위키피디아 문서로부터 트리플 추출과 RDF 그래프 생성

이순웅○ 최기선
한국과학기술원 시맨틱웹 연구센터
{swlee, kschoi}@world.kaist.ac.kr

Triple Extraction for RDF Graph Construction from Wikipedia Articles

SoonWoong Lee○ KeySun Choi
Semantic Web Research Center, KAIST

요 약

웹이 발전하면서 점점 더 많은 정보가 웹을 통해 생성되고 공유되고 있다. 하지만 정보의 급격한 증가로 인해 정작 정확한 정보를 찾는 것은 오히려 더 어려워지고 있고, 이로 인해 특히 구조화되지 않은 텍스트에 대한 정확한 정보 검색의 필요성이 증가하고 있다. 본 논문에서는 위키피디아 문장들로부터 RDF 트리플을 추출하고 이를 하나의 연결된 RDF 그래프로 구성함으로써 효과적인 정보 검색을 수행하는 방법을 제안하고자 한다. 트리플 추출 방법은 문장에 대한 파스 트리를 탐색함으로써 이루어지는데, 약 81%의 정확도를 나타내었다. 최종적으로 생성되는 RDF 그래프는 입력 문장들의 문법적인 요소만을 고려하기 때문에 방법이 단순하지만 그래프 탐색을 통해 다양한 쿼리에 대한 정보 검색이 가능하다.

위키피디아, RDF, 트리플

1. 서 론

웹상에 정보가 늘어나면서, 정확한 정보를 찾는 것은 오히려 더욱 어려워지고 있다. 이러한 정보 과다로 인한 문제를 해결하기 위해서 효과적인 검색 기술이 요구되어 지고 있다. 이에 따라 데이터를 어떻게 구조화 할 것인지와 구조화된 데이터에서 어떻게 하면 정확한 정보를 찾을 것인지에 대한 시도가 많이 이루어지고 있다. DBPedia[1]는 그러한 시도 중 하나로써, 위키피디아의 정보를 RDF 트리플 형태로 바꾸는 것을 목표로 하고 있다. 이를 통해 기존 키워드 방식 검색에 비해 훨씬 정교한 검색 결과를 제공하고자 한다. 하지만 웹상의 많은 데이터가 구조화되지 않은 형태 즉 자연어 텍스트로 존재한다. 이는 사람이 정보를 기록하는 가장 손쉬운 방법 중의 하나가 자연어 텍스트로 기록하는 것이기 때문이다. 이러한 구조화되지 않은 데이터에 대해 효과적인 정보 검색을 하기 위해서는 데이터를 기계가 해석할 수 있는 구조화된 형태로 바꾸어야 한다. 본 논문에서는, 자연어 텍스트를 구조화된 형태의 하나인RDF 그래프 형태로 바꾸는 방법을 제안하고자 한다. 이 때 자연어 텍스트는 매우 다양하고 방대한 범위이기 때문에 본 연구에서는 위키피디아의 영어 문장으로 한정한다. 위키피디아 영어 문장으로 정한 이유는, 많은 사람들에 의해 정제된 문장이기 때문에 기계로 처리하기에 비교적 수월한 자연어 문장이라고 가정하였기 때문이다. 따라서 제안하는 시스템의 입력은 위키피디아 영어 문장의 집합이며 출력은 하나로 연결된 RDF 그래프이다. 이때 의미적인 정보는 고려하지 않고 단지 문법적인 요소만 고려하여 RDF 그래프를 구성하게 된다. 문법적인 요소만 고려하는 비교적 단순한 방법을 사용하더라도, 구성된 RDF 그래프에 대한 탐색을 통해 기존 키워드 방식으로 찾기 어려운 정보들을 검색할 수 있음을 보이고자 한다. 이 논문의 구성은 다음과 같다. 2장에서는 코퍼스로부터 자동으로 트리플을 추출하는 방법에 대한 기존 연구들을 살펴본다. 3장에서는 본 논문에서 제안하는 자동 트리플 추출 방법 및 RDF 그래프 생성 방법을 기술한다. 추출된 트리플에 대한 평가 결과가 4장에 기술되며, 5장에서는 본 논문의 결론을 맺고 앞으로의 연구 방향에 대해 기술한다.

2. 관련 연구

비구조화된 대량의 텍스트로부터 트리플을 추출하고자 하는 연구가 활발히 이루어지고 있는데, 그 중 KnowItAll[2]은 공개된 첫 번째 시스템이었다. 대량의 웹 코퍼스에서 비지도학습 방법으로 트리플을 추출하기 위해 패턴을 사용하였는데, 초기에 지정된 패턴에 해당하는 트리플들만 추출하는 방법을 사용하였다. 패턴은 품사 태깅 정보와 개체명 인식기 등을 사용하여 정의하였다. KnowItAll을 개량한 것이 TextRunner[3]이다. KnowItAll과 마찬가지로 대량의 웹 코퍼스로부터 트리플을 추출하는 시스템인데, 자기감독학습 방법을 사용하였다. 가장 큰 특징은, 학습 데이터를 자동으로 생성한다는 것이다. 이를 위해, 먼저 주어진 코퍼스 중 일부에 대하여 품사 태깅을 한 후 자질로써 사용한다. 그리고 구문 분석을 통하여 실제로 트리플로 구성될 수 있는지 아닌지를 판별하는 것이다. 위 학습 데이터를 활용하여 베이지안 분류기를 학습시키고, 이것을 활용하여 대량의 코퍼스로부터 자동으로 트리플을 추출하는 것이다. Ramakrishnan et al.[4]은 생물학 분야의 문장들에서 트리플을 추출하는 방법을 연구하였다. 분야의 특성상 복잡한 개체명이 많기 때문에 이를 트리플로 올바르게 추출하는 방법을 제안하였다. 먼저 문장에 대해 구문 분석을 수행 후 토큰간의 의존 관계 타입을 살펴봄으로써 트리플을 추출하는 방법을 사용하였다. Dong-Hyun Choi et al.[5]은 영어 문장으로부터 자동으로 트리플을 추출하는 방법을 제안하

였다. 하나의 문장에 있는 모든 정보를 하나의 트리플로 표현하기 위해, 트리플 내에 AND를 사용하여 여러 개의 개념들을 연결한 표현 방식이 특징이다. 본 논문에서 제안하는 방법은, 트리플을 추출하기 위해 먼저 문장에 대해 구문 분석을 수행 후 파스트리를 활용한다는 면에서 [2]와 [3] 보다는 [4]와 [5]의 방법과 비슷하다. 하지만 본 연구에서는 위키피디아 문장을 대상으로 하고 있으며, 동사-논항 구조를 활용한 트리플 표현 방법 및 추출 방법을 사용한다는 면에서 위 연구들과 차이점이 있다.

3. 제안하는 방법

위키피디아 영어 문장들을 하나의 연결된 RDF 그래프로 생성하는 방법은 2가지 단계로 이루어진다. 첫 번째 과정은 위키피디아 영어 문장으로부터 트리플을 추출하는 부분이며, 두 번째 과정은 추출된 트리플들로부터 하나의 연결된 RDF 그래프를 생성하는 부분이다. 이렇게 생성된 RDF 그래프에 대해 그래프 탐색을 통해 정보 검색이 수행된다.

3.1. 트리플 형태

주어진 위키피디아 영어 문장 집합에 대해 먼저 각 문장별로 트리플들을 추출하게 된다. 이때 트리플은 아래와 같이 정의된다.

<subject, property, object>

subject와 object사이에는 property의 관계가 존재한다는 것을 나타낸다. 입력 문장 및 추출된 트리플의 예제는 그림 1과 같다. Input은 입력 문장을 나타낸다. SPO-Triples은 다른 관련 연구[4][5]에서 볼 수 있는 트리플의 형태로, 동사에 해당하는 부분이 트리플의 property에 해당이 된다. 주어진 문장에 2개의 동사 manufactured와 is가 있으므로 2개의 SPO-Triples가 생성될 수 있다. 이때 각 트리플의 subject와 object로는 각 동사의 논항들이 나타나게 된다. PA-Triples은 본 논문에서 제안하는 방법에서 사용하는 트리플의 형태이다. SPO-Triples와 다른 점은, 문장에서 동사에 해당하는 부분이 트리플의 subject에 위치한다는 것이다. property에는 arg1, arg2, arg3 중 하나가 나타나는데, 각각 주어, 직접 목적어, 간접 목적어를 의미한다. 예제에서 주어진 문장에 2개의 동사가 있고, 각 동사의 주어 및 직접 목적어가 문장에 나타나 있으므로 총 4개의 PA-Triples가 생성될 수 있다. 또한 아래의 예제에서 PA-Triples를 2개씩 짝지어 합치면 하나의 SPO-Triples를 얻을 수 있게 된다. 즉 SPO-Triples이 표현할 수 있는 모든 정보는 PA-Triples로도 표현할 수 있다. PA-Triples XML은 제안하는 방법으로 구현된 시스템에서 실제로 사용하고 있는PA-Triples의 XML 형태이다. PA-Triples와의 차이점은 XML 형태라는 것 외에, subject type과 object type이라는 것이 추가됐다는 점이다. subject type은 subject로 사용된 동사의 원형을 나타내고, object type은 object로 사용된 논항의 중심어를 나타낸다. subject type과 object type은 이후 단계인 RDF 그래프 생성 과정에서 사용된다.

- **Input**

Z Portable is a handheld game console manufactured by X Computer.

- **SPO-Triples**

<X Computer, manufactured, handheld game console>
<Z Portable, is, handheld game console>

- **PA-Triples**

<manufactured, arg2, handheld game console>
<manufactured, arg1, X Computer>
<is, arg2, handheld game console>
<is, arg1, Z Portable>

- **PA-Triples XML**

```
<triplet><subject type="MANUFACTURE">manufactured</subject>
    <predicate>arg2</predicate>
    <object type="CONSOLE">handheld game console</object></triplet>
<triplet><subject type="MANUFACTURE">manufactured</subject>
    <predicate>arg1</predicate>
    <object type="COMPUTER">X Computer</object></triplet>
<triplet><subject type="BE">is</subject>
    <predicate>arg2</predicate>
    <object type="CONSOLE">handheld game console</object></triplet>
<triplet><subject type="BE">is</subject>
    <predicate>arg1</predicate>
    <object type="PORTABLE">Z Portable</object></triplet>
```

그림 1. 추출된 트리플 예제

3.2. 트리플 추출 방법

입력 문장으로부터 PA-Triples 형태의 트리플을 추출하는 과정은 다음과 같다. 먼저 하나의 문장에 대하여 구문 분석을 수행하여 파스트리를 추출한다. 이는 기존에 나와 있는 구문 분석 파서들을 사용할 수 있는데, 본 논문에서는 Enju[6] 파서를 사용하였다. 추출된 파스트리에 대하여 트리를 탐색하며 동사-논항 구조를 추출하고 이를 트리플 형태로 변환하게 된다. 이에 대한 알고리즘은 그림 2에 나타나 있다. 입력으로 받는 node는 탐색을 시작할 파스트리의 노드를 나타낸다. 위 알고리즘은 입력으로 받은 노드의 하위 노드들 중 먼저 동사에 해당하는 노드를 찾는다. 그런 후에 구문분석을 통해 얻어진 의존 관계 정보를 이용하여 파스트리 상에서 위 동사와 주어 및 목적어 관계에 있는 다른 노드들을 찾아낸 후 이들을 동사와 함께 하나의 트리플로써 추출하게 된다. 즉 하나의 명사구가 하나의 용어로써 주어 또는 목적어로 트리플에 포함된다. 이때 논항에 해당하는 노드가 동사일 경우에는 해당 노드로부터 위의 과정이 재귀적으로 반복된다.

Algorithm: extractPATriples(parse_tree, start_node, visit_list)

Input:

parse_tree=parse tree which contains dependency relations between the sentence tokens
start_node=start node to visit
visit_list=a list of visited nodes

Output:

visited_txt=tokens below the visited nodes
trilist=a list of extracted triples

```
1  visited_txt=""
2  trilist={}
3  verb_node=None
4  if start_node==None: return visited_txt, trilist
5  if start_node==LeafNode: visited_txt=start_node.text
6  verb_node=find a start_node's child node whose POS-tag is verb
7  if verb_node!=None:
8     add verb_node to visit_list
9     foreach arg_node in verb_node's subject or object
10       add arg_node to visit_list
11       subject_text=verb_node.text
12       property_text=get dependency type between verb_node and arg_node
13       object_text, subtrilist = extractPATriples(parse_tree,arg_node,visit_list)
14       add subtrilist to trilist
15       add a triple<subject_text, property_text, object_text> to trilist
16    end
17    return subject_txt, trilist
18 else:
19    foreach node in start_node's below nodes:
20       add node to visit_list
21       subtxt, subtrilist = extractPATriples(parse_tree,node,visit_list)
22       add subtrilist to trilist
23       txt=txt+subtxt
24    end
25 endif
26 return txt, trilist
```

그림 2. 파스트리로부터 PA-Triples을 추출하는 알고리즘

3.3. 트리플로부터 RDF 그래프 생성 방법

앞의 트리플 추출 과정을 통해 주어진 문장 집합으로부터 트리플 집합이 생성된다. 그런 후에 트리플 집합을 하나의 연결된 RDF 그래프로 만들게 된다. RDF 그래프에서 노드는 클래스 또는 인스턴스에 해당이 되며, 간선은 클래스 또는 인스턴스 간의 관계를 나타낸다. 그림3은 주어진 2개의 문장으로부터 최종적으로 생성된 RDF 그래프를 보여준다. 이와 같이 PA-Triples를 RDF 그래프로 만들기 위해 먼저 'Event'라는 클래스와 'Entity'라는 클래스를 정의한다.

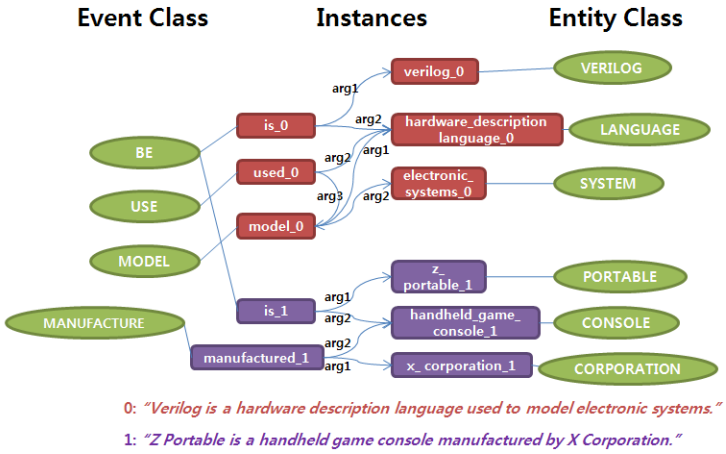

그림 3. 트리플로부터 생성된 RDF 그래프 예제

3.3.1 Event 클래스 및 인스턴스

'Event' 클래스는 동사들을 연결하는 최상위 클래스이며, '동사 클래스'들을 하위 클래스로 가진다. '동사 클래스'는 동사의 기본형을 이름으로 하는 클래스들이다. 예를 들면 'Make', 'Build', 'Release' 등을 이름으로 갖는다. 그림3의 Event Class 부분이 '동사 클래스'의 예제이다. 그리고 각 '동사 클래스'는 '동사 인스턴스'들을 인스턴스로 가진다. '동사 인스턴스'는 실제로 문장에 나타난 동사들을 나타내는데, 이름으로는 나타난 동사 문자열 뒤에 언더바와 문장 인덱스가 붙는다. 예를 들면 'used_2'는 인덱스 2인 문장에서 나타난 'used'라는 동사를 '동사 인스턴스'로 나타낸 것이다. 그림3의 Instances 중 Event Class 와 연결된 노드들이 '동사 인스턴스'의 예제이다.

3.3.2 Entity 클래스 및 인스턴스

'Entity' 클래스는 논항들을 연결하는 최상위 클래스이며, '논항 클래스'들을 하위 클래스로 가진다. '논항 클래스'는 논항의 중심어를 이름으로 하는 클래스들이다. 예를 들면 논항의 문자열이 'electronic systems'일 경우 중심어인 'system'이 '논항 클래스'의 이름이 된다. 그림3의 Entity Class 부분이 '논항 클래스'의 예제이다. 그리고 '논항 클래스'는 '논항 인스턴스'들을 인스턴스로 가진다. '논항 인스턴스'는 실제로 문장에 나타난 논항들을 나타내는데, 이름으로는 나타난 논항 문자열 뒤에 언더

바와 문장 인덱스가 붙는다. 예를 들면 'hardware_description_language_2'는 인덱스 2인 문장에서 나타난 'hardware description language'를 말한다. 그림3의 Instances 중 Entity Class 와 연결된 노드들이 '논항 인스턴스'의 예제이다.

3.3. RDF 그래프 탐색을 통한 정보 검색

위 과정을 통해 생성된 RDF 그래프는 다음과 같은 이점이 있다. 첫째는 RDF 트리플들로 이루어졌기 때문에, RDF에 대한 쿼리 언어인 SPARQL을 그대로 사용하여 그래프 탐색이 가능하다는 점이다. 두번째는 각 문장의 동사-논항 정보를 포함하고 있기 때문에, 이 정보를 활용한 정보 검색이 가능하다는 점이다. 예를 들어 'X Corporation이 생산한 것이 무엇인가?'라는 사용자 쿼리에 대해 아래와 같은 트리플 형태로 검색이 가능하다

<X Corporation, manufactured, ?>

그림 3의 예제에서 위와 같은 쿼리를 처리하는 과정은 다음과 같다. 먼저 'Entity Class'의 모든 인스턴스들에 대하여 X Corporation이라는 문자열을 이름에 포함하는 인스턴스들을 찾아본다. 예제에서 'x_corporation_1'을 찾게 되면, 이를 'arg1'이나 'arg2' 혹은 'arg3'로 가지는 '동사 인스턴스'가 존재하는지 찾아본다. 이에 대한 SPARQL 쿼리는 아래와 같다.

```
SELECT ?subject ?prop ?obj WHERE
{
subject ?prop ?obj
FILTER regex(str(?subject), "", "i")
FILTER regex(str(?prop), "arg", "i")
FILTER regex(str(?obj), "x_corporation_1", "i")
}
```

위 쿼리의 결과로부터 <manufactured_1, arg1, x_corporation_1>과 같은 트리플을 찾을 수 있다. 이제 쿼리에 대한 답을 찾기 위해 'manufactured_1'과 'arg2'의 관계에 있는 인스턴스를 찾아야 한다. 이를 위한 SPARQL 쿼리는 아래와 같다.

```
SELECT ?subject ?prop ?obj WHERE
{
?subject ?prop ?obj
FILTER regex(str(?subject), "manufactured_1", "i")
FILTER regex(str(?prop), "arg2", "i")
FILTER regex(str(?obj), "", "i")
}
```

위 쿼리의 결과로부터 <manufactured_1, arg2, handheld_game_console_1> 트리플을 찾을 수 있고, handheld game console이 쿼리에 대한 답이 된다. 즉 SPARQL 쿼리를 조합함으로써 트리플 형태의 검색이 가능한 것이다. 만약 동사-논항 정보가 포함되지 않은 상태에서 'X Corporation'과 'manufactured' 키워드로만 찾는다면, 두 키워드를 모두 포함하는 모든 문장이 나타날 것이고 그 중에는 위 쿼리에 대한 답이 아닌 문장들도 나타나게 된다.

4. 평 가

제안한 시스템은 주어진 입력 문장들에 대해 각각의 문장별로 PA-Triples을 추출한 후 하나의 연결된 RDF 그래프로 생성해준다. 생성된 RDF그래프의 평가를 위해 그래프를 이루는 각각의 PA-Triples을 평가하였는데, 2개의 PA-Triples 쌍을 같은 의미를 가지는 1개의 SPO-Triple로 변환하여 평가하였다. 같은 정보를 갖지만 사람이 평가하기에는 2개의 PA-Triples보다는 1개의 SPO-Triple 평가가 더 쉽기 때문이다. 정확도 평가는 추출된 트리플들이 주어진 문장의 내용에 대하여 맞는지 틀린지를 수작업으로 판단하는 방식으로 수행하였다. 이를 위해 9,872개의 위키피디아 페이지를 무작위로 선정한 후 첫 문장들만을 추려내어 트리플 추출을 시도하였다. 그 결과 34,288개의 PA-Triples이 추출되었다. 그리고 같은 subject를 가지면서 arg1과 arg2에 대한 트리플이 존재하는 PA-Triples 2개를 하나의 SPO-Triples로 만드는 방법을 통하여 총 14,097개의 SPO-Triples을 얻을 수 있었다. SPO-Triples의 subject 즉 동사들에 대해 출현 빈도수가 높은 상위 7개는 표1과 같았다.

표 1. Subject에 따른 출현 빈도수

Subject	Frequency
is	7,607
was	1,063
developed	273
are	264
created	158
produced	99
released	83

이 중 be동사를 제외하고 출현 빈도수가 높은 상위 4개의 동사 즉, developed, created, produced, released에 대한 트리플들을 모아 수작업을 통해 평가를 하였다. be동사를 평가에서 제외한 이유는 상대적으로 트리플 추출하기 쉬운 문장이 많았기 때문이다. 수작업 평가는 2명의 평가자에 의해 중복 수행되었다. 트리플 평가 결과는 표 2와 같다.

표 2. 트리플 평가 결과

	Total	Correct	Incorrect	Precision
developed	273	220	53	0.81
created	158	132	26	0.84
produced	99	78	21	0.79
released	83	64	19	0.77
Total	613	494	119	0.81

평균 약 81%의 정확도를 보임을 알 수 있다. 또한 잘못되었다고 평가된 트리플들의 유형을 파악해 본 결과는 표3과 같다.

표 3. 트리플 오류 유형 및 빈도수

	Frequency(Percentage)
Subject 오류	28(23.5%)
Object 오류	63(52.9%)
Subject & Object오류	4(3.4%)
원문 오류	3(2.5%)
평가 오류	8(6.7%)
기타	13(10.9%)

Subject 오류는 SPO-Triples에서 subject 즉 주어에 해당하는 부분이 잘못 되었다는 것을 말하고, Object 오류는 SPO-Triples에서 object 즉 목적어에 해당하는 부분이 잘못 되었다는 것을 말한다. Subject & Object 오류는 주어와 목적어에 해당하는 부분 둘 다 잘못된 경우를 말한다. 원문 오류는 원문장에 문법적인 오류가 있는 경우이며, 평가 오류는 2명의 평가자가 서로 다르게 평가를 한 경우이다. 빈도수에서 알 수 있듯이, SPO-Triples에서 object 즉 목적어를 잘못 생성한 경우가 절반 가까이를 차지하였다. 이는 제안하는 트리플 추출 방법이 파스트리에 의존적인 특성상 파싱 에러로 인한 오류가 많다고 볼 수 있다.

5. 결 론

본 논문에서는 향상된 위키피디아 정보 검색을 위해 위키피디아 문장들로부터 RDF 트리플을 추출한 후 이를 하나의 RDF 그래프로 변환하는 시스템을 제안하였다. 이때 의미적인 정보는 고려하지 않고, 문법적인 요소만 사용하였다. 생성된 RDF 그래프는 문장의 동사-논항 정보를 포함하며, 동사 원형과 중심어를 통해 하나로 연결된 형태이다. 따라서 단순 키워드 검색 뿐 아니라 그래프 탐색을 통해 동사-논항 정보를 필요로 하는 쿼리에 대해서도 처리가 가능하다. 하지만 파서 에러로 인해 잘못 생성되는 트리플이 많음을 알 수 있었다. 따라서 앞으로 어떻게 파서에 대한 의존도를 줄일 것인지가 추가로 연구되어야 한다. 이를 위해 구문 분석보다 비교적 성공률이 높은 부분 구문 분석을 먼저 수행 후 그 결과를 활용해 문장을 단순화하는 방안에 대해 연구를 진행 중이다. 그리고 문법적인 요소만 사용하여RDF 그래프를 구성하였는데, 의미적인 정보를 어떻게 포함할 것인지에 대한 연구가 진행 중이다.

감사의 글

본 논문은 지식경제부 및 정보통신연구진흥원의 정보통신선도기반기술개발사업과 2009년도 두뇌한국 21사업의 지원을 받아 수행되었습니다.

참고 문헌

[1] DBPedia, http://www.dbpedia.org

[2] Oren Etzioni, Michael Cafarella, Doug Downey, Stanley Kok, Ana-Maria Popescu, Tal Shaked, Stephen Soderland, Daniel S. Weld and Alexander Yates, Web-scale information extraction in knowitall, *Proceedings of the 13th international conference on World Wide Web*, pp. 100 - 110, 2004.

[3] Alexander Yates and Oren Etzioni, Unsupervised Resolution of Objects and Relations on the Web, *Proceedings of NAACL HLT*, pp. 121 - 130, 2007.

[4] Ramakrishnan Cartic, Mendes Pablo N., Gama Rodrigo A., Ferreira Guilherme C., and Shet Amit P., Joint Extraction of Compound Entities and Relationships from Biomedical Literature, *IEEE/WIC/ACM International Conference on In Web Intelligence and Intelligent Agent Technology*, Vol. 1, pp. 398-401, 2008.

[5] Dong-Hyun Choi and Key-Sun Choi, Automatic Triplet Relation Extraction by Dependency Parse Tree Traversing, *EKAW Poster*, 2008.

[6] Kenji Sagae, Yusuke Miyao and Jun'ichi Tsujii, HPSG Parsing with Shallow Dependency Constraints, *Proceedings of the 45th Annual Meeting of the ACL*, 2007.