論文94-31B-3-2

온라인 방식의 자연언어 해석기 설계

(Design of On-Line Natural Language Parser)

禹堯涉*, 崔炳旭**

(Yo Seop Woo and Byung Uk Choi)

要約

자연언어 시스템은 일반적으로 속도가 느린 단점이 있다. 대화형 시스템에서 사용자가 기다리는 시간이 과다하다면 실용적이라고 할 수 없다. 본 논문에서는 문장을 터미날에 입력하는 것과 동시에 해석을 병행 수행하는 온라인 자연언어 해석기를 설계한다. 문장이 입력되는 시간동안 대부분의 형태소 해석과 상당부분의 구문 의미 해석이 완료될 수 있으므로, 사용자는 즉각적인 응답을 얻을 수 있다. 또한 한국어 해석기를 다중처리 환경에서 구현하여 오프라인 해석과 비교한다. 온라인 해석기의 경우 실시간에 근접한 수행 속도를 얻을 수 있어 유용하다고 판단한다.

Abstract

A natural language processing system usually has the demerit that its processing time is relatively long. If an interactive system makes its user kept waiting long, it can't be said to be practical. In this paper, the on-line natural language parser in which its processing coincides with the sentence's inputting is designed. Since the greater part of morpholgical and syntatic semantic analysis is already performed during the keyboard input, user can get a prompt response. Moreover, the Korean parser is implemented in multitasking environment, and it is compared with an off-line parser. The on-line parser can be considered to be efficient for its real time processing.

Ⅰ. 서론

자연언어를 해석하는 시스템에 있어서 그 처리 속도가 실시간에 이르지 못하는 것이 현실이다. 기계번역과 같이 오프라인 일괄 처리 (off-line batch processing)에서는 시간적 제약이 비교적 덜한 경우이다. 그러나 질의 응답이나 자연언어 인터페이스 시스템, 전화 통역등에 사용되는 대화형 번역 시스템에서는 실시간 응답이 절실하다고 할 수 있다. 터미날을 통하여 자연언어 시스템을 사용하는 경우를 예로 들면, 사용자는 원하는 결과를 얻기 위해 문장을 입력시키는 시간을 포함 수초 내지는 수십초의 시간을 기다려야 하므로 이러한 자연언어 시스템은 실용적일 수 없는 난점을 가진다. 본 논문에서는 이러한 문제

* 正會員, 仁川大學校 情報通信工學科
(Dept. of Information & Telecommunication, Inchon Univ.)
** 正會員, 漢陽大學校 電子通信工學科
(Dept. of Electronic Communication, Hanyang Univ.)
接受日字 : 1993年 9月 7日

를 해결하기 위하여 온라인 언어 해석기 (on-line parser)의 설계와 구현에 관하여 논하고자 한다.

사용자가 문장을 입력하는 것과 병행하여 문장해석을 진행하는 방법이 제시된다면, 터미날에 문장을 입력하는 통상 수초이상의 시간을 시스템이 언어 해석에 사용할 수 있으므로, 문장 입력 종료와 거의 동시에 언어 해석을 완료할 수 있게 된다. 즉, 오프라인 언어해석에서는 실시간 처리가 곤란하지만, 대화형 시스템 (interactive system)일 경우에는 언어 해석 시간의 상당 부분을 문장 입력 시간에 포함시킬 수 있으므로 빠른 응답을 기대할 수 있다. 자연언어처리 시스템에서 대부분의 시간을 소모하는 부분이 언어 해석부이므로, 이러한 측면에서 보면 입력과 동시에 결과를 출력하는 실시간 시스템의 가능성을 보여준다 하겠다.

따라서 본 논문에서는 문장 전체가 아니라 어절 단위의 입력에 따라 처리를 수행하는 자연언어 해석 방법론을 제시한다. 어절이 입력되면 형태소 해석을 위한 처리기가 동작되고, 동작 중에 다른 어절이 입력되면 또다른 처리기가 병행하여 동작한다. 또 복수개의 형태소 처리기가 동작하는 동안 먼저 종료된 처리기는 구문 해석기를 동작시키고 자신은 또다른 입력 어절을 형태소 해석하기 위해 대기하는 방법의 온라인 언어 해석기이다. 이러한 방법론을 실행하기 위해서는 처리의 병렬 또는 다중화 방식이 유용하다. 따라서 본 논문에서는 자연언어 해석의 병렬성 (paralellism)을 정의하고, 이를 처리하는 알고리즘을 제안한다. 또한 다중처리 환경에서 이를 구현하여 제안된 방법의 유용성을 보인다. 복수개의 처리기를 병행하여 동작시키는 것은 단순히 동일한 프로그램을 여러개 동시에 동작시키는 것과는 달리, 처리기 간의 작업 배분, 데이타 전달, 복수의 처리기 관리등의 방법이 필요하고, 또한 중복 해석 방지등이 보증되는 알고리즘이 제시되어야 한다.

기존의 선진 제국의 연구에서 병렬 처리 방법에 따른 차트나 LR 해석[1,2,3] 등이 제안되어 있다. 병렬 차트 방식은 병렬 시스템 환경에서 데이타의 중복 처리 없이 언어를 해석하는 장점이 있다. 본 논문에서 제안하는 방법도 이러한 점을 고려하였고, 한국어등에 적합하도록 형태소 해석부도 포함하며 구문 해석기와 병행하여 구동할 수 있고, 온라인 및 오프라인 처리가 모두 가능하다. 국내에서도 다중 환경에서의 언어 해석 방법이 제안된 예[4]가 있으나, 언어 해석의 병렬성에 대한 정의가 상이하고 단지 오프라인 처리의 효율 증대를 위한 것이므로 본 논문과는 관점이 다르다고 판단된다.

Ⅱ. 언어해석의 병렬성 고찰

언어해석을 다중 또는 병렬 방식으로 처리하려면 대상으로 하는 해석기가 어떤 부분에서 병렬성을 갖는가를 우선 분석하여야 하며, 기존의 연구 결과들이 제시되어 있다.[5] 본 논문에서 고려하는 해석기는 온라인 또는 오프라인으로 입력된 문장을 먼저 어절단위로 해석한다. 이 과정에서 형태소 해석이 수행되며 이때 각 어절간의 병렬성이 존재한다. 그림 1에서 띄어쓰기로 구분된 각 어절은 별도의 프로세스 P1 ~ P5에 의해 독립적으로 형태소 해석될 수 있으며 동시에 병행 처리가 가능하다. 오프라인 입력이라면 5개의 프로세스가 동시에 형태소 해석을 수행할 수 있고, 온라인의 경우라면 키보드 입력에 의한 지연 시간을 두고 P1부터 P5가 순차 또는 병행하여 해석될 수 있다.

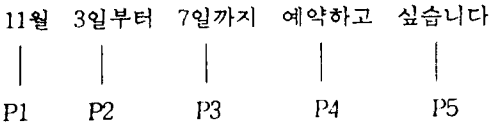

그림 1. 형태소 해석의 병렬성
Fig. 1. Parallism in Morphological Analysis.

또한 문법 규칙의 적용에 있어서도 복수 개의 어절 해석 결과와 복수 개의 규칙이 조합적으로 적용되는 방식이므로 병렬성이 있다. 그림 2는 부분 해석목 (partial parsed tree) 간의 병렬성을 나타낸 것이다. R1, R2에 해당하는 두개의 구구조 규칙은 상호 독립적이므로, 별도의 프로세스에 의해 병행하여 적용될 수 있다. 그림 3은 해석의 애매함에 따른 병렬성을 나타낸 것이다. "11월 3일부터 7일까지"에 대한 두가지 해석이 애매함을 발생시키고, 이때 규칙 적용

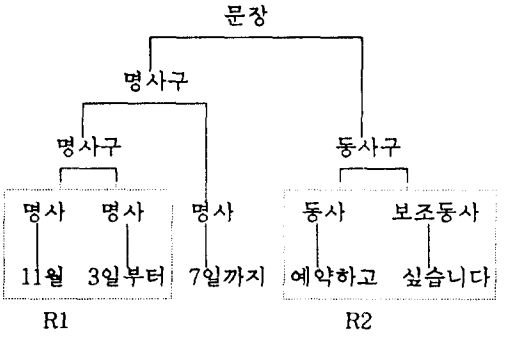

그림 2. 부분 해석목 간의 병렬성
Fig. 2. Parallism among Partial Trees.

에 따라 두 개의 해석목을 구성하는 것은 병행하여 처리될 수 있다. 이때 P1과 P4, P2와 P3 두 프로세스 군 사이에는 병렬성이 있다고 할 수 있다.

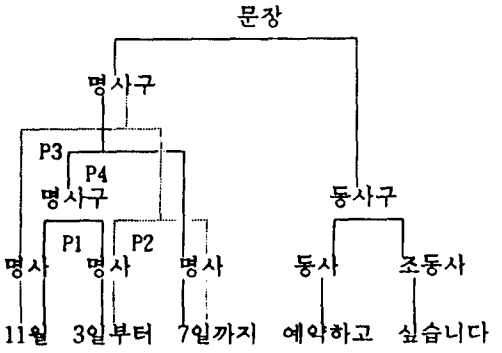

그림 3. 애매성에 의한 병렬성
Fig. 3. Parallism in Ambiguity.

본 논문에서는 이상에서 언급한 3가지 병렬성을 고려한다. 이것은 자연언어 해석 과정중 형태소 해석과 구문/의미 해석 과정을 포함한다. 물론 이외에도 다른 측면의 병렬성을 고려할 수 있다. 예를 들면, PATR-II [6] 와 같이 단일화 문법을 주석 (annotation) 부가 방식에 의해 구현하면 단일화되는 속성들이 병행 처리될 수 있는 가능성이 있다. 그림 4는 PATR-II 규칙의 일례이다. 그림에서 구구조 규칙이 적용될 때 수반되는 a) ~ d)의 주석들은 병행하여 수행될 가능성이 있다. 본 논문에서 정의한 문법 규칙들도 이러한 주석 부가 방식에 기반을 두고 있으므로 연구가 진행 중이지만, 여기서는 포함시키지 않는다. 이것은 각 주석들 간에 선후 관계가 존재하지 않는다는 보장을 할 수 없기 때문이다. 주석이 순수한 단일화 연산만으로 구성된다면 비교적 용이하겠으나, 최근에 문법 구성의 목적에 따라 단일화 연산의 범위를 벗어나는 주석들이 사용되는 경우도 있기 때문이다. [7,8] 또한 단일화와 같은 연산 자체에도 원칙 (principles) 등에 따라 병렬성이 존재한다고 할 수 있으나 [5], 본 논문에서는 단일화 문법을 대상으로 한 것이 아니므로 제외한다.

```
S -> NP VP:                          -> CFG
   <S head> = <VP head>              -> annotation a)
   <S head form> = finite                          b)
   <VP subcat first> = <NP>                        c)
   <VP subcat rest> = end.                         d)
```

그림 4. 주석 부가에 의한 규칙 일례
Fig. 4. An Example of Rules with Annotations.

Ⅲ. 온라인 해석기의 설계

해석기는 시스템 전반을 제어하는 주프로세스 (main process)와 형태소 해석을 담당하는 부프로세스 (child process), 구문 및 의미 해석을 담당하는 부프로세스로 구성한다. 다중 처리 방식의 온라인 해석기를 설계하기 위해서는 몇가지 고려해야 할 사항이 있다. 기본적으로 프로세스간의 작업이 중복되지 않는다는 보장이 필요하다. 논문에서는 프로세스들이 작업해야 할 내용들을 공유 메모리 (shared memory) 형태의 큐 (queue)에 저장하고, 형태소나 구문 해석 프로세스들이 필요에 따라 인출하여 작업하는 방식으로 시스템을 설계하였다. 자신이 인출한 작업이 끝나면 그 결과를 주 프로세스에 전달하고, 주 프로세스는 이 결과를 분석하여 새로 진행되어야 할 작업들을 다시 큐에 추가한다. 해석에 사용되는 데이타뿐만 아니라 작업할 내용들이 저장되고 순차적으로 인출되므로 중복 처리의 부담이 없다. 또한 부프로세스간의 간섭을 최소화하기 위해서는 어절 단위의 독립성이 가정되어야 한다. 따라서 어절의 형태소 해석은 어절내에 한정된 정보를 사용하며, 어절 단위의 해석 결과가 구문 및 의미해석 과정에 전달되는 방식을 사용한다. [9] 복합문 어미의 제약 [10] 등에 필요한 어절 간의 정보 전달은 단일화 문법과 같은 속성 천이 방법 [11,13] 을 사용하여 구문 해석 과정에서 처리하도록 하였다.

다중 환경은 병렬 처리 시스템과 달리 H/W S/W 자원이 한정되어 있다. 부프로세스를 필요로 할 때마다 실행 (exec)하려면 시간 소요가 과다하므로, 형태소와 구문 해석 전담 프로세스를 구분하여 시스템 초기화 때 기억 장치에 적재하는 방식으로 설계한다. 이때 각 프로세스의 수는 외부에서 제어할 수 있도록 한다. 따라서 시스템 환경에 따라 최적의 경우를 결정해야 보다 효율적이며, 정해진 텍스트의 분석 결과를 이용하여 제한적으로 자동화 하는 방법도 고려될 수 있다. 그러나 본 논문은 병렬처리가 아니라 다중화 방식이어서 자원 (system resources) 이 한정되므로, 다중 프로세스에 의한 효율 증대보다는 온라인에 의한 처리 속도 향상을 주 목표로 하고 시스템을 설계하였다.

주프로세스와 부프로세스 사이의 정보 전달은 앞서 설명한 공유메모리와 신호 (signal)을 통해 수행된다. 비활성 상태 (sleep)의 부프로세스를 주프로세스가 신호로 활성화 (wake)하면, 부 프로세스는 공유메모리를 검색하고 작업을 할당받아 해석을 진행하고, 할당된 작업이 종료하면 결과를 공유 메모리에

(1) 변수 (Global variable)

```
bool   on_line                ; 온라인/오프라인 선택
int    usable_morph_process   ; 사용가능한 형태소 해석 부프로세스의 갯수
int    usable_rule_process    ; 사용가능한 규칙 적용 부프로세스의 갯수
int    input_end              ; 문장입력 종료여부 (Button)
         = NULL               ;       -> 입력중
         = TRUE               ;       -> 종료
int    no_pair_queue          ; pair_queue에 있는 elements 갯수
int    no_edge_queue          ; edge_queue에 있는 elements 갯수
Queue  sentence               ; 문장저장
int    read_sentence_ptr      ; 읽어서 형태소 해석할 (현재) 위치
int    add_sentence_ptr       ; 새 어절 입력시 삽입될 위치

Queue  edge_queue             ; 형태소 해석된 결과를 순차저장
int    start[x,y]             ; 입력 어절의 시작 위치 x를 갖는 (sentence 상의)
                              ; edge_queue의 요소 번호 y를 쌍으로 하는 배열
int    end[x,y]               ; 입력 어절의 끝 위치 x를 갖는 (sentence상의)
                              ; edge_queue의 요소 번호 y를 쌍으로 하는 배열

Queue  pair_queue             ; 규칙 적용을 시도할 pair지정 (인접성 우선)
int    current_pair_queue_ptr ; 현재 시도할 pair 위치
int    last_pair_queue_ptr    ; pair_queue의 끝 위치
```

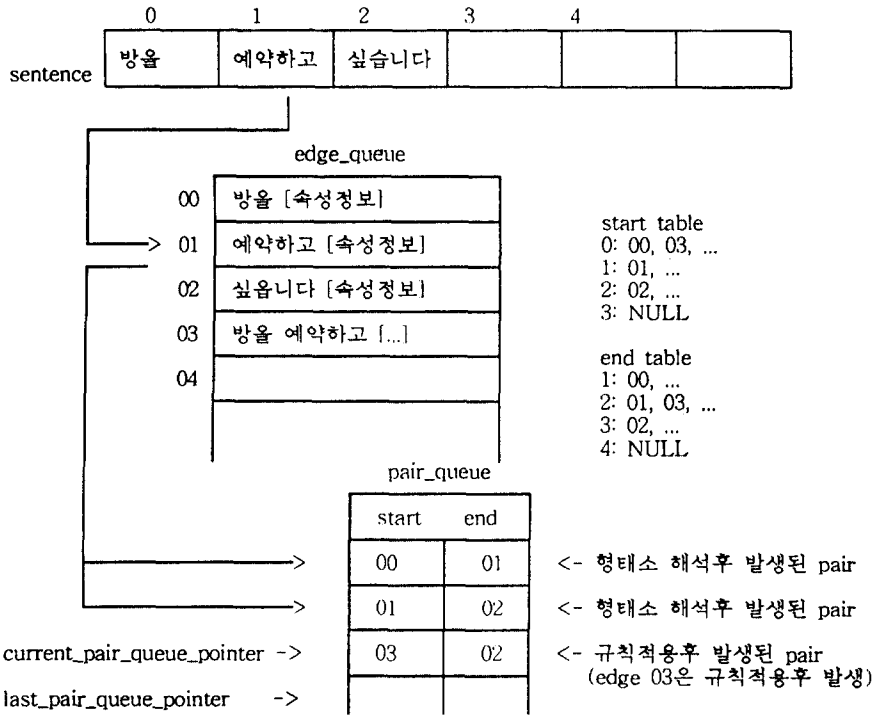

그림 5. 해석 알고리즘 (I)
Fig. 5. Parsing Algorithm (I).

기록한 후 주 프로세스에 신호를 보내고 비활성(sleep) 상태가 된다. 언어해석을 위한 알고리즘을 그림 5에 개략적으로 보인다.

그림에서 (1)은 시스템 전반에 사용되는 전역 변수 또는 공유 메모리의 자료구조 일례이다. usable_morph_process, usable_rule_process는 다중 프로세스로 사용되는 형태소 해석과 규칙적용 프로세스의 갯수이다. 따라서 이를 1로 한정하고 변수 on_line을 FALSE로 하면 오프라인의 단일 프로세스 시스템이 가능하다. sentence는 입력된 문장을 저장하는 배열이고, 새로운 어절이 입력되면 형태소 해석이 수행되어 그 결과가 edge_queue에 저장된다. edge_queue

(2) 주프로세스 (Main Process)

```
TAG1:   if  (usable_morph_process == MAX_NO_OF_MORPH)
            and (usable_rule_process == MAX_NO_OF_RULE)
                                    /* 부프로세스가 모두 종료됨      */
            and (sentence != NULL)  /* 문장입력이 시작되었음        */
            and (input_end == TRUE) /* 문장입력이 종료되었음        */
            and (no_edge_queue == 0) /* edge_queue 처리가 종료됨    */
            and (no_pair_stack == 0)  /* pair_stack 처리가 종료됨   */
        then
            call generator or application:    /* 생성 또는 기타 응용 과정으로 */

TAG2:   if ((usable_morph_process > 0) or (usable_rule_process))
        then      /* 사용가능한 프로세스가 있으면 */
          if read_sentence_ptr < add_sentence_ptr
          then  /* 이미 입력된 어절중 형태소 해석 안한 어절이 있으면 */
                Read Next Phrase            /* 다음 어절 입력받음    */
                read_sentence_ptr++

                usable_morph_process--
                Send Signal To Wakeup A Morphological Child Process

          if (pair_queue != 0)
          then
                /* pair_queue에 요소가 있으면   */
                usable_rule_process--
                Send Signal To Wakeup A Rule_Apply Child Process
          else
                /* 사용가능한 프로세스가 없으면 */
                wait()  /* 부프로세스 중 하나가 종료할 때까지 대기 */
                goto TAG2
          goto TAG1
```

(3) 형태소 해석 부프로세스

```
STEP 1 : Performing Morphological Analysis (with Demons)
STEP 2 : Write Into edge_queue
STEP 3 : Write Into start, end
STEP 4 : Write Into pair_queue
STEP 5 : usable_morph_process++
STEP 6 : return
```

(4) 규칙적용 부프로세스

```
TAG3:   if (current_pair_queue_ptr < last_pair_queue_ptr)

                Read From current_pair_queue_ptr
                current_pair_queue_ptr++
                Apply A Rule

                if (규칙적용 success)
                then Write Into edge_queue
                if (input_end == NULL) and (usable_process != 0)
                then goto TAG3

        usable_rule_process++
        return
```

그림 5. 해석 알고리즘 (Ⅱ)

Fig. 5. Parsing Algorithm (Ⅱ).

는 차트 해석[1,2,11]에서의 edge_pool 또는 agenda (chart)에 해당하며, 공유 메모리 상에 존재한다. 구문/의미 해석이 진행되어 새로운 부분 해석 결과가 생성되면 이 또한 edge_queue에 저장된다. pair_queue도 공유 메모리 상에 존재하는 데이터 구조이다. 새로운 edge가 edge_queue에 추가될 때마

다 main proccess는 이 edge와 결합할 가능성이 있는 또다른 edge를 찾아 pair_queue에 저장한다. 규칙 적용 프로세스는 이 pair_queue에서 job을 할당받으므로 프로세스간의 중복 처리 가능성은 없다. 또

```
NP -> VP[+관형형, -content]  NP
   (fequal       '($dau1 head syn change)        'detm)
   (fnotequal    '($dau1 head syn content)       '+)
   (fconcatpos)
   (fset         '($moth cat)                    'np)
   (fset         '($moth head syn)               '($dau2 head syn))
   (fset         '($moth head sem)               '($dau2 head sem))
   (fset         '($moth head sem (fnext '($dau2 head sem)) mod)
                 '($dau1 head sem))
   (fvarcn       '($moth head sem (fnext '($dau2 head sem)) ref)
                 '($moth head sem (fnext '($dau2 head sem)) mod (fselectcase)))
   (fvarcn       '($moth subcat dau1)            '($dau1))
   (fvarcn       '($moth subcat dau2)            '($dau2))
```

그림 6. 문법규칙의 일례

Fig. 6. An Example of Grammatical Rules.

```
bool VP_NP1_rule(NODEP mother, NODEP daughter1, NODEP daughter2)
                                                  /* NP -> VP NP */
{
  int tmp_feature, list1[MAXDEPTH], list2[MAXDEPTH];

  list1[0] = HEAD, list1[1] = SYN, list1[2] = CHANGE, list1[3] = 0;
  if(fequalvalue(daughter1, list1, "DETM") == FALSE)
    return FALSE;
  list1[0] = HEAD, list1[1] = SYN, list1[2] = CONTENT, list1[3] = 0;
  if(!fequalvalue(daughter1, list1, "+") == FALSE)
    return FALSE;
  clear_node(mother);
  fconcatpos(mother, daughter1, daughter2, TRUE);
  list1[0] = CAT, list1[1] = 0;
  if(fsetvalue(mother, list1, "NP") == FALSE)
    return FALSE;

  list1[0] = HEAD, list1[1] = SYN, list1[2] = 0;
  fsetlist(mother, list1, daughter2, list1);
  list1[0] = HEAD, list1[1] = SEM, list1[2] = 0;
  fsetlist(mother, list1, daughter2, list1);
  list2[0] = HEAD, list2[1] = SEM, list2[2] = 0;
  tmp_feature = fnext(daughter2, list2);
  list1[0] = HEAD, list1[1] = SEM, list1[2] = tmp_feature;
  list1[3] = VMOD, list1[4] = 0;
  fsetlist(mother, list1, daughter1, list2);
  list1[0] = HEAD, list1[1] = SEM, list1[2] = 0;
  tmp_feature = fnext(daughter2, list1);
  list1[2] = tmp_feature, list1[3] = REF;
  list2[0] = HEAD, list2[1] = SEM, list2[2] = tmp_feature, list2[3] = VMOD;
  list2[4] = fselectcase(daughter1, daughter2);
  list2[5] = 0;
  fvalcn(mother, list1, mother, list2);
  list1[0] = VAL, list1[1] = 0;
  fsetlist(mother, list1, daughter2, list1);
  list1[0] = SUBCAT, list1[1] = HEAD, list1[2] = 0;
  fvarcn(mother, list1, daughter2);
  list1[1] = REST;
  fvarcn(mother, list1, daughter1);
  return TRUE;
}
```

그림 7. precompile된 규칙의 일례

Fig. 7. An Example of Precompiles Rules.

한 main_process가 규칙을 적용할 edge 쌍을 결정할 때, 일반적인 차트 해석에서는 edge_queue를 전부 검색해야 하지만 본 시스템에서는 각 edge와 배열 sentence의 위치를 대응시키는 start, end 테이블을 그림과 같이 설정함으로써, 단지 이 테이블만을 검색하여 결정할 수 있다. (2)는 주프로세스의 처리 알고리즘이고, (3)은 형태소 해석 부프로세스 (4)는 규칙 적용 구문/의미 해석 부프로세스의 처리 과정이다.

앞에서 부프로세스간의 간섭을 최소화하기 위해 어절 단위의 독립성을 가정하였다. 그러나 실제 관용표현이나 영역제한에 따른 어휘 정의등에 의해 다어절 어휘가 다수 존재하므로, 이의 처리가 다중 온라인 방식의 난점이라고 할 수 있다. '신라 호텔 예약부'와 같은 일반어나 '-기 때문에', '-ㄹ 수 있'등과 같은 기능어들이 이에 해당한다. 띄어 쓰기로 구분된 각 어절이 각기 다른 프로세스에 의해 해석된다면 이를 하나의 어휘로 취급하는 것이 불가능하다. 본 논문에서는 이를 해결하기 위하여 하나의 프로세스가 복수개 어휘를 인출하게 되면 다음 어절을 참조하기 위해 계속 대기하는 방법을 정의하였다. 예를 들면, '신라', '신라 호텔', '신라 호텔 예약부', '신라 시대'등의 어휘는 모두 '신라'라는 키 (key) 값으로 사전에 등록한다. '신라 '가 입력되면 형태소 해석 프로세스 #1이 기동되어 네 개의 어휘를 사전에서 인출한다. 이 상태에서 처리가 가능한 사전 항목 '신라'에 대해 형태소 해석을 수행하고 그 결과를 공유 메모리에 저장한다. 나머지 세개의 사전 항목을 부프로세스 내부 스택에 저장하고 다음 어절 입력까지 대기 상태가 된다. 다음 어절 '호텔을'이 입력된다면, 주프로세스는 대기 상태의 #1 프로세스에 신호를 보내 계속 처리를 지시하고, 별도의 형태소 해석 프로세스 #2에 '호텔을'을 처리하도록 한다. 프로세스 #1은 내부 스택과 비교하여 사전 항목 '신라 호텔'에 대해 형태소 해석 처리를 하여 그 결과를 다시 공유 메모리에 저장한다. 이때 내부 스택에 두 사전 항목이 남아 있으므로 계속 다어절 처리를 위해 대기할 지 판단하게 된다. '신라 시대'는 '호텔을'이라는 입력과 상충되므로 제거되고, '신라 호텔 예약부'는 입력의 기능어 '-을'을 수용하지 못하므로 제거된다. 내부 스택이 비게 되면 다어절 처리가 종료되어 형태소 해석기는 sleep한다. 'single room', ' John F Kennedy'와 같은 한국어 이외의 자종이 입력될 경우에도 다어절 어휘와 동일한 방식으로 처리한다.

반대로 한 어절을 두 어절 이상으로 분리하여 처리할 필요가 있는 경우도 있다. 예를 들면 '오고가는'과 같이 주로 띄어 쓰기 오류등에 의한 경우가 많다. 이 어절을 담당한 형태소 프로세스는 두 어절로 분리하여 해석하며, 그 두 개의 결과를 edge_queue에 각각 저장한다. 설계한 시스템에서는 '예약하기를'과 같은 명사화 어미 부분도 동사구 '예약하'와 명사구 '기를'로 나누어 구문 해석을 진행하는데, 이때에도 어절 분리 과정이 사용된다.

본 연구에서의 문법 규칙은 기존의 PATR-II 등과 같은 주석 (annotations) 부가 방식[6,11]을 사용한다. 그러나 선언적 주석 (declarative annotations) 대신에, 주석에서 함수를 호출하는 방식의 절차적 주석 (procedural annotations)을 부가하는 규칙 기술 방법을 정의하였다.[9] 단일화 문법의 예를 든다면, 기존의 선언적 주석 부가에서 단일화 이외의 함수가 필요한 경우가 규칙의 대상 영역에 따라 제기되었다.[7,8] 비단일화 함수를 처리하기 위해서는 선언적인 주석 제어 방식만으로 불가능하므로, 본 논문에서와 같은 규칙 기술 방식이 적합하다고 판단된다. 또한 주석에 사용되는 함수에 따라 단일화 문법을 포함한 다양한 문법을 수용할 수 있다. 그림 6은 본 논문의 규칙 일례로 구구조 규칙에 복수개의 선언적 주석이 지정되어 있다. f-로 시작하는 요소는 함수이고 $moth $dau1 $dau2는 각기 구구조 규칙에 대한 상위노드 (mother node), 하위노드 (daughter node)에 포함된 속성 구조를 나타낸다. 이러한 주석 형태의 규칙은 작성자에게는 용이한 형태지만, 시스템에서 실행시 해석되어야 하므로 다소 시간이 소요된다. 효율을 제고하기 위해 본 논문에서는 규칙을 precompiler에 의해 C 언어로 바꾸어 사용한다. 그림 7은 precompile된 규칙의 일례이다.

Ⅳ. 해석기 구현 및 고찰

본 논문에서 설계한 시스템을 SUN-SPARCstation 상에서 C 언어로 구현하였고, 이를 한일 기계번역 시스템과 질의 응답 시스템에 접속하는 연구가 진행중이다. 소스 코드 규모는 사전을 제외하고 5000라인 정도이고, 실험에는 1000여개의 어휘 항목과 60여개의 문법 규칙을 정의하여 사용하였다. 실험 대상 문장으로 호텔 예약 영역에서 발생하는 100여개 패턴의 문장을 선정하였다.

그림 8은 해석 결과의 일례이다. 문장은 명사들이 격조사 없이 반복되는 경우이다. 회화문에서 흔히 문장을 간단하게 하기 위해 사용되는 표현으로, 명사간의 관계는 구문적으로나 의미적으로나 애매성을 많이 내포하게 된다. 그림에 보인 것은 여러 애매한 해석 결과중 경험적인 선정 과정을 거쳐 가장 가능성이 높

$ 문장을 입력하세요 : 손님 성씨의 영어 스펠을 하나씩 불러 주십시요.

```
********** 입력 문자열 정보 **************
str[0] : 손님                str[1] : 성씨의
str[2] : 영어                str[3] : 스펠을
str[4] : 하나씩              str[5] : 불러
str[6] : 주십시요.

* 시작 어절 번호 *          * 끝 어절 번호 *
      0 : 00                     3 : 00
      3 : 01                     6 : 01
      6 : 02                     9 : 02
      9 : 03                    12 : 03
     12 : 04                    15 : 04
     15 : 05                    18 : 05
     18 : 06 07                 21 : 06 07

********** 규칙적용할 짝 **************
daughter1 : 00. daughter2 : 01
daughter1 : 01. daughter2 : 02
daughter1 : 02. daughter2 : 03
daughter1 : 03. daughter2 : 04
daughter1 : 04. daughter2 : 05
start edge : 0,  end egde : 21

POS    손님 성씨의 영어 스펠을 하나씩 불러 주십시요
CAT    VP
HEAD   SYN    VFORM      ADVB
              TENSE      PRES
              CHANGE     TERM
              HUMBLE     +
              NEED       +
       SEM    PRED       부르
              ADVMOD     하나씩
              ACC        REF   스펠
                         NMOD  REF   영어
                               NMOD  REF   성씨
                                     NMOD  REF   손님
VAL    SELF   OR-R  Z
       SUBJ   OR-R  AA B
SUBCAT        HEAD  POS   하나씩 불러 주십시요
              CAT   VP
              HEAD  SYN   VFORM      ADVB
                          TENSE      PRES
                    SEM   PRED       불러_주십시요
                          ADVMOD     하나씩
              VAL   SELF  OR-R       Z
                    SUBJ  OR-R       AA B
                    ACC   OR-R       AA
              SUBCAT      HEAD       PTR->불러 주십시요
                          REST       PTR->하나씩
              REST  CAT   POS   손님 성씨의 영어 스펠을
                    CAT   NP
                    HEAD  SYN   NFORM      NORMAL
                                NUMBER     SINGULAR
                          SEM   ACC   REF  스펠
                                      NMOD REF   영어
                                           NMOD REF   성씨
                                                NMOD REF   손님
                    VAL         SELF  OR-R J
                    SUBCAT      HEAD  PTR->스펠을
                                REST  PTR->손님 성씨의 영어
```

그림 8. 해석 결과의 일례

Fig. 8. An Example of Parsing.

은 결과를 선정한 것이다. 해석된 하위범주는 SUBCAT |HEAD|PTR-〉AA '등과 같이 AA를 해석한 속성 구조를 지시하는 포인터로 반복적으로 연결하고 있다.

오프라인 해석에 대해 온라인 방식의 유용성을 검토하기 위해 몇가지 문장의 해석 결과를 보인다. 그림 9는 테스트한 문장의 일레이고, 그림 10은 해석 수행 시간을 측정한 것이다. 그림 9의 각 문장에 대해 사용자가 문장을 입력하는 소요시간, 오프라인 해석시 입력 종료부터 결과 출력까지의 소요시간을 표시한 것이다. 또한 온라인 해석시 프로세스 갯수에 따른 문장 종료후 결과 출력 시간도 기술되어 있다. 이때 그림의 해석 시간은 각 문장을 10회씩 해석한 평균값이다. 그림에서 보는 바와 같이 오프라인 해석에서 사용자가 대기하는 시간에 비해 온라인 해석이 월등하게 빠른 출력을 내고 있다. 오프라인일 때 문장 입력후 수초간 사용자가 대기하여야 하지만, 온라인이라면 1초 미만의 실시간 응답을 얻을 수 있다. 이것은 문장 입력 동안에 해석이 거의 종료되기 때문이다. 따라서 대화형 시스템일 경우 온라인 해석 방법이 유용하다고 판단할 수 있다. 그림 9의 문장이외에도 300여 문장을 실험에 사용하였고, 대체로 그림 10과 같은 결과를 얻을 수 있었다

온라인 해석에서 프로세스의 갯수에 따라 다소 소요 시간이 감소하지만 기대보다는 다소 미진하다. 이것은 논문에서는 검증을 위해 사용한 사전 항이나 규칙의 갯수가 적기 떠문이라 생각되며, 실용적인 시스템이라면 더 많은 규칙과 사전 항목이 필요하고 이에 따라 부분 해석목의 갯수와 애매성이 급증하여 소요시간이 늘어나는 것이 일반적이므로 이때는 프로세스의 갯수가 시스템의 성능에 보다 큰 영향을 미칠 것으로 판단된다. 이것은 현재 문장을 해석하는데 지정된 프로세스가 최대로 사용되지 않기 때문에, 즉 사용자의 입력 시간보다 해석의 진행 속도가 빠르므로 부프로세스들이 해석을 종료하고 다음 어절을 위해 대기하는 시간이 과다하기 때문이다. 문장 해석이 복잡해져서 사용자의 입력 시간과 비슷한 정도가 된다면, 자원의 한계까지는 프로세스의 갯수가 많은 경우가 보다 빠른 응답을 낼 수 있을 것으로 판단된다.

논문에서 구현한 온라인 해석기를 한일 번역 시스템에 포함시키는 연구가 현재 수행되고 있다. 터미날간의 한일/일한 번역 시스템을 대상으로 수행중인 연구의 일부로서 한국어 해석기 부분에 해당한다. 또한 호텔 예약 담당하는 질의응답 시스템과 도서 관리에 관한 데이타베이스 인터페이스 시스템[9]에 적용하는 연구가 진행중이다.

1. 안녕하십니까, 저 9월 20일부터 10월 5일까지 보름동안 보통실을 좀 예약하고 싶은데요.
2. 예, 손님 성씨의 영어 스펠을 하나씩 불러 주십시오.
3. 어, 그럼 single, double, twin 중에서 원하시는 객실의 종류를 좀 말씀해 주십시오.
4. 안녕하십니까, 여기는 한양 호텔 예약부입니다.
5. 9월 30일 부터 10월 2일 까지 2박 3일 간 인데요.

그림 9. 테스트 문장의 일례
Fig. 9. Examples of Tested Sentences.

문 장	문장입력시간 (평균 초)	오프라인해석시간 (단일프로세스)	프로세스수에 따른 평균 해석시간		
			5	10	20
1	23.20	5.02	1.21	0.71	0.52
2	11.05	2.17	0.24	0.19	0.14
3	22.78	2.27	0.27	0.17	0.15
4	11.03	3.32	0.14	0.08	0.07
5	9.57	0.94	0.06	0.04	0.03

그림 10. 해석 시간의 비교
Fig. 10. Comparison by Parsing Time.

Ⅴ. 결론

본 논문에서는 대화형 자연언어 시스템이 보다 실시간에 가까운 응답을 제공하도록 온라인 방식의 자연 언어 해석기를 설계하였다. 이때 어절 단위의 입력과 동시에 적절한 프로세스가 할당되어 해석이 진행되도록 하기 위하여, 언어 해석기의 병렬성에 관한 고찰을 통해 다중 처리 방식의 시스템을 구성하였다. 제안된 방식의 유용성을 보이기 위해 한국어 해석 시스템을 구현하였고, 사용자가 터미날에 문장을 입력하는 동안 병행하여 언어 해석을 진행함으로써 문장 입력 종료와 거의 동시에 해석을 종료하는 결과를 얻을 수 있었다.

논문은 온라인 방식에 중점을 두고 있지만, 시스템 자원의 제약을 벗어나서 다중 프로세스의 효율을 제고하려면 주석및 단일화등 세부 연산, 기타 생성기등의 병렬성에 관한 연구를 통하여 병렬처리 시스템을 구축하는 것이 바람직하다고 판단된다. 또한 문법에 관하여도 이론적 측면에서 문법의 온라인 해석 가능성 (on-line parsability)에 관한 연구가 필요하다고 하겠다. 본 연구에서는 이미 입력된 문장의 수정을 고려하고 있지 않지만 실용적인 측면에서 어절의 수정이나 삽입 삭제에 관한 적절한 방법이 제시되어야 하며, 따라서 현재 이러한 연구가 진행중이다.

參 考 文 獻

[1] R.Trehan, P.F.Wilk, "A Parallel Chart Parser for the Commited Choice Non - Deterministic *Logic Languages*", *Logic Programming, Proc. of Fifth International Conference and Symposium*, vol. 1, pp.212-232, MIT Press, 1988

[2] R.Grishman and M.Chitrao, "Evaluation of Parallel Chart Parser", *Second Conference on Applied Natural Language Processing*, pp.71-79, 1988

[3] T.Mine, R.Taniguchi, M.Amamiya, "An Efficient Parallel Parsing Algorithm for Context -Free Grammar", *PRICAI 90*, pp.239-244, 1990

[4] 유병기, 임인칠, "한일 기계번역 시스템에서 형태소 분석부의 다중처리 기법", 한국정보과학회 학술발표대회 논문집 vol.19, no.1, 1992

[5] 加藤進, "素性構造の單一化に基23づくパーサの並列化手法の效率", 日本 情報處理學會 自然言語處理研究會 研究報告 77-2, vol.90, no. 40, pp.1-8, 1990

[6] S.M.Shieber, *An Introduction to Unification Based Approaches to Grammar*, CSLI, 1986

[7] S.M.Shieber, F.C.N.Pereira, Gertjan van Noord, R.C. Moore, "Semantic Head-Driven Generation", *Computational Linguistics*, vol.16, no.1, pp30-42, Mar. 1990

[8] 上田良寬, 小暮潔, "タイプ付き素性構造を用いた生成過程の宣言的制御", 日本 情報處理學會 自然言語處理研究會 研究報告 76-5, vol.90, no.17, pp.1-8, 1990

[9] 우요섭, 최병욱, "데이터베이스를 위한 자연언어 인터페이스 NAULI의 설계 및 구현 (I) - 언어 해석 과정을 중심으로 -", 대한전자공학회 논문지, vol.28-B, no.4, pp1-12, Apr. 1991.

[10] 권재일, 국어의 복합문 연구, 집문당, 1990.

[11] G.Gazdar, C.Mellish, *Natural Language Processing in LISP*, Addison-Wesley, 1989

[12] R.E.Frederking, *Integrated Natural Language Dislogue - A Computational Model*, Kluwer Academix Publishers, 1988

[13] M.Johnson, *Attribute-Value Logic and the Theory of Grammar*, CSLI, 1988

[14] M.Tomita, *Efficient Parsing for Natural Language*, Kluwer Academic Publishers, 1986.

著 者 紹 介

禹 堯 涉(正會員)

1963年 11月 16日生. 1986年 2月 한양대학교 전자통신공학과(학사). 1988年 2月 한양대학교 대학원 전자통신공학과(석사). 1992年 2月 한양대학교 대학원 전자통신공학과(박사). 1992年 3月 ~ 현재 시립인천대학교 정보통신공학과(조교수). 주관심 분야는 인공지능, 자연언어이해, 휴먼 인터페이스 등임.

崔 炳 旭(正會員) 第 28卷 B編 第 4號 參照

현재 한양대학교 전자통신공학과 교수