두 단계 구문 규칙을 이용한 후-실패 완화 기법

정한민°, 최승권, 김영길, 심철민

전자통신연구원 컴퓨터·소프트웨어 기술연구소
지식처리연구팀

A Post-Fail Softening Mechanism Using Two-level Syntactic Grammar

Hanmin Jung, Sung-Kwon Choi, Young-Kil Kim, Chul-Min Sim

Knowledge Processing Research Team
Electronics and Telecommunication Research Institute
{jhm, skchoi, ygkim, cmsim}@seri.re.kr

요 약

웹 상에서의 자연어 처리 응용에 관한 연구가 활발히 진행되면서, 웹 문서에서 자주 나타나는 특징인 문장 표현에 있어서의 자유로움은 문장 처리의 기술적인 어려움을 가중시킨다 특히, 문장 내의 부분적인 비정형적 표현이 흔히 나타남으로 인해 구문 분석이 자주 실패하게 되는데, 이에 대한 강건성 확보를 위해서 실패에 대한 복구 작업이 필요하다 우리는 두 단계 구문 규칙을 사용하여 1 단계에서 정형적 표현의 문장들을 처리하고, 2 단계에서 구문 규칙의 제약을 완화하여 부분적인 비정형적 표현까지도 포용할 수 있도록 한다 후-실패 완화 과정에서 일관성 있게 구문 규칙을 사용함으로써 효율적인 구조적 파스 트리의 복구도 가능하게 하며, 차트의 재사용을 통해 구조적 애매성과 분석 시간의 단축을 가져온다

1. 서론

최근에 인터넷의 발달과 함께 웹 (WWW) 상에서의 자연어 처리 응용에 대한 연구가 활발히 이루어지고 있다 [Chandrasekar & Srinivas 1997] [Zajac & Casper 1997] 특히, 기계번역 분야에서는 바벨 [Transgate], j-SEOUL, 한글아니 [Idetect] 등과 같은 웹 기반 일한 번역 시스템이나 잉코르, TransNet 등과 같은 영한 번역 시스템이 상용화 단계에까지 이르고 있다 그렇지만, 웹 문서의 특성들 중의 하나인 무제한적인 전문 분야와 문장 표현의 자유로움은 웹 기반 자연어 처리 응용 프로그램들이 고려해야 하는 기술적 문제들을 크게 증가 시키는 결과를 가져오며 이에 대한 해결책으로 전문 분야 인식과 강건한 문장 분석 기법 등이 필요하다 특히 기계번역 시스템의 강건성은 예측하지 못한 문장들에 대한 처리가 필수적인 핸에서 반드시 갖추어야 할 특성이다 일반적으로 번역 과정의 어느 부분에서 문제가 발생할 지 모르므로 각 모듈별로 고유한 강건성을 위한 기법들이 필요하게 된다 본 논문은 특히 강건한 구문 분석을 위해 도입한 분석 실패의 복구 방법에 초점을 맞추어 기술한다 우리는 이를 후-실패 완화라고 명명하는데, 이는 실패가 발생한 후에 이를 복구를 통해 완화하려는 기법이기 때문이다. 우리는 긴 문장에 대한 구문 분석의 실패를 미리 방지하고자 하는 전-실패 완화와 강하게 결합되어 작동한다는 의미에서 전-, 후-실패 완화리는 용어를 사용한다 Menzel 은 자연어 처리 응용 시스템의 신뢰도 향상을 위한 필수적인 요소로 강건성을 언급하였는데 특히 오류 기대 (error anticipation)와 제약 완화 기법 (constraint relaxation techniques)의 일반적인 사용을 설명하였다 [Menzel 1995] 후-실패 완화는 결국 제약 완화 기법의 일종으로 입력 문장 처리를 위한 구문 규칙의 시스템적인 제약 완화이며, 전-실패 완화는 기존의 문법 내에서 처리할 수 있도록 미리 오류 후보들을 적절히 제거하는 것이다 후-실패 완화와 관련된 연구들은 그 중요성에 비해 국내외에서 아직까지 많지 않은 실정이다 [Lee *et al* 1995]는 구문 분석기 자체에 최소 오류 인식을 위한 알고리즘을 도입하여 구문 규칙 자체가 오류를 허용하도록 하였으며, [Rose & Waibel 1994] [Rose & Lavie 1997]은 통계학적 기법과 심볼릭 기법의 결합을 이용하여 사용자와 상호작용을 가지면서 의미 표현 구조의 복구에 관한 연구를 하였다 그렇지만, 전자는 후-실패 완화 라기보다는 오류를 예측하여 미리 구문 규칙에 반영한 형태이며, 후자는 우리와 같은 규칙 기반 기계번역에서가 아니라 음성 번역과 같이 제한되고 예측이 가능한 분야에 적용 가능한 것으로 일반적인 텍스트 기계번역에는 부적당하다

본 논문에서 제시하는 후-실패 완화 엔진은 우리가 자체적으로 개발한 웹 기반 영한 기계번역 시스템인 FromTo/EK [Sim *et al* 1998]에 구문 분석 의 강건성 기법을 도입한 것으로. 예측 과정 없이 이미 분석에 실패한 문장을 대상으로 최대한의 자연스러운 대역어 생성을 위한 구조적인 복구를 목표로 한다 우리는 구문 분석을 위해 차트 파싱을 이용하며, 폭발적인 차트의 증가를 억제하고자 chart pruning 을 파싱 중에 시도한다 2 장은 구문 분석의 실패 원인들과 이에 대한 해결 방안의 하나인 두 단계 구문 규칙에 대해 기술하며, 3 장과 4 장에서는 후-실패 완화 엔진과 알고리즘을 설명한다

2. 구문 분석 실패와 두 단계 구문 규칙

차트 기반의 구문 분석이 실패하는 경우는 크게 다음과 같으며, 문장 표현이 자유롭고 문장의 길이가 길수록 실패하는 비율이 더욱 높아진다

1 구문 규칙의 부재 입력 문장을 처리할 수 있는 구문 규칙의 전부나 일부가 없어서 구문 분석이 실패하는 경우로, 실세계를 빈영하기 위한 구문 규칙의 근본적인 한계에 그 원인이 있다 이에 대한 복구 방안으로는 부분적으로 성공한 피스 트리들을 정해진 원칙 (최장 일치나 가중치 적용 등)에 따라 병합하는 방법이 있다

2 지나친 또는 강도 높은 구문 규칙의 제약 입력 문장을 포함할 수 있는 구문 규칙이 존재하지만 그 규칙이 가지는 제약 조건이 엄격하여 해당 구문들 간의 접속 관계를 밝혀내지 못하는 경우이다 이에 대한 복구 방안으로는 완화된 규칙을 이용한 문법 재적용이나 부분적으로 성공한 파스 트리들의 비구조적 병합이 있다

3 차트의 폭발 차트 파싱의 경우에, active와 inactive edge들과 agenda를 사용하여 파싱을 수행한다 그렇지만, 문장의 길이가 길어짐에 따라 유한한 시스템 자원 내에서 존재하는 이 edge들의 수가 증가하며. 결국 한계를 넘어 차트가 폭발하는 경우도 생긴다 이 경우에 완전한 파스 트리의 생성까지 이르지 못하게 되므로 이미 생성된 피스 트리들을 적절히 병합하는 작업이 필요하다

우리는 위와 같은 요인들에 의해 발생하는 구문 분석 실패를 구조적으로 복구하여 자연스러운 대역어 생성을 돕고자 후-실패 완화 엔진을 도입한다 후-실패 완화는 기존의 제약적인 구문 규칙과 제약이 완화된 구문 규칙을 모두 이용하여 구조적인 파스 트리 복구를 수행

한다 구문 규칙은 두 단계로 구성되어 1 단계에서는 제약 조건을 가지고 정규적 표현의 문장을 처리하고자 시도하며, 만일 실패하는 경우에는 2 단계의 제약이 완화된 구문 규칙을 이용하여 일부 비정규적 표현의 문장들까지도 처리하고자 시도한다 웹 기반 기계번역 시스템의 성능은 얼마나 많은 문장들을 강건하게 번역할 수 있느냐에 있다 그렇기 때문에 웹에서 특히 자주 나타나는 일부의 비정규적 표현을 가지고 있는 문장들까지도 처리 대상으로 할 필요가 있다 이런 경우에 구문 규칙의 제약 완화는 분석할 수 있는 대상을 확장 시켜 주면서도 처음부터 비정규적 처리를 허용하지 않아 애매성을 증가시키지 않는 (구문 규칙의 1 단계와 2 단계로의 구분 이유) 장점을 가진 두 단계 구문 규칙이 효과적이다 또한, 2 단계의 제약 완화 구문 규칙의 이용은 1 단계에서 생산한 부분 파스 트리들을 구조적으로 복구할 수 있도록 한다 물론, 단순한 파스 트리 병합이나 휴리스틱을 이용하여 복구를 수행할 수도 있지만, 이는 결국 일관된 분석/복구 기법이 아닌 임시방편인 것이다 우리는 두 단계 구문 규칙 적용을 위해 1 단계의 470 여 개, 2 단계의 110 여 개의 구문 규칙들을 사용한다

다음은 1 단계 규칙 (level 0)과 2 단계 규칙 (level 1)의 예를 보여준다 우리의 구문 규칙은 ACFG 문법 기술을 따르며, 규칙 선언부, 제약 부분, 문법 실행 부분으로 구성된다 2 단계 구문 규칙은 제약 부분이 아주 이약하거나 없는 것이 특징으로 부분적인 비정규적 표현을 허용할 수 있다.

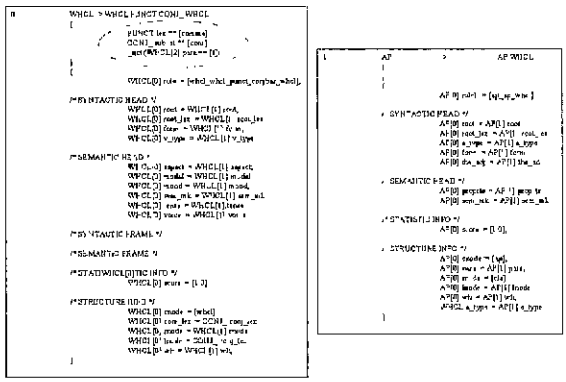

그림 1 구문 분석 실패 복구를 위한 두 단계 구문 규칙
(왼쪽은 정형적 표현 처리를 위해
제약 (동그라미 부분)을 가진 1 단계,
오른쪽은 부분적인 비정형적 표현 처리를 위해
제약을 완화하거나 없앤 2 단계 구문 규칙)

3. 후-실패 완화 엔진

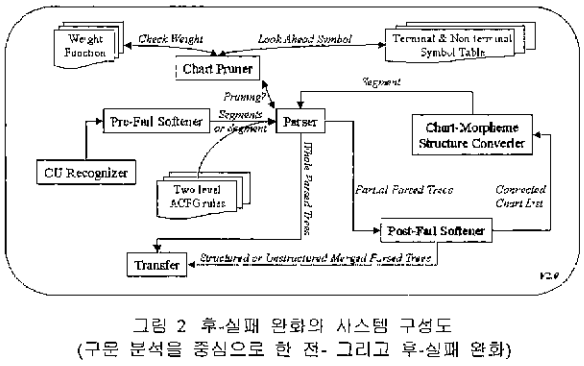

그림 2 후-실패 완화의 시스템 구성도
(구문 분석을 중심으로 한 전- 그리고 후-실패 완화)

후-실패 완화 엔진의 주 작업은 구문 분석에 실패한 파스 트리의 복구이다 구문 분석기를 중심으로 전-실패 완화와 후-실패 완화 엔진이 번역 시스템의 강건성을 위해 위치한다 전-실패 완화에 의해 세그멘트 분리가 되거나 또는 안된 문장이 구문 분석기의 입력이 되며, 분석이 성공한 경우에는 후-실패 완화 과정 없이 전체 파스 트리가 변환기로 넘어간다 그렇지만, 분석에 실패한 경우에는 2 단계 구문 규칙을 이용한 구문 분석이 진행된다 이 단계에서 차트-형태소 구조 변환기가 사용되는데, 이는 차트의 재사용을 통해 (차트의 재사용은 2 단계 구문 분석 시간을 단축할 수 있을 뿐만 아니라, 구조적 애매성을 줄일 수 있다) 파스 트리의 복구를 시도한다 (그림 4) 지나친 또는 강도 높은 구문 규칙의 제약으로 인한 구문 분석 실패는 이 과정을 통해 구문 규칙의 부재나 차트의 폭발로 인한 실패는 문장의 부분들에 해당하는 구문 파스 트리들의 병합으로 복구한다. 차트의 폭발을 방지하기 위해서 도입된 차트 pruner 는 가중치를 이용하여 규칙의 적용 깊이를 조절하고, 구문 태그의 look ahead 방식을 도입하여 적용이 불가능한 차트들을 미리 제거한다

4. 후-실패 완화 알고리즘

우리의 후-실패 완화는 두 단계 구문 규칙을 이용하며, 차트의 재사용을 통해 구문 분석에 실패한 파스 트리를 복구한다. 그림 3 은 후-실패 완화 알고리즘을 보여주는데, 크게 1 단계 (level 0) 구문 규칙을 이용하는 주 분석부, 2 단계 (level 1) 구문 규칙을 이용하는 재 분석부, 그리고 차트의 재사용과 구조적 애매성 감소를 위한 차트-형태소 구조 변환부로 나누어진다 앞에서도 언급한 바와 같이 후-실패 완화 알고리즘은 세 가지의 구문 분석 실패 요인에 따라 다른 처리 과정을 가진다. 주 분석부에서는 구문 규칙이 성공한 경우와 차트의 폭발을 다루며, 재 분석부에서는 지나친 또는 강도 높은 구문 규칙의 제약과 구문 규칙의 부재를 다룬다 차트-형태소 구조 변환부에서는 차트의 재사용을 위해 입력 형태소들이 아닌 최장 일치 원칙에 의해 전체 문장을 포함하는 차트들의 리스트를 만들어 2 단계 구문 규칙의 재 분석부로 넘긴다

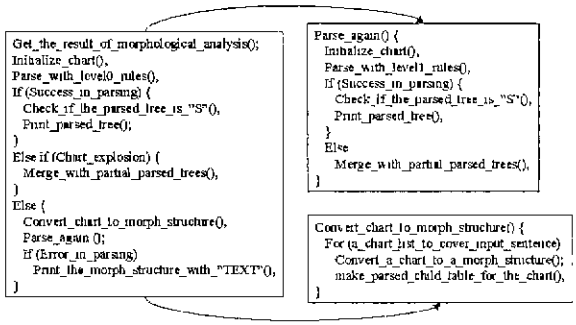

그림 3 후-실패 완화 알고리즘
(1 단계 구문 규칙을 이용한 분석부,
2 단계 구문 규칙을 이용한 분석부, 차트-형태소 구조 변환)

1 단계 구문 분석이 실패하면, 좌우 최장 일치를 이용하여 전체 문장을 포함할 수 있는 차트들의 리스트를 만들어 이를 파스 자식 테이블로 저장한다 (그림 4 의 [1]) 이 이유는 inactive edge 의 인덱스만 가지고 있을 경우, 이 edge 들이 재사용되면 edge 뿐만 아니라 그것이 가지고 있던 자식들에 대한 모든 구조적 정보를 잃게 되기 때문이다 이를 위해 구조적인 자식 정보들을 파스 자식 테이블에 저장함으로써 2 단계 구문 규칙이 적용된 후에 구조적인 파스 트리 복구를 효율적으로 수행할 수 있게 한다 (그림 4 의 [2]) [3]에서는 이 차트들의 리스트를 기존의 구문 규칙에 그대로 적용할 수 있도록 하기 위해 구문 분석기의 원래 입력 형태인 형태소 분석 결과 구조로 변환한다 이것이 차트-형태소 구조 변환으로, 차트 구조와 형태소 구조 간의 매핑을 시도한다 2 단계 구문 규칙을 적용하면 [3]에서 만들어진 차트들의 리스트로부터 새로운 inactive edge 들이 생산된다 (그림 4 의 [4]) 이들은 파스 자식 테이블을 가리키고 있던 [3]의 리스트로부터 위치 정보를 얻어 이 테이블로부터 기존의 자식 정보들을 얻어낸다 (그림 4 의 [5]) 이런 과정을 통해 구조적인 파스 트리 복구에 성공하게 되며, 이 결과는 변환 이후로 넘어기 보다 자연스러운 대역어를 생성할 수 있도록 한다

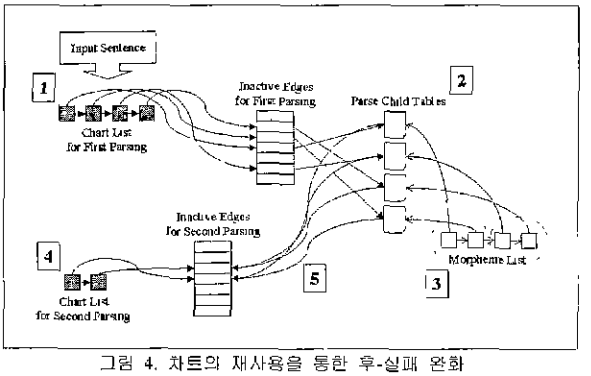

그림 4. 차트의 재사용을 통한 후-실패 완화

다음은 문장 "I love"에 대한 처리 예를 보여준다. 첫번째 파스 트리는 타동사 "love"의 결합 관계 실패로 인하여 세그멘트 형태로 출력된 구문 분석 실패를 보여준다. 두번째 파스 트리는 단순한 비구조적 형태로 세그멘트들을 "TEXT" (웹 상에서 다양한 형태의 문장들을 처리하기 위한 "S"의 상위 태그)로 묶어 변환 과정으로 넘겨 단어 단위나 구 단위 번역을 할 수 있도록 한 예이다. 세번째는 후-실패 완화에 의해 1 단계 구문 분석에서 생성된 "NP"와 "VERB_"를 'SM'으로 묶고, "SM"과 "PUNCT"를 "S"로 묶어 구조적으로 파스 트리를 복구한 예이다. 결국, 번역 결과는 두번째의 경우에는 "나 사랑한" (두 어절 간의 의미적 연결의 부재)이, 세번째의 경우에는 "나는 사랑한다"가 된다.

<1 구문 분석 실패>
(NP ((g_level 0)(RANK_VALUE 1.000000)(DET_LEX NIL)(LNODE NIL)(RNODE NIL)(CNODE PRONBAR)(SCORE 1.0)(CASE SUBJ)(SUBCAT CENT)(ROOT_LEX I)(ROOT I)(RULE NP_PRONBAR))
(PRON ((g_level 0)(RANK_VALUE 1.000000)(RNODE NIL)(LNODE NIL)(SCORE 1.0)(SUBCAT CENT)(PERSON 1)(NUMBER S)(CASE SUBJ)(ROOT_LEX I)(ROOT I)(RULE PRONBAR_PRON))
(PRON ((RANK_VALUE 1.000000)(LEX I)(TOKEN_ID 0)(DICTYPE 1)(NUM 1S 2S 3S 1P 2P 3P)(ROOT I)(CAT PRON)(SUBCAT CENT)(PERSON 1)(NUMBER S)(CASE SUBJ)(V_TYPE)(A_TYPE)(N_TYPE)(S_TYPE)(TRV_K_GCODE1)(ROOT_LEX I))))
(VERB ((g_level 0)(RANK_VALUE 1.000000)(RNODE NIL)(LNODE NIL)(CNODE VERBBAR)(TOKEN_ID 1)(SCORE 1.0)(VOICE ACTIVE)(TENSE PRES)(SEM_MK STATIVE)(V_TYPE T1 T3 T4)(FORM INF PRES)(ROOT_LEX love)(ROOT LOVE)(RULE VERBBAR_VERB))
(VERB ((RANK_VALUE 1.000000)(LEX LOVE)(TOKEN_ID 1)(DICTYPE 1)(FORM INF PRES)(ROOT LOVE)(CAT VERB)(V_TYPE T1 T3 T4)(SEM_MK STATIVE)(A_TYPE)(N_TYPE)(S_TYPE)(TRV_K_GCODE1)(ROOT_LEX love)))))
(PUNCT ((RANK_VALUE 1.000000)(PARSE_NO 1)(LEX PERIOD)(TOKEN_ID 2)(FORM PERIOD)(CAT PUNCT)(V_TYPE)(A_TYPE)(N_TYPE)(S_TYPE)(TRV_K_GCODE1)(ROOT_LEX love))))

<2 단순한 비구조적 파스 트리 복구>
(dummy T
(TEXT ((TENSE PRES)(MOOD DECL))
(NP ((g_level 0)(RANK_VALUE 1.000000)(DET_LEX NIL)(LNODE NIL)(RNODE NIL)(CNODE PRONBAR)(SCORE 1.0)(CASE SUBJ)(SUBCAT CENT)(ROOT_LEX I)(ROOT I)(RULE NP_PRONBAR))
(PRON ((g_level 0)(RANK_VALUE 1.000000)(RNODE NIL)(LNODE NIL)(SCORE 1.0)(SUBCAT CENT)(PERSON 1)(NUMBER S)(CASE SUBJ)(ROOT_LEX I)(ROOT I)(RULE PRONBAR_PRON))
(PRON ((RANK_VALUE 1.000000)(LEX I)(TOKEN_ID 0)(DICTYPE 1)(NUM 1S 2S 3S 1P 2P 3P)(ROOT I)(CAT PRON)(SUBCAT CENT)(PERSON 1)(NUMBER S)(CASE SUBJ)(V_TYPE)(A_TYPE)(N_TYPE)(S_TYPE)(TRV_K_GCODE1)(ROOT_LEX I))))
(VERB ((g_level 0)(RANK_VALUE 1.000000)(RNODE NIL)(LNODE NIL)(CNODE VERBBAR)(TOKEN_ID 1)(SCORE 1.0)(VOICE ACTIVE)(TENSE PRES)(SEM_MK STATIVE)(V_TYPE T1 T3 T4)(FORM INF PRES)(ROOT_LEX love)(ROOT LOVE)(RULE VERBBAR_VERB))
(VERB ((RANK_VALUE 1.000000)(LEX LOVE)(TOKEN_ID 1)(DICTYPE 1)(FORM INF PRES)(ROOT LOVE)(CAT VERB)(V_TYPE T1 T3 T4)(SEM_MK STATIVE)(A_TYPE)(N_TYPE)(S_TYPE)(TRV_K_GCODE1)(ROOT_LEX love)))))
(PUNCT ((RANK_VALUE 1.000000)(PARSE_NO 1)(LEX PERIOD)(TOKEN_ID 2)(FORM PERIOD)(CAT PUNCT)(V_TYPE)(A_TYPE)(N_TYPE)(S_TYPE)(TRV_K_GCODE1)(ROOT_LEX love))))

<3 후-실패 완화에 의한 구조적 파스 트리 복구>
(dummy T
(TEXT ((TENSE PRES)(MOOD DECL))
(S ((g_level 1)(RANK_VALUE 1.000000)(PARSE_NO 1)(RULE S_SM_PUNCT))
(SM ((RANK_VALUE 1.000000)(PARSE_NO 1)(MISSING_DOBJ)(SUBJ T)(LNODE PHR)(RNODE NIL)(CNODE VERBBAR)(SCORE 1.0)(VOICE ACTIVE)(TENSE PRES)(SEM_MK STATIVE)(MOOD DECL)(V_TYPE T1)(FORM INF PRES)(ROOT_LEX love)(ROOT LOVE)(RULE SM_NP_VERBBAR))
(NP ((g_level 0)(RANK_VALUE 1.000000)(DET_LEX NIL)(LNODE NIL)(RNODE NIL)(CNODE PRONBAR)(SCORE 1.0)(CASE SUBJ)(SUBCAT CENT)(ROOT_LEX I)(ROOT I)(RULE NP_PRONBAR))
(PRON ((g_level 0)(RANK_VALUE 1.000000)(RNODE NIL)(LNODE NIL)(SCORE 1.0)(SUBCAT CENT)(PERSON 1)(NUMBER S)(CASE SUBJ)(ROOT_LEX I)(ROOT I)(RULE PRONBAR_PRON))
(PRON ((RANK_VALUE 1.000000)(LEX I)(TOKEN_ID 0)(DICTYPE 1)(NUM 1S 2S 3S 1P 2P 3P)(ROOT I)(CAT PRON)(SUBCAT CENT)(PERSON 1)(NUMBER S)(CASE SUBJ)(V_TYPE)(A_TYPE)(N_TYPE)(S_TYPE)(TRV_K_GCODE1)(ROOT_LEX I))))
(VERB ((g_level 0)(RANK_VALUE 1.000000)(RNODE NIL)(LNODE NIL)(CNODE VERBBAR)(TOKEN_ID 1)(SCORE 1.0)(VOICE ACTIVE)(TENSE PRES)(SEM_MK STATIVE)(V_TYPE T1 T3 T4)(FORM INF PRES)(ROOT_LEX love)(ROOT LOVE)(RULE VERBBAR_VERB))
(VERB ((RANK_VALUE 1.000000)(LEX LOVE)(TOKEN_ID 1)(DICTYPE 1)(FORM INF PRES)(ROOT LOVE)(CAT VERB)(V_TYPE T1 T3 T4)(SEM_MK STATIVE)(A_TYPE)(N_TYPE)(S_TYPE)(TRV_K_GCODE1)(ROOT_LEX love)))))
(PUNCT ((RANK_VALUE 1.000000)(PARSE_NO 1)(LEX PERIOD)(TOKEN_ID 2)(FORM PERIOD)(CAT PUNCT)(V_TYPE)(A_TYPE)(N_TYPE)(S_TYPE)(TRV_K_GCODE1)(ROOT_LEX love))))

5. 결론

우리는 웹 기반 기계번역 시스템이 반드시 갖추어야 하는 특성들 중의 하나인 구문 분석기의 강건성을 위해 파스 트리 복구를 주 작업으로 하는 후-실패 완화 알고리즘을 제시하였다. 규칙 기반 기계번역이 가지는 규칙의 적용 한계성을 극복하며 일부 비정규적 표현을 가진 문장들까지도 포용하기 위해 차트의 재사용을 통한 구조적인 파스 트리 복구 과정을 보여주었다. 우리는 두 단계 구문 규칙을 사용하여 1 단계에서 정형적 표현의 문장들을 처리하고, 2 단계에서 구문 규칙의 제약을 완화 또는 제거하여 애매성의 증가 없이 규칙의 포용 범위를 넓히고자 하였다. 특히 일관성 있는 구문 규칙의 적용은 기존의 구문 분석 모델을 변경하지 않고도 효율적인 구조적 파스 트리의 복구를 가능하게 한다. 이를 위해 차트-형태소 구조 변환을 이용하여 기존의 시스템 자원을 재활용하고, 파스 자식 테이블에 구조적인 구문 분석 결과를 보관하고 이들의 위치 정보만을 이용하는 방식을 사용하였다.

앞으로는 지나친 또는 강도 높은 구문 규칙의 제약으로 인한 구문 분석 실패 뿐만 아니라 구문 규칙의 부재 및 차트 폭발로 인한 실패까지도 구조적으로 복구할 수 있는 연구에 초점을 맞출 예정이다.

참고 문헌

[Chandrasekar & Srinivas 1997] R. Chandrasekar and B. Srinivas, Gleaning Information from the Web: Using Syntax to Filter out Irrelevant Information, *In Proceedings of Natural Language Processing for the World Wide Web, AAAI-97 Spring Symposium*, 1997

[Rose & Waibel 1994] C. Rose and A. Waibel, Recovering From Parser Failures: A Hybrid Statistical / Symbolic Approach, cmp-lg/9407025, 1994

[Rose & Lavie 1997] C. Rose and A. Lavie, An Efficient Distribution of Labor in a Two Stage Robust Interpretation Process, *In Proceedings of EMNLP*, 1997

[Lee *et al* 1995] K. Lee, C. Kweon, J. Seo, and G. Kim, A Robust Parser Based on Syntactic Information, *In Proceedings of EACL*, 1995

[Menzel 1995] W. Menzel, Robust Processing of Natural Language, *In Proceedings of the 19th German Conference on Artificial Intelligence*, 1995

[Sim *et al* 1998] C. Sim, H. Jung, S. Yuh, T. Kim, D. Park, and H. Kwon, An Implementation of English-to-Korean Machine Translation System for HTML Documents, *In Proceedings of the IASTED International Conference on Artificial Intelligence and Soft Computing*, 1998

[Zajac & Casper 1997] R. Zajac and M. Casper, The Temple Web Translator, *In Proceedings of Natural Language Processing for the World Wide Web, AAAI-97 Spring Symposium*, 1997

[Idetect] http://japan.idetect.com/

[Transgate] http://www.bora.net/transgate/