Automated story comprehension

This notebook shows how to extract background knowledge and relations from a story text. To this end, we use Stanford's NLP parser available to download here (https://nlp.stanford.edu/software/lex-parser.html#Download).

We parsed the following story (`story.txt` file):

```
Mary was sleeping.
Her phone rang.
She was annoyed.
Mary answered the phone.
Ann told the good news to Mary.
```

with

```
java -cp "*" -Xmx2g edu.stanford.nlp.pipeline.StanfordCoreNLP -annotators
 tokenize,ssplit,pos,lemma,ner,parse,dcoref -file story.txt
```

This resulted in an xml file `story.txt.xml` that contains all the necessary data to extract relevant information from the story.

The next step is to extract this information and represent it as Prolog facts. To this end, we wrote the Python code included below.

In [1]:

```
# Import libraries
from xml.etree.ElementTree import parse as xml_parse
from pprint import pprint
```

In [2]:

```
# Read in the xml file with the parsed story
e = xml_parse("story.txt.xml").getroot()
```

In [3]:

```
# path_tags is a list of tuples: [(tag, attribute, attribute_id), ...]
def get_xml_struct(root, path_tags):
    current = root
    for tag, attribute, attribute_value in path_tags:
        for c in current:
            if c.tag == tag:
                if attribute is not None and attribute_value is not None:
                    if c.attrib.get(attribute) == attribute_value:
                        current = c
                        break
                else:
                    current = c
                    break
    return current

def get_attrib_value(root, dep_type, dependent=True):
    identifiers = {}
    for c in root:
        if c.tag == "dep":
            if c.attrib.get("type", "") == dep_type:
                for d in c:
                    if dependent and d.tag == "dependent":
                        return d.attrib.get("idx", "")
                    identifiers[d.tag] = d.attrib.get("idx", "")
    return identifiers

def get_token_lemma(tokens, token_id):
    for t in tokens:
        if t.tag == "token" and t.attrib.get("id", "") == token_id:
            for i in t:
                if i.tag == "lemma":
                    return i.text
    return "none"
```

In [4]:

```
# get sentences ids
sentence_ids = []
t = get_xml_struct(e, [("document", None, None), ("sentences", None, None)])
for c in t:
    id = c.attrib.get("id")
    if id is not None:
        sentence_ids.append(id)

print "Sentence ids:"
print sentence_ids
```

```
Sentence ids:
['1', '2', '3', '4', '5']
```

In [5]:

```
sentence_dependencies = {}
for sid in sentence_ids:
    sentence_structure = {"root":"none", "nsubj":"none", "dobj":"none",
"nmod":"none"}
    # get basic-dependencies
    basicd = get_xml_struct(e, [("document", None, None), ("sentences", None, No
ne), \
                                ("sentence", "id", sid), ("dependencies",
"type", "basic-dependencies")
                               ]
                           )
    # get lemmas
    tokens = get_xml_struct(e, [("document", None, None), ("sentences", None, No
ne), \
                                ("sentence", "id", sid), ("tokens", None, None)
                               ]
                           )

    for i in ["nsubj", "nsubjpass"]:
        it = get_token_lemma(tokens, get_attrib_value(basicd, i)).lower()
        if it != "none":
            sentence_structure["nsubj"] = it
    for i in ["root", "dobj", "nmod"]:
        it = get_token_lemma(tokens, get_attrib_value(basicd, i)).lower()
        if it != "none":
            sentence_structure[i] = it
    sentence_dependencies[sid] = sentence_structure

print sentence_dependencies
```

```
{'1': {'dobj': 'none', 'root': 'sleep', 'nmod': 'none', 'nsubj': 'ma
ry'}, '3': {'dobj': 'none', 'root': 'annoy', 'nmod': 'none', 'nsub
j': 'she'}, '2': {'dobj': 'none', 'root': 'ring', 'nmod': 'none', 'n
subj': 'phone'}, '5': {'dobj': 'news', 'root': 'tell', 'nmod': 'mar
y', 'nsubj': 'ann'}, '4': {'dobj': 'phone', 'root': 'answer', 'nmo
d': 'none', 'nsubj': 'mary'}}
```

In [6]:

```
predicates_string = ""
for i in sorted(sentence_dependencies.keys()):
    si = sentence_dependencies[i]
    si["time"] = i

    x = "s(1) :: %(root)s(%(nsubj)s, %(dobj)s, %(nmod)s) at %(time)s." % si

    x = x.lower()
    predicates_string += x + "\n"
```

Processing output

`xml` parsing gives us the following logical representation of the story:

In [7]:

```
print predicates_string,
```

```
s(1) :: sleep(mary, none, none) at 1.
s(1) :: ring(phone, none, none) at 2.
s(1) :: annoy(she, none, none) at 3.
s(1) :: answer(mary, phone, none) at 4.
s(1) :: tell(ann, news, mary) at 5.
```

In [8]:

```
# Save the rules to the `extracted_knowledge.pl` file
with open("extracted_knowledge.pl", "w") as rules_file:
    rules_file.write(predicates_string)
```

In [ ]:

```

```