Typeful Ontologies with Direct Multilingual Verbalization

Krasimir Angelov and Ramona Enache

Department of Computer Science and Engineering,
Chalmers University of Technology and University of Gothenburg
{krasimir,ramona.enache}@chalmers.se

Abstract. We have developed a methodology for representation of ontologies in a strictly typed language with dependent types. The methodology is supported by an experiment where we translated SUMO (Suggested Upper-Merged Ontology) to GF (Grammatical Framework). 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. SUMO is the largest open-source ontology describing over 10,000 concepts and the relations between them, along with a number of first-order axioms, which are further on used in performing automated reasoning on the ontology. GF is a type-theoretical grammar formalism mainly used for natural language applications. Through the logical framework that it incorporates, GF allows a consistent ontology representation, and thanks to its grammatical features the ontology is directly verbalized in a number of controlled natural languages.

Keywords: ontologies, type theory, knowledge representation, automated reasoning, natural language generation.

1 Introduction

The constantly growing amount of formal knowledge has brought about the necessity of a coherent and unambiguous representation of ontologies which can further be processed automatically. As a consequence, a number of ontology description languages like KIF [1], OWL [2], CycL [3] and Gellish [4] has emerged. However, the focus in all these languages is on the knowledge representation and consequently, they are mainly descriptive, leaving tasks such as consistency checking or natural language generation to external tools. Moreover, most languages are based on some kind of untyped first-order logic with predicates which occasionally allows higher-order constructions. They aim to maximize the expressivity with the cost of allowing set theoretical paradoxes to be expressed (Section 3). Also, because of the lack of a type system, one can easily extend such ontologies with axioms which are not well-formed. Although type information in these languages is often provided in the form of logical assertions, the validation of correctness is left to a reasoner which may or may not be able to find all problems. Even with a complete and decidable reasoner, if the ontological language has the open-world assumption, a potential problem might be left undiscovered, if it

M. Rosner and N.E. Fuchs (Eds.): CNL 2010, LNAI 7175, pp. 1–20, 2012.

is not stated explicitly that certain classes in the ontology are disjoint. This is a problem when dealing with large coverage ontologies. Or, for example, a predicate could be applied to an argument of the wrong type, or a small change in the signature of a function could lead to the update of all its occurrences. If all these checks are manual, then this is a resource-consuming and error-prone process. In contrast, database systems are equipped with rigid database schemas which ensure that the information is always kept consistent. The programming languages community was also dealing with that from the very beginning of computer science and has developed many different type systems.

We have developed a methodology for encoding of ontologies in strictly typed language based on type theory with dependent types. As a proof-of-concept, an experiment with SUMO [5], the largest open-source ontology available today. The implementation language of choice is GF [6]. The result is a controlled language which could be used to formulate new axioms in SUMO or to render existing axioms in natural language. Further on, we analyzed the difference of expressivity compared to the original ontology and also the other benefits that one can get from encoding an ontology in GF.

SUMO consists of 2 upper-level ontologies (*Merge*, *Mid-level-ontology*) describing general concepts, and 29 domain-specific ontologies for finances (*FinancialOntology*), geographical concepts (*Geography*), and others. The ontology is written in a dialect of KIF (Knowledge Interchange Format [1]), called SUO-KIF, which permits the declaration of concepts in a human-readable form, featuring support for expressing first-order predicate calculus constructions. However, due to the modelling of the hierarchy in SUMO, which treats functions and relations as ordinary concepts, it is possible to express second-order logic constructions in SUO-KIF, such as quantification over functions and relations.

The SUMO ontology has natural language translations for the *Merge* module in 12 languages. The translations are based on a set of string templates, which are combined by concatenation. They are hand-written and cover the ontology partially. However, the templates are not expressive enough to handle various natural language phenomena such as case and gender agreement or phonetic mutations. We will show how these problems were solved by using GF.

GF is a type-theoretical grammar formalism which distinguishes between abstract and concrete syntax. The abstract syntax is a logical framework based on Martin-Löf's type theory [7], in which the application domain can be described in an abstract language-neutral manner. The concrete syntax is a mapping of the abstract syntax into some controlled natural language. Since it is possible to have multiple concrete syntaxes, linked to the same abstract syntax, the abstract syntax acts as a semantic interlingua which allows simultaneous translation into multiple controlled languages.

We consider the abstract syntax of GF as a kind of ontology description language and translate some of the axioms from SUMO to statements in the abstract syntax of a grammar. Other axioms, those related to the natural language generation from SUMO, are used to generate the concrete syntax. The rest of the axioms are just converted to abstract syntax trees and used in the automated theorem prover for reasoning.

The development of a grammar for a new controlled natural language from scratch would involve an ad hoc implementation for low-level linguistic details such as word

order, agreement, etc. This is simplified by using the resource grammar library [8] developed in GF. The library provides an abstract syntax for common general-purpose natural language constructions and concrete syntaxes corresponding to 16 languages. The usage of the library ensures that the rendering is always syntactically correct and reduces the development effort for new application grammars. The resource grammar library was used for the generation of the concrete syntax of SUMO.

Another advantage of GF is the portability of the grammars, via PGF [9] – a runtime binary format, which can be used by applications written in Haskell, Java, JavaScript, C and Python - through the GF runtime system. In this way, the GF grammars can be embedded in user applications. GF has been used in various large-scale projects such as the dialogue system research project TALK [10], the educational project WebALT [11], the verification tool KeY [12], and the project in multilingual translation MOLTO [13].

Regarding the automated reasoning and the checking for consistency [14], SUMO was mapped to TPTP-FOF [15], a standard untyped first-order logic language, which is accepted by most theorem provers. There is an annual competition held during the premier conference in automated deduction, CADE[1], which awards prizes for finding inconsistencies in one of the two upper ontologies from SUMO, based on these mappings[2]. A similar translation from SUMO-GF to TPTP is provided. The translated ontology is checked for consistency and is used for making inferences on the abstract syntax trees or natural language, with the aid of an automated theorem prover (Section 6).

SUMO is also associated with a knowledge engineering environment – Sigma [16], which can be used for intelligent browsing of the ontology, optimized natural language generation and automated reasoning [17]. An alternative system with similar capabilities is the KSMSA browser[3]. The web user interface of GF also evolved in the direction of ontology browsing. Since his interface is still under development, we will give an overview of it in Section 7.

From the total number of ontologies that SUMO provides, 17 were translated into GF. These are: *Merge* and *Mid-level-ontology* – the upper ontologies and 12 domain ontologies. The remaining ontologies can also be ported to GF using the same techniques, in a semi-automatic way.

The advantages of representing the SUMO ontology in GF are the possibility to type-check the axioms and the definitions at an early stage and also to generate natural language of a higher syntactical quality. The translation to GF, is also an in-depth analysis of SUMO and the benefits that a type system in general, and GF in particular, could bring to ontology development.

2 The Abstract Syntax of SUMO-GF

The two languages SUO-KIF and GF have been created for different purposes and have evolved in different ways. It is not surprising that the translation of SUMO from SUO-KIF to an abstract syntax in GF is not trivial. Still we will show that the different

[1] `http://www.cadeinc.org/`
[2] `http://www.cs.miami.edu/~tptp/Challenges/SUMOChallenge/`
[3] `http://virtual.cvut.cz/ksmsaWeb/main`

ontological concepts - from classes and taxonomical relations to complex logical axioms have natural representations in GF.

2.1 The Taxonomy

The most central component of every ontology is the taxonomy of classes, and this is the starting point from where we begin the ontology modelling in GF.

Knowledge representation languages like OWL, KIF and CycL do not set a sharp border between classes and instances. In fact, the classes are just instances of one special class which is the class of all classes. In SUMO the special class is called *Class* and there is a predicate *subclass* which is used to assert the taxonomical relations. For example, the axiom:

$$(subclass\ Human\ Hominid) \tag{1}$$

asserts that the class *Human* is a subclass of *Hominid*. Furthermore, there is an axiom stating that everything that is a subclass of *Entity* is also an instance of *Class* and viceversa:

$$\begin{aligned}(<=>\ &(instance\ ?CLASS\ Class)\\ &(subclass\ ?CLASS\ Entity))\end{aligned}$$

Since the *subclass* relation is transitive and *Entity* is the most general class, from the axiom:

$$(subclass\ Class\ SetOrClass)$$

it follows that *Class* is itself an instance of *Class*:

$$(instance\ Class\ Class)$$

This kind of cyclic relations were proven to be inconsistent because they lead to different kinds of paradoxes (Section 3). The other two popular languages OWL and CycL are not exceptions and similar examples could be constructed in them as well. This seems to be a common mistake because the first version of Martin-Löf's [18] type theory suffered from the same inconsistency which was first demonstrated with Girard's paradox [19]. The problem was resolved in the later versions of the theory [7] by introducing the concepts of small and big types. In the context of SUMO, this would be translated as a restriction which states that *Class* cannot be an instance of *Class* because it is too big to fit as an instance of itself. The abstract syntax of GF is a logical framework consistent with the modern type theory, so if we want to model ontologies like SUMO in GF we have to resolve the conflict.

GF distinguishes between values and types. Every value belongs to some type but none of the types could be a value as well, so it is not possible for a type to belong to another type. The solution for the cyclic relation in SUMO is to declare that *Class* is a type:

$$\textbf{cat}\ Class;$$

Now the classes will be values of type *Class*. For instance:

$$\begin{aligned}\textbf{fun}\ &\mathit{Entity} : \mathit{Class};\\ &\mathit{Hominid} : \mathit{Class};\\ &\mathit{Human} : \mathit{Class};\end{aligned}$$

Essentially, we cut the class *Class* from the common hierarchy and move it to another level (also known as universe in type theory).

Once we have a way to define classes in the abstract syntax we could also define the taxonomy. In SUMO, the taxonomy is encoded by using the *subclass* predicate. In GF, we can translate *subclass* either as a function or as a type. Since we want to be able to statically check the axioms for well-formedness we choose to represent the predicate as a type:

$$\textbf{cat}\ \mathit{SubClass}\ \mathit{Class}\ \mathit{Class};$$

then the human-hominid relation could be asserted as:

$$\textbf{fun}\ \mathit{Human\_Class} : \mathit{SubClass}\ \mathit{Human}\ \mathit{Hominid}; \tag{2}$$

Here, the *SubClass* type is an example of a dependent type. The dependent types are not just simple identifiers, but have in addition indices of some type. In this case, *SubClass* is a type indexed by two values of type *Class*. In the case of *Human_Class* those are *Human* and *Hominid*.

Note that while in the original SUMO axiom (1) we had just a logical assertion, in GF we have to assign an unique identifier (*Human_Class*) to it. In type theory this is deeply rooted in Curry—Howard's correspondence, but it is interesting that a similar kind of "labeling" of assertions is now emerging in OWL via the Named Graphs standard [20].

Semantically the *subclass* predicate in SUMO encodes the reflexive transitive closure of the taxonomic relation, while the immediate subclass relation is encoded using the predicate *immediateSubclass*. To take this into account we define one more type:

$$\textbf{cat}\ \mathit{Inherits}\ \mathit{Class}\ \mathit{Class};$$

Strictly speaking the *SubClass* type is the translation of the predicate *immediateSubclass* and *Inherits* is the translation of *subclass*. However we choose to read simple *subclass* axioms such as (1) as assertions for immediate subclassing and thus the conversion tool will generate the *SubClass* type in GF. The reason for this is that this would let us do some reasoning with the ontology by using only the tools that are already available in GF. Our intuition is that this still preserves the principal information from SUMO.

From the atomic *SubClass* axioms we can easily infer the reflexive-transitive closure *Inherits*. All that is needed is to add two inference rules. The inference rules in type theory are nothing else but functions with some specific type signatures:

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

The type of function *inhz* states that every class c inherits itself, i.e. this is the reflexivity axiom. The second function *inhs* expresses the transitivity over *SubClass*, i.e. if c_1 is a subclass of c_2, and c_2 inherits c_3 then c_1 inherits c_3.

The inference rules can be applied using the inference engine built into GF. For example, from the GF shell the user can use the `gt` command to generate an expression of a given type:

```
SUMO> gt -cat="Inherits Human Hominid"
(inhs Human Hominid Hominid Human_Class (inhz Hominid))
```

In type theory the types are seen as logical propositions and the existence of a value of a given type is interpreted as an evidence for the validity of the proposition. The value is also a constructive recipe for building the proof from the axioms in the theory. In Section 2.4 we will use it to generate explanations in natural language for the proofs.

Some of the classes in SUMO have two or more superclasses. For instance *Human* is both a *CognitiveAgent* and a *Hominid*. In other situations it is necessary to quantify over instances of the union of two or more classes. For that purpose we added two of the primitive operations from description logic – intersection and union of classes:

$$\begin{aligned} &\textbf{fun}\ both : Class \to Class \to Class; && \text{– intersection} \\ &\quad either : Class \to Class \to Class; && \text{– union} \end{aligned}$$

With the help of these primitives, the full definition of the class *Human* is:

$$\textbf{fun}\ Human\_Class : SubClass\ Human\ (both\ CognitiveAgent\ Hominid); \qquad (3)$$

The reasoning with these two new primitives can be axiomatized with three new inference rules:

$$\begin{aligned} &\textbf{fun}\ bothL : (c_1, c_2 : Class) \to SubClass\ (both\ c_1\ c_2)\ c_1; \\ &\quad bothR : (c_1, c_2 : Class) \to SubClass\ (both\ c_1\ c_2)\ c_2; \\ &\quad eitherC : (c_1, c_2, c_3 : Class) \to \\ &\qquad SubClass\ c_1\ c_3 \to SubClass\ c_2\ c_3 \to SubClass\ (either\ c_1\ c_2)\ c_3; \end{aligned}$$

The first two state that the intersection class of any two classes c_1 and c_2 is a subclass of both c_1 (function *bothL*) and c_2 (function *bothR*). The third function (*eitherC*) states that if two classes c_1 and c_2 are both subclasses of c_3, then their union class is also a subclass of c_3. Now, with the extended definition for *Human* (3), the proof that every Human is a kind of Hominid will use the function *bothR*:

```
SUMO> gt -cat="Inherits Human Hominid"
(inhs Human (both CognitiveAgent Hominid) Hominid Human_Class
(inhs (both CognitiveAgent Hominid) Hominid Hominid
(bothR CognitiveAgent Hominid) (inhz Hominid)))
```

At least in some cases, the criterion which distinguishes the members of a given class from the super class is formally specified. In this case the criterion is specified in SUMO as an axiom. In our encoding we found it handy to use an encoding which uses the

KappaFn function. *KappaFn* is a function in SUMO which takes a logical formula and returns the class of all instances for which the formula is valid. The type of the function in GF is:

$$\textbf{fun}\ KappaFn : (c : Class) \to (\,Var\ c \to Formula) \to Class; \tag{4}$$

It takes as arguments the superclass c and the logical formula and returns the subclass. The type $(\,Var\ c \to Formula)$ indicates that the argument itself is a function which takes a variable of class c and returns a formula. Every instance of c for which the formula is true is also a member of the new subclass. Using *KappaFn* it is trivial to define the class *NegativeRealNumber* as a subclass of *RealNumber*:

$$\begin{aligned}&\textbf{fun}\ NegativeRealNumber : Class;\\ &\textbf{def}\ NegativeRealNumber = KappaFn\ RealNumber\ (\backslash N \to lessThan\ \ldots);\end{aligned}$$

Again for the inference of the transitive closure to work we need an inference rule:

$$\begin{aligned}\textbf{fun}\ kappa : (c : Class) \to (p : Var\ c \to Formula) \to\\ SubClass\ (KappaFn\ c\ p)\ c;\end{aligned}$$

which defines the semantics of *KappaFn*, i.e. that the new class is a subclass of the argument of the function.

2.2 Instances

Once we have the taxonomy of the ontology we can proceed with adding some instances. Similarly with the classes we will distinguish between direct instances of a class and generalized instances. The instances will be defined as values of one of the following types:

$$\begin{aligned}\textbf{cat}\ &El\ Class;\\ &Ind\ Class;\end{aligned}$$

The type $Ind\ c$ is assigned to all instances with principal class c, while $El\ c$ is the type of all direct instances of c together with the instances of its subclasses. There is an injection between this two types:

$$\textbf{fun}\ el : (c_1, c_2 : Class) \to Inherits\ c_1\ c_2 \to Ind\ c_1 \to El\ c_2;$$

The function *el* injects an instance with principal class c_1 into the type of the generalized instances of c_2, if there is an evidence that c_1 is a subclass of c_2 (the argument $Inherits\ c_1\ c_2$). For example in the CountriesAndRegions module of SUMO there is an instance for the city of London:

$$\textbf{fun}\ LondonUnitedKingdom : Ind\ EuropeanCity;$$

The class *EuropeanCity* is a subclass of *City* so it is possible to do the coercion. The following expression is the injection of *LondonUnitedKingdom* into the generalized instances of *City*:

$$\begin{array}{l} el\ EuropeanCity\ City \\ \quad (inhs\ EuropeanCity\ City\ City\ EuropeanCity\_Class\ (inhz\ City)) \\ \quad LondonUnitedKingdom \end{array}$$

2.3 Functions, Predicates and Logical Formulas

In SUMO, all functions and predicates are represented as instances of a descendant of *Relation*, and the expected classes of the arguments and the result are stated as axioms in the ontology. For example the definition of the *AbsoluteValueFn* function is:

$$\begin{array}{l} (instance\ AbsoluteValueFn\ UnaryFunction) \\ (domain\ AbsoluteValueFn\ 1\ RealNumber) \\ (range\ AbsoluteValueFn\ NonnegativeRealNumber) \end{array}$$

Here the predicates *domain* and *range* specify the class of the first argument and the class of the returned value. The class of *AbsoluteValueFn* itself is *UnaryFunction* which encodes the fact that this is a function with only one argument. From this SUMO axioms we generate a type signature in GF:

$$\textbf{fun}\ AbsoluteValueFn : El\ RealNumber \to Ind\ NonnegativeRealNumber;$$

Note that with our implementation we impose the closed world assumption. The argument of *AbsoluteValueFn* is declared of type $El\ RealNumber$, and the only way to construct a value of that type is to combine an instance of some subclass c of *RealNumber* with a proof object of type:

$$Inherits\ c\ RealNumber$$

If this object cannot be constructed from the current state of the knowledge base then the application of *AbsoluteValueFn* is not possible.

The predicates are declared in a way very similar to the functions. The only difference is that while the functions return some instance, the predicates are used to create logical formulas. In the original ontology, there is already a class called *Formula* which represents the class of all well-formed SUO-KIF formulas. In principle the predicates could return $Ind\ Formula$ but there are two reasons for which we choose not to do that. The first reason is that if *Formula* is kept as a class then this would allow quantification over logical formulas which is not supported in first-order logic. The second reason is that when the logical axioms are translated to natural language then *Formula* will correspond syntactically to a sentence while *Ind* corresponds to a noun phrase, and this would make the verbalization of the ontology difficult. Instead we declared *Formula* as a type:

$$\textbf{cat}\ Formula;$$

The last piece that is needed to be able to write logical axioms in GF is to add the standard logical quantifiers and connectives:

$$\begin{aligned}
&\textbf{cat } \mathit{Var}\ \mathit{Class};\\
&\textbf{fun } \mathit{var} : (c_1, c_2 : \mathit{Class}) \to \mathit{Inherits}\ c_1\ c_2 \to \mathit{Var}\ c_1 \to \mathit{El}\ c_2;\\
&\\
&\textbf{fun } \mathit{exists} : (c : \mathit{Class}) \to (\mathit{Var}\ c \to \mathit{Formula}) \to \mathit{Formula};\\
&\quad\ \mathit{forall} : (c : \mathit{Class}) \to (\mathit{Var}\ c \to \mathit{Formula}) \to \mathit{Formula};\\
&\textbf{fun } \mathit{not} : \mathit{Formula} \to \mathit{Formula};\\
&\quad\ \mathit{and}, \mathit{or}, \mathit{impl}, \mathit{equiv} : \mathit{Formula} \to \mathit{Formula} \to \mathit{Formula};
\end{aligned}$$

The only specific thing here is how the variables are introduced by the quantifiers. The first argument of the quantifier (function *exists* or *forall*) is the class over which the function quantifies. The second argument is the formula over which it scopes. The quantified variable itself is a high-order argument of type $\mathit{Var}\ c$. This type plays a role similar to the role of El. While the former denotes some known instance, for Var we neither know the instance, nor its principal class. This is reflected for example in natural language generation where the grammatical gender is deduced from the class of the variable instead of the instance itself. This special treatment of variables allows the generation of more fluent natural language. Still the *var* function allows the coercion from type *Var* to *El*.

With the usage of quantifiers and connectives all axioms from SUMO, which were not already converted to type signatures in GF, can be converted to abstract syntax trees. For example the SUO-KIF formula:

$$\begin{aligned}
&(=> (\mathit{instance}\ ?P\ \mathit{Wading})\\
&\quad (\mathit{exists}\ (?W)\ (\mathit{and}\ (\mathit{instance}\ ?W\ \mathit{BodyOfWater})\ (\mathit{located}\ ?P\ ?W))))
\end{aligned}$$

is converted to the following abstract syntax tree in GF:

$$\mathit{forall}\ \mathit{Wading}\ (\backslash P \to \mathit{exists}\ \mathit{BodyOfWater}\ (\backslash W \to \quad \mathit{located}\ (\mathit{var}\ P)\ (\mathit{var}\ W)))$$

Note that this is more than just a syntactic conversion because the quantifiers in GF expect explicit class information while in SUMO this is encoded with *instance* predicates.

2.4 Proofs in Natural Language

As it was mentioned in Section 2.1, the proofs in GF are explicitly represented as abstract syntax trees. Since the abstract syntax trees could also have linearizations in the concrete syntax, it is possible to render the proofs in the same controlled natural language encoding the ontology. For example the following command in the GF shell:

```
SUMO> gt -cat="Inherits Human Primate" | l -lang=SUMOEng
```

will derive a proof for $\mathit{Inherits}\ \mathit{Human}\ \mathit{Primate}$ and will linearize the proof in English. The text contains some HTML tags, so when it is rendered in a web browser it looks like a bullet list:

- *human is a subclass of both cognitive agent and hominid*
- *hominid is a subclass of primate*

The natural language rendering can be used to generate end-user explanations for the inferences in the ontology.

3 Russell's Paradox

Russell's paradox [21] was first discovered in naïve set theory. It stems from the assumption that for every logical proposition there is a set of entities which satisfy the proposition. This was shown to be inconsistent with the example of the set of all sets which are not members of themselves. Such a set cannot exist because then it will be simultaneously a member and not a member of itself. The design of SUMO follows naïve set theory and the *KappaFn* function is exactly the way to build sets from propositions. Using the function, the paradox can be expressed as:

$$(instance\ (KappaFn\ "x"\ (not\ (instance\ x\ x)))\ (KappaFn\ "x"\ (not\ (instance\ x\ x))))$$

The reasoning with SUMO is sound only because the *KappaFn* function is not axiomatised and the automated theorem provers cannot make any inferences.

The paradox is principally avoided in the GF translation by first discarding the predicate *instance* and second by making the class *Class* into a type. This results into a completely different signature for KappaFn (4) which would make the above statement incorrect even if we still had the predicate *instance*.

4 Verbalization

Apart from the advantages that the GF type system provides, for the natural language generation the benefits of using GF are considerably more substantial. The present work deals with the generation of natural language for the two upper ontologies - *Merge* and *Mid-level-ontology* in 3 languages: English, Romanian and French, as a proof-of-concept for the capabilities of GF to host a controlled language for ontologies.

For English, about 7,000 concepts and relations have been translated to natural language. For Romanian and French, only a small number of examples, that illustrate the advantages of GF over a template-based generation, were built. This is due to the fact that there are no large coverage lexicons for those languages in GF yet.

A typical SUMO template is the predicate `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 structure of the templates is rather simple, and works reasonably just for morphologically 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 or the various moods of a sentence, depending on its usage.

This solution is not compositional and leads to incorrect constructions in languages with a rich morphology such as Romanian. For example the verbalization of "the inverse of the square root of X" in Romanian would require the combination of two templates and would render: *inversa lui rădăcina pătrată a lui X*, which is considerably different from the correct form - *inversa rădăcinii pătrate a lui X*. One reason is that the translation of "square root" should be in Genitive case, whereas the template only has the Nominative one, and in Romanian the two forms are different. The second is the matter of the possessive preposition, which in Romanian needs to agree with its object. The template provides the masculine form as default, but *rădăcina pătrată a lui X* is feminine. 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, which cannot be handled by the SUMO templates. In addition to this, for French there is also the problem of phonetic mutations, such as for the usage of a verb with negative polarity. In case that the verb starts with a vowel, the form of the particles used to express negation changes, and this is a mutation that SUMO doesn't handle, because the templates provide only one value for the particles. It goes without saying that the French and Romanian resource grammars offer solutions for these problems, so that the natural language generation in SUMO-GF is syntactically correct for compositions of patterns also.

Moreover, 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 languages that have gender agreement [22]. This is a very common feature for Romance and Slavic languages, where there is a gender differentiation. The SUMO templates simply assume that all the variables have masculine gender, while in GF, the wrapper function `var`, that has access to the class of the variable also, would assign a proper gender to the variable. Since variables can only 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 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 accidentally change the gender of an ordinary instance.

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

$$forall\ Animal\ (\backslash A \rightarrow exists\ Animal\ (\backslash B \rightarrow smaller\ (var B)\ (var A)))$$

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 type of feminine gender, such as house we would have that:

$$forall\ House\ (\backslash A \rightarrow exists\ House\ (\backslash B \rightarrow smaller\ (var B)\ (var A)))$$

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 axioms are not taken from SUMO, but they are just two examples that illustrate 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 examples, although few, show the advantages of GF in developing a set of multilingual aligned syntactically-correct controlled languages for describing ontologies.

Besides axioms, we can also generate natural language for `SubClass`, `Ind` declarations and higher-order functions. For example:

beverage is a subclass of food
blue is an instance of primary color
"x is equal to y" is an equivalence relation

Our work provides natural language generation in English for the two biggest modules *Merge* and *Mid-level-ontology* and two domain specific: *Elements* - featuring chemical substances and *Mondial* - featuring countries and cities of the world. As mentioned before, a total of almost 7 000 objects and 500 relations from SUMO were verbalized. This process is done automatically for objects and semi-automatically for relations, and uses the GF resource grammar.

The automatic process takes advantage of the camel case representation of SUMO concepts. For example, *BodyOfWater* will be rendered as ”body of water” and parsed by GF as a noun phrase. Instances are parsed as GF noun phrases, while classes are parsed as GF common nouns, which are similar to noun phrases, only that they have variable number, gender and other morphological features. In this way, the representation of *BodyOfWater* will also contain the plural form "bodies of water", which we can use for generating natural language constructions. For functions and predicates the missing arguments are replaced by some dummy variables and the procedure is semi-automatical, using the original SUMO templates and hand-written verbalizations which are further on parsed as noun phrases for functions and clauses with polarity for predicates. For example, the binary predicate `parent` will be verbalized as "o1 is the parent of o2" and parsed to a GF abstract syntax tree. This method allows generalizations, so the 2 negative forms are "o1 is not the parent of o2" and "o1 isn’t the parent of o2" are automatically obtained from this. For the two domain specific ontologies, the information is extracted from the SUMO predicate `name` that gives the English verbalization of the concepts. As a result, our approach renders verbalization of a large number of entries from the ontology, with a high rate of automation, ensuring syntactical correctness of the generated phrases. For example :

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.

For the same axiom, the SUMO templates generate:

For all unique list ?LIST holds for all ?NUMBER1, ?NUMBER2 holds if ?NUMBER1th element of ?LIST is equal to ?NUMBER2th element of ?LIST, then ?NUMBER1 is equal to ?NUMBER2

The optimized natural language generation mechanism from the Sigma system would render the axiom as:

* *If a list is an instance of unique list*
* *then for all a positive integer and positive integer*
  - *if positive integerth element of list is equal to positive integerth element of list*
  - *then positive integer is equal to positive integer*

Further optimizing of the code by anaphora generation and a list-like structure of the arguments for better readibility is possible, like in the proof rendering.

5 Evaluation

During the translation of SUMO to GF, we discovered a number of small inconsistencies in the original ontologies like mismatches between instances and classes, usage of undefined objects and usage of functions with a wrong number of arguments. This represents almost 8% of the total number of axioms from SUMO and was determined automatically during the type-checking phase. In addition to this, we left out the higher-order logic constructions such as quantifications on `Formula` or axioms with higher-order functions.

However, there are some types of axioms which could not be ported to SUMO-GF, such as the ones that use quantification over classes, negative type declarations and axioms which use the predicates `subclass`, `range` or `domain`. In addition to this, we mention the class of axioms which feature conditional type declarations. For example:

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

The type declaration for *BEV* appears as a consequence of the fact that it is used in the process of *Drinking*. The total number of axioms which are lost in translation is about 23%. Our observations suggests that those axioms could be paraphrased and incorporated in the type system but this would require manual work with every axiom.

6 Automated Reasoning in SUMO-GF

Since SUMO offers a generous amount of information in a first-order logic format, it represents an excellent source for automated reasoning, especially in the context of SUMO-GF where one can perform automated reasoning on natural language.

As mentioned before, the TPTP-FOF translations of the 2 upper SUMO ontologies are used yearly in the ATP competition. We have shown already in Section 2 that a limited kind of ontological reasoning is possible by using GF alone. Unfortunately, the reasoner in GF is not as optimized as current state of the art theorem provers. However, to take advantage of the tools that already exists, we translated the 17 SUMO-GF ontologies to TPTP-FOF, checked them for consistency and used them for solving small inferences.

Since TPTP is an untyped system, whereas GF is strongly typed, the information about types needs to be reformulated, with the aid of an additional predicate `hasType`, that resembles the original `instance` predicate from SUMO.

For subclasses, the translation reflects the possibility of coercing from the subclass to the superclass:

$$\textbf{fun}\ \mathit{Adjective\_Class} : \mathit{SubClass\ Adjective\ Word};$$

and would be translated to TPTP as:

```
fof(axMerge2, axiom, (![X]:
    (hasType(type_Adjective, X)=>hasType(type_Word, X)))).
```

For instance declarations, we have a simpler translation pattern:

$$\textbf{fun}\ \mathit{Flat} : \mathit{Ind\ ShapeAttribute};$$

will be translated to TPTP as:

```
fof(axMerge686, axiom,
     hasType(type_ShapeAttribute, inst_Flat)).
```

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

```
type_ShapeAttribute (inst_Flat)
```

We did not choose this method, since the SUMO classes are not just used as types, in typing declarations, but also as arguments for some functions. By using classes as predicates, one couldn't unify the two occurrences in first-order logic.

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: $\sim$, $\&$, $|$ and $\Rightarrow$. For the *both* and *either* constructors, the built-in $\&$ and $|$ are used again:

$$\textbf{fun}\ \mathit{Togo} : \mathit{Ind}\ (\mathit{both\ Country\ Nation});$$

will be translated to TPTP as:

```
fof(axmond72, axiom,
     hasType(type_Country, inst_Togo) &
               hasType(type_Nation, inst_Togo)).
```

As for 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:

$$\begin{array}{l}(=>\ (\mathit{equal}\ ?\mathit{THING1}\ ?\mathit{THING2})\\ \quad(\mathit{forall}\ (?\mathit{CLASS})\\ \qquad(<=>\ (\mathit{instance}\ ?\mathit{THING1}\ ?\mathit{CLASS})\\ \qquad\quad(\mathit{instance}\ ?\mathit{THING2}\ ?\mathit{CLASS}))))\end{array}$$

could not 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 specific mechanisms for dealing with the built-in equality from TPTP [23]. 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` for consistency with the type declarations. For example, the axiom (**??**) was translated to TPTP as:

```
fof(axMid9, axiom, ![Var_P]:
 hasType(type_Wading, Var_P) => ?[Var_W]:
 hasType(type_BodyOfWater, Var_W) & f_located(Var_P,Var_W)).
```

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. In this way the function name is used as a macro for its body. This is the same approach as in [14].

The resulting files have been checked with the first-order theorem prover**E** [24]. **E** is a multiple award-winning theorem prover which is freely available and is based on an 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 [25]. 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, which was exceeded for the upper-ontologies, due to the increased complexity of the axioms they contain.

Regarding typical inferences that could be solved on the existing data, we used the problems from the SUMO webpage [4].

The category of axioms that the SUMO to TPTP translation can express, but the SUMO-GF to TPTP cannot are mainly the ones that got lost in the SUMO to SUMO-GF translation. In addition to this, there are the nested predicates, quantifications over `Formula` and the class-forming function `KappaFn`. They are used in SUMO-GF only for language generation. The loss is almost 23% of the total number of the axioms. The coverage 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, it is worth investigating if the results could be better, if one chooses the typed version of TPTP. [5].

7 End-User Interface

An important component of the GF distribution is the front-end user interface. While the grammarians are supposed to use the GF shell plus some development environment

[4] `http://sigmakee.cvs.sourceforge.net/viewvc/sigmakee/KBs/tests/`
[5] `http://www.cs.miami.edu/$~$tptp/TPTP/Proposals/TypedFOF.html`

for writing grammars, end users should have the option to use some more comfortable interface. GF comes with a generic web-based interface [26] which could be specialized further for particular applications. In relation with SUMO, the interface was extended with features which make the relation of the ontology with the concrete syntax more transparent.

While in Sigma the knowledge engineers are supposed to write the axioms in KIF, in GF they could do it in a controlled natural language. The problem with all controlled languages is that the user has to learn how to write content which is in the scope of the grammar. The successful use of predictive editors in helping the users build constructions within the bounds of the controlled language was investigated in [27]. In GF there is a similar predictive editor (fig. 1) which guides the authoring by generation of suggestions. In the example shown the user has just started adding a reference to a variable and the editor suggests the list of all variables in the current scope which start with "NU". The same kinds of suggestions are offered for every word in the sentence. Furthermore, the user could at any time select a phrase (the highlighted phrase on the picture) and

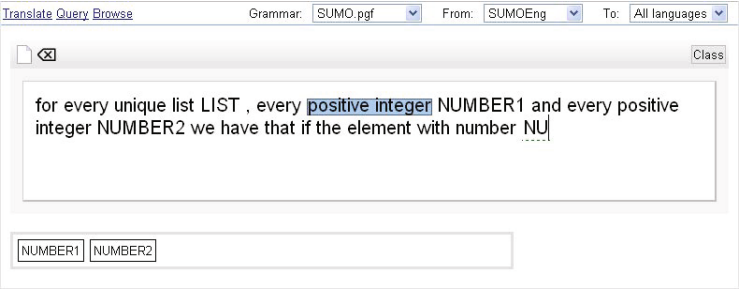

Fig. 1. Text editor for authoring SUMO axioms using controlled natural language

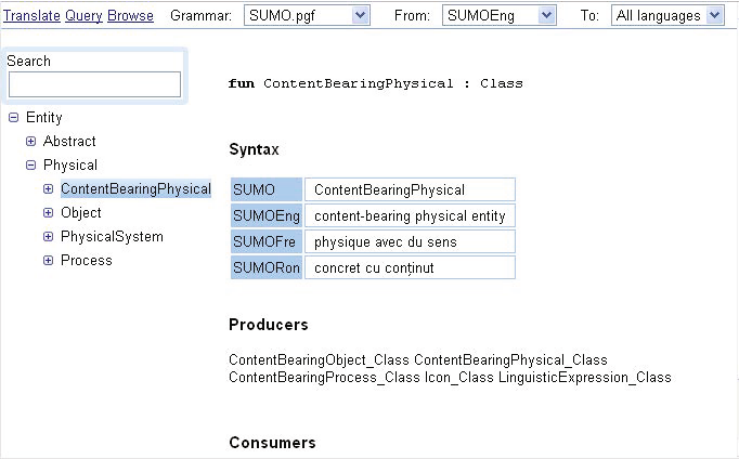

Fig. 2. Browser for combined ontology and syntax exploration

see in the upper-right corner the corresponding ontological type of the phrase (*Class* in this case). If the axiom is not well-formed, i.e. contains a type error for example, then the error is immediately reported and the corresponding phrase is underlined. This very much resembles an IDE for programming languages, except for the fact that the input is a kind of natural language.

For nontrivial ontologies of the scale of SUMO it is often helpful to have an overall view of the ontology. The browsing functionalities of Sigma very much fulfil the requirements. A similar browser (fig. 2) for the abstract syntax of the grammars was developed for GF. In the case of SUMO-GF, this corresponds exactly to the taxonomy plus the signatures of all functions and predicates. The user sees the class hierarchy on the left-hand side and can start with the exploration of any class, or could use the search box in the upper-left corner to find a class or function by name.

8 Related Work

At the moment 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 do not have a type system.

A first exception is the programming language prototype Zhi#[6], which is a novel language for encoding ontologies. It has a static type-system and it is compiled to C#. It benefits from using the C# built-in types and functions, but also the syntax looks very much like C# and it is not very intuitive for most users.

A more notable example is CASL (Common Algebraic Specification Language) [28]. It introduces the notions of strongly-typed and structured ontologies and provides a strong formal structure for representing them. However, it deals only with the algebraic side of the specifications, whereas GF has a built-in natural language generation component, in addition to the robust type system.

Compared to these languages, GF is the only system which combines a strongly typed framework for ontology description with a direct multilingual verbalization. To our knowledge, the current work is the first representation of an ontology in type theory 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. robustness to the representation.

Regarding natural language generation, there are many notable applications that verbalize ontologies. Most of them however, have only English as target. A notable exception is the KPML project [29], which provides natural language generation for 10 languages. Another interesting case is the Gellish ontology which provides direct verbalization for English, German and Dutch. However, there is considerably less progress for Slavic and Romance languages, due to their complexity. The GF approach has built-in mechanisms for verbalization via the resource grammars, which provide syntactically correct translations. Moreover, GF also has support for multilingual translation.

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

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

the ResearchCyC ontology [25]. However, SUMO is the most well-known case of an ontology which is checked for consistency every year, as part of a 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 [30] provides automated reasoning for the SUMO ontology with KRHyper [31]. KRHyper is a theorem prover for first-order logic that implements hyper tableaux, and a version of it that deals with equality is also available [7]. The tool can answer questions posed in normal English, by using the wordnet mappings and a simple parser, in order to infer the SUMO expression that should be checked.

9 Future Work

The current work explores aspects of data modelling, 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. These directions can be extended in a more comprehensive manner and lead to stand-alone projects.

One interesting possibility would be to generate higher-quality natural language, following the idea[8] of truncating the hierarchy even 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* 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*.

Another interesting application would be to build a user interface, where users could ask questions and get answers from the theorem prover via the GF to TPTP translation. If the prover provides a trace of the proof search, this could be converted back to a GF tree and used for generation of proof explanations in natural language. When dealing with more complex proofs, more work is needed for rendering readable natural language. A comprehensive reference for natural language generation from proofs is [32].

10 Conclusion

The work investigates the representation of upper ontologies in the type-theoretical functional language GF, which provides mechanisms for direct verbalization as a controlled language, having the SUMO ontology as a use case. The results obtained show

[7] `http://www.uni-koblenz.de/$~$bpelzer/ekrhyper/`

[8] `http://www.ontologyportal.org/student.html`

a consistent improvement from the multilingual natural language generation point of view, in terms of effort, scalability and syntactical correctness of the obtained text. Moreover, the type system, while still preserving a comparable coverage, prevents type errors that could lead to inconsistencies in the ontology. Also the editor makes it easier for users to interact with the ontology by adding content in natural language. From a knowledge engineering point of view, GF offers obvious advantages for encoding ontologies, as the framework defined and applied for SUMO is general enough to fit a large range of ontologies.

References

1. Ganesereth, M.R., Fikes, R.E.: Knowledge interchange format. Technical Report Logic-92-1, Stanford University (June 1992)
2. Bechhofer, S., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D.L., Patel-Schneider, P.F., Stein, L.A.: OWL web ontology language reference (2009)
3. Cycorp: The syntax of CycL (2002)
4. Van Renssen, A.: Gellish: A Generic Extensible Ontological Language. PhD thesis, Delft University, PhD thesis (2005)
5. Niles, I., Pease, A.: Towards a standard upper ontology. In: FOIS 2001: Proceedings of the International Conference on Formal Ontology in Information Systems, pp. 2–9. ACM, New York (2001)
6. Ranta, A.: Grammatical Framework: A Type-Theoretical Grammar Formalism. The Journal of Functional Programming 14(2), 145–189 (2004)
7. Martin-Löf, P.: Constructive mathematics and computer programming. In: Cohen, Los, Pfeiffer, Podewski (eds.) Logic, Methodology and Philosophy of Science VI, pp. 153–175. North-Holland, Amsterdam (1982)
8. Ranta, A.: The GF resource grammar library. Linguistic Issues in Language Technology 2(2) (2009)
9. Angelov, K., Bringert, B., Ranta, A.: PGF: A Portable Run-Time Format for Type-Theoretical Grammars. Journal of Logic, Language and Information (2009)
10. Ljunglöf, P., Amores, G., Cooper, R., Hjelm, D., Lemon, O., Manchin, P., Perez, G., Ranta, A.: Multimodal Grammar Library, TALK. Talk and Look: Tools for Ambient Linguistic Knowledge. IST-507802. Deliverable 1.2b (2006)
11. Caprotti, O.: WebALT! Deliver Mathematics Everywhere. In: Proceedings of SITE 2006, Orlando, March 20-24 (2006)
12. Ahrendt, W., Baar, T., Beckert, B., Bubel, R., Giese, M., Hähnle, R., Menzel, W., Mostowski, W., Roth, A., Schlager, S., Schmitt, P.H.: The key tool. Technical report in computing science no. 2003-5, Department of Computing Science, Chalmers University and Göteborg University, Göteborg, Sweden (2003)
13. MOLTO - Multilingual Online Translation. European Project (2010-2012)
14. Pease, A., Sutcliffe, G.: First order reasoning on a large ontology. In: Proceedings of the CADE-21 Workshop on Empirically Successful Automated Reasoning on Large Theories, ESARLT (2007)
15. Sutcliffe, G.: The TPTP problem library and associated infrastructure. Journal of Automated Reasoning 43, 337–362 (2009) 10.1007/s10817-009-9143-8
16. Pease, A.: The sigma ontology development environment. Working Notes of the IJCAI 2003 Workshop on Ontology and Distributed Systems, vol. 71 (2003)

17. Trac, S., Sutcliffe, G., Pease, A.: Integration of the tptpworld into sigmakee. In: Proceedings of IJCAR 2008 Workshop on Practical Aspects of Automated Reasoning (PAAR 2008), vol. 373 (2009)
18. Martin-Löf, P.: A theory of types (1971) (unpublished)
19. Girard, J.Y.: Interpretation fonctionnelle et elimination des coupures de l'arithmetique d'ordre superieur, Paris (1972)
20. W3C: Named graphs (2004)
21. Russell, B.: Principles of Mathematics. Cambridge University Press, Cambridge (2011)
22. Ranta, A.: Structures grammaticales dans le français mathématique. Mathématiques, informatique et Sciences Humaines 138, 139, 5–56, 5–36 (1997)
23. Slagle, J.R.: Automatic theorem proving with built-in theories including equality, partial ordering, and sets. J. ACM 19, 120–135 (1972)
24. Schulz, S.: E - a brainiac theorem prover (2002)
25. Ramachandran, D., Reagan, P., Goolsbey, K.: First-orderized researchcyc: Expressivity and efficiency in a common-sense ontology. Papers from the AAAI Workshop on Contexts and Ontologies: Theory, Practice and Applications (2005)
26. Bringert, B., Angelov, K., Ranta, A.: Grammatical framework web service. In: EACL (Demos), 9–12 (2009)
27. Schwitter, R., Ljungberg, A., Hood, D.: ECOLE — a look-ahead editor for a controlled language. In: Proceedings of EAMT-CLAW 2003, pp. 141–150 (2003)
28. Lüttich, K.: Development of Structured Ontologies in CASL. PhD thesis, University of Bremen, PhD thesis (2007)
29. Bateman, J.A.: Enabling technology for multilingual natural language generation: the kpml development environment. Nat. Lang. Eng. 3(1), 15–55 (1997)
30. Baumgartner, P., Suchanek, F.M.: Automated reasoning support for sumo/kif (2005); (manuscript, Max-Planck Institute for Computer Science)
31. Wernhard, C.: System Description: KRHyper. Fachberichte Informatik 14-2003 (2003)
32. Fiedler, A.: Natural Language Proof Explanation. In: Hutter, D., Stephan, W. (eds.) Mechanizing Mathematical Reasoning. LNCS (LNAI), vol. 2605, pp. 342–363. Springer, Heidelberg (2005)