Semantic Web 0 (2016) 1–17
IOS Press

Question Answering over Biomedical Linked Data with Grammatical Framework

Editor(s): Christina Unger, Bielefeld University, Germany; Axel-Cyrille Ngonga Ngomo, University of Leipzig, Germany; Philipp Cimiano, Bielefeld University, Germany; Sören Auer, University of Bonn & Fraunhofer IAIS, Germany; George Paliouras, NCSR Demokritos, Greece
Solicited review(s): Kaarel Kaljurand, Nuance Communications, Austria; Dana Dannélls, University of Gothenburg, Sweden; Christina Unger, Bielefeld University, Germany; Anonymous reviewer

Anca Marginean [a,*]
[a] *Department of Computer Science, Technical University of Cluj Napoca, 401446, Cluj-Napoca, Romania*
E-mail: anca.marginean@cs.utcluj.ro

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, besides 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 amount of data from various domains such as government, 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 [15], 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 is described by means of large terminologies. Connections between datasets are encouraged by the very essence of the concept of linked data. Consequently, the lack of 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. At the same time, building a restricted natural language 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. The 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

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

used. With an increasing popularity, controlled natural languages (CNLs) aim at giving an intuitive representation of formal representations with a trade-off between precision of formal languages and ambiguity, but high expressivity, of natural languages. A comparative analysis of controlled natural languages can be found in [13].

In this line, we propose a system, GFMed, based on application grammars manually built with Grammatical Framework (GF) [17]. All GFMed resources are available at `http://cs-gw.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 grammars. This relation is also analyzed in [8], [12], while GF is also used for linked data in [5], [7]. The targeted linked data is 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 challenge* (QALD-4) [18]. GFMed's CNL consists of a grammar for question answering that covers questions from QALD benchmark and that is evaluated on them.

DrugBank is part of the project Bio2RDF [4] and contains chemical, pharmacological and pharmaceutical data about drugs with comprehensive drug target information. Diseasome provides information about human disease-gene networks, 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 *owl: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.

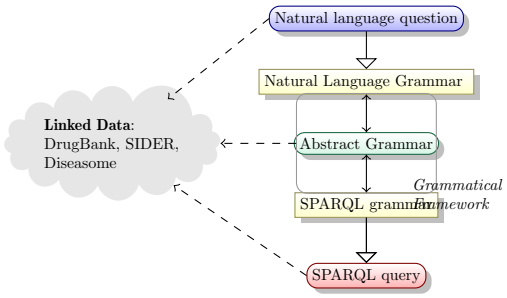

Fig. 1. Querying linked data with natural language

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.

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 [17]. 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 syn-

```
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}};
mkNotFilter : Triplet -> Statement=
     \t -> {s="FILTER NOT EXISTS {" ++ t.subj ++ t.prop.s ++ t.obj ++ "}"; 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 which create graph patterns

```
mkEmptyTriplet : PropertyT -> Triplet =
 \p -> {subj=""; prop=p; obj=""};
addSubj : Str -> Triplet -> Triplet=
 \s, vp -> {subj=s; prop=vp.prop; obj=vp.obj};
```

Fig. 3. Operators which create a `Triplet`

tax of the CNL is defined by the concrete grammars for natural languages. In case of the SPARQL concrete grammar, since SPARQL is a formal language, the patterns described in this grammar could also be considered as the 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 preferred 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 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. We covered basic elements of SPARQL 1.1, with a limited support for term constraints, aggregates and negation, and no support for property paths of length different from 1, optional graph pattern, or assignment.

Two main types were defined: `Triplet` and `Statement`. The type

$$\texttt{Triplet} : \texttt{Type} = \{\texttt{subj},\ \texttt{obj} : \texttt{Str};\ \ \texttt{prop} : \texttt{PropertyT}\}$$

is a structure with three components: two string components for the subject and the object, and a structure for the property. The structure

$$\texttt{PropertyT} : \texttt{Type} = \{\texttt{s} : \texttt{Str};\ \texttt{vt} : \texttt{ValueType}\}$$

for the property includes two fields: a string for the name of the property and a field of the enumerated type `ValueType`. This field stores the type of the values of the property. The enumerated type has two values: `String` and `Number`. The type of the property is important in using the correct operator inside of `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

$$\texttt{Statement} : \texttt{Type} = \{\texttt{s} : \texttt{Str};\ \texttt{extra} : \texttt{Str};\ \texttt{aggreg} : \texttt{AggregationType}\}$$

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 aggregates and solution modifiers, such as `GROUP BY`, `ORDER BY` or `LIMIT` are required. The field `extra`

contains the expression for the solution modifiers in case they exist. When two statements are concatenated, their `extra` fields are concatenated.

A set of operators were defined on these types. There are operators that create incomplete triple patterns, such as `mkEmptyTriplet` or `addSubj` from 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 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 to a string that includes the variables to appear in the query results and to a graph pattern. The final string is built according to whether solution modifiers are part of the query or not.

2.2. Brief description of GFMed CNL

The GFMed CNL aims at 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: i) by their name (e.g. *lepirudin*, *rickets*, *fever*), ii) by a named category to which they belong (e.g. *approved drugs*, *metabolic class*), or iii) by a category defined through a certain property, as the ones mentioned in Table 1 (e.g. *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 entity references are obtained from the three datasets and are part of the lexicon. For the third type, a set of about forty words is 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*. The 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 the three targeted datasets are not all providing for DLs descriptions or ontologies, approaching them from a DL 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

```
mkQuery : Str -> Statement -> Str=
    \var, b -> case b.aggreg of {
                    yes =>"select "++ var ++ "where {" ++ b.s ++ "}" ++ b.extra;
                    no => "select distinct" ++ var ++ "where {" ++ b.s ++ "}" };
mkInnerQuery : Str -> Statement -> Str=
    \var, b-> "{ select " ++ var ++ "where {"++ b.s++"}}";
mkAskQuery: Statement -> Str=
    \b -> "ask{" ++ b.s ++ "}";
```

Fig. 4. Operators which create SELECT and ASK queries

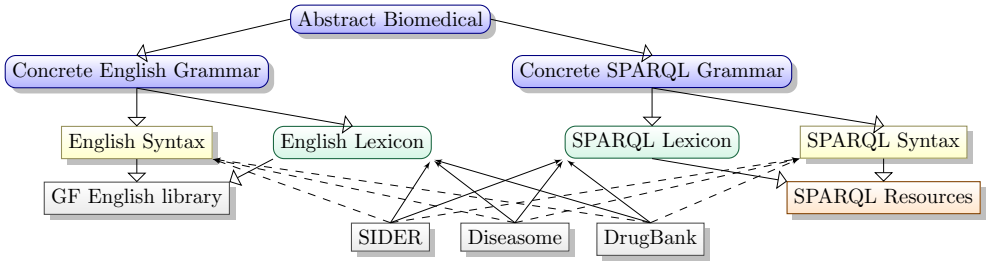

Fig. 5. The main grammars and resources used by GFMed

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`}
`DrugBankProperty`	CN	`MeltingPoint, GeneralFunction, DosageForm, PredictedWaterSolubility, Manufacturers, Indication, Target, Interacting, FoodInteraction,`...
`SIDERProperty`	CN	SideEffect
`DiseasomeProperty`	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 a `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 ...`)

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 `SIDERDrug` 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 filled in during the linearization of different restriction functions. One of them must be a resource or a previously introduced variable. In the later case there must exist a triplet where this variable is bound. The other one is filled in 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.

Table 3
Examples of class expressions and assertions in Diseasome

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

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.

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 ∃`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, ∃`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

```
WithPossibleDrug : DrugClass -> DiseaseClass;                                  - - diseases treated with D
WithPossibleDrugCriterion : DrugClass -> CriterionForDiseaseClass;             - - treated with D
WithPossibleDrugCriterionClSlash : DrugClass -> CriterionForDiseaseClassClSlash;   - - D is used for
WithPossibleDrugCriterionNP : DrugClass -> CriterionForDiseaseClassNP;         - - D as possible drug
WithPossibleDrugCriterionAdv : DrugClass -> CriterionForDiseaseClassAdv;       - - with D as possible drug
WithPossibleDrugCriterionRCl : DrugClass -> CriterionForDiseaseClassRCl;       - - whose possible drug is D
WithPossibleDrugCriterionRCl_VP : CriterionForDrugClassVP -> CriterionForDiseaseClassRCl;
                                                    - - whose possible drug interacts with D
WithPossibleDrugCriterionRCl_Adj : CriterionForDrugClassAdj -> CriterionForDiseaseClassRCl;
                                                    - - whose possible drug is associated with D
```

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

```
WithPossibleDrug[Criterion] 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`

`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 corresponding DL expression ensures composability of functions from the concrete grammars based on the their types.

Each DL constructor can be expressed in natural language in more than one way, 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 the 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. The first 6 functions from Figure 6 model restrictions 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 restrictions on the inverse property of `PossibleDrug` are defined. They allow for 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 parsed as `(WithPossibleDrugsCriterion (SingleDrug DB00060))`, and its linearization is the SPARQL 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 ques-

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

Fig. 8. Functions for restrictions on datatype properties

tion, 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 solubility of XX`. This is parsed 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 properties, are not covered in the current version of the 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 treats number of properties, or `HighestValue`, and `LowestValue` which deal with values of properties. For example, `the least common chromosome location` is interpreted as `[LowestNumberValue ChromosomeLocation]`, where `ChromosomeLocation` is a `DrugBank` property. SPARQL aggregation and solution modifiers are used for these constructors, as it can be seen in the SPARQL linearization of the function `HighestNumber`:

```
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"};
```

(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

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

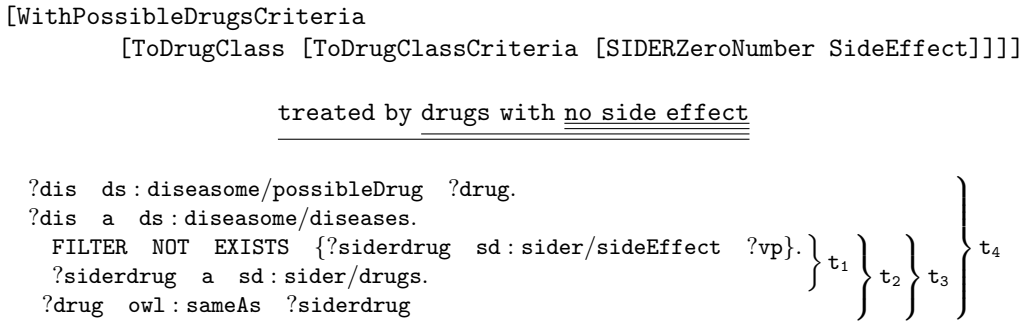

Fig. 10. Abstract tree and concrete linearizations in English and SPARQL for an example with two transformation functions:

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 from 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, the phrase `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 linearizations 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 the function `ToSIDERDrugClass_Without` is the function `mkSIDERDrugStatement`. The clear separation between domain-dependent and -independent fragments from the majority of the SPARQL linearizations facilitates 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, i.e. based on `owl:sameAs`. 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 `owl:sameAs` 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 to one class, one criterion, or to a list of classes or criteria for classes (Figure 11). The questions mostly deal with resource classes and criteria for these classes and

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

Fig. 11. Functions for queries

less with properties. An exception to this rule is the function `WhatPropertyValue`. This function applies to arguments of type `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]]]`.

The advantage of the described approach is the flexibility in the composition of trees and 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 [DDrugClass-CriterionVP t_2]]

where

t_2=[AdjToDrugClass [PossibleDrugs-ForCriterionAdj t_1]]

is the tree for the class of drugs that are used for diseases in t_1. The tree

t_1=[ToDiseaseClass [WithPossibleDrugs-Criterion [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 to a query whose result consists of 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 translating SPARQL queries into natural language questions, for the phrases that are not altered by the pre/post-processing of GFMed. For example, the query

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

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) ? ⋃_i {SPARQLLin_i, abstractTree_i}
        : FAIL
end function
```

is translated into several NL expressions, such as `give me possible drugs for breast cancer`, `which drug treats breast cancer`, `list the drugs that are used for breast cancer`.

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 interfaces with the 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 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 at 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 that will be explored as future work is 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).

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 of GFMed in Task 2 of QALD-4

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

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 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 QALD-4. The results on the test questions are shown in 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 from `brand names`.

```
- - 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
WithPossibleDrugsCriteria dc =
      mkAP Treated_A2 dc;
PossibleDrugsForCriterionAdj disc =
      mkAP UsedFor_A2 disc;
```

Fig. 12. Functions from lexicons and incomplete grammar

4. First steps towards a multilingual system

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

4.1. Multilingual grammar

The default GF mechanism for building multilingual application grammars 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.

In Figure 12, two two-place adjectives are defined in the English and the Romanian lexicons. In the incomplete grammar, they are used in linearizations of two criteria for either the property `possibleDrugs` or its inverse. Figure 12 also contains an example of a property from DrugBank, `Route of Elimination`, as it is defined in the Romanian lexicon.

For efficiency reasons, some constructions were not included in the GF Romanian library [9]. The main problems in using the 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 (a, al, ai, ale, alor). 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 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 the addition of other languages requires effort mostly only for the translation of the lexicon, which can be done with simple dictionaries, except for the domain-dependent names of the queried resources.

4.2. 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 built 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.

[1] `www.drg.ro`

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

Bio2RDF [4] is a project that aims at providing linked data for the Life Sciences and includes many medical resources formalized in 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 a Romanian lexicon is to rely on a mapping between English and Romanian names without the use of ICD classification. For other languages this could be an acceptable solution, but 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 of 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`.

With this analysis, the current solution for building the Romanian lexicon for diseases follows the following 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 example, for the disease with OMIM code 180920, Orphanet mentions two mappings to ICD-

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

classified disorders, both having the status of *not decided* (ND), and one of them has Q38.4 code.

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 the name of the disease in Orphanet, since for example, in Diseasome, the phrase *type ii* 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 from 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, the fridge magnet type interface included in GF might be appropriate, since it supports incremental parsing and completion.

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 a richer lexical layer is the ontology-lexicon model *lemon* [8],[16]. 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 [19]. 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 the classes and 20% of the properties. Similat to 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 a similar way to DL.

Manual development of the grammars strongly restricts the scalability of our approach. The semantics of the targeted 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 the automatic derivation of lexicons in *lemon* format is described in [21].

SQUALL [10] is another controlled natural language that allows for a translation into SPARQL queries, relying 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, the 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 the biomedical domain was created with Attempto Controlled English (ACE) for stating facts about interaction between proteins [14]. In contrast to this, GFMed CNL is a querying language, and the queried data is linked data, so it requires a translation of the identified meaning to SPARQL. The transitions between concrete syntax, semantics and then back to concrete syntax are easily captured with GF concrete and abstract grammars. Furthermore, the CNL from [14] is restricted to English, while GFMed in-

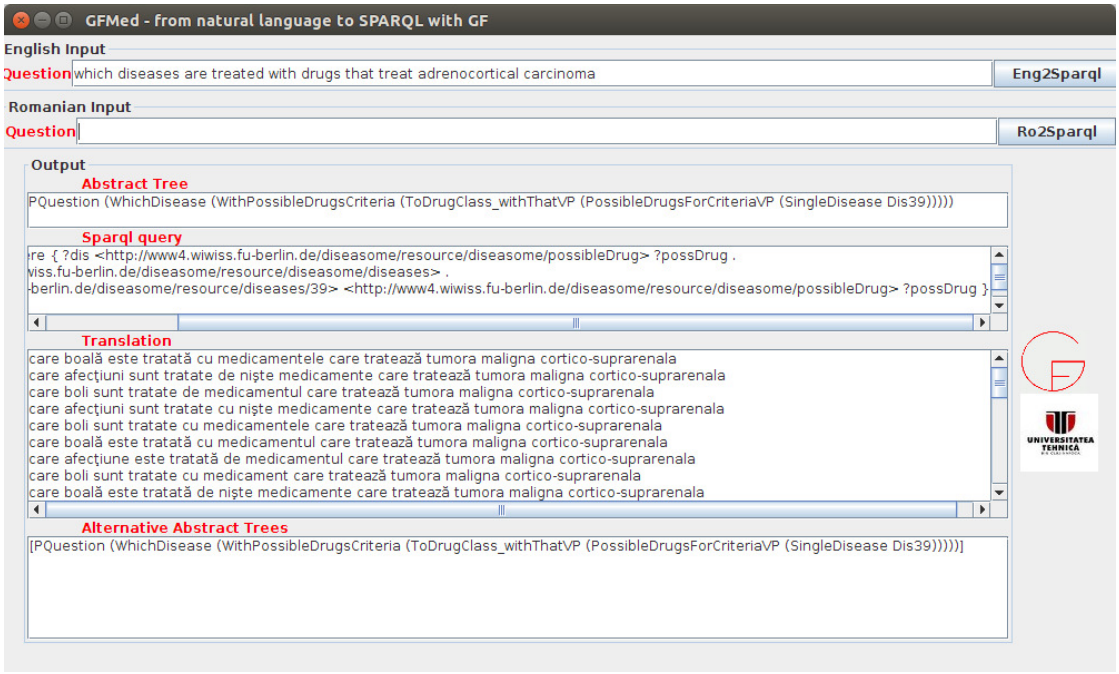

Fig. 13. An example of the multilingual version of GFMed, with the Romanian term for disorder

cludes also a controlled natural language for Romanian.

An incremental construction 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 but 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 within 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 existing resources for processing this language hamper the application of many approaches based on learning and pattern matching and 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 with 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 an extension of the queried datasets with, for example, linked data about medical publications possible. 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: i) the split between criteria for a class and the class, with criteria expressed with different syntactic categories, ii) the transformation functions, either from one criteria to a class, or from one dataset to another, iii) the operators from the SPARQL resource. In the same time, the manual character of the language building process limits the scalability and claims for future work in the automatic derivation of GF functions from ontologies extended with lexical layer.

The proposed system also addresses multilinguality. Romanian was investigated in addition to English. 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. E. Fuchs, editors, *Controlled Natural Language: Second International Workshop, CNL 2010, Marettimo Island, Italy, September 13-15, 2010. Revised Papers*, pages 1–20, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. doi:10.1007/978-3-642-31175-8_1.

[2] K. Angelov and A. Ranta. Implementing Controlled Languages in GF. In N. E. Fuchs, editor, *Controlled Natural Language: Workshop on Controlled Natural Language, CNL 2009, Marettimo Island, Italy, June 8-10, 2009. Revised Papers*, pages 82–101, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. doi:10.1007/978-3-642-14418-9_6.

[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, 2003.

[4] F. Belleau, M.-A. Nolin, N. Tourigny, P. Rigault, and J. Morissette. Bio2RDF: Towards a Mashup to Build Bioinformatics Knowledge Systems. *Journal of Biomedical Informatics*, 41(5):706–716, 2008. http://dx.doi.org/10.1016/j.jbi.2008.03.004.

[5] M. Damova, D. Dannélls, and R. Enache. Multilingual Retrieval Interface for Structured Data on the Web. In *Workshop of Natural Language Interfaces for Web of Data (NLIWoD), International Semantic Web Conference (ISWC), Trentino, Italy*, 2014.

[6] M. Damova, D. Dannélls, R. Enache, M. Mateva, and A. Ranta. Multilingual Natural Language Interaction with Semantic Web Knowledge Bases and Linked Open Data. In P. Buitelaar and P. Cimiano, editors, *Towards the Multilingual Semantic Web: Principles, Methods and Applications*, pages 211–226. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014. doi:10.1007/978-3-662-43585-4_13.

[7] D. Dannélls, A. Ranta, R. Enache, M. Damova, and M. Mateva. Multilingual Access to Cultural Heritage Content of the Semantic Web. In *Proceedings of the 7th Workshop on Language Technology for Cultural Heritage, Social Sciences, and Humanities*, pages 107–115, Sofia, Bulgaria, 2013. Association for Computational Linguistics.

[8] B. Davis, R. Enache, J. Grondelle, and L. Pretorius. Multilingual Verbalisation of Modular Ontologies Using GF and lemon . In T. Kuhn and N. E. Fuchs, editors, *Proceedings of Controlled Natural Language: Third International Workshop, CNL 2012, Zurich, Switzerland, August 29-31, 2012*, pages 167–184, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. doi:10.1007/978-3-642-32612-7_12.

[9] R. Enache, A. Ranta, and K. Angelov. An Open-Source Computational Grammar for Romanian. In A. Gelbukh, editor, *Proccedings of Computational Linguistics and Intelligent Text Processing: 11th International Conference, CICLing 2010, Iaşi, Romania, March 21-27, 2010*, pages 163–174, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. doi:10.1007/978-3-642-12116-6_14.

[10] S. Ferré. SQUALL: A Controlled Natural Language as Expressive as SPARQL 1.1. In E. Métais, F. Meziane, M. Saraee, V. Sugumaran, and S. Vadera, editors, *Proceedings of Natural Language Processing and Information Systems: 18th International Conference on Applications of Natural Language to Information Systems, NLDB 2013, Salford, UK, June 19-21, 2013*, pages 114–125, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. doi:10.1007/978-3-642-38824-8_10.

[11] K.-I. Goh, M. E. Cusick, D. Valle, B. Childs, M. Vidal, and A.-L. Barabási. The Human Disease Network. *Proceedings of the National Academy of Sciences*, 104(21):8685–8690, 2007. doi:10.1073/pnas.0701361104.

[12] J. 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: Principles, Methods and Applications*, pages 67–82. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014. doi:10.1007/978-3-662-43585-4_5.

[13] T. Kuhn. A Survey and Classification of Controlled Natural Languages. *Computational Linguistics*, 40(1):121–170, 2014. doi:10.1162/COLI_a_00168.

[14] T. Kuhn, L. Royer, N. E. Fuchs, and M. Schröder. Improving Text Mining with Controlled Natural Language: A Case Study for Protein Interactions. In U. Leser, F. Naumann, and B. Eckman, editors, *Proceedings of Data Integration in the Life Sciences: Third International Workshop, DILS 2006, Hinxton, UK, July 20-22, 2006*, pages 66–81, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. doi:10.1007/11799511_7.

[15] 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. http://dx.doi.org/10.1016/j.websem.2007.03.003.

[16] J. McCrae, D. Spohr, and P. Cimiano. Linking Lexical Resources and Ontologies on the Semantic Web with Lemon. In G. Antoniou, M. Grobelnik, E. Simperl, B. Parsia, D. Plexousakis, P. Leenheer, and J. Pan, editors, *Proceedings of The Semantic Web: Research and Applications: 8th Extended Semantic Web Conference, ESWC 2011, Heraklion, Crete, Greece, May 29-June 2, 2011*, pages 245–259, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. doi:10.1007/978-3-642-21034-1_17.

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

[18] C. Unger, C. Forascu, V. Lopez, A. N. Ngomo, E. Cabrio, P. Cimiano, and S. Walter. Question Answering over Linked Data (QALD-4). In L. Cappellato, N. Ferro, M. Halvey, and W. Kraaij, editors, *Working Notes for CLEF 2014 Conference, Sheffield, UK, September 15-18, 2014*, volume 1180 of *CEUR Workshop Proceedings*, pages 1172–1180, Aachen, Germany, Germany, 2014. CEUR-WS.org.

[19] C. Unger, J. McCrae, S. Walter, S. Winter, and P. Cimiano. A lemon lexicon for DBpedia. In H. Sebastian, F. Agata, B. Caroline, M. Pablo, and K. Dimitris, editors, *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*, pages 103–108, Aachen, Germany, Germany, 2013. CEUR-WS.org.

[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 M. Ganzha, L. Maciaszek, and M. Paprzycki, editors, *Proceedings of the 2014 Federated Conference on Computer Science and Information Systems, Warsaw, Poland, September 7-10, 2014*, pages 281–288. IEEE, 2014. doi:10.15439/2014F323.

[21] S. Walter, C. Unger, and P. Cimiano. ATOLL - A framework for the automatic induction of ontology lexica. *Data & Knowledge Engineering*, 94, Part B:148–162, 2014. http://dx.doi.org/10.1016/j.datak.2014.09.003.

[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, *Proceedings of Natural Language Processing and Chinese Computing: Third CCF Conference, NLPCC 2014, Shenzhen, China, December 5-9, 2014*, pages 333–344, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg. doi:10.1007/978-3-662-45924-9_30.

[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. *Web Semantics: Science, Services and Agents on the World Wide Web*, 7(3):166–176, 2009. http://dx.doi.org/10.1016/j.websem.2009.07.005.

[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*, pages 313–324, New York, NY, USA, 2014. ACM. doi:10.1145/2588555.2610525.