규칙 기반 영한 기계번역에서의 구문 규칙 컴파일러

김성동*
*한성대학교 컴퓨터공학과
e-mail:sdkim@hansung.ac.kr

Syntactic Rule Compiler in Rule-based English-Korean Machine Translation

Kim, Sung-Dong
Dept. of Computer Engineering, Hansung University

요 약

규칙 기반의 영한 기계번역 시스템의 구문 분석 시스템은 영어의 구문 구조를 기술하는 규칙 부분과 규칙을 적용하여 차트 파싱을 수행하는 실행 부분으로 구성된다. 구문 규칙은 문맥 자유 문법의 형식으로 기술되는데, 기술된 구문 규칙을 적용하여 파싱을 실행하는 실행 부분은 C 언어 함수로 표현되므로, 구문 규칙을 C 언어 함수로 변환해야 한다. 본 논문에서는 문맥 자유 문법 형식으로 기술된 구문 규칙을 C 언어 함수로 변환하는 도구인 구문 규칙 컴파일러를 개발하였다. 구문 규칙 컴파일러는 자동적으로 구문 규칙을 C 언어 함수로 변환함으로써 영한 기계번역 시스템의 성능 개선 과정에서 빈번하게 발생하는 구문 규칙의 생성과 수정을 용이하게 하여 번역 성능을 개선하는 작업을 지원한다.

1. 서론

규칙 기반 기계번역(rule-based machine translation: RBMT) 방법은 기계번역 방법 중 가장 오래된 역사를 가지고 있으며 언어학 기반(linguistic-based) 방법이라고도 불린다. 최근에는 통계적 기계번역(statistical machine translation) 방법에 대한 연구가 활발하여 통계적 방식의 기계번역 시스템이 많이 개발되어 사용되고 있다. 그러나 영어와 한국어처럼 매우 상이한 언어를 대상으로 하는 영한 기계번역은 통계적 방법만으로는 정확한 번역을 생성하기 매우 어려우며, 규칙 기반 방법을 적용하고 부분적으로 통계적 방법을 활용하여 모호성 문제를 해결하는 것이 바람직하다는 판단이다. 본 논문에서 대상으로 하는 영한 기계번역 시스템은 규칙 기반 방법으로 개발된 시스템이며 영한 기계번역 시스템의 구성 요소 중 구문 분석기(parser)를 주제로 한다.

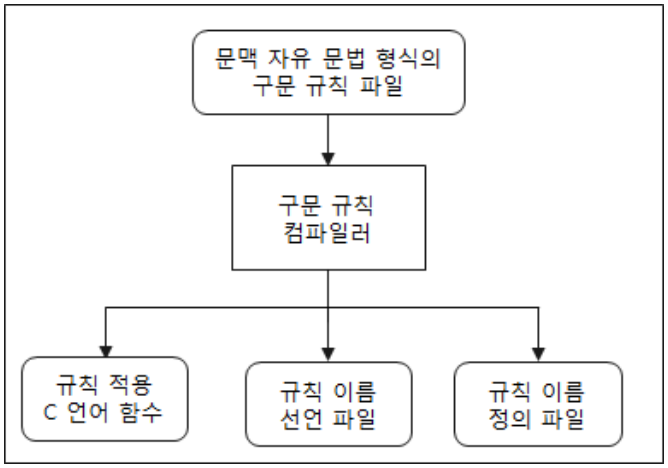
규칙 기반 영한 기계번역 시스템의 구문 분석기는 영어의 구문 구조에 대한 정보를 포함하는 지식부(knowledge part)와 지식을 활용하여 구문 분석을 수행하는 실행부(action part)로 구성된다. 지식부는 문맥 자유 문법(context-free grammar)로 영어의 구문 구조를 표현하고 실행부는 문법을 적용하여 입력 문장의 구조를 분석하는데, 이때 차트 파싱(chart parsing)[1]을 수행한다. 즉 실행부는 C 언어 함수로 표현된다. 따라서 문맥 자유 문법으로 기술된 구문 규칙을 C 언어 함수로 변환해야 하며, 본 논문에서는 이러한 변환을 자동적으로 수행하는 구문 규칙 컴파일러(syntactic rule compiler)를 개발하였다.

구문 규칙은 영한 기계번역 시스템을 개발할 때 한번만 기술되는 것이 아니고 시스템의 성능 개선 과정에서 지속적으로 수정될 수 있을 뿐만 아니라, 새로 추가 또는 삭제될 수 있다. 결과적으로 문맥 자유 문법으로 기술된 구문 규칙에 대응하는 C 언어 함수도 지속적 수정과 추가, 삭제가 이루어지는데, 사람에 의한 변환은 매우 많은 노력을 요구하기도 하지만 오류의 가능성도 매우 높다. 제안하는 구문 규칙 컴파일러는 문맥 자유 문법으로 기술된 구문 규칙을 자동적으로 C 언어 함수로 변환하기 때문에 사람의 노력을 줄이고 변환에서의 오류도 제거할 수 있기 때문에 지속적인 영한 기계번역 시스템의 개선을 지원할 것이다.

본 논문은 다음과 같이 구성된다. 2장에서는 구문 규칙 컴파일러의 입력과 출력을 분석하고 3장에서는 구문 규칙 컴파일러의 구조를 설명한다. 4장에서는 앞으로의 과제를 제시하며 논문을 마무리한다.

2. 구문 규칙 컴파일러의 입력과 출력

구문 규칙 컴파일러는 문맥 자유 문법 형식으로 기술한 영어 구문 규칙 파일을 입력받아 차트 파싱을 수행하는 C 언어 함수를 출력한다. 또한 구문 분석 실행부의 완전한 구성을 위해 필요한 정보를 포함하는 파일들을 함께 생성해야 한다. (그림 1)은 구문 규칙 컴파일러의 입출력 파일을 보여주며 (그림 2)는 입력 파일의 예이다.



(그림 1) 구문 규칙 컴파일러의 입출력 파일 관계

```
NP010: NOUN(COORDED=0)
->  NP(%NOUN, cat='NP, OBJP=1, SUBJP=1,
      RefIndicToScore NOUN POSSESIVE 1 0.5,
      score=1.02)
```

(그림 2) 구문 규칙 컴파일러의 입력 예

(그림 2)는 하나의 명사(NOUN)가 명사구(NP)가 된다는 규칙을 나타내는데, 각각의 규칙은 (그림 3)과 같은 형식으로 기술된다. 규칙 이름(RuleName)으로 시작하고 ':' 다음부터 하나 이상의 문맥 자유 문법의 오른편(right hand side: RHS) 기호가 나오고 '->' 기호 다음에 왼편(left hand side: LHS) 기호가 위치한다. 기호 다음의 괄호 안에는 규칙 적용의 조건을 나타내는 *condition*과 규칙이 적용될 때 규칙 적용의 결과를 기록하는 *action*이 함께 기술된다.

```
RuleName: RHS(condition) { RHS(condition) }
        -> LHS(action)
```

(그림 3) 구문 규칙 기술 형식

(그림 4) ~ (그림 6)은 구문 규칙 컴파일러의 3가지 출력 파일의 예를 보여준다. (그림 4)에서와 같이 입력 규칙의 RHS와 LHS에 대해서 각각 1개의 C 함수가 생성된다. RHS에 대한 함수는 규칙 적용의 조건을 검사하는 루틴이고, LHS에 대한 함수는 규칙 이름을 그대로 함수 이름으로 사용하며 규칙 적용의 결과 트리를 생성하는 명령어로 구성된다. (그림 5)의 규칙 이름 선언은 차트 파싱 과정에서 적용할 함수를 결정할 때 이용되고, (그림 6)의 규칙 이름 정의는 적용된 규칙을 확인할 때 이용되기 때문에 구문 규칙 컴파일러에서 함께 생성해야 한다.

```
Tree* NP010noun(Tree* noun)
{
  if ( GetIndic(noun, COORDED)) )
    return noun;
  else return NULL;
}
Tree* NP010(Tree* noun)
{
  Tree* x;
  double          pScore;

  x = CopyAvm(noun);
  pScore = noun->score;
  SetScore(x, 1.0);
  SetCat(x, NP);
  SetIndic(x, OBJP, 1);
  SetIndic(x, SUBJP, 1);
  RefIndicToScore(x, noun, POSSESIVE, 1, 0.5);
  CalcRuleScore(x, 1.02);
  CalcScore(x, pScore);
  return x;
}
```

(그림 4) 구문 규칙 컴파일러의 출력 예: 규칙 적용 C 언어 함수

```
enum Rule_ID1 {
    NP010noun=0,  NP010, ... };
```

(그림 5) 구문 규칙 컴파일러의 출력 예: 규칙 이름 선언 파일

```
char* RuleNameTable1[] = {
    "NP010noun",  "NP010", ... };
```

(그림 6) 구문 규칙 컴파일러의 출력 예: 규칙 이름 정의 파일

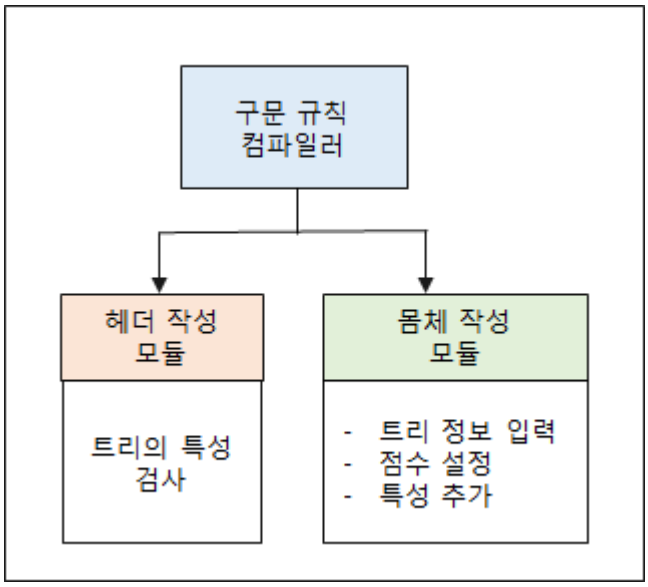
3. 구문 규칙 컴파일러의 구조

구문 규칙 컴파일러는 (그림 3)의 형식으로 작성된 구문 규칙을 이용하여 규칙 적용을 수행하는 C 언어 함수를 생성한다. C 언어 함수는 헤더(header)와 함수 몸체(function body)로 구성되는데, C 언어 함수의 구조를 BNF로 나타내면 (그림 7)과 같다.

```
1. <C_function> ::== <header> <function_body>
2. <header> ::== <return_type> function_name
                 ( <parameter_list> )
3. <parameter_list> ::== <parameter> |
                         <parameter_list> , <parameter>
4. <parameter> ::== <data_type> var_name
5. <function_body> ::== { <var_decl>
                          <statement_list> }
```

(그림 7) C 언어 함수 구조

구문 규칙 컴파일러가 생성하는 C 언어 함수의 경우 <return_type>은 파싱 트리 구조체(그림 4의 Tree)가 된다. 그리고 매개 변수의 데이터 형 또한 파싱 트리 구조체가 되므로 <data_type>은 Tree가 된다[1]. 따라서 구문 규칙 컴파일러는 헤더를 작성하는 모듈과 함수 몸체를 작성하는 모듈로 구성된다. (그림 8)은 구문 규칙 컴파일러의 구조를 보여준다.



(그림 8) 구문 규칙 컴파일러의 구조

헤더를 작성 모듈은 (그림 3)에서 나타난 'RuleName', 'RHS', 'LHS'를 이용하여 (그림 7)의 'function_name'을 작성하고 <parameter_list>를 구성한다. 몸체 작성 모듈은 (그림 3)의 'condition'과 'action'을 이용하여 조건 검사와 생성된 파싱 트리에 정보를 추가하는 명령어를 생성한다. 'condition' 부분은 구문 규칙의 적용 가능성 여부를 확인하는 조건들이 제시되어 있으므로 해당하는 파싱 트리가 제시된 조건을 만족하는지를 확인하는 명령어(그림 4의 GetIndic() 함수)를 생성한다. 'action' 부분은 구문 규칙 적용 결과로 생성된 트리에 대해서 정보를 추가하고 트리의 점수(score)를 설정하고 특성 값을 추가하는 명령어(그림 4의 SetCat(), SetIndic(), CalcRuleScore() 등의 함수)를 생성한다.

4. 결론

본 논문에서는 문맥 자유 문법 형식으로 기술된 영어 구문 규칙을 영한 기계번역 시스템의 구문 분석 모듈인 C 언어 함수로 번역하는 구문 규칙 컴파일러를 개발하였다. 구문 규칙 컴파일러는 C 언어 함수의 헤더를 작성하는 모듈과 함수 몸체를 작성하는 모듈로 구성된다. 헤더 작성 모듈은 규칙 이름과 구문 규칙의 LHS, RHS 기호를 이용하여 함수 헤더를 작성하고, 몸체 작성 모듈은 구문 규칙의 LHS와 RHS를 인지하여 각각에 따른 적절한 명령어를 생성한다. 전문가에 의해 기술된 영어 구문 규칙을 자동적으로 C 언어 함수로 변환함으로써 전문가에 의한 영어 구문 규칙의 기술과 개선이 용이하게 됨으로써 영한 기계번역 시스템의 구문 분석 시스템을 지속적으로 개선할 수 있다.

앞으로 구문 규칙을 관리하는 시스템을 개발하여 규칙의 검색과 수정을 지원하게 함으로써 보다 쉽게 영한 기계번역 시스템의 구문 분석 시스템을 개선할 수 있도록 할 예정이며, 본 논문에서 개발한 구문 규칙 컴파일러는 구문 규칙 관리 시스템의 구성 요소로서 역할을 하여 영한 기계번역 시스템의 지속적 성능개선에 기여할 것으로 기대한다.

Acknowledgement

이 논문은 2010년도 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임(No. 2010-0010815).

참고문헌

[1] Terry Winograd, "Language as a Cognitive process: Vol. 1, Syntax," Addison Wesley, 1983.

1) 실제로는 파싱 트리 구조체의 포인터를 <return_type>와 매개 변수의 <data_type>으로 한다.