Semantic Web 0 (2015) 1–0
IOS Press

Question Answering over BioMedical Linked Data with Grammatical Framework

Anca Marginean [a,*]
[a] *Department of Computer Science, Technical University of Cluj Napoca, 401446 Cluj-Napoca, Romania*

Abstract. The blending of linked data with ontologies leverages the access to data. GFMed introduces grammars for a controlled natural language targeted towards biomedical linked data 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. The semantic and linguistic chunks correspond to Description Logic constructors. Problems and solutions for querying biomedical linked data with Romanian, beside English, are also considered in the context of GF.

Keywords: querying linked data, description logics, controlled natural language, multilingual system, Grammatical Framework

1. Introduction

Linked Data means using the Web to connect related data. A large number of data from various domains such as government data, education, life sciences, art and others were made available in the context of the Linked Open Data 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 [14], or in Controlled Natural Languages [6], [5], [10], or incremental query building [24].

The suitability of interfaces in natural languages for querying linked data is justified by more reasons. Frequently, the linked data are described with the use of large terminologies. Connections between datasets are encouraged by the very essence of the concept of linked data. Consequently, the lack of previous and detailed knowledge about the structure of the data makes the querying task tedious, even for users well adjusted to the semantic web technologies. A controlled natural language could hide the complexity from the user, without important limitations on the expected expressivity. In the same time, being a restricted natural language, building it with a well defined semantic is feasible, especially in the context of the large adoption of ontologies and more recently, of their lexicalization [8].

Querying databases with meaning representation languages that rely on natural language is an old idea. CHILL system [23] used such a language for querying Geobase, a set of Prolog facts. The specialized parser for the language was learned through inductive logic programming. More recently, statistical machine learning was used. With an increasing popularity, controlled natural languages (CNLs) aim at giving an intuitive representation to formal representations, by making a trade-off between precision of formal languages and ambiguity, but high expressivity, of natural languages. A comparative analysis of controlled natural languages is done in [12].

In this line, we propose a system (GFMed) based on application grammars manually built with Grammatical Framework (GF) [16]. All GFMed resources are available at `http://cs-gw.`

[*]Corresponding author. E-mail: anca.marginean@cs.utcluj.ro

utcluj.ro/~anca/GFMed. The goal is to provide a natural, yet precise way of querying linked data, with the help of a subset of natural language. The relation between the meaning of expressions in natural language and the constructors of description logics guided the process of building the CNL. This relation is also analyzed in [8], [19], while GF is also used for linked data in [5], [7]. The targeted linked data are biomedical data, described in Diseasome, Sider and DrugBank. These three sets were proposed in Task 2, *Biomedical question answering over linked data*, of the *Question Answering over Linked Data* (QALD-4) [17].

DrugBank is part of the project Bio2RDF [4] and gives chemical, pharmacological and pharmaceutical data about drugs with comprehensive drug target 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 three datasets are connected: diseases from Diseasome are related to drugs described in DrugBank, DrugBank drugs are related to Sider drugs with the relation *sameAs* and there are side effects from Sider that are also diseases.

Grammatical Framework is a special purpose programming language with a high support for multilingual applications. Therefore, starting from the grammars proposed for querying in English, a multilingual system was investigated, with Romanian as the second natural language.

The article is structured in 6 sections. Section 2 introduces the GF resource library that we propose for SPARQL, together with a very brief description of the proposed CNL. Section 3 describes the main functions from the abstract and concrete grammars which define the controlled language. Section 4 introduces our first attempt in building a multilingual language for querying biomedical data. Related work is analyzed in Section 5, while some conclusions are drawn in Section 6.

2. Controlled natural languages with GF for linked data

The general workflow of a system for querying linked data with GF is detailed in Figure 1. The user inputs the query in natural language and a SPARQL query is generated with the help of the GF grammars.

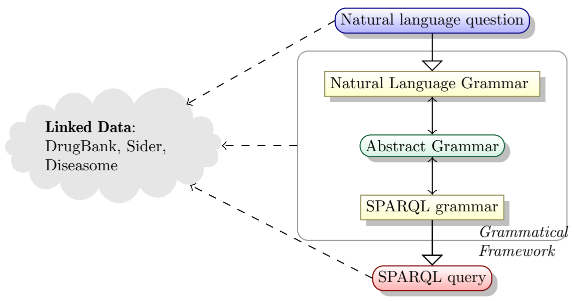


Fig. 1. Querying linked data with natural language

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 defined and for each function, 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.

GF has comprehensive libraries for syntax, lexicon and inflections in 36 languages [16]. CNLs built with GF, including ours, rely on these libraries for syntax, morphological paradigms useful for introduction of new elements in the lexicon, and coordination.

The abstract grammar of a CNL built with GF stands for the CNL's semantic model [2]. The syntax of the CNL is defined by the concrete grammars for natural languages. In case of SPARQL concrete grammar, since SPARQL is a formal language, the patterns described in this grammar could also be considered for CNL's semantic.

2.1. SPARQL resource library

The first step in using GF for querying linked data is to have proper support for the language SPARQL. There is another SPARQL-based retrieval interface for structured data [6], [5], yet, we preffered to define our own resource for SPARQL, with a set of types and operators. In a previous attempt to query touristic linked data [20], we did

```
mkStatement : Triplet → Statement =
        \vp → {s = vp.subj + +vp.prop.s + +vp.obj; extra = ""; aggreg = no};
mkStatementwithAddit : Triplet → Str → Statement =
        \vp, adit → {s = vp.subj + +vp.prop.s + +vp.obj; extra = adit; aggreg = yes};
mkFilterStatement : PropertyT → Str → Str → Statement =
        \p, x, v2 → case   p.vt   of   {
                String ⇒ {s = "FILTER(regex(" + +v2 + +", " + +x + +",'i'))";  extra = ""; aggreg = no};
                Number ⇒ {s = "FILTER(" + +v2 + +" = " + +x + +")"; extra = "";  aggreg = no}};
mkNotFilter1 : Statement → Statement =
        \t → {s = "FILTER NOT EXISTS{" + +t.s + +"}"; extra = ""; aggreg = no};
addStatement : Triplet → Statement → Statement =
        \vp, st → let   s1 = mkStatement   vp
                      in   {s = s1.s + +"." + +st.s; extra = st.extra; aggreg = st.aggreg};
addStatement2 : Statement → Statement → Statement =
        \st1, st2 → {s = st1.s + +"." + +st2.s; extra = st1.extra + +st2.extra;
                aggreg = case st1.aggreg of { yes ⇒ yes;
                                              no ⇒ st2.aggreg}   };
```

Fig. 2. Operators that create graph patterns

not use any GF module of type resource in writing the SPARQL concrete grammar. But, SPARQL-dedicated types and operators ease the process of building compact SPARQL grammars.

Two main types were defined: Triplet and Statement. The type $Triplet : Type = \{subj,\ obj : Str;\ prop : PropertyT\};$ is a structure with three components: two string components for the subject and the object, and a structure for the property. The structure $PropertyT : Type = \{s : Str;\ vt : ValueType\}$ for the property includes also two fields: a string for the name of the property and a field of the enumerated type $ValueType$. The field stores the type of the values for the property. The enumerated type has two values: *String* and *Number*. The type of the property is important in using the correct operator inside the FILTER expressions. For the moment, only *equality* and the *regex* operators are included. By default, the *regex* operator is used for properties that have strings as values, while the *equality* operator is used for the properties with numerical values.

The type $Statement : Type = \{s : Str;\ extra : Str;\ aggreg : AggregationType\};$ is a structure with three fields. It corresponds to a graph pattern formed by one or more triple patterns. The first component is the string expressing the graph pattern and it will be part of the WHERE clause. The third component, *aggreg*, has two possible values, *yes* and *no*, stating whether optional statements, as GROUP BY, ORDER BY or LIMIT clauses are required. The field *extra* contains the expression for the optional clauses in case they exist. When two statements are concatenated, their *extra* fields are concatenated too if they exist.

A set of operators were defined on these types. There are operators that create incomplete triple patterns, as *mkEmptyTriplet* or *addSubj* (Figure 3). A subset of the operators that create graph patterns is mentioned in Figure 2. The operator *mkStatement* concatenates the fields of a *Triplet* and creates a graph pattern with only one triple pattern. The operator *mkStatementWithAddit* creates a statement with a non-empty string for the *extra* field. The operator *mkFilterStatement* details the

$$\begin{aligned}
&\mathit{mkEmptyTriplet} : \mathit{PropertyT} \rightarrow \mathit{Triplet} = \\
&\backslash p \rightarrow \{\mathit{subj} = \text{""}; \mathit{prop} = p; \mathit{obj} = \text{""}\};
\end{aligned}$$

$$\begin{aligned}
&\mathit{addSubj} : \mathit{Str} \rightarrow \mathit{Triplet} \rightarrow \mathit{Triplet} = \\
&\backslash \mathit{ss}, \mathit{vp} \rightarrow \{\mathit{subj} = \mathit{ss}; \mathit{prop} = \mathit{vp.prop}; \\
&\qquad\qquad \mathit{obj} = \mathit{vp.obj}\};
\end{aligned}$$

Fig. 3. Operators which create a *Triplet*

$$\begin{aligned}
&\mathit{mkQuery} : \mathit{Str} \rightarrow \mathit{Statement} \rightarrow \mathit{Str} = \\
&\backslash \mathit{var}, b \rightarrow \mathit{case} \quad \mathit{b.aggreg} \quad \mathit{of}\{ \\
&\quad \mathit{yes} \Rightarrow \text{"}\mathit{select}\text{"} {+}{+} \mathit{var} {+}{+} \\
&\qquad\quad \text{"}\mathit{where}\{\text{"} {+}{+} \mathit{b.s} {+}{+} \text{"}\}\text{"} {+}{+} \mathit{b.extra}; \\
&\quad \mathit{no} \Rightarrow \text{"}\mathit{select} \quad \mathit{distinct}\text{"} {+}{+} \mathit{var} {+}{+} \\
&\qquad\quad \text{"}\mathit{where}\{\text{"} {+}{+} \mathit{b.s} {+}{+} \text{"}\}\text{"} \quad \};
\end{aligned}$$

$$\begin{aligned}
&\mathit{mkInnerQuery} : \mathit{Str} \rightarrow \mathit{Statement} \rightarrow \mathit{Str} = \\
&\backslash \mathit{var}, b \rightarrow \text{"}\{\mathit{select}\text{"} {+}{+} \mathit{var} {+}{+} \\
&\qquad\quad \text{"}\mathit{where}\{\text{"} {+}{+} \mathit{b.s} {+}{+} \text{"}\}\}\text{"};
\end{aligned}$$

$$\begin{aligned}
&\mathit{mkAskQuery} : \mathit{Statement} \rightarrow \mathit{Str} = \\
&\backslash b \rightarrow \text{"}\mathit{ask}\{\text{"} {+}{+} \mathit{b.s} {+}{+} \text{"}\}\text{"};
\end{aligned}$$

Fig. 4. Operators which create SELECT and ASK queries

default behavior for FILTER expressions for the two defined property types. In order to support other datatypes, the enumerated type $ValueType$ must be extended with other types, and a new field can be added to $PropertyT$ allowing explicit specification in this function of the behavior for properties with a certain type of values. The last two operators build graph patterns with more than one triple pattern. The operator *addStatement* adds a triple pattern to a graph pattern, while the function *addStatement2* concatenates two graph patterns.

Finally, there is a set of operators that create the strings for SELECT and ASK queries (Figure 4). The operator *mkQuery* applies on a string that includes the variables to appear in the query results and on a graph pattern. The final string is built according to whether optional SPARQL clauses are part of the query or not.

2.2. Brief description of GFMed CNL

The GFMed CNL aims for querying biomedical knowledge bases. The set of interrogative or imperative sentences includes *which*, *what*, *are there*, *list* and *give me* type sentences. The entities for which some data are queried are: diseases, drugs, genes, targets, side-effects. The entities can be referred: (a) by their name: *lepirudin*, *rickets*, *fever*, (b) by a named category to which they belong: *approved drugs*, *metabolic class*, or (c) by a category defined through a certain property (as the ones mentioned in table 1): *drugs that treat rickets*, *diseases associated to lrrc8*. The expressivity of the CNL is enlarged by the fact that these three types can be composed, e.g. *side-effects of drugs that treat diseases associated to growth hormone 1.* Besides questions about these entities, certain information about properties are also covered: *highest value of melting point*, *least common chromosomal location.*

Table 1
Classes of resources to express in CNL

drugs that are possible drugs for a disease
drugs that interact with another drug or food
drugs that target/are involved in something
drugs with/without a certain side effect
drugs with a certain value for a datatype property
diseases that have a drug as possible drug
diseases associated to a gene
diseases that are subtype of another disorder
diseases with a certain value for a datatype property
side effects of a drug
genes associated to a disease
food that interacts with a drug

The first two types of entities references are obtained from the three datasets and are part of the lexicon. For the third type, a set of words, in order of tens, are included in the lexicon, e.g. *number*, *of, distinct, involve*, *target*, *drug*, together with the names of the properties defined by the terminology of the three datasets, e.g. *predicted solubility*, *affected organisms, brand name, route of elimination.* Next section gives a detailed description of how this CNL is built.

3. DL based questions for biomedical linked data

Description Logics (DLs) [3] 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

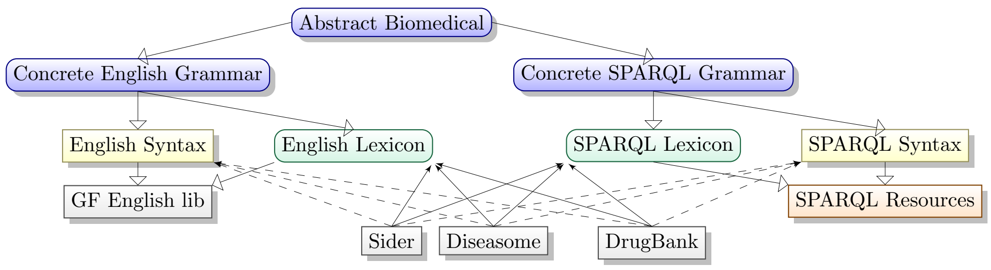


Fig. 5. The main grammars and resources used by GFMed

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 chunks that have straightforward translations to SPARQL and are highly composable.

GFMed, the proposed system, 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 post-processing 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 5 shows the main grammars: an abstract grammar and two concrete grammars. For each concrete grammar, lexicons derived from SIDER, Diseasome and DrugBank were generated. Many syntactic structures in both English and SPARQL were driven by the datasets' 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 datasets. We also introduced the categories DrugBankProperty, DiseasomeProperty and SIDERProperty for describing the properties of the same datasets. 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 2 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. GFMed'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.

In the concrete SPARQL grammar built on top on our SPARQL resource library, each property from the targeted datasets is linearized to a value of type $Triplet$, initially with 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 i) the name of the new variable and ii) the body that includes complete triplets and possible aggregations or filters in a $Statement$ structure.

When dealing with *hasValue* restrictions, the SPARQL linearization must include different types of filters according to the datatype of the property. 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.

Table 2
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-For$\mathcal{X}$Class$\mathcal{Y}$		criterion 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 ...*

Table 3
Examples of class expressions and assertions in Diseasome

DL Constructor	Examples
existential restriction $\exists$ `R.C`	$\exists$ *PossibleDrug.ApprovedDrugs* - diseases treated with at least one drug from the category of *ApprovedDrug*
	$\exists\, PossibleDrug^{-1}.DiseasesWithDegree1$ - drugs that treat diseases with *Degree* 1
universal restriction $\forall$ `R.C`	$\forall$ *PossibleDrug.*$\{$*Bextra*$\}$ - diseases treated only with *Bextra*
individual assertion `a:C`	*Rickets* : *Disease* - *Rickets* is a disease
role assertion `(a,b):R`	(*Rickets*, *Calcitriol*) : *PossibleDrug* - *Rickets* has *Calcitriol* as possible drug

3.1. Building Trees for Property Restrictions

In DLs, there are two types of roles or properties: object properties and datatype properties. Object properties relate individuals of two concepts, while datatype properties relate an individual of 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 Object properties and datatype properties are treated differently in the GFMed 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*. Here, $FeverInducingDrug$ stands for all drugs that have fever as a side effect and it is a *value restriction* with value $Fever$ on the property $SideEffect$. Some more examples are given in table 3. The correspondence between linguistic phrases of our CNL and their DL corresponding expression ensures composability of functions from the concrete grammars based on the their types.

```
WithPossibleDrug : DrugClass → DiseaseClass;                              -- diseases treated with D
WithPossibleDrugCriterion : DrugClass → CriterionForDiseaseClass;                     -- treated with D
WithPossibleDrugCriteriaClSlash : DrugClass → CriterionForDiseaseClassClSlash;          -- D is used for
WithPossibleDrugCriteriaNP : DrugClass → CriterionForDiseaseClassNP;             -- D as possible drug
WithPossibleDrugCriteriaAdv : DrugClass → CriterionForDiseaseClassAdv;      -- with D as possible drug
WithPossibleDrugCriteriaRCl : DrugClass → CriterionForDiseaseClassRCl; -- whose possible drug is D
WithPossibleDrugCriteriaRCl_VP : CriterionForDrugClassVP → CriterionForDiseaseClassRCl;
                                                          -- whose possible drug interacts with D
WithPossibleDrugCriteriaRCl_Adj : CriterionForDrugClassAdj → CriterionForDiseaseClassRCl;
                                                     -- whose possible drug is associated with D
```

Fig. 6. Functions for diseases expressed as restrictions on the property *PossibleDrug*

```
WithPossibleDrugs   dc = -- SPARQL linearization for ∃PossibleDrug.⟨drugclass⟩
    let   disc = addSubj   "?dis"   (addObj   dc.var   PossibleDrug)
    in   {var = "?dis";   body = addStatement2   (addStatement2   (mkStatement   disc)
                                                        (mkDiseaseStatement   "?dis"))
                                               dc.body};
```

Fig. 7. SPARQL linearization function for diseases expressed as restrictions on the property *PossibleDrug*

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. Each DL constructor identified at a conceptual level corresponds to more functions which build trees at the concrete English level, one for each possible syntactic structure. 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. Figure 6 shows functions that model restriction on the property *possibleDrug* with values in *DrugClass*. Their SPARQL Linearization, which is one for all, is mentioned in Figure 7.

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 classes formed from only one named drug are recognized by the CNL, *hasValue* restrictions on object properties can be treated in the same way as existential restrictions. Other properties treated similarly to *possibleDrug* are *associatedGene, sideEffect, target, interactionDrug1.*

Within this approach, *treated by interferon beta-1a* is identified as *(WithPossibleDrugsCriterion (SingleDrug DB00060))*, and its SPARQL linearization is the graph pattern *?dis ds:diseasome/possibleDrug db:drugs/DB00060 . ?dis a ds:diseasome/diseases.*

Datatype properties 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 Figure 8. 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 post-processing. 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 the dummy value and the question to be parsed becomes *Give me the side effects of drugs with a*

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

Fig. 8. Functions for restrictions on datatype properties

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.

Cardinality restrictions, either on datatype or object property, are not covered in the current version of CNL, but they can be included without much effort: the current SPARQL resource already has support for inner queries. However, the number of functions for restrictions on object properties would double. The conjunction of DL classes that correspond to main entities of the CNL is covered for a limited subset, for example *causes fever and anemia*. The disjunction operator is not covered, while the negation is partially covered, e.g. *without side effects*, *without fever as side effect*.

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. Optional SPARQL clauses are used for these constructors, as it can be seen in the SPARQL linearization of the function *HighestNumber*:

(HighestNumber (DbToProperty Indication))
the highest number of indications
?drug db:drugbank/indication ?vp. – WHERE clause count(distinct ?vp) as ?c – SELECT clause group by ?drug order by desc(?c) limit 1 – the *extra* field
(WithPossibleDrugsCriteria (ToDrugClass GasState))
treated by drugs with gas state
?dis ds:diseasome/possibleDrug ?drug . ?dis a ds:diseasome/diseases . ?drug db:drugbank/state ?state . FILTER(regex(?state , 'gas' , 'i'))

Fig. 9. Examples for datatype properties

```
HighestNumber  p =
let hn = addSubj  "?drug"  (addObj "?vp" p)
in{
var = "COUNT(DISTINCT ?vp) as  ?c, ?drug";
body = mkStatementwithAddit  hn
        "GROUP BY ?drug ORDER BY desc(?c)
          LIMIT 1"};
```

Figure 9 details two examples. The first uses the function *HighestNumber*. The resulting graph pattern for the WHERE clause is stored in the component *s* of the structure *Statement* (from the SPARQL resource library), and the description for the aggregation function is stored in the component *extra* of the same structure *Statement*. A second example includes a filter on the value of the property *state* from DrugBank.

3.2. 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 lineariza-

[WithPossibleDrugsCriteria [ToDrugClass [ToDrugClassCriteria [SiderZeroNumber SideEffect]]]]

$$\underline{\underline{treated\ by\ \underline{drugs\ with\ \underline{\underline{no\ side\ effect}}}}}$$

$$\left.\begin{array}{l} ?dis\quad ds:diseasome/possibleDrug\quad ?drug.\\ ?dis\quad a\quad ds:diseasome/diseases.\\ \left.\left.\begin{array}{l} \left.\begin{array}{l} FILTER\quad NOT\quad EXISTS\quad \{?siderdrug\quad sd:sider/sideEffect?vp\}.\\ ?siderdrug\quad a\quad sd:sider/drugs.\end{array}\right\} t_1\\ ?drug\quad owl:sameAs\quad ?siderdrug\end{array}\right\} t_2\right\} t_3\end{array}\right\} t_4$$

Fig. 10. Abstract tree and concrete linearizations in English and SPARQL for an example with two transformation functions:

tion, 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 for Rickets*, there is a transformation from CriterionForDrugClassAdj to DrugClass that adds the noun *drugs* to the linearization of the AP. In the same way, *drugs that are used for Rickets* is obtained from the same criterion by another transformation function. When building SPARQL queries, these transformation functions do not alter the linearization of the Criterion, because the corresponding SPARQL triplets are already completely built. The only exception from this rule is met for the *negation* of a criterion. For example, *fever as side effect* is the criterion, *drugs with fever as side effect* is a drug class, and *drugs without fever as side effect* is its negation. The functions *ToSiderDrugClass* and *ToSiderDrugClass_Without* are both defined on the same criterion, and their linearization in SPARQL are the following:

```
ToSiderDrugClass    csdc = csdc;
ToSiderDrugClass_Without    csdc =
{var = csdc.var;
body = addStatement2
            (mkNotFilter1    csdc.body)
            (mkSiderDrugStatement    csdc.var)};
```

The only domain-dependent part from the SPARQL linearization of *ToSiderDrugClass_Without* is the function *mkSiderDrugStatement*. The clear separation between domain dependent and independent fragments from the majority of the SPARQL linearizations makes facile both the extension of the CNL and porting the CNL to another domain.

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 linearization introduces a new variable *?siderdrug* that is related with a *sameAs* statement to the variable of the function's parameter:

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

An example for the composition of two transformation functions is given in Figure 10. The innermost tree t_1 is a criterion for a class of Sider drugs [*SiderZeroNumber SideEffect*]. The tree t_2=[*ToDrugClassCriteria* t_1] is determined by a transformation function between the sets Sider and DrugBank, while t_3 = [*ToDrugClass* t_2] corresponds to a transformation from a criterion to a class.

3.3. 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 (Figure 11). 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

$WhichDisease2 : DiseaseClass \rightarrow Question;$	- - which are the diseases caused by D?
$WhichDisease : CriterionForDiseaseClass \rightarrow Question;$	- - which diseases are caused by D?
$WhichTargetAdj : ValueRestrictionAdj \rightarrow Question;$	- - which targets are involved in XX?
$WhatPropertyValue : PropertyClass \rightarrow Question;$	- - which is the least common chromosome location?

Fig. 11. Functions for queries

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]]].*

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.

The grammars can be used not only for translating natural language questions into SPARQL, but also for SPARQL queries to natural language questions, for the phrases that are not involved in the pre/post-processing of GFMed. For example, the next query is identified as being the SPARQL linearization of 17 different abstract trees.

```
SELECT DISTINCT ?possDrug
WHERE { ds:diseases/173
        ds:diseasome/possibleDrug ?possDrug }
```

3.4. 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.*

A special case of this trimming is done for situations where a list of free text values is included

Algorithm 1 NaturalLanguage2SPARQL

```
toLowerCase(question)
replacedText=""
answer=TRANSLATION(question)
if (answer does not contain FAIL) then
    find abstractTree_k with minimal size
    return SPARQLLin_k of abstractTree_k
else
    while (answer contains FAIL)&& (question is not empty) do
        replacedText+=lastWord(question)
        question=removeLastWord(question)
        answer=TRANSLATION(question+XX)
    end while
end if
if (answer does not contain FAIL) then
    find abstractTree_k with minimal size
    query← substitute(XX, replacedText, SPARQLLin_k)
    return query
end if

function TRANSLATION(phrase)
    abstractTree_i ← PARSE(phrase)
    SPARQLLin_i ← LINEARIZE(abstractTree_i)
    return $(i > 0)$ ? $\bigcup_i${SPARQLLin_i, abstractTree_i}
            : $FAIL$
end function
```

in the question. Question 13 from 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*, and YY with *serotonin*. A much more efficient preprocessing alternative would be to use the morphological analysis from GF and replace those words that are not found in the lexicon.

The transformation functions from a dataset to another are not protected against redundant application. It is possible to transform a drug from DrugBank to a drug from Sider, just to return back to a drug from DrugBank. Therefore the parsing step could return more alternative abstract trees of variable size, with redundant application of the transformation functions as one factor that increases size. The post-processing module searches for the abstract tree with the smallest size, where the size of a tree is the number of included nodes. 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.

3.5. Generated Lexicons

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 (Figure 5).

These lexicons where generated from the data sources available on the sites of the three datasets, either by using SPARQL endpoints, or by parsing RDF files. Also, the same results were obtained from 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 also 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

Table 4
Number of resources described in lexicon

Dataset / **Resource**	**Distinct Ids**	**Distinct Names**	**Considered properties**
DrugBank / *Drug*	1470	22872	*db:name*, *db:synonym* *db: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	*sd:sideEffect-Name*

Table 5
Results for GFMed in Task2 of QALD4

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

drug includes alternatives expressed by values of the 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 4 shows the number of resources identified in this way, giving both the number of distinct IDs and distinct names for each category.

3.6. Evaluation

The system was evaluated against training and test questions of Task 2 in QALD4 and obtained the overall evaluation from the table 5. 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, the lexicon should treat the *names* differently compared to *brand names*.

4. First steps towards a multilingual system

Grammatical Framework is a programming language for multilingual grammar applications. Therefore, starting from GFMed's concrete grammars for English, the first steps were made towards a multilingual application. The Romanian language was considered, beside English.

The default GF mechanism for building multilingual grammar applications is through incomplete grammars that are language independent. Such a grammar was created for the described CNL without any significant changes. The incomplete grammar is extended by two concrete grammars, one for English and one for Romanian. Lexicons were generated for both languages. The name of the properties from the schemas of the three sets were translated manually in Romanian, resulting in two lexicons, one for each language.

In Figure 12, two two-place adjectives are defined in the English and the Romanian lexicons. The criterion for drugs based on the inverse property of the *possibleDrugs* property is defined in the incomplete grammar using the functions from the lexicons. Figure 12 also contains an example of a property from DrugBank, *MeltingPoint*, as it is defined in Romanian lexicon.

From efficiency reasons, some constructions were not included in the GF Romanian library [9]. The main problems in using GF library for Romanian were identified for expressing relational nouns and genitive relative phrases. Unlike English prepositions, Romanian prepositions have cases, and the nouns and adjectives have different forms for different cases. GF's resource library has a good support for these cases, but we experienced a problem with expressing nouns in the genitive case due to the expected presence of the possessive article. The correct forms are *solubilitatea medicamentului*, *solubilitate* **a** *medicamentului*, *efect advers* **al** *medicamentului*, where the presence of the possessive article is variable. A proper solution is to add rules for dealing with this variable presence. For the moment, the surrogate solution was

```
- - English lexicon
UsedFor_A2 = P.mkA2 "used" for_Prep;
Treated_A2 = P.mkA2 "treated" by8means_Prep
          |P.mkA2 "treated" with_Prep;

- - Romanian lexicon
UsedFor_A2 = P.mkA2 (P.mkA "utilizat") for_Prep;
Treated_A2 = P.mkA2 (P.mkA "tratat") with_Prep
        |P.mkA2 (P.mkA "tratat") by8means_Prep;
Route_of_Elimination_CN = mkCN
   (P.mkN2 (P.mkN "cale" "căi")
             (P.mkPrep "de" Ac))
   (mkNP (P.mkN "eliminare"));

- - incomplete grammar
PossibleDrugsForCriterionAdj disc =
        mkAP   UsedFor_A2   disc;
PossibleDrugsForCriterionVP disc =
        mkVP   Treat_V2   disc;
```

Fig. 12. Functions from lexicons and incomplete grammar

to use the preposition *pentru (for)*, which requires a noun in accusative. For example, the phrase *side effect of the drug* corresponds to *efect advers pentru medicamentul*, which means *side effect for the drug*.

Nonetheless, these problems, particular to the chosen language, are less relevant for the current goal, which is to test the possibility to add a new language. The incomplete grammar covers the majority of the functions from our CNL. This justifies the conclusion that addition of other languages requires effort mostly only for the translation of the lexicon, which can be done with simple dictionaries, except for the names of the queried resources.

4.1. The Romanian names for human diseases

Apart from having phrases of common sense knowledge expressed in different languages, an important issue in building multilingual systems is the domain specific terminology. When there are structured versions of the terminology in different languages, the problem is easier to solve. For example, Medical Dictionary for Regulatory Activities (MedDRA) is a multilingual terminology developed in order to provide a single standardized international medical terminology which can be used for regulatory communication and evaluation of data pertaining to medicinal products for human use. Unfortunately, Romanian is not included in the set of used languages.

However, based on existing classifications of disorders as OMIM (Online Mendelian Inheritance in Man), ICD (International Classification of Diseases) and Orphanet, we investigated a solution to build a Romanian lexicon for the named diseases from Diseasome.

OMIM captures the relation between genes and disorders. Each disorder has an OMIM code. This catalog is continuously updated. The initial bipartite graph from Diseasome was build on the OMIM version from 2005 [11], while the 2012 version of Diseasome was extended with drugs.

ICD-10 classification is the 10th revision of the International Classification of Diseases and Related Health Problems. There is a Romanian version of the Australian version ICD-10-AM, that is used in Romanian hospitals. Diagnosis Related Group (DRG)[1] provides for an application that includes this classification.

Orphanet is a portal for rare diseases, led by a consortium of around 40 countries. One of its freely accessible services is a classification of diseases elaborated using existing published expert classifications[2]. The included alignments between disorders and external terminologies or resources, as OMIM, ICD10, MeSH (Medical Subject Headings), UMLS (Unified Medical Language System) and MedDRA, are characterized in order to specify if the terms are perfectly equivalent (exact mappings) or not. There are versions for 7 languages, but again Romanian is not included.

Bio2RDF is a project that aims at providing Linked Data for the Life Sciences [4] and includes many medical resources formalized with RDF. DBpedia also contains information for diseases, including OMIM, MeSH or ICD identifiers, and names in different languages. One way to use it for the task of building Romanian lexicon is to rely on a mapping between English and Romanian names without the use of ICD classification. If in case of other languages this could be an acceptable solution, in case of Romanian, the small number of DBpedia resources in this language make it inappropriate. The alternative is to rely on DBpedia for identification of ICD codes from OMIM codes or names, followed by the use of the Romanian version for ICD-10AM. A first problem is the incompleteness of the OMIM and ICD codes in DBpedia. Another problem is the inconsistency with the referred classifications. For example, for *Acute myeloid leukemia*, DBpedia gives the OMIM code *602439*, which in OMIM is moved to 601626.

[1]`www.drg.ro`
[2]`www.orphadata.org/cgi-bin/inc/product1.inc.php`

Table 6
Romanian terms for diseases

Label in Diseasome	ICD10	Romanian term
beta-ureidopropionase deficiency	E79.8 (NTBT)	(eng)+alte tulburari de metabolism al purinelor si pirimidinei
abetalipoproteinemia	E87.6 (NTBT)	(eng)+deficit în lipoproteine
aplasia of lacrimal and salivary glands, 180920	Q38.4 (ND)	malformatii congenitale ale glandelor si canalelor salivare
hepatocellular carcinoma	C22.0 (E)	carcinom al celulei hepatice

With this analysis, the current solution for building the Romanian lexicon for diseases follows the next steps:

1. the labels of the diseases are obtained from Diseasome
2. the ICD code is searched in the Linked Data version of OMIM from Bio2RDF. The search is done either on the disease's label or on the OMIM code extracted from the label (in case it is included). The used endpoint is `http://cu.omim.bio2rdf.org/sparql`.
3. the ICD code is searched in Orphanet, again based on the name or the OMIM code. In case it is found, the type of the mapping is extracted, too.
4. the Romanian terms are extracted from the Romanian classification ICD-10-AM based on the identified ICD code. If the ICD code was obtained at *step 2*, the mapping is considered exact. Otherwise, the type of Orphanet mapping is analyzed, and in case it is an exact mapping, only the Romanian term is kept. In case the mapping is not exact, the initial English label is concatenated to the Romanian term.

Table 6 includes some examples of obtained terms. For the disease with OMIM code 180920, Orphanet mentions two mappings to ICD classified disorders, both having the status of *not decided.*

Results The steps 2 and 3 are alternatives for finding the ICD code. None of them is complete, and their combination is also incomplete. From 4213 different Diseasome resources, using only Orphanet, 1210 Romanian terms were found, including all the mapping types, not only the exact mappings. By using also OMIM Bio2RDF, the number increases to 1815 terms. Their correctness relies solely on the existing data in the queried resources. It must be emphasized that the process of building the lexicon can be improved through i) a more detailed filtering by name of the disease in Orphanet, since for example, in Diseasome, *type ii* phrase is used and in Orphanet it appears as *type 2*; ii) the extensive use of synonyms of diseases from Orphanet. Medical competence is needed to validate the resulted lexicon and to solve mappings different than the exact one.

A question parsed and linearized with GFMed grammars that include the generated lexicon for Romanian terms of diseases in shown in Figure 13. The Java GUI of GFMed is meant only for testing the grammars combined with the pre- and post-processing steps. For the end user, incremental parsing and completion as in the fridge magnet type interface included in GF are appropriate.

5. Related work

With the recent boost in available linked data and ontologies, the interest in extending the lexical context of ontologies increased as well [1]. A recent result in extending ontologies with richer lexical layer is the ontology-lexicon model *lemon* [8],[15]. This model proposes design patterns for the most common lexicalizations of labels from ontologies. The model was used in a manual approach of building an ontology-derived lexicon for DBpedia [18]. The building process consists of creating descriptions of verbalizations for classes and properties from ontologies. A significant part of DBpedia ontology was covered, 98% of classes and 20% of properties. Common with this approach, GFMed grammars are built manually, starting from the schemas of datasets to be queried. Patterns are identified in our approach starting from DL constructors, mainly restrictions on properties. The patterns are described directly in GF, based on our own SPARQL resource library. The GF functions correspond to DL constructors, facilitating their composition in similar way to DL.

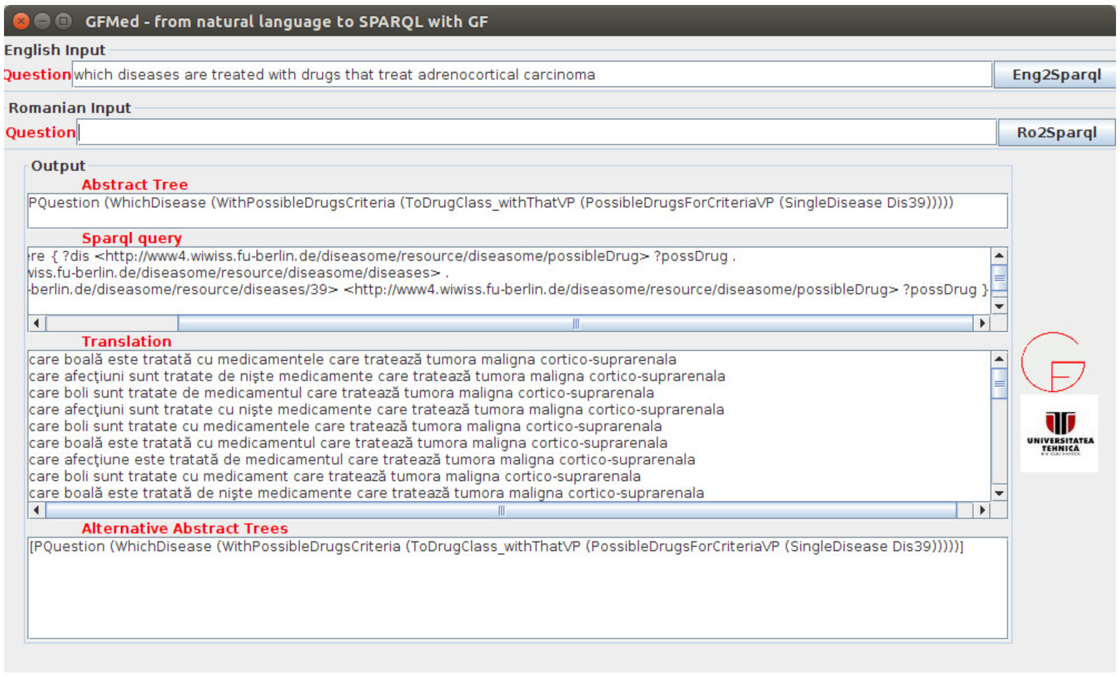
Manual development of the grammars strongly restricts the scalability of our approach. The semantic of the aimed linked data, biomedical data from Diseasome, Sider and DrugBank, is narrower compared to DBpedia. Nevertheless, the required precision in tackling medical data can be obtained with a manual approach. A very recent result in automatic derivation of lexicons in *lemon* format is described in [21].

SQUALL [10] is another controlled natural language that allows SPARQL queries and updates and it relies on Montague grammars. Unlike SQUALL, GFMed is appropriate for multilingual development due to the fact that is a controlled natural language built with GF. GF was previously used in multilingual systems for querying linked data in [7], [6], [5]. Compared to cultural heritage linked data, biomedical domain calls for high composability of the recognized expressions due to the strong relations between the involved entities, like drugs and diseases.

Another CNL for biomedical domain was created with Attempto Controlled English (ACE) for stating facts about interaction between proteins [13]. In contrast to this, GFMed CNL is a querying language, and the queried data are Linked Data, so it requires a translation of the identified meaning to SPARQL. The transitions between concrete syntax, semantic and then back to concrete syntax are easily captured with GF concrete and abstract grammars. Furthermore, the CNL from [13] is restricted to English, while GFMed includes also a controlled natural language for Romanian.

An incremental built of queries is described in [24]. Relevant concepts and properties are identified at each step and the user can choose one. The use of the ontology is the shared point with our approach, but in our case the user does not interact with the ontology and it uses natural language to express his needs.

From a completely different perspective, [22] and [25] propose learning and pattern matching for querying linked data in natural language. Ambiguity and named entity recognition are not easy to tackle withing these approaches, unlike the case of controlled natural languages. But their important advantage is scalability and domain independence. Nevertheless, if we consider Romanian instead of English as querying language, the limited



which diseases are treated with drugs that treat adrenocortical carcinoma

```
PQuestion (WhichDisease (WithPossibleDrugsCriteria (ToDrugClass_withThatVP (PossibleDrugsForCriteriaVP (SingleDisease Dis39)))))
```

```
re { ?dis <http://www4.wiwiss.fu-berlin.de/diseasome/resource/diseasome/possibleDrug> ?possDrug .
viss.fu-berlin.de/diseasome/resource/diseasome/diseases> .
berlin.de/diseasome/resource/diseases/39> <http://www4.wiwiss.fu-berlin.de/diseasome/resource/diseasome/possibleDrug> ?possDrug }
```

```
care boală este tratată cu medicamentele care tratează tumora maligna cortico-suprarenala
care afecţiuni sunt tratate de nişte medicamente care tratează tumora maligna cortico-suprarenala
care boli sunt tratate de medicamentul care tratează tumora maligna cortico-suprarenala
care afecţiuni sunt tratate cu nişte medicamente care tratează tumora maligna cortico-suprarenala
care boli sunt tratate cu medicamentele care tratează tumora maligna cortico-suprarenala
care boală este tratată cu medicamentul care tratează tumora maligna cortico-suprarenala
care afecţiune este tratată de medicamentul care tratează tumora maligna cortico-suprarenala
care boli sunt tratate cu medicament care tratează tumora maligna cortico-suprarenala
care boală este tratată de nişte medicamente care tratează tumora maligna cortico-suprarenala
```

```
[PQuestion (WhichDisease (WithPossibleDrugsCriteria (ToDrugClass_withThatVP (PossibleDrugsForCriteriaVP (SingleDisease Dis39)))))]
```

Fig. 13. An example of the multilingual version for GFMed, with Romanian term for disorder

resources existing for processing this language, hamper the application of many approaches based on learning and pattern matching. In the same time, they sustain approaches based on GF and its resources for Romanian.

6. Conclusion

A controlled natural language for querying biomedical linked data was introduced. The language is able to cover questions over more datasets, complex questions with different linguistic structures, and questions that involve lists and free text. It is built within Grammatical Framework by following a methodology based on DL constructors. The functions defined in GFMed's grammar are highly composable, due to their relation with DL constructors. This relation, together with the fact that the main categories of the abstract grammar map to the types of the main entities from the queried datasets, makes possible extension of the queried datasets with, for example, linked data about medical publications. A general GF resource for SPARQL was also introduced. The steps followed in building the language are not specific to the biomedical area. We consider that porting to a different domain is facilitated by three elements: (1) the split between criteria for a class and the class, with criteria expressed with different syntactic categories (2) the transformation functions, either from one criteria to a class, either from one dataset to another, (3) the operators from SPARQL resource. In the same time, the manual character of the language building process limits the scalability and claims for future work in automatic derivation of the GF functions from ontologies extended with lexical layer.

The proposed system addresses also multilinguality. Romanian was investigated near English as natural language. In order to obtain Romanian terms for diseases from Diseasome, a method based on more international classifications was analyzed, employing mainly the resources which follow the principles of linked data.

References

[1] K. Angelov and R. Enache. Typeful ontologies with direct multilingual verbalization. In M. Rosner and N. Fuchs, editors, *Controlled Natural Language*, volume 7175 of *Lecture Notes in Computer Science*, pages 1–20. Springer Berlin Heidelberg, 2012.

[2] K. Angelov and A. Ranta. Implementing controlled languages in gf. In N. Fuchs, editor, *Controlled Natu-*

ral Language, volume 5972 of *Lecture Notes in Computer Science*, pages 82–101. Springer Berlin Heidelberg, 2010.

[3] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, New York, NY, USA, 2003.

[4] F. Belleau, M.-A. Nolin, N. Tourigny, P. Rigault, and J. Morissette. Bio2rdf: Towards a mashup to build bioinformatics knowledge systems. *J. of Biomedical Informatics*, 41(5):706–716, Oct. 2008.

[5] M. Damova, D. Dannells, and R. Enache. Multilingual retrieval interface for structured data on the web. In *Proceedings of NLIWod 2014, ISWC'2014, Trento, Italy*, 2014.

[6] M. Damova, D. Dannells, M. Mateva, R. Enache, and A. Ranta. Natural language interaction with semantic web knowledge bases and linked open data. In *Towards multilingual Semantic Web*, pages 211–226. Springer, 2014.

[7] D. Dannells, A. Ranta, R. Enache, M. Damova, and M. Mateva. Multilingual Access to Cultural Heritage Content of the Semantic Web. In *Language Technologies for Cultural Heritage Workshop at ACL'2013*, 2013.

[8] B. Davis, R. Enache, J. van Grondelle, and L. Pretorius. Multilingual verbalisation of modular ontologies using gf and lemon. In T. Kuhn and N. Fuchs, editors, *Controlled Natural Language*, volume 7427 of *Lecture Notes in Computer Science*, pages 167–184. Springer Berlin Heidelberg, 2012.

[9] R. Enache, A. Ranta, and K. Angelov. An open-source computational grammar for romanian. In A. Gelbukh, editor, *Computational Linguistics and Intelligent Text Processing*, volume 6008 of *Lecture Notes in Computer Science*, pages 163–174. Springer Berlin Heidelberg, 2010.

[10] S. Ferr. Squall: A controlled natural language as expressive as sparql 1.1. In E. Mtais, F. Meziane, M. Saraee, V. Sugumaran, and S. Vadera, editors, *Natural Language Processing and Information Systems*, volume 7934 of *Lecture Notes in Computer Science*, pages 114–125. Springer Berlin Heidelberg, 2013.

[11] K.-I. Goh, M. E. Cusick, D. Valle, B. Childs, M. Vidal, and A.-L. Barabsi. The human disease network. *Proceedings of the National Academy of Sciences*, 104(21):8685–8690, 2007.

[12] T. Kuhn. A survey and classification of controlled natural languages. *Computational Linguistics*, 40(1):121–170, March 2014.

[13] T. Kuhn, L. Royer, N. Fuchs, and M. Schrder. Improving text mining with controlled natural language: A case study for protein interactions. In U. Leser, F. Naumann, and B. Eckman, editors, *Data Integration in the Life Sciences*, volume 4075 of *Lecture Notes in Computer Science*, pages 66–81. Springer Berlin Heidelberg, 2006.

[14] V. Lopez, V. Uren, E. Motta, and M. Pasin. AquaLog: An ontology-driven question answering system for organizational semantic intranets. *Web Semantics: Science, Services and Agents on the World Wide Web*, 5(2):72 – 105, 2007. Software Engineering and the Semantic Web.

[15] J. McCrae, D. Spohr, and P. Cimiano. Linking lexical resources and ontologies on the semantic web with lemon. In *Proceedings of the 8th Extended Semantic Web Conference on The Semantic Web: Research and Applications - Volume Part I*, ESWC'11, pages 245–259, Berlin, Heidelberg, 2011. Springer-Verlag.

[16] A. Ranta. The GF Resource Grammar Library. *Linguistic Issues in Language Technology*, 2(2), 2009.

[17] C. Unger, C. Forascu, V. Lopez, A.-C. N. Ngomo, E. Cabrio, P. Cimiano, and S. Walter. Question answering over linked data (qald-4). In *Working Notes for CLEF 2014 Conference, Sheffield, UK*, pages 1172–1180, 2014.

[18] C. Unger, J. McCrae, S. Walter, S. Winter, and P. Cimiano. A lemon lexicon for dbpedia. Proceedings of 1st International Workshop on NLP and DBpedia, co-located with the 12th International Semantic Web Conference (ISWC 2013), October 21-25, Sydney, Australia. CEUR Workshop Proceedings, 2013.

[19] J. van Grondelle and C. Unger. A three-dimensional paradigm for conceptually scoped language technology. In P. Buitelaar and P. Cimiano, editors, *Towards the Multilingual Semantic Web*, pages 67–82. Springer Berlin Heidelberg, 2014.

[20] B. Varga, A. D. Trambitas-Miron, A. Roth, A. Marginean, R. R. Slavescu, and A. Groza. LELA - A natural language processing system for romanian tourism. In *Proceedings of the 2014 Federated Conference on Computer Science and Information Systems, Warsaw, Poland, September 7-10, 2014*, pages 281–288, 2014.

[21] S. Walter, C. Unger, and P. Cimiano. Atoll - a framework for the automatic induction of ontology lexica. *Data & Knowledge Engineering*, 94, Part B(0):148 – 162, 2014.

[22] K. Xu, S. Zhang, Y. Feng, and D. Zhao. Answering natural language questions via phrasal semantic parsing. In C. Zong, J.-Y. Nie, D. Zhao, and Y. Feng, editors, *Natural Language Processing and Chinese Computing*, volume 496 of *Communications in Computer and Information Science*, pages 333–344. Springer Berlin Heidelberg, 2014.

[23] J. M. Zelle and R. J. Mooney. Learning to parse database queries using inductive logic programming. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence - Volume 2*, AAAI'96, pages 1050–1055. AAAI Press, 1996.

[24] G. Zenz, X. Zhou, E. Minack, W. Siberski, and W. Nejdl. From keywords to semantic queries - incremental query construction on the semantic web. *Journal Web Semantic*, 7(3):166–176, 2009.

[25] L. Zou, R. Huang, H. Wang, J. X. Yu, W. He, and D. Zhao. Natural language question answering over rdf: A graph data driven approach. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, SIGMOD '14, pages 313–324, New York, NY, USA, 2014. ACM.