Controlled Natural Language as Interface Language to the Semantic Web

Rolf Schwitter

Centre for Language Technology, Macquarie University,
Sydney, NSW 2109, Australia
schwitt@ics.mq.edu.au

Abstract. In this paper, I will show how a controlled natural language (CNL) can be used as an interface language to the Semantic Web. Instead of working with a formal language based on RDF that is difficult to write and understand for non-specialists, I will argue that a CNL can be employed to describe resources on the Web (via assertional statements) and to construct ontologies (via terminological statements). I will present a complete rule set written in CNL that allows for efficient reasoning over the assertional and terminological knowledge with the help of a model builder. There is no need to formally encode this knowledge in an RDF-based notation. Everything can be described in a uniform way on the level of the controlled natural language provided that we support the user of the CNL with an intelligent writing tool.

1 Introduction

The vision of the Semantic Web is to extend the current Web in a way in which information is given well-defined meaning enabling computers and people to work in cooperation [2]. To a certain degree, cooperation between computers can be achieved by annotating information (assertional knowledge) on the Web with machine-processable data and by linking these annotations to ontologies (terminological knowledge). In the ideal case, ontologies can be combined with rule languages so that new (entailed) information can be inferred and questions about assertional (and terminological) knowledge can be answered with the help of reasoning services [7].

However, cooperation between computers and people on a world-wide scale can hardly be achieved via RDF-based formal languages such as RDFS [3] and OWL [14]. People (in particular non-specialists) need to be able to add new machine-readable information to a Web site. They need to be able to express their questions in a familiar notation, and to read and understand information derived from a piece of potentially distributed knowledge.

What is urgently needed is a high-level interface language to the Semantic Web that abstracts away from these RDF-based formal notations. I will show that a well-designed controlled natural language (CNL) is an ideal candidate to increase the transparency of the Semantic Web and to empower non-specialists with a "seemingly informal" language to work in cooperation with computers.

Bhanu Prasad (Editor): IICAI-05, pp. 1699-1718, 2005.

Usually a CNL is defined as a well-defined subset of a natural language that has been restricted with respect to its grammar and its lexicon. Grammatical restrictions result in less complex and less ambiguous sentences, while lexical restrictions reduce the size of the lexicon and the meaning of the lexical entries for a particular domain [10]. This definition is a good starting point for our undertaking, but we need to keep in mind that we are going to design a CNL that has the same formal properties as an ontology language layered on top of RDF; this seriously restricts the expressivity of the CNL.

2 OWL and OWL Lite$^-$

The Web Ontology Language OWL is based on description logic and comes in three increasingly expressive layers: OWL Lite, OWL DL, and OWL Full [14]. Ironically, OWL Full and OWL DL are not suitable for reasoning over large and distributed ontologies, since there exist no efficient reasoning algorithms for these languages [5]. Even OWL Lite, the least expressive layer of these languages, has constructors (e.g. equality, disjunction and negation) that considerably complicate the implementation of efficient reasoning algorithms.

Recently, it has been argued that the intersection of description logic with logic programs can provide a straightforward computational pathway for reasoning and interoperability on the Semantic Web [7, 9]. In particular, OWL Lite$^-$ has been identified as the maximal subset of OWL which can be expressed in the deductive database language Datalog [5]. Datalog corresponds to Horn clauses with range-restricted universally quantified variables (all variables in the head of a clause occur also in the body), without function symbols (of arity greater than zero) and without negation (see [6]). It has been shown that about 77% of all current ontologies developed for the Semantic Web fall under the OWL Lite$^-$ subset [16].

OWL Lite$^-$ is layered on top of RDF which relies on eXtensible Markup Language (XML) for syntax, Uniform Resource Identifiers (URIs) for naming and RDFS Schema (RDFS) for describing meaning and relationships of terms. OWL Lite$^-$ uses RDF and RDFS constructors whenever the required functionality already exists for a lower layer. In a nutshell: OWL Lite$^-$ consists of the following constructors whose meaning will become clear in the subsequent discussion:

```
Individuals:              rdf:type
Simple classes:           owl:Class
                          rdfs:subClassOf
Simple properties:        owl:objectProperty
                          rdfs:subPropertyOf
                          rdfs:domain, rdfs:range
Property characteristics: owl:TransitiveProperty
                          owl:SymmetricProperty
                          owl:inverseOf
Property restrictions:    owl:Restriction
                          owl:onProperty
                          owl:allValuesFrom
```

Additionally, OWL Lite$^-$ provides two constructors (*owl:equivalentClass* and *owl:equivalentProperty*) that can be used to map between classes and properties from different ontologies.

3 OWL Lite$^-$ plus Rules in CNL

As discussed in the last section, the syntax of OWL Lite$^-$ relies on RDF. RDF is based on the idea of identifying things (= resources) using URIs and describing these things in terms of simple properties and property values.

Imagine trying to state that someone named Nora Yuen supervises someone named John Smith. This can be encoded in RDF as follows:

```
<rdf:Description rdf:ID=‘‘nora_yuen’’>
  <ex:supervise rdf:resource=‘‘john_smith’’>
</rdf:Description>
```

Here *nora_yuen* is the identified resource, *supervise* is a simple property, and *john_smith* is a property value. The URIs for the resource and the property value are the ones of the current document and the prefix *ex* indicates that the URI for the property is the one specified in the namespace declaration of the document. A straightforward way to express the above statement in CNL is:

```
Nora Yuen supervises John Smith.
```

In CNL (as well as in RDF terminology), the subject ‘Nora Yuen’ identifies the resource of the statement, the predicate ‘supervises’ identifies the property of the statement and the object ‘John Smith’ identifies the value of that property. The namespaces for these terms are not displayed here. As we will see later in Section 3.4, namespaces are handled by an intelligent text editor that supports the writing process of CNL. Please note that the user **does not** need to learn the syntactic rules of the CNL, since these rules are enforced by the text editor via a look-ahead mechanism [15].

3.1 Making Assertional Statements in CNL

The basic syntactic structure for making assertional statements in CNL consists of a simple sentences that have a subject-predicate-object pattern and variations that can be mapped into “triples”. Here are a few assertional statements in CNL (that we will feed later for illustration purposes to the model builder):

```
Nora Yuen is a linguist and supervises John Smith.
John Smith who is a friend of Kylie Miller is a PhD student.
Kylie Miller is drilled by Nora Yuen.
```

The first sentence with the coordinator *and* is a compound one and is equivalent to the following two simple statements in CNL:

```
Nora Yuen is a linguist. Nora Yuen supervises John Smith.
```

The second sentence – mentioned above – with the subordinator *who* is a complex one and corresponds to the two simple statements:

```
John Smith is a friend of Kylie Miller. John Smith is a PhD student.
```

The third sentence is a passive construction and can be interpreted as the inverse of the statement:

```
Nora Yuen drills Kylie Miller.
```

if the corresponding relationship between *drill* and *be drilled by* is specified in the ontology via a terminological statement (see next section).

3.2 Making Terminological Statements in CNL

In contrast to assertional statements that are most likely to be made by users with different kind of computational background, terminological statements will most probably be uttered by knowledge engineers in order to construct an ontology. Terminological statements speak about classes, properties, instances of classes, and various kinds of relationships between instances, classes and properties. The following statement in CNL

```
The property 'supervise' has the type 'object property'.
```

talks for example about the *'supervise'* property and assigns a specific type to it. This statement can be written in an abbreviated form in CNL:

```
'supervise' has the type 'object property'.
```

Below are a number of terminological statements with the corresponding "naturalized" OWL Lite$^-$ constructors in predicate position (for example, *'has the domain'* or *'is a subproperty of'*):

```
'supervise' has the domain 'professor'.
'supervise' has the range 'PhD student'.
'teach' has the type 'object property'.
'teach' has the domain 'academic'.
'teach' has the range 'student'.
'drill' is the inverse of 'be drilled by'.
'drill' has the equivalent property 'instruct'.
'drill' is a subproperty of 'teach'.
'professor' is a sublcass of 'academic'.
'linguist' is a subclass of 'researcher'.
'researcher' is a subclass of 'scientist'.
'PhD student' is a subclass of 'student'.
```

This terminological knowledge is used by the model builder for reasoning purposes and by the look-ahead text editor to guide the writing process of assertional statements.

3.3 Making Conditional Statements in CNL

In contrast to OWL Lite$^-$, the CNL allows for expressing rules in form of conditional sentences to build an axiomatic framework for reasoning. The antecedent (and consequent) of a conditional statement can be complex, for example:

```
If E has a property P whose value is V and R has the type 'range
     restriction' and R is on P and R has all values from C
   then E has the type R.
```

As this example shows, the CNL allows for variables (*E, P, V, R, C*) in rules that directly translate into variables in the formal representation. Note that complex consequents will be distributed automatically during the translation.

3.4 Writing in CNL

Writing in CNL is supported by a look-ahead text editor. This editor can be used either to write an assertional specification, to construct an ontology or to build an axiomatic rule set. The user does not need to learn the rules of the CNL explicitly, since he is guided by the look-ahead editor while the text is written [15].

For an assertional specification, the user first selects the ontologies he wants to work with via a menu. Thereby the text editor becomes "ontology-aware" and guides the writing process via look-ahead categories and handles namespaces. Let's imagine that the user wants to make the subsequent assertional statement in CNL:

```
Nora Yuen supervises John Smith.
```

The editor first displays a look-ahead category for the subject position:

[ProperNoun]

After entering the name 'Nora Yuen' (that denotes a unique individual) the editor displays further look-ahead categories (partially) derived from the syntactic information of the grammar and from the available terminological knowledge:

[who – is – has – FullVerb]

The user either directly types an approved word or selects it from a context menu that will provide additional information about the namespaces. After entering, for example, the verb 'supervises', the look-ahead editor asks for the object of the statement, and so on.

4 Processing CNL

Specifications in CNL are translated via a Definite Clause Grammar (DCG) [13] into a format that can directly be processed by SATCHMO [4, 12]. SATCHMO is a model-generation based theorem prover that takes a set of first-order formulae as input and tries to generate a finite satisfying model for them by combining a forward chaining strategy for normal cases with a backward chaining strategy for special cases.

4.1 Syntax of SATCHMO Rules

In SATCHMO, all first-order formulae are uniformally represented in an implicational rule format of the form:

```
ANTECEDENT ---> CONSEQUENT.
```

The antecedent *ANTECEDENT* of a rule is either *true* or a single atomic formula or a conjunction of atomic formulae. The consequent *CONSEQUENT* is either a single atomic formula or *false* or a disjunction of atomic formulae. Since we are working with the Datalog subset of first-order logic, we will end up with only one single atomic formula in the consequent of the rule but with no disjunctions.

Let's first have a look at a few translations and then discuss in the next section how these translations are produced automatically. For instance, the assertional statement

```
Nora Yuen is a linguist and supervises John Smith.
```

is translated into two facts that are represented as two SATCHMO rules of the form:

```
true ---> term([‘Nora’,‘Yuen’],[rdf:type],[ex:linguist]).
true ---> term([‘Nora’,‘Yuen’],[ex:supervise],[‘John’,‘Smith’]).
```

A terminological statement such as

```
‘supervise’ has the domain ‘professor’.
```

results in a similar translation with an *rdfs* constructor:

```
true ---> term([ex:supervise],[rdfs:domain],[ex:professor]).
```

And finally a conditional statement such as

```
If E has a property P whose value is V and R has the type ‘range
     restriction’ and R is on P and R has all values from C
   then E has the type R.
```

is translated into the following rule:

```
term(E,P,V), term(R,[rdf:type],[owl:restriction]),
term(R,[owl:onProperty],P), term(R,[owl:allValuesFrom],C) --->
  term(E,[rdf:type],R).
```

Please note that variables in SATCHMO rules are range-restricted in the same way as in Datalog. That means, whenever a variable occurs in the consequent of a rule, then the same variable must also occur in the antecedent of that rule. This restriction guarantees that the knowledge base and any derived consequences are always variable-free.

4.2 Translating CNL into SATCHMO Rule Format

As aforementioned, statements in CNL can be translated automatically into the SATCHMO rule format. The syntax of the CNL is specified here by a DCG (that can be processed by a chart parser - if required). Specifying the grammar gives you a parser (and a generator) for free. In the following, I will show how this can be done starting from two simple assertional statements in CNL:

```
Nora Yuen is a linguist.
Nora Yuen supervises John Smith.
```

Both sentences are instantiations of a subject-predicate-object pattern. The first sentence starts with a proper noun in subject position followed by a verb (copula) in predicate position, and a noun phrase consisting of an indefinite article plus a noun in object position. The second sentence starts also with a proper noun, followed by a full verb but has a proper noun in object position. These structures can easily be described by a DCG extended by additional arguments and Prolog goals in the body of the grammar rules:

```
sentence(C1-C2) -->
  proper_noun(N,S),
  verb_phrase(a,_,N,S,C1-C2),
  ['.'].

verb_phrase(a,-,N,S,C1-[true--->term(S,P,O)|C2]) -->
  verb(N,P),
  noun_phrase(a,N,O,C1-C2).

noun_phrase(T,s,O,C-C)  -->
  ( [a], noun(s,O) ; proper_noun(s,O) ).

verb(s,M) -->
  { lexicon([v,s],W,M) }, W.
noun(s,M) -->
  { lexicon([n,s],W,M) }, W.
proper_noun(s,M) -->
  { lexicon([pn,s],W,M) }, W.

lexicon([n,s],[linguist],[ex:linguist]).
lexicon([n,s],[linguist],[ex:professor]).
lexicon([pn,s],['Nora','Yuen'],['Nora','Yuen']).
lexicon([pn,s],['John','Smith'],['John','Smith']).
lexicon([v,s],[is],[rdf:type]).
lexicon([v,s],[supervises],[ex:supervise]).
```

As these grammar rules show, each non-terminal symbol in the grammar has been extended by one or more arguments. For example, the term `C1-C2` represents a difference list and is responsible for building up the required SATCHMO rule(s) while the sentence is parsed. The constant `a` in the verb phrase indicates that this grammar rule is designed for processing assertional statements in CNL, and

the variable *N* is used to control number agreement between the proper noun and the verb. Finally, the variables *S*, *P*, and *O* guarantee that a SATCHMO rule of the form *true--->term(S,P,O)* can be constructed in the verb phrase via unification using information available from the lexicon.

The above DCG processes the two input sentences and produces the required SATCHMO rules:

```
true ---> term([‘Nora’,‘Yuen’],[rdf:type],[ex:linguist]).
true ---> term([‘Nora’,‘Yuen’],[ex:supervise],[‘John’,‘Smith’]).
```

Sentences written in CNL can be coordinated and thereby have the potential to express complex statements such as:

```
Nora Yuen is a linguist and supervises John Smith.
```

The subsequent DCG rule deals with verb phrase coordination:

```
verb_phrase(T,+,N,S,C1-C3) -->
  verb_phrase(T,-,N,S,C2-C3),
  [and],
  verb_phrase(T,_,N,S,C1-C2).
```

The constants + and - in the second argument position stand for a ‘coordinated’ and ‘non-coordinated’ verb phrase. The underscore _ stands for an anonymous variable and indicates that this verb phrase can be either coordinated or non-coordinated.

Sentences in CNL can be embedded into other sentences such as the relative sentence *who is a linguist* in:

```
Nora Yuen who is a linguist supervises John Smith.
```

The following DCG rules takes care of relative sentences:

```
sentence(C1-C3)  -->
  proper_noun(N,S),
  rel_sentence(a,_,N,S,C2-C3),
  verb_phrase(a,_,N,S,C1-C2),
  [‘.’].

rel_sentence(T,-,N,S,C1-C2) -->
  [who],
  verb_phrase(T,-,N,S,C1-C2).

noun_phrase(T,s,O,C1-C2)  -->
  ( [a], noun(s,O) ; proper_noun(s,O) ),
  rel_sentence(T,_,N,O,C1-C2).
```

Not only verb phrases, but also relative sentences can be coordinated in CNL as the subsequent DCG rule illustrates:

```
rel_sentence(T,+,N,S,C1-C3) -->
  rel_sentence(T,-,N,S,C2-C3),
  [and],
  rel_sentence(T,_,N,S,C1-C2).
```

This rule deals with coordinated relative sentences such as in:

```
Nora Yuen who is a linguist and who is a professor
  supervises John Smith.
```

The DCG presented so far has two interesting properties: it is declarative and bi-directional. Bi-directionality allows us not only to analyse sentences in CNL and to produce rules in SATCHMO format but also to generate sentence starting from a set of rules. This is of particular interest, if we intend, for instance, to display all statements in CNL that are entailed in a theory or answer questions in CNL using a given theory. For example, if we take the three SATCHMO rules in

```
generate(S) :-
  C = [true--->term([‘Nora’,‘Yuen’],[rdf:type],[ex:linguist]),
       true--->term([‘Nora’,‘Yuen’],[rdf:type],[ex:professor]),
       true--->term([‘Nora’,‘Yuen’],[ex:supervise],[‘John’,‘Smith’])],
  sentence([]-C,S,[]).
```

as starting point, then our DCG fragment will generate a number of syntactically different sentences in CNL that have the all same meaning, for example:

```
Nora Yuen is a linguist and is a professor and supervises John Smith.
Nora Yuen who is a linguist and who is a professor
  supervises John Smith.
Nora Yuen who is a linguist is a professor and supervises John Smith.
```

4.3 Model Generation

Starting from the empty interpretation, SATCHMO works by attempting to generate a model of its input rules of the form `A--->C` by searching for a violated rule of which the antecedent `A` is true in the current model but the consequent is not. In the simplest case, a new consequent `C` is satisfied by adding a single atomic formula to the model. Thereby the consequent is made true in the model and the rule under investigation is no longer violated. This procedure iterates and if no violated rule remains left, then the model is complete (for details see [1]).

A modified version of the original SATCHMO program [12] that uses an accumulator instead of the (Prolog) knowledge base is given below:

```
generate_model(M) :-
  generate_model([],M).

generate_model(M1,M2) :-
  violated_instance(C,M1), !,
```

```
  disjunction_consequent(A,C),
  generate_model([A|M1],M2).
generate_model(M,M).

violated_instance(C,M) :-
  (A--->C),
  satisfy_antecedent(A,M),
  \+ satisfy_consequent(C,M).

satisfy_antecedent(true,M).
satisfy_antecedent(A,M) :-
  member(A,M).
satisfy_antecedent((A,As),M) :-
  member(A,M),
  satisfy_antecedent(As,M).

satisfy_consequent(A,M) :-
  member(A,M).
satisfy_consequent((A;As),M) :-
  member(A,M).
satisfy_consequent((A;As),M) :-
  satisfy_consequent(As,M).

disjunction_consequent(X,X) :-
  \+ X = false,
  \+ X = (Y;Ys).
disjunction_consequent(X,(X;Ys)).
disjunction_consequent(X,(Y;Ys)) :-
  disjunction_consequent(X,Ys).
```

Note that in our case we **do not** need to test for disjunctions in the consequent of a rule, since OWL Lite$^-$ does not allow for disjunction.

5 Rule Set for Reasoning in CNL

In this section, I will introduce the rule set – written in CNL – that is required for reasoning over assertional and terminological knowledge. Note that this rule set is **not** available in the ontology language OWL Lite$^-$ and would have to be encoded in a completely separate rule language such as TRIPLE [8]. In the subsequent discussion, I will follow the discussion in Harth and Decker [8] but show how class and property hierarchies as well as property characteristics can be directly expressed in CNL.

5.1 Class and Property Hierarchies

Subclass property. The most important taxonomic construction to model class hierarchies is the subclass property. This property relates a specific class to a more general class. The subclass property is transitive and can be used to model class hierarchies via the following rule written in CNL:

```
If C1 is a subclass of C2 and C2 is a subclass of C3
   then C1 is a subclass of C3.
```

This rule guarantees, for example, that if 'linguist' is a subclass of 'researcher' and 'researcher' is a subclass of 'scientist', then 'linguist' is a subclass of 'scientist'.

Additionally, we need a rule that makes sure that the type of an individual takes the class hierarchy into account:

```
If C1 is a subclass of C2 and E has the type C1
   then E has the type C2.
```

This rule ensures, for example, that if 'linguist' is a subclass of 'researcher' and 'Nora' has the type 'linguist', then 'Nora' has the type 'researcher'.

Subproperty. The subproperty relation behaves in a similar way as the subclass property and allows for defining hierarchies of properties in CNL using the following rule:

```
If P1 is a subproperty of P2 and P2 is a subproperty of P3
   then P1 is a subproperty of P3.
```

This rule states, for example, that if 'drill' is a subproperty of 'teach' and 'teach' is a subproperty of 'inform', then 'drill' is a subproperty of 'inform'.

Additionally, we need a rule that applies a property hierarchy to individuals:

```
If P1 is a subproperty of P2 and E has P1 whose value is V
   then E has P2 whose value is V.
```

This rule ensures, for example, that if 'drill' is a subproperty of 'teach' and 'Nora' has the property 'drill' whose value is 'Kylie', then 'Nora' has the property 'teach' whose value is 'Kylie'.

Domain restriction. In OWL Lite$^-$ the domain of a property can be restricted to a specific class ensuring that only individuals of this class occur in the subject position. The following rule handles this:

```
If E has the property P whose value is V and P has the domain D
   then E has the type D.
```

This rule guarantees, for example, that if 'Nora' has the property 'supervise' whose value is 'John' and the property 'supervise' has the domain 'professor', then 'Nora' has the type 'professor'.

Range restriction. The range of a property can be restricted in a similar way to a specific class making sure that only individuals of this class occur in the object position:

```
If E has the property P whose value is V and P has the range R
   then V has the type R.
```

This rule states, for example, that if 'Nora' has the property 'supervise' whose value is 'John' and the property 'supervise' has the range 'PhD student', then 'John' has the type 'PhD student'.

5.2 Property Characteristics

Object property. In OWL Lite$^-$ only object properties are allowed, but no datatypes as in OWL Lite. That means that the subject and object position must be realised by an individual of a specific class:

```
'object property' has the domain 'class'.
'object property' has the range 'class'.
```

The rules for domain and range restriction introduced above cover the required inferences.

Transitive property. The next rule generalizes transitivity for properties of type *'transitive property'*:

```
If E has the property P whose value is W and W has P whose value is V
   and P has the type 'transitive property'
   then E has P whose value is V.
```

The rule ensures, for example, that if 'Nora' has the property 'is ancestor of' whose value is 'Carla', and 'Carla' has also the property 'is ancestor of', but with the value 'Fabian', and the property 'is ancestor of' has the type 'transitive property', then 'Nora' has the property 'is ancestor of' whose value is 'Fabian'.

Symmetric property. A symmetric property is a property that is true in both directions:

```
If E has the property P whose value is V
   and P has the type 'symmetric property'
   then V has P whose value is E.
```

This rule states, for example, that if 'John' has the property 'is a friend of' whose value is 'Kylie' and the property 'is a friend of' has the type 'symmetric property', then 'Kylie' has the property 'is a friend of' whose value is 'John'.

Similar to a subproperty, we can restrict the range and the domain of a symmetric property:

```
If P has the type 'symmetric property' and has the range R
   then P has the domain R.
If P has the type 'symmetric property' and has the domain D
   then P has the range D.
```

Inverse property. A property is the inverse property of another property when the variables in the subject and object position of the first property switch their argument positions in the second property:

```
If E has the property P1 whose value is V
   and the property P2 is the inverse of P1
   then V has P2 whose value is E.
```

This rule guarantees, for example, that if 'Nora' has the property 'drill' whose value is 'Kylie' and the property 'be drilled by' is the inverse of 'drill', then 'Kylie' has the property 'be drilled by' whose value is 'Nora'.

The following statements in CNL define that the inverse property is a symmetric property and that the subject position and object position of an inverse property are realised by individuals (as indicated by *'object property'*).

```
'inverse of' has the type 'symmetric property'.
'inverse of' has the domain 'object property'.
'inverse of' has the range 'object property'.
```

Class equivalence. In order for ontologies to have the maximum impact, they need to be sharable and re-usable. When two ontologies describe the same class, then a mechanism is needed to state that the two classes are equivalent. That means we need to be able to systematically describe that every individual of one class is also an individual of the other class.

Here are two statements in CNL that describe specific characteristics of class equivalence:

```
'equivalent class' has the type 'symmetric property'.
'equivalent class' has the type 'transitive property'.
```

The first statement says that class equivalence is a symmetric property. The rules for symmetry introduced above take this statement into consideration. The second statement says that class equivalence is transitive. The general rule for transitivity introduced above takes care of this statement.

The following rule performs the needed inferences on the type hierarchy and infers that an individual of one class is equivalent to an individual of another class:

```
If C1 is an equivalent class of C2 and E has the type C2
   then E has the type C1.
```

The next rule is used for completion of the rule set and states that class equivalence is reflexive:

```
If C1 is an equivalent class of C2 and E has C1 whose value is V
   then E has C1 whose value is V.
```

The subsequent three rules guarantee that all occurrences of equivalent classes in either subject, predicate, or object position of a statement are replaced:

```
If C1 is an equivalent class of C2 and E has C1 whose value is V
   then E has C2 whose value is V.
If C1 is an equivalent class of C2 and C2 has the property P
      whose value is V
   then C1 has P whose value is V.
If C1 is an equivalent class of C2 and E has the property P
      whose value is C1
   then E has P whose value is C2.
```

Property equivalence. Similar to class equivalence, we can formulate statements and rules in CNL that deal with property equivalence.

The subsequent two statements are straightforward and state symmetry and transitivity of equivalent properties:

```
'equivalent property' has the type 'symmetric property'.
'equivalent property' has the type 'transitive property'.
```

The following two rules deal with property hierarchies and reflexivity of equivalent properties:

```
If P1 is an equivalent property of P2 and E has P1 whose value is V
   then E has P1 whose value is V.
If P1 is an equivalant property of P2 and E has P1 whose value is V
   then E has P2 whose value is V.
```

and finally the next two rules take care of the replacement of equivalent properties in the subject and object position of a statement:

```
If P1 is an equivalent property of P2 and P2 has the property P
      whose value is V
   then P1 has P whose value is V.
If P1 is an equivalent property of P2 and E has the property P
      whose value is V
   then E has P whose value is P2.
```

5.3 Property Restrictions

So far, we have seen how to restrict the range and the domain of properties in a global way. The following rule allows to set a local range restriction on a property taking both the property and the domain of a statement into account:

```
If E has a property P whose value is V and R has the type 'range
     restriction' and R is on P and R has all values from C
   then E has the type R.
```

The rule ensures, for example, that if 'Nora' has the property 'teach' whose value is 'Kylie' and the class 'academic' has the type 'range restriction', and the restricted property is 'teach', and all values are from the domain 'student', then 'Nora' has the type 'academic'. In brief, the rule licenses the inference: if Nora teaches Kylie and Kylie is a student, then Nora is an academic.

6 Evaluation

Given the rule set introduced in the previous section, the terminological knowledge presented in Section 3.2, and the assertional knowledge in Section 3.1, SATCHMO can generate a satisfying model.

Since this model consists of terms of the form

```
term([‘John’,‘Smith’],[rdf:type],[ex:student]).
```

it is straightforward to extract all entailed assertional statements in CNL:

```
John Smith is a student. Kylie Miller is a student.
Nora Yuen is a scientist. Nora Yuen is a researcher.
Nora Yuen is an academic. Nora Yuen is a professor.
Nora Yuen instructs Kylie Miller. Nora Yuen teaches Kylie Miller.
Nora Yuen drills Kylie Miller.
```

as well as all entailed terminological statements (only a small subset of them is displayed here):

```
‘linguist’ is a subclass of ‘scientist’.
‘drills’ has the type ‘object property’.
‘drills’ is a subproperty of ‘instructs’.
‘is drilled by’ is the inverse of ‘drills’.
‘is drilled by’ has the type ‘object property’.
```

The generated model can now be used to answer questions in CNL, such as:

```
Who instructs Kylie?
Does Nora teach Kylie Miller and instruct John?
What does Nora do?
What type does John Smith have?
```

Questions are first translated into SATCHMO format, solution(s) are looked up in the model, and answers are generated in CNL.

7 Conclusions

In this paper, I presented a CNL that can be used to express the same sort of knowledge as the Web Ontology Language OWL Lite$^-$ but in a “seemingly informal” notation. In contrast to OWL Lite$^-$ that does not have direct rule support, I showed how a complete rule set for reasoning with the assertional and terminological knowledge can be specified in CNL. Statements and rules written in CNL can be translated automatically into a format that can be further processed by a model builder. The model builder generates all entailed statements in CNL and allows for question answering over the generated model. The writing of statements and rules in CNL is supported by a look-ahead text editor. The user does not need to worry about the syntactic rules of the CNL and is guided while writing a specification. In summary: CNL can replace an RDF-based ontology language, allows for expressing rules for reasoning in a transparent way, and can empower people to work in cooperation with computers without the need to formally encode the knowledge.

Acknowledgments

The research reported here is supported by the Australian Research Council, Discovery Project DP0449928. The author would also like to thank Marc Tilbrook and two anonymous referees for their careful reviews and precious suggestions.

References

1. S. Abdennadher, F. Bry, N. Eisinger, T. Geisler. 1995. The Theorem Prover Satchmo: Strategies, Heuristics, and Applications (System Description). *Research Report PMS-FB-1995-3*, May, Institut for Informatics, Ludwig Maximilians University, Munich, Germany.
2. T. Berners-Lee, J. Hendler, Ora Lassila. 2001. The Semantic Web. In: *Scientific American*. May 17.
3. D. Brickley, R. V. Guha. 2004. RDF Vocabulary Description Language 1.0: RDF Schema. *W3C Recommendation*, 10 February 2004.
4. F. Bry and A. Yahya. 2000. Minimal Model Generation with Positive Unit Hyper-Resolution tableaux. *Journal of Automated Reasoning*, Vol. 25, Issue 1, July, 35–82.
5. J. de Bruijn, A. Polleres, D. Fensel. 2004. WSML Deliverable, D20 v0.1, OWL Lite$^-$. *WSML Working Draft*, July 18.
6. H. Garcia-Molina, J. D. Ullman, J. D. Widom. 2002. *Database Systems: The Complete Book*. Prentice Hall.
7. B. N. Grosof, I. Horrocks, R. Volz, S. Decker. 2003. Description logic programs: Combining logic programs with description logic. In: *Proceedings of the Twelfth International World Wide Web Conference (WWW 2003)*, pp. 48-57.
8. A. Harth, S. Decker. 2004. D20v02.2 OWL Lite$^-$ Reasoning with Rules. *WSML Working Draft*, November 23.
9. I. Horrocks, P. F. Patel-Schneider. 2003. Three theses of representation in the semantic web. In: *Proceedings of the Twelfth International World Wide Web Conference (WWW 2003)*, pp. 39-47.
10. W. O. Huijsen. 1998. Controlled Language - An Introduction. In: *Proceedings of CLAW 1998*. Pittsburgh, pp. 1-15.
11. F. Manola, E. Miller. 2004. RDF Primer. *W3C Recommendation*, 10 February 2004.
12. R. Manthey and F. Bry. 1988. Satchmo: a theorem prover implemented in prolog. In E. Lusk and R. Overbeek, (eds.), *Proceedings of CADE-88*, Vol. 310 of LNCS, Springer-Verlag, 415–434.
13. F. C. N. Pereira, S. M. Shieber 1987. *Prolog and Natural-Language Analysis*. CSLI Lecture Notes, Number 10, Chicago University Press, Stanford.
14. M. K. Smith, C. Welty, D. L. Mc Guinness. 2004. OWL Web Ontology Language. Guide. *W3C Recommendation*, 10 February 2004.
15. R. Schwitter, M. Tilbrook. 2004. Controlled Natural Language meets the Semantic Web. In: A. Asudeh, C. Paris, S. Wan (eds.). *Proceedings of the Australasian Language Technology Workshop 2004*, Macquarie University, 8th December 2004, pp. 55-62.
16. R. Volz. 2004. Web Ontology Reasoning with Logic Databases. *PhD thesis*, AIFB, Karlsruhe.

Appendix

```
% ============================================================================
% Translated Rule Set for OWL Lite- Reasoning
% ============================================================================

% ----------------------------------------------------------------------------
% Subclass property
% ----------------------------------------------------------------------------

term(C1,[rdfs:subClassOf],C2),
term(C2,[rdfs:subClassOf],C3) --->
  term(C1,[rdfs:subClassOf],C3).

term(C1,[rdfs:subClassOf],C2),
term(E,[rdf:type],C1) --->
  term(E,[rdf:type],C2).

% ----------------------------------------------------------------------------
% Subproperty
% ----------------------------------------------------------------------------

term(P1,[rdfs:subPropertyOf],P2),
term(P2,[rdfs:subPropertyOf],P3) --->
  term(P1,[rdfs:subPropertyOf],P3).

term(P1,[rdfs:subPropertyOf],P2),
term(E,P1,V) --->
  term(E,P2,V).

% ----------------------------------------------------------------------------
% Domain restriction
% ----------------------------------------------------------------------------

term(E,P,V),
term(P,[rdfs:domain],D) --->
  term(E,[rdf:type],D).

% ----------------------------------------------------------------------------
% Range restriction
% ----------------------------------------------------------------------------

term(E,P,V),
term(P,[rdfs:range],R) --->
  term(V,[rdf:type],R).
```

```
% ------------------------------------------------------------------------
% Object property
% ------------------------------------------------------------------------

true ---> term([owl:objectProperty],[rdfs:domain],[owl:class]).
true ---> term([owl:objectProperty],[rdfs:range],[owl:class]).

% ------------------------------------------------------------------------
% Transitive property
% ------------------------------------------------------------------------

term(E,P,W),
term(W,P,V),
term(P,[rdf:type],[owl:transitiveProperty]) --->
  term(E,P,V).

% ------------------------------------------------------------------------
% Symmetric property
% ------------------------------------------------------------------------

term(E,P,V),
term(P,[rdf:type],[owl:symmetricProperty]) --->
  term(V,P,E).

term(P,[rdf:type],[owl:symmetricProperty]),
term(P,[rdfs:range],R) --->
  term(P,[rdfs:domain],R).

term(P,[rdf:type],[owl:symmetricProperty]),
term(P,[rdfs:domain],D) --->
  term(P,[rdfs:range],D).

% ------------------------------------------------------------------------
% Inverse property
% ------------------------------------------------------------------------

term(E,P1,V),
term(P2,[owl:inverseOf],P1) --->
  term(V,P2,E).

true --->
  term([owl:inverseOf],[rdf:type],[owl:symmetricProperty]).

true --->
  term([owl:inverseOf],[rdfs:domain],[owl:objectProperty]).

true --->
  term([owl:inverseOf],[rdfs:range],[owl:objectProperty]).
```

```
% ------------------------------------------------------------------------
% Class equivalence
% ------------------------------------------------------------------------

true --->
  term([owl:equivalentClass],[rdf:type],[owl:symmetricProperty]).

true --->
  term([owl:equivalentClass],[rdf:type],[owl:transitiveProperty]).

term(C1,[owl:equivalentClass],C2),
term(E,[rdf:type],C2) --->
  term(E,[rdf:type],C1).

term(C1,[owl:equivalentClass],C2),
term(E,C1,V) --->
  term(E,C1,V).

term(C1,[owl:equivalentClass],C2),
term(E,C1,V) --->
  term(E,C2,V).

term(C1,[owl:equivalentClass],C2),
term(C2,P,V) --->
  term(C1,P,V).

term(C1,[owl:equivalentClass],C2),
term(E,P,C1) --->
  term(E,P,C2).

% ------------------------------------------------------------------------
% Property equivalence
% ------------------------------------------------------------------------

true --->
  term([owl:equivalentProperty],[rdf:type],[owl:symmetricProperty]).

true --->
  term([owl:equivalentProperty],[rdf:type],[owl:transitiveProperty]).

term(P1,[owl:equivalentProperty],P2),
term(E,P1,V) --->
  term(E,P1,V).

term(P1,[owl:equivalentProperty],P2),
term(E,P1,V) --->
  term(E,P2,V).
```

```
term(P1,[owl:equivalentProperty],P2),
term(P2,P,V) --->
  term(P1,P,V).

term(P1,[owl:equivalentProperty],P2),
term(E,P,P1) --->
  term(E,P,P2).

% ------------------------------------------------------------------------
% Property restriction
% ------------------------------------------------------------------------

term(E,P,V),
term(R,[rdf:type],[owl:restriction]),
term(R,[owl:onProperty],P),
term(R,[owl:allValuesFrom],C) --->
  term(E,[rdf:type],R).
```