GFMed: Question Answering over BioMedical Linked Data with Grammatical Framework

Anca Mărginean

Technical University of Cluj Napoca, 401446 Cluj-Napoca, Romania,
anca.marginean@cs.utcluj.ro

Abstract. This paper reports on the participation of the system GF-Med in QALD-4 question answering challenge for Biomedical interlinked data. GFMed introduces grammars for a controlled natural language targeted towards biomedical information and the corresponding controlled SPARQL language. The grammars are described in Grammatical Framework and introduce linguistic and SPARQL phrases mostly about drugs, diseases and relationships between them.

1 Introduction

Linked Data means using the Web to connect related data. A large number of data from various domains such as government data, medicine, education, life sciences, literature, art and others were made available in the context of the Linked Open Data (LOD) initiative built around *DBpedia*.

One of the greatest challenges of this new big set of data is querying it. In order to fill the gap between end users and formal languages like SPARQL more approaches emerged: querying in full natural language [8], in Controlled Natural Languages [4], or incremental query building [9].

The medical domain excels in the quantity of existing data, and lately, there is a large interest in making biomedical data available in RDF format (Bio2RDF project [2]). Task 2, *Biomedical question answering over linked data*, of the *Question Answering over Linked Data* (QALD-4) lab, proposes querying of the following three biomedical datasets: DrugBank, Diseasome and SIDER. DrugBank is part of the project Bio2RDF, while the other two are not. DrugBank gives drug (chemical, pharmacological and pharmaceutical) data with comprehensive drug target (sequence, structure, and pathway) information. Diseasome provides information about human disease-gene network, while SIDER relates drugs to their adverse reactions. The Linked Data version of Diseasome publishes a network of 4300 disorders and disease genes, as well as possible drugs for diseases. SIDER includes 4192 side effects, 996 drugs and 99423 drug/side-effect pairs.

The objective of Task2 of QALD-4 is to search solutions for querying these three datasets considering also their integration. The key challenge, as stated by the organizers, is to translate the users' information needs into a form such that they can be evaluated using standard Semantic Web query processing and inferencing techniques. The current system proposes a natural language query

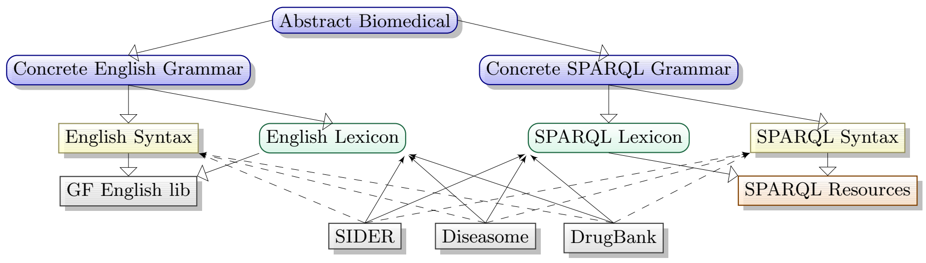

Fig. 1. Main GFMed grammars and used resources

approach based on translation from a controlled natural language to SPARQL. The translation relies on grammars defined in Grammatical Framework [7]. Our main objective was to test the semantic coverage of our application domain grammar against the questions proposed in Task 2 of QALD-4.

2 System Overview

Grammatical Framework (GF) is a special-purpose programming language for writing grammars based on a typed functional programming language. GF grammars are divided into abstract and concrete grammars. An abstract grammar defines categories and functions. Each category stands for a set of trees. Functions produce trees of certain categories. The linearization types and functions are defined in concrete grammars. For each category, a linearization type is needed and each function requests a linearization function. Based on the abstract grammar and the concrete grammars for each language, GF is able to translate a phrase from one language to another by parsing it first into an abstract tree and then linearizing it by means of the concrete grammars.

Description Logics (DLs) [1] are a family of knowledge representation languages that can be used to represent the knowledge of an application domain in a structured and formally well-understood way. In the description logic $\mathcal{ALC}$, concepts are built using the set of constructors formed by negation, conjunction, disjunction, value restriction, and existential restriction. Extensions of $\mathcal{ALC}$ introduce inverse roles, number restrictions ($\mathcal{N}$, $\mathcal{Q}$) and concrete domains($\mathcal{O}$). Even though the three targeted datasets are not all providing for DLs descriptions or ontologies, approaching them from a DLs perspective indicates ways to efficiently split possible questions in semantic parts that have straightforward translations to SPARQL and are highly composable.

GFMed consists mainly of a GF grammar for the application domain given by SIDER, Diseasome and DrugBank datasets. GFMed also includes some minor preprocessing of questions and postprocessing of translation results, mainly in order to deal with structures involving numeric values, e.g. values for water solubility, or free text, like different names of foods. Figure 1 shows the main grammars: an abstract grammar and two concrete grammars. For each concrete

Table 1. Main categories of GFMed grammars

Category	English Category	Examples and Short Explanation
$\mathcal{X}$	NP	$\mathcal{X} \in \{Drug, TargetConcept, Gene, Disease, SideEffect, SiderDrug\}$
DrugBank-Property	CN	MeltingPoint, GeneralFunction, DosageForm, PredictedWaterSolubility, Manufacturers, Indication, Target, Interacting, FoodInteraction,...
Sider-Property	CN	SideEffect
Diseasome-Property	CN	AssociatedGene, PossibleDrug, ClassDegree, Degree, Class, Size, SubtypeOf, ChromosomalLocation
Property	CN	any kind of property from the above three
$\mathcal{X}Class$	NP	classes formed of a single named $\mathcal{X}$ entity: *Lepirudin*, or from drugs described by a criterion: *drugs that target Prothrombin* DrugClass, TargetClass, SideEffectClass, SiderDrugClass, DiseaseClass, GeneClass, PropertyClass
Criterion-ForXClassY		criteria for getting a class of X, expressed by an Y syntactic structure
	NP	*Lepirudin as possible drug*
	Adv	*with Lepirudin as possible drug*
	AP	*treated with Lepirudin, indicated for Fever*
	VP	*treat Tuberculosis*
	RCl	*whose possible drug is Lepirudin, whose possible drugs interact with Lepirudin*
	ClSlash	*Lepirudin is used for*
Question	QS	*which drugs interact with food*
Utterance	Utt	utterances from affirmative clauses *List the drugs that...* or from question clauses *What are the drugs that ...*

grammar, lexicons derived from SIDER, Diseasome and DrugBank were generated. Many syntactic structures in both English and SPARQL grammars were driven by the datatsets' terminology.

For domain-specific applications, the GF abstract grammar must state the main semantic categories and trees of the language. For GFMed we introduced the following categories: Drug, Target, Disease, Gene, SideEffect and SIDER Drug corresponding to the main resources in the targeted datatsets. We also introduced the categories DrugBankProperty, DiseasomeProperty and SIDER-Property for describing the properties of the same datatsets. For each mentioned resource category, classes of these semantic entities are described, resulting DrugClass, DiseaseClass, GeneClass, TargetClass and SideEffectClass. Trees for these categories are built either from a single named resource, or from a restriction on a property. For each $\mathcal{X}$Class there are one or more CriterionFor$\mathcal{X}$Class categories, where the class can be obtained from the Criteria for that class. For example, *drugs that interact with food* is a DrugClass tree, while *interact with food* is a CriterionForDrugClass. In other words, trees of type CriterionFor$\mathcal{X}$Class are subtrees of $\mathcal{X}$Class trees. Table 1 depicts the main categories together with their English linearization category and some examples or explanations.

The core of the abstract grammar consists of functions to build trees. GF-Med's functions can be categorized in (i) functions that describe *property restrictions*, (ii) functions for *transforming* a criterion of a class into a class or for transformations between different types of classes and properties, (iii) functions expressing queries.

2.1 Building Trees for Property Restrictions

In DLs, there are two types of roles or properties: object properties and datatype properties. Object properties relate two concepts, while datatype properties relate one concept to a value of a certain datatype. For example, the object property *SideEffect* connects the resources *PenicillinG* and *Fever*. Differently, the *Mechanism of Action* property relates the drug *Lepirudin* to a string value.

Object properties and datatype properties are treated differently in the GF-Med grammar. When it comes to object properties, the DL *existential restriction* $\exists R.C$ on property R describes the set of individuals having as value of the property (role) R an individual from the concept C. For example, $\exists$ *PossibleDrug.FeverInducingDrug* is a restriction on property $PossibleDrug$ whose interpretation, if it exists, is the set of all diseases that have at least one possible drug from the class *FeverInducingDrug*. $FeverInducingDrug$ stands for all drugs that have fever as a side effect and it is a *value restriction*, with value $Fever$, on the property $SideEffect$. We believe that even though the targeted datasets are not described in DL, in order to have language constructors able to be composed based on their type, the functions to build trees in either English or SPARQL can be described similarly to DL constructors. This approach is also justified by the functional style of the chosen grammatical framework.

Each DL constructor can be expressed in natural language in more ways, either as noun phrase (NP), verbal phrase (VP), adjectival phrase (AP), verb-phrase-modifying adverb (Adv), relative clause (RCl) or clause with some missing part (ClSlash). These syntactic categories are defined by GF library. To each DL constructor identified at a conceptual level correspond more functions to build trees at concrete English level, one for each possible syntactic structure. Figure 2 shows functions that model restriction on the property $PossibleDrug$ with values in $DrugClass$. In a similar manner, functions for restriction on the inverse property of $PossibleDrug$ are defined. They allow statements about drugs used to treat a certain disease or a disease class. For all object properties, the abstract and concrete grammars include sets of functions to express existential and value restrictions on them. Since classese formed from only one named drug are allowed, value restrictions on object properties can be treated in the same way as existential restrictions.

When it comes to restrictions on datatype properties, the English methods to express them are not anymore particular to each property, therefore it is possible to treat all with the same set of functions. Some examples are described in Fig. 3.

$WithPossibleDrug : DrugClass \rightarrow DiseaseClass$;- - diseases treated with
$WithPossibleDrugCriterion : DrugClass$
 $\rightarrow CriterionForDiseaseClass$;- - treated with Lepirudin;
$WithPossibleDrugCriteriaClSlash : DrugClass$
 $\rightarrow CriterionForDiseaseClassClSlash$;- - Lepirudin is used for
$WithPossibleDrugCriteriaNP : DrugClass$
 $\rightarrow CriterionForDiseaseClassNP$;- - Lepirudin as possible drug
$WithPossibleDrugCriteriaAdv : DrugClass$
 $\rightarrow CriterionForDiseaseClassAdv$;- - with Lepirudin as possible drug
$WithPossibleDrugCriteriaRCl : DrugClass$
 $\rightarrow CriterionForDiseaseClassRCl$;- - whose possible drug is Lepirudin
$WithPossibleDrugCriteriaRCl\_VP : CriterionForDrugClassVP$
 $\rightarrow CriterionForDiseaseClassRCl$;- - whose possible drug interacts with
$WithPossibleDrugCriteriaRCl\_Adj : CriterionForDrugClassAdj$
 $\rightarrow CriterionForDiseaseClassRCl$;- - whose possible drug is associated with
$WithPossibleDrugCriteriaRCl\_NP : CriterionForDrugClass$
 $\rightarrow CriterionForDiseaseClassRCl$; - - whose possible drug have NP

Fig. 2. Functions for diseases expressed as restrictions on the property PossibleDrug

The property becomes one of the functions' parameters. The most important issue is that it is not possible to include all actual values in the grammar, because the set of values is not finite. This issue can not be completely solved in GF. The proposed solution is to include in the grammar generic trees with a dummy string. If the translation to SPARQL succeeds, the dummy value is replaced in the generated query during postprocessing. Since the values for these restrictions tend to appear at the end of the question, e.g. *Give me the side effects of drugs with a solubility of 3.24e-02 mg/mL*, in the preprocessing phase the string value is replaced with XX and the question to be parsed becomes *Give me the side effects of drugs with a solubility of XX*. This is identified as *[GiveSiderProperty SideEffect [ToDrugClass [ValueRestriction Solubility]]]*, where *SideEffect* indicates the object property whose value is asked for. The content of the innermost brackets represents the drugs indicated by the transformation to DrugClass of a value restriction on the datatype property *Solubility*. Another possible solution for covering numerical values for these restrictions could be based on the GF support for integers and floating point numbers.

Other described constructors include *HighestNumber*, *LowestNumber*, *ZeroNumber*, which are focused on the number of properties, or *HighestValue*, and *LowestValue* which are focused on values of properties. For example, *the least common chromosome location* is interpreted as [LowestValue ChromosomeLocation], where *ChromosomeLocation* is a DrugBank property.

$$\begin{aligned}
\mathit{ValueRestriction} &: \mathit{DrugBankProperty} \rightarrow \mathit{CriterionForDrugClass} \\
&\text{- - solubility of XX} \\
\mathit{ValueRestrictionAdj} &\rightarrow \mathit{CriterionForTargetClass} \\
&\text{- - involved in XX} \\
\mathit{ValueRestrictionRCl} &: \mathit{DrugBankProperty} \rightarrow \mathit{CriterionForDrugClassRCl} \\
&\text{- - whose route of elimination involves XX} \\
\mathit{DiseaseValueRestriction} &: \mathit{DiseasomeProperty} \rightarrow \mathit{CriterionForDiseaseClassNP} \\
&\text{- - chromosomal location of XX} \\
\mathit{DiseaseValueRestrictionRCl} &: \mathit{DiseasomeProperty} \rightarrow \mathit{CriterionForDiseaseClassRCl} \\
&\text{- - whose subtype involves XX} \\
\mathit{LowestNumber} &: \mathit{Property} \rightarrow \mathit{CriterionForDrugClass} \\
&\text{- - lowest number of side effects} \\
\mathit{DiseaseWithLowestValue} &: \mathit{DiseasomeProperty} \rightarrow \mathit{CriterionForDiseaseClassNP}; \\
&\text{- - with lowest size} \\
\mathit{LowestNumberValue} &: \mathit{Property} \rightarrow \mathit{PropertyClass}; \\
&\text{- - least common chromosome location}
\end{aligned}$$

Fig. 3. Functions for Value Restrictions

2.2 Transformation Functions

For composability reasons, *transformation* functions are defined for getting from a criterion to a class, or for getting from one dataset to another. The former are important for English linearization, while the latter play an important role in SPARQL linearization.

The first transformation functions take criteria and build on them the upper level linguistic structures needed in queries. For example, in order to get to the Noun Phrase *drugs used for Rickets* from the Adjectival Phrase *used from Rickets*, there is a transformation from CriterionForDrugClassAdj to DrugClass that adds the noun *drugs* to linearization of the AP. When building SPARQL queries, these transformation functions do not alter the linearization of the Criterion, because the corresponding SPARQL triplets are already completely built. All the English alternatives for expressing a conceptual DL constructor have the same SPARQL linearization. This is somehow expected, as SPARQL is a formal language tightly related to DLs.

The second type of transformations deals with queries requesting access to more datasets. In this case, English linearization does not alter the object of transformation, while the SPARQL linearization introduces new variables and *sameAs* statements. For example, the function *DBToSiderDrug* converts the class of DrugBank drugs to the class of Sider drugs. Its SPARQL linerization introduces a new variable ?*siderdrug* that is related with a *sameAs* statement

$$\begin{aligned}
WhichDisease2 &: DiseaseClass \rightarrow Question; \\
&\text{- - which are the diseases caused by Lepirudin?} \\
WhichDisease &: CriterionForDiseaseClass \rightarrow Question; \\
&\text{- - which diseases are caused by Lepirudin?} \\
WhichTargetAdj &: ValueRestrictionAdj \rightarrow Question; \\
&\text{- - which targets are involved in XX?} \\
WhatPropertyValue &: PropertyClass \rightarrow Question; \\
&\text{- - which is the least common chromosome location?}
\end{aligned}$$

Fig. 4. Functions for queries

to the variable of the function's parameter. *addStatement2* and *mkSameAsStatement* are operators introduced in the proposed SPARQL resource library:

```
DBToSiderDrug : DrugClass -> SiderDrugClass;
DBToSiderDrug d={var="?siderdrug";      /* concrete SPARQL */
     body=addStatement2 (mkSameAsStatement "?siderdrug" d.var)
                        d.body};
DBToSiderDrug d=d;                      /* concrete English */
```

2.3 Functions for Queries

Several types of queries were identified: *give*, *list*, *which*, *what*, *for/with which*, and *is/are there*. They are applied on one class, one criterion, or on a list of classes or criteria for classes (see Fig. 4). The questions deal mostly with resource classes and criteria for these classes and less with properties. An exception to this rule is the question *WhatPropertyValue*. This question treats *PropertyClass* instead of a resource class, because it queries for information about a property class and not about a property of some resource. For example, the question *which is the least common chromosome location* is parsed to the abstract tree *[WhatPropertyValue [LowestNumberValue [DBToProperty ChromosomeLocation]]]*. Its SPARQL linearization requires aggregation and sort operations.

The advantage of taking the described approach is the flexibility in composition of trees/constructors, based on their types and transformation functions. For example, *drugs that interact with the drugs used for diseases treated by tetracycline* is parsed to the abstract tree t_3=[ToDrugClass_withThatVP [DDrugClassCriterionVP t_2]], where t_2=[AdjToDrugClass [PossibleDrugsForCriterionAdj t_1]] is the tree for the class of drugs that are used for diseases in t_1. t_1=[ToDiseaseClass [WithPossibleDrugsCriterion [SingleDrug DB00759]]] stands for a DiseaseClass of diseases treated by tetracycline. DB00759 is the DrugBank ID for tetracycline. The abstract tree t_3 is linearized in the SPARQL concrete grammar. By running the query, we get drugs which interact with tetracycline, and also other drugs used to treat the same diseases as tetracycline.

Algorithm 1 English2SPARQL

```
toLowerCase(question)
replacedText=""
answer=TRANSLATION(question)
if !(answer contains FAIL) then
    Find $Abstract\_Tree_k$ with minimal length
return $SPARQL\_Linearization_k$ of $Abstract\_Tree_k$
else
    while (answer contains FAIL) && !empty(question) do
        replacedText+=lastWord(question)
        question=removeLastWord(question)
        answer=TRANSLATION(question+XX)
    end while
end if
if !(answer contains FAIL) then
    Find $Abstract\_Tree_k$ with minimal length
    $query \leftarrow substitute(XX, replacedText, SPARQL\_Linearization_k)$
return query
end if

function TRANSLATION(EN phrase)                                ▷ GF Rest Service
    $Abstract\_Tree_i \leftarrow PARSE(ENphrase)$
    $SPARQL\_Linearization_i \leftarrow LINEARIZE(Abstract\_Tree_i)$
return $(i > 0)\ ?\ \bigcup_i \{SPARQL\_Linearization_i, Abstract\_Tree_i\} : FAIL$
end function
```

2.4 Pre- and Post-processing

GF comes with an HTTP server that supports REST services for its main functionality, as translation or parsing. GFMed includes (i) the abstract grammar and the concrete grammars for English and SPARQL described previously, and (ii) a Java standalone application that consumes GF translation service based on these grammars.

The standalone application includes a preprocessing module, a module for consuming the translation service, and a post-processing module. Algorithm 1 describes the main steps of the translation from a natural language to SPARQL.

Preprocessing includes a simple transformation of the question to lowercase, and a failure handling method. When the translation module gets a failure from the server, the failure handling method repeatedly trims the last word of the question and replaces the trimmed sequence with the dummy string XX. This is done in order to deal with value restrictions, for example *drugs with water solubility of 3.24e-02 mg/mL*. It can be observed that the part which represents the value is formed by the last two words, and not only the last one.

A special case of this trimming is done for situations where a list of free text values is included in the question. Question 13 from the QALD test set is an example for this situation: it includes the phrase *drugs whose mechanism of action* *involves norepinephrine and serotonin*, with *mechanism of action* as a datatype property. In this case, the preprocessing includes a step where the question is split by the string *and*. Thus, the previously mentioned phrase becomes *drugs whose mechanism of action involves XX and YY*. In case the translation works, XX is replaced with *norepinephrine* in the resulting query, and YY with *serotonin*. It must be mentioned that in case one of the free text values includes more than one word, this method fails.

After a successful translation, the postprocessing module searches for the abstract tree with the smallest length. This is needed because is is possible to have more alternatives for translating the questions, mainly due to the transformation functions. Once the tree is found, its SPARQL linearization is extracted. In case it was a value restriction, solved by the failure handling method, some replacements are done.

3 Employed Resources

3.1 Language Resources

GFMed is built mainly as a domain-specific application grammar described in GF. The GF services of parsing, linearization, translation and completion based on two concrete grammars and an abstract grammar are employed. The completion service supports assisted query building by suggesting at each step possible words to continue.

GF has support for syntax, lexicon and inflections in 36 languages. It comes with a comprehensive library for the English language [7]. GFMed English concrete grammar relies on this library when it comes to syntax, morphological paradigms used to introduce new elements in the lexicon, and coordination.

The situation is different for SPARQL. GF does not provide a library for it, even though there are applications using GF for querying Linked Data in Natural Language [3]. Therefore, GFMed proposes a resource library for SPARQL[1] which defines categories and operators for dealing with RDF resources and properties, with partial or complete triplets, and with aggregation, filtering, and sort operations.

In the concrete SPARQL grammar built on top of this library, each property from the targeted datasets is linearized to a triplet structure that has null object and subject. These two are completed within linearization of different restriction functions: one of them must be a resource or a previously introduced variable, case in which there must exist a triplet where this variable is bound. The other one is completed with a newly introduced variable that will be either included in the SELECT clause of the query, or will become the subject or the object of another triplet, when more functions are composed. In order to be able to do this, the linearizations of $\mathcal{X}Class$ or of the associated criteria are structures consisting of the name of the new variable and the body that includes complete triplets and possible aggregations or filters.

[1] It can be accessed at http://cs-gw.utcluj.ro/%7Eanca/GFMed

Table 2. Number of distinct resources described in lexicon

Dataset	Resource Type	Distinct IDs	Distinct names	Considered properties
DrugBank	Drug	1470	22872	$drugbank : name$, $drugbank : synonym$, $drugbank : brandName$
DrugBank	Target	4553	3784	$drugbank : name$
Diseasome	Disease	4213	3642	$diseasome : name$
Diseasome	Gene	3919	4328	$rdfs : label$, $owl : sameAs$
SIDER	SideEffect	1737	2398	$sider : sideEffectName$

When dealing with value restrictions, the SPARQL linearization must include different types of filters according to the datatype of the property: regex-based filters for string, and equality for integer or real datatypes. In order to identify the correct filter, SPARQL linearization of each DrugBank, Diseasome, Sider property includes also the type, Number or String, in addition to its complete name.

3.2 Generated Lexicons

It must be emphasized that GF grammars must know, at compilation time, all the tokens that are part of the analyzed text. Therefore, GFMed includes lexicons for both SPARQL and English formed of all drugs, targets, diseases, genes, and side effects extracted from the three datasets (see Fig. 1).

In the early stages of GFMed's development, these lexicons where generated from data sources available on the sites of the three datatasets, either by using SPARQL endpoints, or by parsing RDF files. In the end, the lexicons where generated by executing SPARQL queries on the QALD-provided endpoint. Special attention was given to side effects, drugs, and genes. For the same ID of a side effect more synonym names are known, expressed through the property *sideEffectName*. For one drug ID in DrugBank, there are more names and synonyms. Furthermore, as the name, the synonyms and the brand names of a drug can appear in a question, English linearization of each drug includes alternatives expressed by values of properties *name*, *synonym*, and *brandName*. For Genes, besides the property $rdfs : label$, it was considered the property $owl : sameAs$ that relates some genes to DBpedia resources. Extended names for genes are extracted from these resources.

Table 2 shows the number of resources identified in this way, giving both the number of distinct IDs and distinct names for each category. Even though the number of elements in lexicons is not small, the time of compilation to PGF (Portable Grammar Format) and the running time are not significantly increased. But, if it were to take the same approach for DBPedia for example, the size of the lexica and of the grammars could become a problem.

4 Results and Their Analysis

The system was evaluated against training and test questions of Task 2 in QALD. GFMed had the best results, with the overall evaluation from table 3.

Table 3. Results for GFMed in Task2

Total	Processed	Right	Partially	Recall	Precision	F-measure
25	25	24	1	0.99	1	0.99

GFMed correctly parsed all the questions, except one. It partially parsed question 21, *Give me the drug categories of Desoxyn*, for which it obtained 0.85714 recall and precision 1, meaning that all the answers retrieved by the proposed query were correct, but they were not complete. The reason for this is that Desoxyn is a brand name for drugs with DrugBank IDs DB00182, DB01576, DB01577. We wrongly assumed that one brand name can be associated either to only one drug, or to several drugs but with consistent descriptions. The drug DB00182 has one more category compared to the other two drugs: amphetamines. GFMed identified the drug as being DB01577 so it missed this category. Given the fact that more drugs with different names and different descriptions can have the same brand name, we think that the lexicon should treat the names differently in comparison to the brand names. Instead of having one drug with alternative linearizations for both *name* and *brand name*, it would be better to linearize in English a drug ID only to its name and synonyms. A new category for brand names is required, and consequently, a new transformation function from brand name to drug IDs. Thus, the generated *Where* clause would become:

```
{?drug drugbank:brandName "Desoxyn". ?drug drugbank:drugCategory ?categ}
     instead of currently generated
{drugbank:DB01577 drugbank:drugCategory ?categ}
```

5 Perspectives for Future Work

The set of described abstract and concrete structures can be extended to other DL constructors. Furthermore, transformation functions are valuable for a uniform treatment of different types of resources, properties or restrictions, but they introduce on one hand (i) the possibility to generate incorrect English questions, as *List Lepirudin*, where Lepirudin is transformed to a drug class, and on the other hand (ii) the parsing of a question into several alternative abstract trees which introduce unnecessary *sameAs* statements in the SPARQL linearization. The current solution to take the shortest abstract tree works well, but we plan to investigate ways to avoid the enumerated issues of transformation functions from the grammar itself.

Starting from the GFMed experiment, a general methodology to build such grammars from ontologies in a semi-automatical manner is to be investigated, similar to the approach taken in [5].

Support for multilinguality is one of the GF strengths that we intend to exploit in future work. In order to support querying of the Task2 datasets in a GF recognised language other than English, GFMed English concrete grammar can be moved almost entirely to an incomplete grammar. Querying in a different language can be obtained by extending this incomplete grammar with language-specific exceptions, together with the language-dependent lexicon. A similar experiment to GFMed, but for Romanian language, is described in [6] where DBpedia information about cultural personalities are searched.

6 Conclusions

We consider that our DL-based methodology for building Controlled Natural Language for querying Linked Data was validated by this experiment. The described grammars are expressive enough to cover questions proposed in Task 2 of QALD, addressing querying over more datasets, complex queries with different linguistic structures, and queries that involve lists and free text. By its intrinsic definition, the semantic of any Controlled Natural Language is limited, so our language is also limited. Neththeless, the DL approach can guide its extension in a structured way.

References

1. Baader, F.: The description logic handbook: theory, implementation, and applications. Cambridge university press (2003)
2. Belleau, F., Nolin, M.A., Tourigny, N., Rigault, P., Morissette, J.: Bio2RDF: Towards a mashup to build bioinformatics knowledge systems. Journal of Biomedical Informatics 41(5), 706–716 (2008)
3. Dannells, D., Enache, R., Mateva, M., Ranta, A.: Natural Language Interaction with Semantic Web Knowledge Bases and LOD. Towards the Multilingual Semantic Web, Paul Buitelaar and Philip Cimiano, eds., Springer, (2014)
4. Ferré, S.: Squall: A controlled natural language as expressive as sparql 1.1. In: Métais, E., Meziane, F., Saraee, M., Sugumaran, V., Vadera, S. (eds.) NLDB. Lecture Notes in Computer Science, vol. 7934, pp. 114–125. Springer (2013)
5. van Grondelle, J., Unger, C.: A three-dimensional paradigm for conceptually scoped language technology. In: Buitelaar, P., Cimiano, P. (eds.) Towards the Multilingual Semantic Web. Springer (2014)
6. Marginean, A., Groza, A., Slavescu, R.R., Letia, I.A.: Romanian2SPARQL: A Grammatical Framework approach for querying Linked Data in Romanian language. In: Proceedings of 12th International Conference on Development and Application Systems, Suceava, Romania, May 15-17 (2014)
7. Ranta, A.: Grammatical Framework: Programming with Multilingual Grammars. CSLI Publications, Stanford (2011)
8. Zenz, G., Zhou, X., Minack, E., Siberski, W., Nejdl, W.: Aqualog: An ontology-driven question answering system for organizational semantic intranets. Journal Web Semantics 5(2), 72–105 (2007)
9. Zenz, G., Zhou, X., Minack, E., Siberski, W., Nejdl, W.: From keywords to semantic queries - incremental query construction on the semantic web. Journal Web Semantic 7(3), 166–176 (2009)