# Transforming Controlled Natural Language Biomedical Queries into Answer Set Programs

**Esra Erdem** and **Reyyan Yeniterzi**
Faculty of Engineering and Natural Sciences
Sabancı University
Orhanlı, Tuzla 34956 Istanbul, Turkey

## Abstract

We introduce a controlled natural language for biomedical queries, called BIOQUERYCNL, and present an algorithm to convert a biomedical query in this language into a program in answer set programming (ASP)—a formal framework to automate reasoning about knowledge. BIOQUERYCNL allows users to express complex queries (possibly containing nested relative clauses and cardinality constraints) over biomedical ontologies; and such a transformation of BIOQUERYCNL queries into ASP programs is useful for automating reasoning about biomedical ontologies by means of ASP solvers. We precisely describe the grammar of BIOQUERYCNL, implement our transformation algorithm, and illustrate its applicability to biomedical queries by some examples.

## 1 Introduction

The rapid increase in the popularity and usage of Web leads researchers to store data and make it publicly available in many ways. In particular, to facilitate access to its desired parts, it is stored in a structured form, like ontologies. These ontologies can be queried with an SQL-like formal query language. However, since these ontologies have been developed for and widely used by people that lacks the necessary knowledge in a formal query language, a simpler and more commonly known language is needed to represent queries. A natural language is the perfect answer, but ambiguities in its grammar and vocabulary make it difficult to automate reasoning about queries in natural language. Therefore, to represent queries, we consider a middle ground between these two options: a Controlled Natural Language (CNL).

A CNL is a subset of a natural language, with a restricted grammar and vocabulary, that overcomes the ambiguity of natural languages. Since we consider queries in a specific domain, namely biomedicine, and over specific sources of information, namely biomedical ontologies, a CNL designed and developed for reasoning about biomedical ontologies is sufficient to represent biomedical queries. Essentially, a CNL is a formal language but with a look of a natural language. Therefore, compared to a natural language, a CNL can be easily converted to some other formalisms. This allows us to use automated reasoners, specifically developed for such formalisms, to find answers to queries expressed in a CNL.

One such formalism is Answer Set Programming (ASP) (Baral, 2003). ASP is a new knowledge representation and reasoning paradigm which supports representation of defaults, constraints, preferences, aggregates, etc., and provides technologies that allow us to automate reasoning with incomplete information, and to integrate other technologies, like description logics reasoners and Semantic Web technologies. For instance, in (Bodenreider et al., 2008), the authors illustrate the applicability and effectiveness of using ASP to represent a rule layer that integrates relevant parts of some biomedical ontologies in RDF(S)/OWL, and to compute answers to some complex biomedical queries over these ontologies.

Although CNLs are appropriate for expressing biomedical queries, and methods and technologies



*Proceedings of the Workshop on BioNLP*, pages 117–124,
Boulder, Colorado, June 2009. 

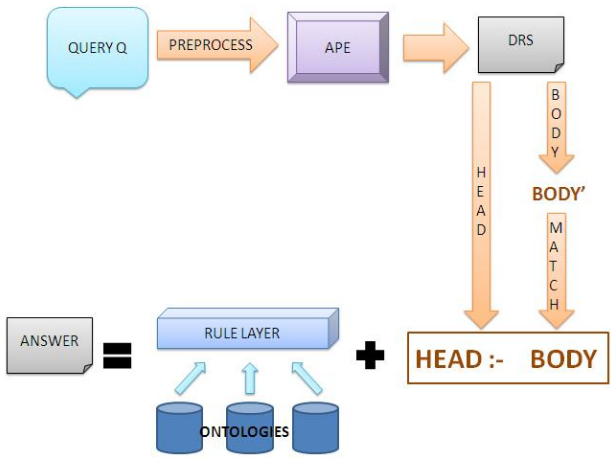


Figure 1: Architecture of the Overall System

of ASP are appropriate for automated reasoning about biomedical ontologies, there is no algorithm to convert a CNL biomedical query into a program. In (Bodenreider et al., 2008), biomedical queries are represented as programs in ASP; however, these programs are constructed manually. However, manually constructing ASP programs to represent biomedical queries is not only time consuming but also requires expertise in ASP. This prevents automating the whole process of computing an answer to a query, once it is given in a CNL.

In this paper, we design and develop a CNL (called BIOQUERYCNL) for expressing biomedical queries over some ontologies, and introduce an algorithm to convert a biomedical query expressed in this CNL into a program in ASP. The idea is to automatically compute an answer to the query using methods of (Bodenreider et al., 2008), once the user types the query. This idea is illustrated in Figure 1.

Similar approaches of using a CNL for querying ontologies have been investigated in various studies. For instance, (Bernstein et al., 2005) considers queries in the controlled natural language, Attempto Controlled English (ACE) (Attempto, 2008), and transforms them into queries in PQL (Klein and Bernstein, 2004) to be evaluated by a query engine. (Bernstein et al., 2006) presents a system that guides the user to write a query in ACE, and translates the query into SPARQL to be evaluated by the reasoner of JENA (Jena, 2008). On the other hand, (Kaufmann et al., 2006) transforms a given natural language query to a SPARQL query (using the Stanford Parser and WORDNET) to be evaluated by a reasoner like that of JENA. Our work is different from these studies in two ways: we consider queries over biomedical ontologies (thus different forms of queries, and vocabulary), and we transform a query into an ASP program to automate reasoning over a rule layer presented in ASP.

Transformations of natural language sentences into ASP has been studied in (Baral et al., 2008) and (Baral et al., 2007). In (Baral et al., 2008), the authors introduce methods to transform some simple forms of sentences into ASP using Lambda Calculus. In (Baral et al., 2007), the authors use C&C tools (CC, 2009) to parse the some forms of natural language input, and perform a semantic analysis over the parser output using BOXER (Boxer, 2009), to do reasoning in ASP. Our work is different in that we consider a CNL to express queries, and introduce a different method for converting CNL to a program in ASP, via Discourse Representation Structures (DRS) (Kamp, 1981).

In the rest of the paper, first we briefly discuss ASP with some examples (Section 2). Then we define the grammatical structure of BIOQUERYCNL and give some examples (Section 3). Next, we introduce our algorithm for transforming a BIOQUERYCNL query into an ASP program and explain it by an example (Section 4). We conclude with a discussion of challenges related to the implementation of our algorithm (Section 5) and other related problems that we are working on (Section 6).

## 2 Answer Set Programming

Answer Set Programming (ASP) (Lifschitz, 1999; Marek and Truszczyński, 1999; Niemelä, 1999; Baral, 2003) is a new knowledge representation and reasoning paradigm which supports representation of defaults, constraints, preferences, aggregates, etc., and provides technologies that allow us to automate reasoning with incomplete information, and to integrate other technologies, like description logics reasoners and Semantic Web technologies.

In ASP, knowledge is represented as a "program" (a finite set of "rules") whose meaning is captured by its models (called "answer sets" (Gelfond and Lifschitz, 1988)). Answer sets for a program can be computed by "answer set solvers" such as DLV

(DLV, 2009). Consider for instance the program:

```
gene_gene(``ADRB1'',``CHRM5'').
gene_gene(``CHRM1'',``CHRM5'').

chain(X,Y) :- gene_gene(X,Y).
chain(X,Y) :- gene_gene(Y,X).
chain(X,Y) :- gene_gene(X,Z), chain(Z,Y).
```

The first rule expresses that the gene ADRB1 interacts with the gene CHRM5. The second rule expresses that the gene CHRM1 interacts with the gene CHRM5. The third, the fourth, and the fifth rules express a chain of such interactions. In a rule containing `:-`, the left-hand-side of `:-` is called the *head* of the rule, the right-hand-side is called the *body* of the rule. Such a rule `p :- q, r.` is read as "`p` if `q` and `r`". Here the head atom is `p`, and the body atoms are `q` and `r`. The answer set for this program describes that there is a chain of interactions between CHRM1 and CHRM5, ADRB1 and CHRM5, and ADRB1 and CHRM1.

As mentioned above, the language of ASP is expressive enough to represent defaults, constraints, preferences, aggregates, etc.. For instance, the rule

```
treats_2diseases(R) :-
 #count{D:treats(R,D)}>=2, drug(R).
```

describes drugs `R` that treat at least 2 diseases.

## 3 A Controlled Natural Language for Biomedical Queries

We introduce a controlled natural language, called BIOQUERYCNL, to express biomedical queries, whose grammar is shown in Table 1. This grammar should be considered in connection with the given biomedical ontologies. The italic words in the grammar, for instance, represent the information extracted from the related ontologies. We call these italic words ontology functions; the detailed description of these functions are given in Table 2.

With BIOQUERYCNL, the users can ask simple queries, queries with nested relative clauses (with any number of conjunctions and disjunctions), and queries with cardinalities. Some sample queries are given below.

(Q1) Which symptoms are alleviated by the drug Epinephrine?

(Q2) What are the side-effects of the drugs that treat the disease Asthma?

(Q3) What are the genes that are related to the disease Asthma and are targeted by the drug Epinephrine?

(Q4) What are the symptoms of the diseases that are related to the gene ADRB1 or that are treated by the drug Epinephrine?

(Q5) Which genes are targeted by at least 2 drugs and are related to at most 3 diseases?

BIOQUERYCNL is a subset of Attempto Controlled English (ACE) (Attempto, 2008), which can represent a wide range of queries (Fuchs et al., 2008), specialized for biomedical ontologies.

## 4 Converting Controlled Natural Language Queries to Programs

We have implemented an algorithm, QUERY, presented in Algorithm 1, that obtains an ASP rule $Head \leftarrow Body$ from a query $Q$ expressed in BIOQUERYCNL, via transforming $Q$ into a DRS. We will explain the main steps of the QUERY algorithm by an example, considering query (Q4).

**Algorithm 1** QUERY($Q$)

**Input:** A query $Q$
**Output:** An ASP rule $Head \leftarrow Body$

1: $D :=$ Find the DRS of $Q$
2: $Head :=$ HEAD($D$)
3: $Body' :=$ BODY($D$)
4: $Body :=$ POSTPROCESSING($Body'$)
5: $return\ Head \leftarrow Body$

### 4.1 Transforming a CNL Query into DRS

Attempto Controlled English (ACE) text can be converted into Discourse Representation Structures (DRS) (Kamp, 1981) — a variant of the first-order logic that is used for the dynamic interpretation of natural language and systematic translation of natural language into logical form — without any ambiguity, using tools like Attempto Parsing Engine (APE). APE converts ACE text to DRS by an approach similar to (Blackburn and Bos, 2005), as explained in (Fuchs et al., 2008). For instance, APE transforms query (Q4) into the following DRS:

Table 1: The Grammar of BIOQUERYCNL

| | |
|---|---|
| QUERY → | YESNOQUERY \| WHQUERY QUESTIONMARK |
| YESNOQUERY → | DODOESQUERY \| ISAREQUERY |
| WHQUERY → | WHATQUERY \| WHICHQUERY |
| DODOESQUERY → | [ Do \| Does ] $Type()$ $Instance(T)$ PREDICATERELATION |
| ISAREQUERY → | [ Is \| Are ] $Type()$ $Instance(T)$ $Verb(T)$ |
| WHATQUERY → | What BE $Type()$ that PREDICATERELATION |
| WHATQUERY → | What BE OFRELATION that PREDICATERELATION |
| WHATQUERY → | What BE OFRELATIONINSTANCE that PREDICATERELATION |
| WHICHQUERY → | Which $Type()$ PREDICATERELATION |
| OFRELATION → | $Noun(T)$ of $Type()$ |
| OFRELATIONINSTANCE → | $Noun(T)$ of $Type()$ $Instance(T)$ |
| PREDICATERELATION → | ACTIVERELATION (CONNECTOR (that)? PREDICATERELATION)* |
| PREDICATERELATION → | PASSIVERELATION (CONNECTOR (that)? PREDICATERELATION)* |
| ACTIVERELATION → | $Verb(T, T')$ $Type()$ $Instance(T')$ |
| ACTIVERELATION → | $Verb(T, T')$ GENERALISEDQUANTOR PositiveNumber $Type()$ |
| PASSIVERELATION → | BE $Verb(T', T)$ by $Type()$ $Instance(T')$ |
| PASSIVERELATION → | BE $Verb(T', T)$ by GENERALISEDQUANTOR PositiveNumber $Type()$ |
| BE → | is \| are |
| CONNECTOR → | and \| or |
| GENERALISEDQUANTOR → | at least \| at most \| more than \| less than \| exactly |
| QUESTIONMARK → | ? |

Table 2: The Ontology Functions

| | |
|---|---|
| $Type()$ | returns the type information the ontologies keep, ex. gene, disease, drug |
| $Instance(T)$ | returns instances of the type $T$, ex. Asthma for type disease |
| $Verb(T)$ | returns the verbs related to the type $T$, ex. approve for type drug |
| $Verb(T, T')$ | returns the verbs where type $T$ is the subject and type $T'$ is the object, ex. drug treat disease |
| $Noun(T)$ | returns the nouns that are related to the type $T$, ex. symptom for type disease |

```
[A,B,C,D]
query(A,what)-1
predicate(B,be,A,C)-1
relation(C,of,D)-1
object(C,symptoms,countable,na,eq,1)-1
  [E,F,G]
  modifier_pp(F,to,E)-1
  property(G,related,pos)-1
  predicate(F,be,D,G)-1
  object(E,gene_ADRB1,countable,na,eq,1)-1
  v
  [H,I]
  predicate(I,treated,H,D)-1
  object(H,drug_Epinephrine,
         countable,na,eq,1)-1
object(D,diseases,countable,na,geq,2)-1
```

Note that the DRS consists of two kinds of expressions. The lines with a list of uppercase letters, like `[E,F,G]`, describe the domain of the DRS; each uppercase letter is a referent. The rest of the DRS describe the conditions about the domain.

The DRS above contains some predefined predicates, such as `object`, `property`, `predicate`, `query`, etc.. All the nouns, adjectives, verbs, modifiers, etc. are represented with one of them. For instance,

- `object` describes objects and the relevant forms of nouns denoting them (like "diseases")
- `predicate` describes relations that are pro-

duced by different forms of verbs (like "treated"),

- `relation` describes relations that are produced by of-constructions (like "symptoms of disease"),

- `query` describes the form of the query and the objects that the query is referring to.

Ontologies represent relations between concepts. A rule layer over ontologies introduce further concepts integrating them. ASP takes into account relevant concepts and relations to answer a given query about these ontologies. In the biomedical queries we consider, the concepts and instances are represented with `object` and the relations between these concepts are represented with `predicate` and `relation`. The `query` is also important in terms of the type of information the user asks for.

### 4.2 Constructing the Head and the Body Atoms

Once the corresponding DRS is obtained from a given BIOQUERYCNL query, the head and the body atoms are constructed by analyzing the conditions in the DRS, as described in Algorithms 2 and 3.

The HEAD algorithm is about the `query` predicate, which refers to objects or relations that are asked for in the given query. By following the referents, starting from the one mentioned in `query`, the algorithm finds out the type of the information that is asked for in the given query. Consider, for instance, query (Q4). The referent mentioned in `query(A,what)` is `A`. It is mentioned in `predicate(B,be,A,C)-1`, and here it denotes an object with referent `C`. Now let's find where `C` is mentioned: in `object(C,symptoms,countable,na,eq,1)-1` to denote symptoms. Therefore, the query asks for symptoms. Based on this information, Algorithm 2 returns the head of the ASP rules as follows:

```
what_be_symptoms(SYM1)
```

The BODY algorithm analyzes the `predicate` and the `relation` predicates. These two predicates describe relations between objects described by the `object` predicates. The algorithm starts from the `predicate` and the `relation` predicates, and then, by following the referents, it returns the body atoms of the ASP rule. For instance, Algorithm 3 returns the following body atoms for query (Q4):

```
symptoms_of_diseases(symptom_SYM1,
                     disease_DIS1)
diseases_be_related_to_gene(disease_DIS1,
                            gene_``ADRB1'')
drug_treated_diseases(drug_``Epinephrine'',
                      disease_DIS1)
```

These body atoms are given to POSTPROCESSING step, to produce bodies of the ASP rules.

### 4.3 Constructing the ASP Rules

POSTPROCESSING is the last step of the QUERY algorithm. At this step, first the number of rules is determined, and then the body atoms are placed in the bodies of these rules. In ASP, a conjunctive query can be represented by a rule. However, disjunctive queries are represented by several rules with same head but different bodies. For instance, query (Q4) is a disjunctive query (a disjunction of two queries), so there will be two rules representing this query:

```
what_be_symptoms(SYM1) :-
 symptoms_of_diseases(symptom_SYM1,
                      disease_DIS1),
 diseases_be_related_to_gene(disease_DIS1,
                             gene_``ADRB1'').

what_be_symptoms(SYM1) :-
 drug_treated_diseases(drug_``Epinephrine'',
                       disease_DIS1),
 symptoms_of_diseases(symptom_SYM1,
                      disease_DIS1).
```

Next, the predicate names in the bodies of these rules are matched with the names of the already defined predicates in ontologies or in the rule layer over these ontologies. After matching the predicate names, the parameters of the predicates may have to be reordered.

The matching of the predicates very much depends on the user interface (UI). If UI enforces users to use a specific grammar and lexicon while forming the query, then the matching can be done with an easy table look-up method. If the UI allows more flexibility of constructing a query, then the matching algorithm should use some basic Natural Language Processing methods and similarity metrics to find the most probable matching.

After matching the predicates, the ordering of the parameters can be done easily. The BODY algorithm

**Algorithm 2** HEAD($D$)

**Input:** A DRS

**Output:** Head of an ASP rule

1: *query*(*Ref*, *QuestionWord*) // e.g., *query*($A$, *which*) for "Which drug ..."
2: **if** *Ref* is an object **then**
3: *Object* := REFERSTO(*Ref*) // e.g., $A$ refers to a "drug" $DRG1$
4: *Head* := CONCAT(*QuestionWord*, *Object*, *Ref*) // e.g., *which_drug*($DRG1$)
5: **else if** *Ref* is a subject of a predicate **then** // *query*($A$, *what*) for "What are the genes ..."
6: *Object* := REFERSTO(*Ref*) // e.g., $A$ refers to "genes" $GENE1$
7: *Head* := CONCAT(*QuestionWord*, *Predicate*, *Object*, *Ref*) // e.g., *what_be_genes*($GENE1$)
8: **end if**
9: **return** *Head*

returns the body predicates with the parameters. In these parameters, the type and the instance names are kept together. Thus, ordering of those parameters are done just by using the type information. After the ordering is done, the type information part is removed from the parameters.

For instance, after matching the predicates, we get the following ASP rule for query (Q4).

```
what_be_symptoms(SYM1) :-
 disease_symptom(DIS1,SYM1),
 disease_gene(DIS1,``ADRB1'').

what_be_symptoms(SYM1) :-
 treats_disease(``Epinephrine'',DIS1),
 disease_symptom(DIS1,SYM1).
```

With an ASP rule layer over ontologies, and this ASP program, an ASP solver, like DLVHEX (DLVHEX, 2009), returns an answer to query (Q4).

For instance, consider the ASP rule layer, and the gene, disease, drug ontologies of (Bodenreider et al., 2008). The ontologies of (Bodenreider et al., 2008) are obtained from the ontologies PHARMGKB (PharmGKB, 2008), UNIPROT (UniProt, 2008), GENE ONTOLOGY (GO) (GeneOntology, 2008), GENENETWORK database (GeneNetwork, 2008), DRUGBANK (DrugBank, 2008), and the Medical Symptoms and Signs of Disease web page (MedicalSymptomsSignsDisease, 2008). With this rule layer and the ontologies, and the ASP program above, the following is a part of the answer DLVHEX finds to the query above:

```
noisy breathing       faster breathing
shortness of breath   coughing
chest tightness       wheezing
```

### 4.4 Another Example

The algorithm discussed above returns the following ASP program for query (Q5):

```
which_genes(GN1) :-
 2<=#count{DRG1:drug_gene(DRG1,GN1)},
 #count{DIS1:disease_gene(DIS1,GN1)}<=3.
```

Since query (Q5) contains cardinality constraints, the ASP program uses the aggregate `#count`.

More examples of biomedical queries, and the ASP programs generated by our program can be seen at `http://people.sabanciuniv.edu/esraerdem/bioquery-asp/bionlp09/`.

## 5 Implementational Issues

We have implemented the algorithms explained above in PERL. We have used Attempto Parsing Engine APE to convert a given BIOQUERYCNL query into a DRS. Since BIOQUERYCNL is about biomedical ontologies, we provided APE some information about biomedical concepts, such as gene, drug, and words that represent relations between these concepts such as treat, target etc..

However, providing such information is not sufficient to convert all BIOQUERYCNL biomedical queries into programs, mainly due to specific instances of these concepts (consider, for instance, various drug names that appear in ontologies). One way to deal with this problem is to extract from the ontologies all instances of each concept and provide them to APE as an additional lexicon. This may not be the perfect solution since this process has to be repeated when an instance is added to the ontology. An alternative way can be enforcing the user to enter

**Algorithm 3** BODY($D$)

**Input:** A DRS
**Output:** Body of an ASP rule

*Body* := empty string
**for** each predicate $P$ **do**
    // $P$ can be of the form *predicate*(*Ref*, *Verb*, *SubRef*), like *predicate*($H$, *targeted*, $A$)
  *Subject* := REFERSTO(*SubRef*)  // e.g., $A$ refers to "genes" $GENE1$
  **if** $P$ has a verb phrase modifier **then**
    ⟨*Modifier*, *Object*⟩ := REFERSTO(*Ref*)  // e.g., $H$ refers to ⟨ "by", "drug" $DRG1$ ⟩
  **end if**
  **if** $P$ has an object **then**  // $P$ can be of the form *predicate*(*Ref*, *Verb*, *SubRef*, *ObjRef*)
    *Object* := REFERSTO(*ObjRef*)
  **end if**
  *Body* := CONCAT(*Body*, *Subject*, *Verb*, *Modifier*, *Object*)
    // e.g., *genes_targeted_by_drugs*($GENE1$, $DRG1$)
**end for**
**for** each relation $R$ **do**
    // $R$ can be of the form *relation*(*Ref1*, *of*, *Ref2*), like *relation*($C$, *of*, $D$)
  *Object1* := REFERSTO(*Ref1*)  // e.g., $C$ refers to "symptoms" $SYM1$
  *Object2* := REFERSTO(*Ref2*)  // e.g., $D$ refers to "diseases" $DIS1$
  *Body* := CONCAT(*Body*, *Object1*, *"'of"'*, *Object2*)
    // e.g., *symptoms_of_diseases*($SYM1$, $DIS1$)
**end for**
**return** $Body$

the concept name just before the instance (like "the drug Epinephrine") in the query. This is how we deal with instance names, in the current version of our implementations. However, such BIOQUERYCNL queries are not in the language of APE; so, with some preprocessing, we rewrite these queries in the correct syntax for APE.

## 6 Conclusion

We have designed and developed a Controlled Natural Language (CNL), called BIOQUERYCNL, to represent biomedical queries over some ontologies, and provided a precise description of its grammatical structure.

We have introduced an algorithm to convert queries in BIOQUERYCNL to a program in Answer Set Programming (ASP). The idea is to compute answers to these queries automatically, by means of automated reasoners in ASP, over biomedical ontologies in RDF(S)/OWL and a rule layer in ASP integrating these ontologies. Our algorithm can handle various forms of simple/complex disjunctive/conjunctive queries that may contain (nested) relative clauses and cardinality constraints.

We have implemented this algorithm in PERL, and tried it with the ASP rule layer, and the ontologies of (Bodenreider et al., 2008).

One essential part of the overall system is an intelligent user interface that allows a user to enter biomedical queries in BIOQUERYCNL. Design and implementation of such a user-interface is a part of our ongoing work.

## Acknowledgments

Thanks to Tobias Kuhn for his help with ACE. This work is supported by the Scientific and Technological Research Council of Turkey (TUBITAK) grant 108E229.

## References

Attempto. 2008. `http://attempto.ifi.uzh.ch/site/`.

Chitta Baral, Juraj Dzifcak, and Luis Tari. 2007. Towards overcoming the knowledge acquisition bottleneck in answer set prolog applications: Embracing natural language inputs. In *Proc. of ICLP*, pages 1–21.

Chitta Baral, Juraj Dzifcak, and Tran Cao Son. 2008. Using answer set programming and lambda calculus to characterize natural language sentences with normatives and exceptions. In *Proc. of AAAI*, pages 818–823.

Chitta Baral. 2003. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press.

Abraham Bernstein, Esther Kaufmann, Anne Göhring, and Christoph Kiefer. 2005. Querying ontologies: A controlled english interface for end-users. In *Proc. of ISWC*, pages 112–126.

Abraham Bernstein, Esther Kaufmann, Christian Kaiser, and Christoph Kiefer. 2006. Ginseng: A guided input natural language search engine for querying ontologies. In *Jena User Conference*.

Patrick Blackburn and Johan Bos. 2005. *Representation and Inference for Natural Language. A First Course in Computational Semantics*. CSLI.

Olivier Bodenreider, Zeynep Hande Çoban, Mahir Can Doğanay, Esra Erdem, and Hilal Koşucu. 2008. A preliminary report on answering complex queries related to drug discovery using answer set programming. In *Proc. of ALPSWS*.

Boxer. 2009. `http://svn.ask.it.usyd.edu.au/trac/candc/wiki/boxer`.

CC. 2009. `http://svn.ask.it.usyd.edu.au/trac/candc/wiki`.

DLV. 2009. `http://www.dbai.tuwien.ac.at/proj/dlv`.

DLVHEX. 2009. `http://con.fusion.at/dlvhex/`.

DrugBank. 2008. `http://redpoll.pharmacy.ualberta.ca/drugbank/`.

Norbert E. Fuchs, Kaarel Kaljurand, and Tobias Kuhn. 2008. Discourse representation structures for ace 6.0. Technical Report IFI-2008.02, Department of Informatics, University of Zurich.

Michael Gelfond and Vladimir Lifschitz. 1988. The stable model semantics for logic programming. In Robert Kowalski and Kenneth Bowen, editors, *Logic Programming: Proceedings of the Fifth International Conference and Symposium*.

GeneNetwork. 2008. `http://humgen.med.uu.nl/~lude/genenetwork`.

GeneOntology. 2008. `http://www.geneontology.org`.

Jena. 2008. `http://jena.sourceforge.net/`.

Hans Kamp. 1981. A theory of truth and semantic representation. In J. A. G. Groenendijk, T. M. V. Janssen, and M. B. J. Stokhof, editors, *Formal Methods in the Study of Language*, volume 1, pages 277–322.

Esther Kaufmann, Abraham Bernstein, and Renato Zumstein. 2006. Querix: A natural language interface to query ontologies based on clarification dialogs. In *Proc. of ISWC*.

Mark Klein and Abraham Bernstein. 2004. Toward high-precision service retrieval. *IEEE Internet Computing*, 8(1):30–36.

Vladimir Lifschitz. 1999. Action languages, answer sets and planning. In *The Logic Programming Paradigm: a 25-Year Perspective*. Springer.

Victor Marek and Mirosław Truszczyński. 1999. Stable models and an alternative logic programming paradigm. In *The Logic Programming Paradigm: a 25-Year Perspective*. Springer.

MedicalSymptomsSignsDisease. 2008. `http://www.medicinenet.com/symptoms_and_signs/article.htm`.

Ilkka Niemelä. 1999. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence*, 25.

PharmGKB. 2008. `http://www.pharmgkb.org`.

UniProt. 2008. `http://www.ebi.uniprot.org/index.shtml`.