which in Romanian demands a different form of the noun phrase *rădăcina pătrată a lui X*, than the Nominative one which is used in the templates as default. The second is the matter of the possesive preposition, which in Romanian needs to agree with the object, that it will bind to. In this case the possesive preposition has been translated with the masculine singular form, but *rădăcina pătrată a lui X* is feminine singular in Romanian. The correct form of the sentence would be *tangenta rădăcinii pătrate a lui X*, which is considerably different compared to *tangenta lui rădăcina pătrată a lui X*.

The Romanian set of templates also features:

- words that do not exist in any Romanian dictionary:

```
(format ro EndFn "the &%end%t{sfanceputul} lui %1")
```

A correct linearization of `EndFn` in Romanian would be *sfârşitul lui %1*.

- literal translations that do not make sense:

```
(termFormat ro TemperatureMeasure "măsură de temperatură")
```

  A correct linearization of this term would be *unitate de măsură pentru temperatură*.

- obsolete words:

```
(format ro connected "%1 %n{nu} este &%connected%t
                                    {imbinat} cu %2")
```

  In this case the word *imbinat* is rather archaic, and is just used as a regionalism nowadays. A more appropriate lineaziation would be *conectat cu*.

- There is also a number of situations where the possesive preposition was wrongly replaced by another preposition that requires the Accusative case, which has the same form as the Nominative one:

```
(format ro attribute "%2 %n{nu} este un &%attribute%t
                                    {atribut} pentru %1")
```

  where *pentru* is the translation of "for".

For French, although nouns do not have multiple declension forms, there is an agreement in gender and number between nouns and other parts of speech that determines them. In this case, the templates cannot capture the agreement either. For example, when expressing "the union of the complement of Y and X", the combination of templates for French would generate *l'union de le complément de Y et X* from the templates:

```
(format fr ComplementFn "le &%complement de %1")
(format fr UnionFn "l' &%union de %1 et %2")
```

The possessive preposition *de* agrees in gender and number with the object in French, similarly. It just keeps the form *de* when combined with a proper name in singular, or with the feminine definite article. In the current construction, *le* which is the masculine definite article, when combined with *de* gives rise to *du*.

Another distinctive feature of French, that is not handled well by the set of templates that SUMO provides is the negation. In French, negation is expressed with the particles *ne* and *pas* placed before and after the verb. For example: *je fais* (I do), will be negated as *je ne fais pas*. In case the verb starts with a vowel, a phonetical mutation occurs, and *ne* becomes *n'*. SUMO tries to handle this mutation in an incorrect way. For the sentences that have the verb "to be" as predicate, the negation just uses the particle *pas*

```
(format fr agent "%1 est %n un &%agent de %2")
```

will be negated as *%1 est pas un agent de %2*, which is not correct in standard French. Also, for verbs beginning with a vowel:

```
(format fr origin "%1 %n{ne} a %n{pas} pour &%origine %2")
```

the negation would be *%1 ne a pas pour origine %2*, which is incorrect also.

The feature that shows best the advantage of a typed system in general, and of GF, in particular, over sets of templates is the assignment of a gender to the variables, according to the gender of their type. For example, if the variable *f* is a number, which is masculine in French, *f* would have masculine gender also. If instead of number, *f* would have been a function, which is feminine in French, *f* would have feminine gender also. This is a very common feature for Romance and Slavic languages, where there is gender differentiation.

The templates simply assume that all the variables have masculine gender, while in GF, the wrapper function `var`, that has acces to the class of the variable also, would assign a proper gender to the variable. Since variables can just be used after being wrapped with `var`, they will have a correct gender for any usage in a quantified formula.

This behaviour shows the importance of separating between variables and known instances of a class. If `Var` and `Ind` or `El` would have been unified in the same category, we could not use a wrapper function to change the gender, since we might accidentaly change the gender of a known instance. For example, `Vietnam`, which is masculine in Romanian would have the type `Country`, which is feminine in Romanian. If we would change the gender according to the type, it would have been used with feminine gender instead, which is not correct.

A solution to this problem would be to implement the variation of the gender as a parameter, generating variants for both masculine and feminine, and choosing one of them when quantifying, depeding on the gender of the class. This would make the concrete representation of the categories heavier, and would not cover the case when the verb agrees with both the subject and the direct object, or just with the direct object. In French it is the case that some verbs agree in gender with the direct object.

Therefore, keeping variables in a separate category is a more general solution, that would yield correct results, and would make it possible to handle the addition of other languages, with potentially more complicated gender system and agreement cases.

An example of how the gender variation feature works in current implementation is the GF axiom:

```
forall Animal (\A -> exists Animal (\B -> smaller
(var Animal Object ? B) (var Animal Object ? A)))
```

which would be linearized in French as:

> *pour chaque animal A il existe un animal B tel que B est plus petit que A*

where animal is of masculine gender in French. For a feminine noun, such as house we would have that:

```
forall House (\A -> exists House (\B -> smaller
(var House Object ? B) (var House Object ? A)))
```

which would be linearized in French as:

> *pour chaque maison A il existe une maison B telle que B est plus petite que A*

The difference can be observed for the forms of the adjective *petit / petite* and also for the gender variation of the relative pronoun *tel / telle* and of the indefinite article *un / une*. The axioms are not taken from SUMO, are just two examples that ilustrate this linguistic feature, and would not probably hold in general, as the set of animals and the set of houses are finite, and hence noetherian.

The same feature of French also holds for Romanian, and show clearly that the SUMO templates would not generate satisfactory natural language constructions even for non-nested templates, as previously shown.

The current issues that are problematic for Romanian and French could make even harder and error-prone the translation of SUMO into other languages, such as the members of the Slavic family. One of the features of these languages is the presence of declension forms for nouns, as Slavic languages, for instance, have a rich inflectional morphology. Also the presence of moods and aspects, that cannot be covered by the templates.

3.2 NLG from SUMO to English

For the generation of natural language in English, the GF resource grammar for English was used, with the larger dictionary - "Oxford advanced learner's dictionary of current English" of almost 50,000 entries. The linearizations of the original types is the following:

```
Class = CN;
El = NP;
Ind = NP;
Var = PN;
SubClass = {};
SubClassC = {};
Inherits = {};
Desc = CN;
Formula = PolSentence;
[El] = {s1,s2: Case => Str; a: Agr};
[Class] = {s1,s2: Number => Case => Str; g: Gender};
```

where `NP` is the GF category for noun phrase, `PN` is the category fo proper nouns and `CN` is the GF category for common noun.

The reason for linearizing types to common nouns is that when linearizing an axiom of the sort

forall Set (\X → ...) it will look like *for every set X ...*
exists Set (\X → ...) it will look like *there exists a set X ...*

where "every" and the indefinite article "a" are both determiners, and in the type system of the GF resource grammars determiners are combined with common nouns and not noun phrases.

`PolSentence` is a category defined especially for this purpose:

```
PolSentence = {s: SentForm => Pol => Str; flag: Flag};
```

where `Pol` is the parameter for Polarity, defined as:

```
Pol = Pos | Neg;
```

where the negative form is used with its informal variant("don't" instead of "do not"). `SentForm` and `Flag` are parameters that perform some optimizations on the form of the sentence:

```
SentForm = Indep | Attrib;
Flag = ExistS | ForallS NumQuant | NothingS NumQuant;
NumQuant = One | Many;
```

The idea is best expressed by some examples.

```
(=>
  (instance ?MIXTURE Mixture)
  (exists (?PURE1 ?PURE2)
   (and
     (instance ?PURE1 PureSubstance)
     (instance ?PURE2 PureSubstance)
     (not (equal ?PURE1 ?PURE2))
     (part ?PURE1 ?MIXTURE)
     (part ?PURE2 ?MIXTURE))))
```

which is translated into GF as:

```
forall Mixture (\MIXTURE -> exists PureSubstance (\PURE1 ->
exists PureSubstance (\PURE2 -> and (not(equal (var
PureSubstance Entity ? PURE1)) (var PureSubstance Entity ?
PURE2)) (and (part (var PureSubstance Object ? PURE1) (var
Mixture Object ? MIXTURE)) (part (var PureSubstance Object ?
PURE2) (var Mixture Object ? MIXTURE)))))
```

and into natural language via GF as:

> *for every mixture MIXTURE there exists a pure substance PURE1 and a pure substance PURE2 such that PURE1 is not equal to PURE2 and PURE1 is a part of MIXTURE and PURE2 is a part of MIXTURE.*

while the SUMO translation via the KSMSA browser is:

for all mixture ?MIXTURE holds there exist pure substance ?PURE1, pure substance ?PURE2 so that ?PURE1 is not equal to ?PURE2 and ?PURE1 is a part of ?MIXTURE and ?PURE2 is a part of ?MIXTURE.

Also:

```
(=>
   (instance ?LIST UniqueList)
   (forall
     (?NUMBER1 ?NUMBER2)
      (=>
        (equal
          (ListOrderFn ?LIST ?NUMBER1)
          (ListOrderFn ?LIST ?NUMBER2))
        (equal ?NUMBER1 ?NUMBER2))))
```

which is translated to GF as:

```
forall UniqueList (\LIST -> forall PositiveInteger (\NUMBER2
-> forall PositiveInteger (\NUMBER1 -> impl (equal (el Entity
Entity ? (ListOrderFn (var UniqueList List ? LIST) (var
PositiveInteger PositiveInteger ? NUMBER1))) (el Entity
Entity ? (ListOrderFn (var UniqueList List ? LIST) (var
PositiveInteger PositiveInteger ? NUMBER2)))) (equal (var
PositiveInteger Entity ? NUMBER1) (var PositiveInteger Entity
? NUMBER2)))))
```

and into natural language via GF as:

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

while the SUMO translation, via KSMSA is:

for all unique list ?LIST holds for all ?NUMBER1, ?NUMBER2 holds if "h element of ?LIST" is equal to "h element of ?LIST", then ?NUMBER1 is equal to ?NUMBER2

The `NumQuant` parameter allows an more elegant rendering of lists of conjunctions of quantifications. For example, in Merge:

```
(=>
 (and
  (holdsDuring ?T1 (legalRelation ?A1 ?A2))
  (instance ?A1 Organism)
  (instance ?A2 Organism))
(holdsDuring ?T1  (relative ?A1 ?A2)))
```

which will be translated to GF as:

```
forall Organism (\A1 -> forall Organism (\A2 -> forall
TimePosition (\T1 -> impl (holdsDuring (var TimePosition
TimePosition ? T1) (legalRelation (var Organism Human ?
A1) (var Organism Human ? A2))) (holdsDuring (var
TimePosition TimePosition ? T1) (relative (var Organism
Organism ? A1) (var Organism Organism ? A2))))))
```

and into natural language, via GF as:

> *for every organism A1 , every organism A2 and every time position T1 we have that if " A1 is a legal relation of A2 " holds during T1 then " A1 is a relative of A2 " holds during T1 .*

The commas avoid the repetition of the conjunction "and". This axiom was not translated in the KSMSA browser.

The GF translations are more grammatically correct and comprehensive. The purpose of the parameters, is to give a better looking construction when combining two quantified axioms, as seen before. SUO-KIF offers the possibility to keep the quantified variables in a list, while in the GF type system this is not possible. Using the `Flag` and `SentForm` we avoid constructions like:

> *for every entity X we have that for every entity Y ...*

Since `PolSentence` keeps too many intermediate forms in the state, a new category was introduced for the purpose of parsing:

```
SS = {s: Str};
```

that can be built from `PolSentence` with

```
fun UsePolSentence: Pol -> PolSentence -> SS;
```

The function `UsePolSentence` takes the `Indep` form of a `PolSentence`, with a polarity indicated by the `Pol` parameter. The two categories `PolSentence` and `SS` resemble the `Cl` - clause and `S` - sentence from the GF resource grammars. `Cl` differs from `PolSentence` as it has parameters for different tenses and moods, while `PolSentence` has the special parameters for optimizing the natural language generation, but features just the Indicative mood and Present tense. This is an example of a domain specific category that handles a particular situation.

Besides axioms, we can also generate natural language for `SubClass`, `SubClassC` and `Ind` declarations, with the aid of the following functions:

```
fun subClassStm: (c1,c2: Class) -> SubClass c1 c2 -> Stmt;
fun subClassCStm: (c1,c2: Class) -> (p: Var c2 -> Formula)
    -> SubClassC c1 c2 p -> Stmt;
fun instStm: (c: Class) -> Ind c -> Stmt;
```

For example, from the GF declarations:

```
fun Beverage_Class: SubClass Beverage Food;
fun Blue: Ind PrimaryColor;
```

we could generate the phrases

beverage is a subclass of food
and
blue is an instance of primary color.

from the following GF constructions:

```
subClassStm Beverage Food Beverage_Class
instStm PrimaryColor Blue
```

Regarding the SUMO concepts, they are written using the CamelCase convention. For example, `SetOrClass` for *set or class*.

In this way the names of the concepts have been expanded and parsed in GF using the dictionary as `CN` and `NP`. For this purpose, new functions have been added to the GF resource grammar.

```
VerbToNounV2: V2 -> N2; -- discovering
VerbToNoun: V -> N; -- walking is healthy
VerbToGerundA: V -> A; -- singing bird
VerbToParticipeA: V -> A; -- the required number
```

The first two functions represent the way of forming a noun from a verb phrase. While the last two represent ways of forming an adjective.

The first problem is a very interesting one, as for various languages there are various ways of obtaining a noun form from a verb. In English the gerund form of the verb is used as noun, signifying the process of performing the action denoted by the verb. The other 2 languages considered for natural language generation differ in this matter.

For functions the starting point was the set of templates from SUMO, but it covered less than half of the total number of functions that were linearized in GF. For the rest we used a similar procedure as for concept, but the automated process stops there. The next step was to form clauses, inserting the arguments in the right places and parse them with GF as clauses. There is a function that builds a `PolSentence` from a GF clause afterwards.

A small statistic shows that:

- Merge: 1136 declarations – 158 not parsed → 14%
- Mid-level-ontology: 1568 declarations – 111 not parsed → 7%

The final result is that just 10% of the over 2.700 concepts and relations could not be parsed. Typicall such examples are

- specific scientific terms: AlethicAttribute, where the term "alethic" is not in the lexicon

- American English terms, since the dictionary just features the British form: WatercolorPicture, where "color" is written as "colour" in British English.

The higher-order functions from HigherOrder have been translated in a similar manner - processed and parsed as `NP` in GF. Example:

```
(instance equal EquivalenceRelation)
```

has been translated to GF as

```
EquivalenceRelation Entity (\x,y -> equal x y)
```

and into natural language as:

"x is equal to y" is an equivalence relation

while in SUMO it is not linearized to natural language.

14 out of the 23 higher-order functions were linearized in English.

There is a SUMO predicate, `names` that maps a concept with its name in English. For the elements and mondial ontologies, there are names declarations for almost all concepts. In this way, we could generated natural language for these files also. Adding the number of linearizations from these two files we obtain a total of almost 7.000 concepts that have a natural language correspondent.

3.3 NLG for Romanian and French

For Romanian and French the situation is different however, since we do not have a resource as the extended English dictionary so we can just use the normal GF lexicon. From the over 1.000 of concepts from Merge, just 28 could be linearized using the lexicon. Another 24 were obtained from Mid-level-ontology.

The basic function from the type system were linearized in the two languages, and same for the functions that extend the GF resource grammars. Regarding the `VerbToNoun`, in Romanian, the participe form for masculine singular is taken and it can further be combined with articles and other determiners. In French, there is no such systematic way to get the noun form from a verb, so the infinitive form can be used as default, but it cannot be combined with articles and determiners. For the functions `VerbToGerundA` and `VerbToParticipeA`, the gerund and participe forms of the verbs are takens, with the significant difference, that for these languages forms for feminine singular, and masculine and feminine plural are needed also. These forms are part of the representation of a verb already, and can be extracted and used.

Further on, we will describe some approaches to achieve natural language generation for these languages also. The starting point was the English translation which uses the GF categories and functions:

```
House = UseN house_N;
```

This concept would clearly look the same in Romanian and French, also, by taking the linearization of `house_N` from the LexiconRon and LexiconFre.

Other cases are not so fortunate:

```
WaterCloud = ApposCN (UseN water_N) (MassNP (UseN cloud_N));
```

because in English appositions are more productive than in other languages.

In both Romanian and French the translation would look like

```
WaterCloud = AdvCN (UseN cloud_N) (PrepNP part_Prep
             (MassNP (UseN water_N)));
```

A more interesting situation occurs when linearizing the function RoundFn which means the rounding of a number. In Romanian it is expressed as *valoarea rotunjită*, while in French it is *la valeur approchée*. Since this is an idiom the GF functions would look differently.

A better idea is to take the English linearizations and translate them with a tool, like Google Translate, and then parse the result with the GF parser in that language. This approach is not error free either, but it would yield to better results for idioms, and in the context of robust parsing, the small error could be corrected. Robust parsing in GF is currently work in progress. An important condition is to have a sufficiently big lexicon for the languages. Currently GF has extended lexicons for English, Bulgarian and Swedish, and larger lexicons could be imported from open source projects for other languages also.

Compared to the SUMO approach, the GF one yields syntactically correct sentences and is easier to reuse for adding more languages.

Chapter 4

Automated Reasoning

Since SUMO offers a generous amount of information and axioms, but no straightforward way to reason about them, it is normal to search for a translation to some format that would allow automated reasoning with the existing data.

The two most general SUMO ontologies, Merge and Mid-level-ontology have been translated to TPTP - FOF standard, and are used yearly in a competition held by CADE, that awards a prize for finding inconsistencies in the two ontologies.

We translated the GF representations of the SUMO ontologies and the axiom files to TPTP - FOF, checked them for consistency and solved small inferences.

4.1 Translation of GF to TPTP

There is a separation between the gf files that were generated from the SUMO ontologies, and the additional files that contain axioms.

The .gf files contain declarations of classes, subclasses and direct instances. Since TPTP is an untyped system, whereas GF is strongly typed, the information about types has been translated as additional predicates, that ressemble the original `instance` predicate from SUMO.

For subclasses, the translation reflects the possibility of converting from the subclass to the superclass. For example, in Merge-

```
(subclass Adjective Word)
```

which in GF was translated as:

```
fun Adjective_Class: SubClass Adjective Word;
```

and would be further on translated to TPTP as:

```
fof(axMerge2, axiom,
( ! [X]:
```

```
(hasType(type_Adjective, X) =>
hasType(type_Word, X))).
```

Also, for the SubClassC category-

```
fun NonnegativeRealNumber_RealNumber: SubClassC
NonnegativeRealNumber RealNumber (\NUMBER ->
greaterThanOrEqualTo (var RealNumber Quantity ? NUMBER)
(el Integer Quantity ? (toInt 0)));
```

will be translated to TPTP as:

```
fof(axMerge389, axiom,
( ! [X]:
(hasType(type_NonnegativeRealNumber, X) <=>
 (((hasType(type_RealNumber, Var_NUMBER)) &
( f_greaterThanOrEqualTo (Var_NUMBER,0)))))))).
```

The axiom captures the meaning of `SubClassC`, as following:

- any `NonnegativeRealNumber` is a `RealNumber` and is ≥ 0
- if a `RealNumber` is ≥ 0, then it is a `NonnegativeRealNumber`.

For instance definitions, we have a simpler translation pattern.
For example, in Merge-

```
(instance Awake ConsciousnessAttribute)
```

which is translated into GF as:

```
fun Awake: Ind ConsciousnessAttribute;
```

will be translated to TPTP as:

```
fof(axMerge686, axiom,
(hasType(type_ConsciousnessAttribute, inst_Awake))).
```

The identifiers`axMerge2, axMerge686` are the names of the axioms, as TPTP requires all axioms to be given unique names. ! signfies the universal quantifier in TPTP, while ? represents the existential quantifier.

The typing rules for a variable or instance are expressed in TPTP by the `hasType` predicate. The TPTP axiom, asserts the meaning of the original SUMO predicate, by stating that for every variable that is of type `type_Adjective` (is an instance of the class `Adjective`), then it also has the type `type_Word` (is an instance of the class `Word`).

A more commonly used approach for expressing typing declarations in first-order logic is to create a predicate for each type, like

```
type_ConsciousnessAttribute(inst_Awake)
```

. We did not choose this method, since the SUMO classes are not just used as types, in typing declarations, but also as arguments for functions. For functions taking arguments of type `Desc c`, using our approach, SUMO classes can be used directly, and there is no difference between the SUMO class used as a type, and the class passed as argument to a function.

TPTP assumes that identifiers starting with capital letters are variables, while constants and functions should start with a small letter. Hence, when translated to TPTP, classes are written with the prefix "type_", like `Adjective` → `type_Adjective`. Instances are written with the prefix "inst_", like `Yellow` → `inst_Yellow`. Functions names get the prefix "f_", like `property` → `f_property`. The last transformation was necessary as SUMO functions could sometimes start with capital letter, also, like `CardinalityFn`. For variables that appear in quantified formulas, they are written in TPTP with the prefix "Var_" that ensures the fact that they will be treated by the theorem prover as variables.

The functions that manipulate `Formula` objects, such as `not`, `and`,`or`, `impl` and `equiv` have been translated into their corresponding first-order logic operators that are predefined in TPTP: ∼, &, |, and ⇒.

For the `both` and `either` functions, the built-in & and | are used again. For example, in GF:

```
fun Australia: Ind (both Country Nation);
```

will be translated to TPTP as:

```
fof(axmondial5372, axiom,
(hasType(type_Country, inst_Australia) &
hasType(type_Nation, inst_Australia))).
```

where if a variable is of type `both A B` means that it is of type A and of type B. A similar situation holds for `either A B`, where the variable is either of type A or of type B.

The equality operator `equal`, the situation is more complicated. In SUMO, because of the structure of the concepts, it could basically take any arguments, like classes, and relations and instances. In GF, the `equal` function would just take arguments of type `El Entity`, so it would not be possible to test the equality of formulas, functions or classes. In SUMO, `equal` is defined as an `EquivalenceRelation`, with some extra axioms, for the various kinds of arguments that it might take. For instances, the axiom, that verifies a property of equal objects:

```
(=>
  (equal ?THING1 ?THING2)
  (forall (?CLASS)
    (<=>
      (instance ?THING1 ?CLASS)
      (instance ?THING2 ?CLASS))))
```

cannot be translated to GF, as it contains a variable type declaration and quantification over a class. Moreover, a more solid interpretation of equality would be

using at least a congruence relation, not just an equivalence one. SUMO does not have the concept of congruence, while theorem provers that can process first-order logic with equality, usually have optimized treatment of the built-in equality from TPTP. For these reasons, the translation from GF to TPTP, uses the default TPTP equality - `=` for the `equal` function.

The existential and universal quantifiers from SUMO and GF, were translated as the built-in quantifiers from TPTP. The type declarations are expressed with the function `hasType` in a similar manner to the treatment of `SubClass` declarations.

For example, in Mid-level-ontology-

```
(=>
  (instance ?T Paragraph)
  (exists (?S)
   (and
     (instance ?S Sentence)
     (part ?S ?T))))
```

which is translated to GF as:

```
forall Paragraph (\T -> exists Sentence (\S -> part (var
Sentence Object ? S)(var Paragraph Object ? T)))
```

and further on, to TPTP as:

```
fof(axMidLem14, axiom,
( ! [Var_T]:
(hasType(type_Paragraph, Var_T) =>
(( ? [Var_S]:
(hasType(type_Sentence, Var_S) & (f_part(Var_S,Var_T)))))))).
```

A special case is the translation of higher-order axioms to TPTP. In this case, the function call is replaced by the definition of the function, rendering a construction in first-order logic.

For example, in Merge-

```
(instance AdditionFn CommutativeFunction)
```

which is translated in GF as:

```
CommutativeFunction Quantity Quantity (\x, y -> AdditionFn x y)
```

will be translated to TPTP as:

```
fof(axMergeHiO39, axiom,
( ! [Var_x]:
(hasType(type_Quantity, Var_x) =>
(( ! [Var_y]:
(hasType(type_Quantity, Var_y) =>
(f_AdditionFn(Var_x,Var_y) = f_AdditionFn(Var_y,Var_x)))))))).
```

Also, for subrelations:

```
subRelation2El Organism Organism Organism
Organism ? ? (\x, y -> mother x y) (\x, y -> parent x y)
```

will be translated to TPTP as:

```
fof(axMergeSubRel75, axiom,
( ! [Var_x]:
(hasType(type_Organism, Var_x) =>
(( ! [Var_y]:
(hasType(type_Organism, Var_y) =>
(f_mother(Var_x,Var_y) => f_parent(Var_x,Var_y)))))))).
```

In case that the subrelation takes arguments of different types, compared to its ancestor, the types of the subrelation are kept for the representation in TPTP, since they are more restrictive.

4.2 Applications of Automated Reasoning

The resulting files have been checked with the automated theorem prover for first-order logic **E**[11]. It is a multiple award-winner theorem prover which is freely available and is based on equational superposition calculus. It provides support for first-order logic with equality. **E** has been used to check the consistency of the largest ontology currently available - ResearchCyC [12]. The TPTP translations of the GF files were tested for consistency with **E**, and no contradiction was found, given the time limit of 1 hour per file.

A possible application for automated reasoning would be to certify a coercion between two SUMO classes. For example, the coercion from `AnimacyAttribute` to `Attribute`, which is not trivial. In SUMO we have that:

```
(subclass AnimacyAttribute BiologicalAttribute)
(subclass BiologicalAttribute InternalAttribute)
(subclass InternalAttribute Attribute)
```

This problem can be translated to TPTP as:

```
fof (conj2, conjecture,
( ! [X]:
(hasType(type_AnimacyAttribute, X) =>
hasType(type_Attribute, X)))).
```

This time, the TPTP construction is not an `axiom`, but a `conjecture`, which means that it has to be proved by the theorem prover. The disadvantage of most high-performance theorem provers is that if they have more than one conjecture in a file, they would not prove all of them, but would stop if they find one which is provable. This mean that for every problem one must create a different file. For this reason, one cannot prove all the type coercions which are used in the axioms and where the Inherits object is not inferred by the type checker.

Another possible application is to determine if an instance has a particular type. For example, in SUMO-

```
(instance Volt CompositeUnitOfMeasure)
(subclass CompositeUnitOfMeasure UnitOfMeasure)
(subclass UnitOfMeasure PhysicalQuantity)
(subclass PhysicalQuantity Quantity)
```

The question if `Volt` is an instance of `Quantity` can be translated to TPTP as:

```
fof (conj1, conjecture,
(hasType(type_Quantity, inst_Volt))).
```

More complicated problems can be taken from the SUMO webpage [1]. For example, the problem, having the hypotheses:

```
(instance John Man)
(instance Jane Woman)
(instance Mary Woman)
(mother John Mary)
(sibling Jane John)
```

and the conclusion to be proved:

```
(mother Jane Mary)
```

This problem assumes the usage of 3 additional quantified axioms from Merge, the commutativity of `sibling` and the fact that `mother` is a subrelation of `parent`. The translation to GF includes some extra type constraints, that must be proved first, like the coercion `Man` to `Organism` and to `Object`, which are required by the axioms. All these problems are sucessfully proved by the theorem prover.

4.3 Evaluation of the Translation of GF to TPTP

All GF definitions were successfully translated to TPTP. Regarding the axioms, an approximate figure of 94% of the the GF representations of SUMO axioms were translated to TPTP.

The category of axioms that could not be represented in the first-order logic format, is represented by the axioms containing nested predicates. For example, in Merge-

```
(=>
 (and
  (instance ?EDUCATION EducationalProcess)
  (patient ?EDUCATION ?PERSON))
(hasPurpose ?EDUCATION
```

[1] http://sigmakee.cvs.sourceforge.net/viewvc/sigmakee/KBs/tests/

```
(exists (?LEARN)
  (and
    (instance ?LEARN Learning)
    (patient ?LEARN ?PERSON)))))
```

which is translated into GF as:

```
forall EducationalProcess (\EDUCATION -> forall Entity
(\PERSON -> impl (patient (var EducationalProcess Process ?
EDUCATION) (var Entity Entity ? PERSON)) (hasPurpose (var
EducationalProcess Physical ? EDUCATION) (exists Learning
(\LEARN -> patient (var Learning Process ? LEARN)(var Entity
Entity ? PERSON))))))
```

The nested predicates cannot be expessed as a valid formula in first-order logic, as the inductive way of building formulas is the one we showed in an earlier chapter, and can just accept predicates taking terms as arguments and not other predicates.

A number of function that are not significant for the automated reasoning part have not been translated to TPTP. These are function such as:

```
(abbreviation ArraburyQueenslandAirport "AAB")
```

that contain a constant of type `String`.

There exist two translations from Merge and Mid-level-ontology to TPTP[2], for an annual competitions, that awards a prize of 100$ to the contestants that find inconsistencies in any of the two ontologies. These translations are done into TPTP FOF also. Nested predicates and implicit or explicit quantifications over `Formula` were not possible in this translation either. Also the functions which were not significant for automated reasoning in the GF to TPTP translation were not translated to TPTP from SUMO either. The function `KappaFn`, which is a class-forming function that was used to build Russell's paradox, is not translated to TPTP from SUMO, for instance. For higher-order functions and subrelations, the definitions are used as a macro, hence the second-order constructions become first-order and can be used as axioms in TPTP. For functions like `capability` and `playsRoleInEvent`, however, the definitions are not used as a macro, and so, all axioms using these functions could not be expressed.

The category of axioms that the SUMO to TPTP translation can express but not the GF to TPTP are

- axioms where the type declaration of a variable occurs as a consequence of the previous usage of that variable
- axioms containing `subclass`, `domain`, `range` and the other predicates mentioned in the section about the limitations of the SUMO – GF translation.
- negative type declarations

[2]http://www.cs.miami.edu/ tptp/Challenges/SUMOChallenge/

- quantification over classes

The loss is almost 23% of the total number of the axioms. However, the SUMO to TPTP translation, would bring about some type errors, for functions which are not applied to the right number of arguments, and which the TPTP type checker would not accept.

The expressivity of the SUMO to GF to TPTP translation is comparable to the direct SUMO to TPTP translation. It is worth mentioning that the first translation yields to a slightly slower system because of the additional type declarations that need to be checked by the theorem prover. However, for a standard like typed FOF, the presence of types would make the theorem proving and model finding processes more efficient.

Chapter 5

Evaluation

The translation of SUMO into GF, shows both the advantages of a type system in the context of representing knowledge and reasoning about it, and the expressive power of GF in the field of natural language generation and processing where it provides efficient and comprehensive solutions.

The benefits of translating the SUMO ontologies to GF, are notable for the type checking part and the natural language generation one. Moreover, none of the SUMO paradoxes can be expressed in the GF translation, which is closer to the first-order logic formalism.

For the natural language generation part, the current approach had more than twice the coverage of the existing results, with a higher degree of automation and re-usability. Also the benefits of a type system are considerable when expressing some problematic cases of agreement, as shown.

For the automated reasoning part, it generated GF and TPTP translations for all the files, and offered a more comprehensive diagnosis, rejecting SUMO ill-typed axioms from the type checking phase, before they were translated to TPTP. Without the higher-order axioms, the difference between the expressivity of the direct TPTP translation of SUMO and the one from SUMO to GF and further on to TPTP is not considerable.

The ongoing work on a typed version of FOF in TPTP[1], would make the GF translation of the axioms even more efficiently processed by a theorem prover, because the type declarations can be easily extracted from the TPTP translation of the GF axioms.

[1] http://www.cs.miami.edu/ tptp/TPTP/Proposals/TypedFOF.html

Chapter 6

Related Work

Since Semantic Web is a very popular field nowadays, there exists a large number of applications dealing with ontologies and building various applications on top of them.

Regarding the languages that are used to encode ontologies, as mentioned before, the most popular ones, such as KIF, OWL, CycL or Gellish do not have a type system. Since their role is mainly descriptive, they do not impose any restrictions on the data they encode. Some primitive forms of type checking, such as checking that a function is applied to the right number of arguments is done on a higher level, such as when checking the ontology for consistency with an automated prover.

The programming language prototype Zhi#[1] is a novel language for encoding ontologies, which has a static type-system and it is compiled further more to C#. The type system is inspired from the Java and C# type systems, and it benefits from using the C# built-in types and functions. However, the syntax looks very much like the normal C# one, and it is not very intuitive for users that do not have a reasonable imperative programming background.

To our knowledge, the current work is the first representation of an ontology in a strongly typed system with dependent types. The benefits of dependent types are visible when expressing the concepts and relations from SUMO in GF, as they provide better control on their semantics. The usage of dependent types gives elegance and robustness to the representation.

Regarding natural language generation, there are many notable applications that verbalize ontologies. Most of them however, have only English as main target, and do not provide multilingual translations. A notable example is the KPML project [15], which provides natural language generation for 10 languages. Another interesting case is the Gellish ontology which provides direct verbalization for the concepts and relations, and is available for English, German and Dutch. However, for languages with a more complicated inflectional morphology, or languages which feature clitics in the grammar, such as the

[1] http://www.alexpaar.de/zhimantic/ZhiSharp.pdf

members of the Slavic or Romance families, the applications that generate natural language, do not usually render correct constructions for these problematic situations. The GF approach has built-in mechanisms for verbalization via the concrete syntax, and the translations it provides are syntactically correct. Moreover, GF has support for multilingual translation for the languages in the resource library.

Regarding automatic reasoning, there has been work for checking the consistency of all the well-known ontologies. A notable example is the use of the E theorem prover for the ResearchCyC ontology[12]. However, SUMO is the most well-known case of an ontology which is checked for consistency every year, as part of the CADE competition. Compared to the official SUMO translation to TPTP, our approach has a comparable expressivity, rejecting the ill-typed axioms at an earlier stage.

The project OntoNat[6] provides automated reasoning for the SUMO ontology with KRHyper, which is a theorem prover for first-order logic that implements hyper tableaux. It provides better behaviour for non-provable tasks than an ordinary theorem prover for first-order logic. It also features a special treatment of equality from SUMO, which is weaker than the built-in equality from TPTP. Moreover, the project provides a more elaborated treatment of class-forming operators from SUMO and of first-order quantifiers. The project can answer to a question posed in normal English, by using the WordNet mappings and a simple parser, in order to infer the SUMO expression that should be checked.

Chapter 7

Future Work

The current work explores aspects of data modeling, compiling from an untyped system to a typed one and from a typed system to first-order logic, type inference, natural language generation, and automated reasoning. Each of this can be extended in a more comprehensive manner in the future.

Starting with data modeling, it is worth mentioning that there is a dimension of SUMO concepts which was not treated in this project - time. As many discussions on the SUMO forum point out, the 4^{th} dimension, time, which brings about many controversies. In case an object changes one of its attributes, that would determine a change of type, is it the same object ? For example if `Lamb` switches from `NonFullyFormed` to `FullyFormed`, then it becomes a `Sheep`. It might be natural to assume that it is still the same object. However, if a `Human` switches from `Living` to `Dead`, then one might argue that it is not the same object anymore. It would be interesting to model the concept of time in the type system, although it would make it more complicated, and probably, it would not be possible to translate it to FOF anymore.

For the translation of SUMO to GF, one could think of alternative models of the type system that would permit more axioms to be translated, or pattern of rephrasing the higher-order functions and axioms from SUMO in a way that would permit them to be checked by an automated reasoner in FOF.

For the type inference process, it would be interesting to solve a more general problem, which is the inference of some classes of dependent types, such as `Inherits` with the aid of a automated reasoner, on the GF type checking phase.

Regarding the natural language generation, there are many directions for future work. One of them would be to to generate natural language for the other two languages for which GF has a large lexicon, following the approach described in the project, or to use other techniques for reusing the current translations for English, in the process of adding new languages.

Another interesting project would be to generate higher-quality natural language, following the idea[1] that would require truncating the hierarchy even

[1] http://www.ontologyportal.org/student.html

more, separating `Attributes` and `Processes`. Instances of `Attribute` and its subclasses can be linearized as adjective phrases, while instances and subclasses of `Process` are to be linearized as verb phrases. In this way a predicate like

```
(attribute ?X NonFullyFormed)
```

would not be linearized as *non fully formed is an attribute of X* in the best case, but as *X is not fully formed*. For a predicate like

```
(agent Reasoning ?A)
```

we would obtain *A reasons* instead of *A is an agent of reasoning*. This process is not straightforward however, as there are instances of Attribute that are not directly linearizable as adjectives. For example in WMD - `Fever`, which can be transformed into an `AP` as *suffering from fever*, and in Merge - `Contract` and `Promise`, which could even less obviously transformed into `APs`. However the last two are not used in any axiom as attributes, they are just declared as such. Using this approach, the functions that take processes and attributes as arguments would have a better linearization into natural language. This category covers the most widely used predicates, such as `patient`, `agent` and other instances of `CaseRole`, and `attribute`.

Regarding the automated reasoning part, the GF abstract syntax trees representing SUMO axioms can also be translated to THF(typed higher-order logic)[2], as they are typed already, and check the higher-order axioms more thoroughly. Other applications would be a more efficient translation of GF to TPTP, or developing some interesting techniques for complete treatment of some subsets of SUMO, such as arithmetics.

Another interesting application would be to build a user interface for the translation of SUMO to GF, where users could ask questions like *Is London the capital of Great Britain ?* in English or another language for which we have provided natural language generation, and the GF parser would transform it to a GF/TPTP axiom that could be checked by the theorem prover. Its result would be processed by GF into a Yes/No answer, or the affirmative/negative form of the interrogative clause. The result could be influenced by the ambiguity of the parsing. For example, as strange as it seems this question would get a negative answer. There are two concepts that are named London, in mondial -

```
(names "London" Countries-GB-provinces-GreaterLondon-cities-London)
(names "London" Countries-CDN-provinces-Ontario-cities-London)
```

The GF parser returns all the possbile parse trees in alphabetical order, so the London city from Canada would be the first option. Ambiguities are very rare, actually, but it is interesting to find a way to infer the more probable of the two. Future development of the GF parser might feature user provided probabilities to get the most probable parse tree. In the absence of a procedure to decide the most probable parse tree, one might try all the possibilities, and return a positive answer if one of them would lead to a positive result.

[2]http://www.cs.miami.edu/ tptp/TPTP/Proposals/THF.html

The current work gives an insight on these areas, but since they have so many connections to other fields, and have so much potential to be developed, it leaves many problems to be solved in the future. The result of this project deals with some parts of these areas, for building application from SUMO to GF and from GF to TPTP, along with an analysis of all these, that might make open the way to other interesting projects.

Bibliography

[1] Ranta, A.: Grammatical Framework: A Type-Theoretical Grammar Formalism. Journal of Functional Programming 14(2) (March 2004) 145–189

[2] Ranta,A.: The GF Resource Grammar Library.Linguistic Issues in Language Technology, 2009, to appear

[3] Angelov, K., Bringert, B., Ranta, A.: A Portable Run-Time Format for Type-Theoretical Grammars. Journal of Logic, Language and Information, 2009, to appear

[4] Niles, I., Pease, A.: Linking Lexicons and Ontologies: Mapping WordNet to the Suggested Upper Merged Ontology. In Proceedings of the 2003 International Conference on Information and Knowledge Engineering (IKE 03), Las Vegas,2003, p. 23-26

[5] Sutcliffe, G., Schulz, S., Claessen, K., van Gelder, A.: Using the TPTP Language for Writing Derivations and Finite Interpretations . Proc. of International Joint Conference on Automated Reasoning (IJCAR),Lecture Notes in Computer Science,Springer Verlag, August 2006

[6] Baumgartner,P. , Suchanek, F.M.: Automated Reasoning Support for SUMO/KIF. Manuscript

[7] Niles, I., Pease, A.: Towards a standard upper ontology. In Proceedings of the 2nd International Conference on Formal Ontology in Information Systems(FOIS-2001)

[8] Sutcliffe, G.: The TPTP Problem Library and Associated Infrastructure: The FOF and CNF Parts, v3.5.0, Journal of Automated Reasoning, p. 337-362, 2009

[9] Enache, R., Ranta, A., Angelov, K.: An Open-Source Computation Grammar for Romanian, Proceedings of the CICLing 2010, LCNS Springer, to appear

[10] Johanisson, K.: Formal and Informal Software Specifications. PhD thesis, Goteborg University, Chalmers University of Technology, June 2005

[11] Schulz, S.: E – A Brainiac Theorem Prover. Journal of AI communications 15(2/3):111-126, 2002

[12] Ramachandran, D., Reagan, P.,Goolsbey, K.: First-Orderized ResearchCyc: Expressivity and Efficiency in a Common-Sense Ontology. AAAI Workshop on Contexts and Ontologies: Theory, Practice and Applications. Pittsburgh, Pennsylvania, July 2005.

[13] Ranta, A.: Structures grammaticales dans le français mathématique. Mathématiques, informatique et Sciences Humaines., vol. 138 pp. 5-56 and 139 pp. 5-36, 1997.

[14] Ranta, A.: Type-Theoretical grammar. Oxford Science Publications, Clarendon Press, Oxford, 1994.

[15] Bateman, J.: Enabling technology for multilingual natural language generation: the KPML development environment. Journal of Natural Language Engineering, 3(1):15–55, 1997.

[16] Miller, G.: WordNet: A Lexical Database for English. Communications of the ACM Vol. 38, No. 11: 39-41, 1995.