# XML 문서 편집을 위한 추상문법
## (An Abstract Grammar for XML Document Editing)

신 경 희 † 최 종 명 † 유 재 우 ††
(Kyoung-Hee Shin) (Jong-Myung Choi) (Chae-Woo Yoo)

**요 약** 문서내의 태그를 정의하는 문서타입정의(DTD)는 구문구조를 정의하는 XML 문서문법으로 이 문법에 따라 작성되는 XML 문서는 파싱처리로 적합성을 확인해야 한다. XML 문서의 적합성을 확인하기 위한 파싱 방법으로서 프로그래밍 언어의 결정적 파싱은 표준에서 언급한 모든 엘리먼트선언에 대한 결정적 내용 모델에 대한 정의를 만족할 수 없다. 이에 본 논문에서는 적합한 XML 문서 처리를 위하여 구문편집환경에 초점을 맞추고 구문편집에 기본이 되는 DTD의 내부표현과 그에 따른 알고리즘에 대하여 기술한다. 문자열로 표현되는 DTD의 엘리먼트선언과 어트리뷰트선언의 문서 논리구조는 본 논문에서 제시하는 알고리즘에 의해 그래프구조와 테이블구조로 변환되고, 특히 테이블구조의 구문정보는 속성값을 갖는 문맥자유문법형태로 구문지향적 편집기에 이용되는 문법이 된다. 이 문법을 XML 추상문법이라고 하고 문법생성결과 및 구문편집 예를 보인다.

**키워드** : 구문편집, 그래프구조, 테이블구조, 추상문법, 참조속성

***Abstract*** A document type definition(DTD) which defines tags for a document is a XML document grammar that defines syntactic structure of a document. An XML document keeps the rules and must be parsed to check validation. To parse XML document, the deterministic parsing method of programming language is irrelevant because it does not satisfy the definition of deterministic content model in element declaration. In this paper, we consider editing of a valid XML document in syntax-directed editing environment, and we suggest the internal storage representations of syntax in DTD and theirs algorithms. The consequence is that a syntactic structure of textual DTD is transformed into graph and table structures. The table structure of DTD is interpreted the context free grammar which has attribute values and is used in syntax-directed editor for XML. We called this the XML abstract grammar and showed generated results and examples.

**Key words** : DTD, syntax editing, graph structure, table structure, abstract grammar, reference attribute.

## 1. 서 론

XML은 문서와 데이타를 구조적으로 잘 표현할 수 있는 메타 언어[1]로써, 1998년 W3C에서 표준으로 제정되었다. XML 문서는 문서타입정의(DTD : document type defintion)라는 문서 규칙과 이 규칙에 따라 작성되는 문서인스턴스(document instance)로 구성된다. 적합한(valid) XML 문서를 작성하기 위하여, 문서타입정의에서 정의된 문서의 구조 정보를 알고 있어야 하고, 작성된 문서인스턴스가 주어진 문서타입정의의 구문규칙에 따라 작성되었는지 여부를 검사하는 파싱 과정이 필요하다.

XML 문서의 구문 규칙은 문서타입정의의 엘리먼트 선언에 의한 것으로, 일반 프로그래밍 언어의 문법처럼 문맥자유문법으로 표현된다[2,3,4]. 특히 엘리먼트선언의 contentspec[1]부분은 BNF로 표현되는 규칙의 오른쪽 부분에 해당되는 것으로 정규표현으로 이루어지지만 결정적(deterministic)이지는 않다[5,6,7].

표준[1]에서는 '결정적'이라는 것을 모호하지 않는 것(umambigous)으로 다루면서 다음과 같이 해석한다."

• 본 연구는 숭실대학교 교내 연구비 지원으로 이루어졌음.
† 학생회원 : 숭실대학교 컴퓨터학과
khshin@comp.ssu.ac.kr
jmchoi@comp.ssu.ac.kr
†† 종신회원 : 숭실대학교 컴퓨터학부 교수
cwyoo@comp.ssu.ac.kr
논문접수 : 2002년 7월 16일
심사완료 : 2002년 12월 3일

문서인스턴스에서 나타날 수 있는 ... 엘리먼트는 문서인스턴스에서 룩-어헤드없이 오직 하나의 기본적인 내용 토큰(primitive content token)만을 만족해야 한다". 예를 들어 문서타입정의 규칙에 따라 작성된 (x, (y, x)*, y?)와 같은 contentspec을 이용하여 해당 문서인스턴스에 대한 적합성 확인시 XML 파서는 태그 y에 대하여 미리 다음에 오는 태그를 보지 않고는 어떤 y인지 알 수 없다.

적합한 문서인지를 검사하기 위한 XML 파싱은 비록 문서타입정의가 규칙에 맞게 작성되었다 하더라도 모두 결정적이지 않기 때문에 프로그래밍 언어의 결정적 파싱 방법을 사용하는 것은 적당하지 않다. 따라서 적합한 XML 문서를 생성하는 방법은 구문지향적 문서처리환경을 이용하는 것이 바람직한 접근방법으로 생각된다[8]. 구문 지향 편집은 자동 제공되는 언어의 구문 정보에 따라 대화식으로 문서를 작성하는 것으로, 구문적으로 올바름이 보장된다. 그리고 작성된 문서를 추상구문트리에 저장하므로서 점진적인 문서 처리가 가능하다[9,10,11].

이에 본 논문에서는 XML 문서의 구문 지향 편집을 하기 위하여 DTD에서 필요한 구문 정보 생성과 관련 알고리즘에 대하여 기술한다. 이 생성되는 구문정보는 엘리먼트선언을 근거로 하는 문맥자유문법으로, 본 논문에서는 XML 추상문법이라고 명칭하고 그림 1과 같은 생성과정을 거친다.

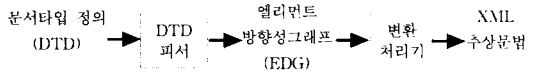


그림 1 XML 추상문법 생성 과정

XML 추상문법을 생성하기 위하여, 먼저 각 문서별로 작성된 문서타입정의는 DTD 파서에 의해 XML 표준에 따라 작성되었는지 여부와 중복된 엘리먼트 선언과 같은 불일치가 있는지를 점검하고, 엘리먼트 선언 정보를 이용하여 문서의 구조를 엘리먼트방향성그래프(EDG : element directed graph)로 생성한다. 생성된 그래프 EDG는 변환 처리기의 4단계 처리에 의해 테이블 구조로 변환하고 XML 구문 편집에 적합한 구문규칙인 추상문법으로 표현되어 구조편집기에서 사용한다.

논문 구성은 먼저 임의 문서타입정의의 정당성을 처리하는 DTD 파서와 파싱결과 생성되는 문서구조 그래프 EDG 생성 알고리즘을 제시한다. 다음으로 그래프로 표현된 문서구조 정보를 추상문법으로 표현하는 알고리즘들을 설명한다. 이 문법은 구문편집환경을 지원하기 위한 구문정보로서, 최적의 XML 추상문법을 생성하기 위하여 알고리즘을 4단계로 구분 설명한다. 끝으로 기술한 알고리즘을 이용하여 생성되는 DTD의 내부표현 결과를 확인하고 이를 이용한 XML 구문편집 예를 보여준다.

## 2. DTD와 그래프 표현

### 2.1 DTD와 파서

문서의 태그명과 그에 따른 속성들 그리고 태그 순서와 같은 XML 문서의 구문 정보를 정의하는 문서타입정의는 사용자의 의도에 따라 작성 가능한 것으로, 반드시 XML 표준에서 정해진 규칙을 따라야 한다. 그러므로 올바른 XML 문서를 작성하기 위하여 먼저 DTD에 대한 점검이 있어야 하는데, 이것은 DTD 파서에 의해 이루어진다. 이 파서는 XML 표준규칙을 메타 문법으로 입력된 DTD가 규칙에 맞게 작성되었는지를 점검하고, 파라메타 엔터티 참조에 따른 확장 및 일반 엔터티 저장 그리고 엘리먼트 선언에 대하여 중복 선언되어 있거나 정의되지 않은 엘리먼트명에 대한 참조 등의 오류를 검사한다.

### 2.2 문서 구조의 그래프 표현

XML 문서의 구조는 선언된 엘리먼트들간의 계층적인 연관성에 의해 형성된다. 하나의 엘리먼트 선언은 방향성을 갖는 트리구조로 표현 가능하고, 트리들간의 연관성으로 형성되는 포리스트는 하나의 전체적인 계층적인 엘리먼트 구조를 형성한다. 엘리먼트 그래프들의 집합으로 형성되는 이 포리스트를 XML 문서의 구조를 나타내는 것으로, 본 연구에서는 이 포리스트를 엘리먼트방향그래프(EDG : element directed graph)이라 명칭한다.

엘리먼트 트리 $t$는 엘리먼트 선언 정보를 표현한 노드와 노드들간을 연결하는 단방향성을 갖는 에지로 이루어진다. 트리를 구성하는 노드는 엘리먼트명과 어트리뷰트 정보를 저장하는 엘리먼트노드(ELEM_NODE)와 *, +, ?와 같이 반복 정도를 나타내는 반복노드(OCCUR_NODE)와, seq와 choice와 같이 발생 순서를 나타내는 지시노드(INDCAT_NODE), 그리고 PCDATA를 나타내는 단말노드(TERM_NODE) 4종류로 구분한다.

<!ELEMENT NAME contentspec>형식의 엘리먼트 선언은 트리 표현시 NAME은 트리 $t_i$의 루트 엘리먼트 노드가 되고, contentspec에 명시되는 name, choice, seq, ?, *, +들은 postfix 방식으로 표기되어 엘리먼트 트리 $t_i$의 서브노드들이 된다. 다음은 파싱된 엘리먼트 선언 정보를 문서 그래프 EDG를 생성하는 알고리즘이다<알고리즘-1>.

```
input:  DTD 파서에 의해 검증된 올바른 엘리먼트 선언 집합 E={e0, e1, e2, ... en}
        ei의 형식은  Name := postfix표기법에 따른 contentspec
output: XML 문서구조를 나타내는 엘리먼트방향그래프(EDG) G
let
        eNameArray = 엘리먼트노드 저장 배열
        ti = 엘리먼트 선언 ei에 해당하는 트리
        aroot, achild_first, anew, aprev= 엘리먼트트리 ti의 노드들
in
        eNameArray ← PCDATA 단말노드
        G ← ∅
        while E의 모든 ei에 대하여 do
        ti ← ∅
          ei의 LHS처리:
                eNameArray에서 NAME 노드 탐색
                if 존재한다면 then
                    aroot ← 해당 NAME의 ELEM_NODE
                else
                    aroot ← NAME을 이용하여 ELEM_NODE 생성
                    eNameArray ← aroot
           ei의 RHS처리:
                while ei의 RHS 정보 다 읽을때 까지 do
                  if 반복인자(? + *)이거나 발생인자(, |)이면 then
                        anew ← 해당 OCCUR_NODE나 INDCAT_NODE생성
                  else
                        eNameArray안에 RHS내의 해당 name의 ELEM_NODE 탐색
                        if 없다면 then
                            anew ← name을 이용하여 ELEM_NODE 생성
                            eNameArray ← anew
                        else
                            anew ← eNameArray내의 해당 name의 ELEM_NODE
                   if  achild_first = NULL  then
                        achild_first ← aprev ← anew
                   if  anew = ELEM_NODE then
                        achild_first에서부터 왼쪽서브노드가 비어있는 것 탐색
                        if 발견되면 then   왼쪽노드에 연결
                        else                    오른쪽노드에 연결
                   else
                        aprev의 오른쪽노드← anew
                od
           aroot의 자식노드 ←  achild_first
           G  ← ( G ∪ ti )
        od
```

알고리즘 1 엘리먼트선언을 이용한 문서그래프 생성 알고리즘

위의 알고리즘에 의해 생성된 XML 문서구조 엘리먼트방향성그래프(EDG) $G$는 다음과 같은 규칙을 갖는다. 그래프 $G$의 루트 노드는 반드시 엘리먼트노드이다. 루트노드인 엘리먼트노드를 제외하고 지시노드는 엘리먼트노드보다 상위에 위치한다. 만일 여러 개의 엘리먼트들이 동일한 연결자로 연결되는 경우, 지시노드의 오른쪽 자식노드를 자신과 동일한 것으로 만들어 연결하여 엘리먼트 개수를 확장하는 우측유도방식을 이용한다. 그래프의 최하위노드는 한개의 단말노드로 모두 연결된다.

## 3. XML 추상문법과 변환처리기

### 3.1 XML 추상 문법

XML 문서를 구문 편집하기 위하여 문서의 논리적인 구조를 정의하는 XML 문법을 본 논문에서는 XML 추상문법이라 하고 다음과 같이 4개의 튜플로 구성되는 문맥자유문법형태로 정의할 수 있다.

**<정의>** XML 추상문법 $XG = (N, \Sigma, S, P)$,

$N$ : 비단말 집합으로 문서의 구조를 정의하는 것으로

엘리먼트명의 집합

$\Sigma$ : 단말기호로서 PCDATA, EMPTY, ANY

$S$ : 시작기호로 첫 번째 엘리먼트명이나 DOCTYPE 선언명과 동일한 비단말 기호

$P$ : 추상문서규칙 $p_i$의 집합.

XML 추상문법의 문서규칙 $p_i$는 seq, choice, ?, *, + 로 표현되는 확장 BNF표현의 엘리먼트선언들을 [12]에서 언급한 "오른쪽 순환적인" 해석을 이용하여 다음과 같은 BNF형태로 재 표현한다.

$$e_0 \rightarrow op(e_1\ e_2\ \ldots\ e_n)$$

여기서 $e_0$ , $e_1$ , $e_2$ , ... $e_n$은 엘리먼트를 나타내고, $\rightarrow$은 왼쪽 엘리먼트 다음으로 나타날 수 있는 엘리먼트들의 리스트를 가리키는 표시이고 op는 연결정보이다.

문서타입정의의 어트리뷰트 선언은 이미 선언된 엘리먼트에 대한 것으로 XML 추상문법에 속성 정의가 가능하다. 일반 프로그래밍 언어의 속성 문법[13,15]은 문맥 자유 문법의 확장으로 문법 기호에 속성 값을 나타내는 것으로, 속성값의 계산은 트리의 상하노드간의 의존성에 따라 합성 속성과 상속 속성으로 구분된다. 그러나 문서타입정의의 어트리뷰트선언에 의한 속성은 계산에 의한 것이기 보다는 엘리먼트 자체에 속성을 정의하는 특징을 갖고 선택적이다. 그러므로 속성을 내포하는 XML 추상문법은 다음과 같이 규칙 왼쪽에 어트리뷰트 정보를 포함한다.

$$e_0\ \{attr_1;\ attr_2;\ \cdots\}_{opt} \rightarrow op(e_1\ e_2\ \ldots\ e_n)$$

$attr$의 구성은 "이름:(타입,디폴트값); ..."의 쌍으로, 하나의 엘리먼트에 여러 개의 속성 타입과 기본값들이 정의될 수 있으므로 $n_1{:}(t_1,dv_{10},\cdots,dv_{1n});\ n_2{:}(t_2,dv_{20},\cdots,dv_{2n});$ ...식으로 표시한다.

$$e_0\ \{n_1{:}(t_1,dv_{10},\cdots,dv_{1n});\ n_2{:}(t_2,dv_{20},\cdots,dv_{2n});\ldots\}_{opt} \rightarrow op(e_1\ e_2\ \ldots\ e_n)$$

특히 어트리뷰트 타입 IDREF, IDREFS는 이미 정의된 ID 속성 값을 확인하고 그 값을 재사용하는 참조(reference)특성을 갖는다. 이와 같이 선택적인 참조속성을 갖는 추상문법을 r-attribute XML추상문법[8]이라 한다.

```
input : 엘리먼트문서그래프(EDG) G
output : XML 추상문법 XG = (N, Σ, S, P)
         tagNameTable : 엘리먼트정보 저장 배열 테이블
let
    T = 그래프 G의 각 노드별 깊이 1인 트리 {t0, t1, t2, t3, ... tn}의 집합
    ti = (a, r)의 구성, a은 노드의 집합(a0, a1, a2), r은 노드간의 에지
in
    tagNameTable ← epcdata
    while T의 모든 트리 ti에 대하여 do
      case (a0 = ELEM_NODE) or (a0 = OCCUR_NODE) :
        e0 ← search_tagNameTable(a0) /* ti의 트리노드 a0처리 */
        e1 ← search_tagNameTable(a1) /* ti의 하위노드 a1처리 */
        if (a1 = ELEM_NODE) then
            pi의 RHS ← one(e1)
        else if (a1 = '+'를 나타내는 OCCUR_NODE) then
            pi의 RHS ← plus(e1)
        else if (a1 = '*'를 나타내는 OCCUR_NODE) then
            pi의 RHS ← star(e1)
        else /* a1이 '?'인 OCCUR_NODE 경우로 2개 규칙생성 */
            pi의 RHS ← null( )
            pi의 RHS ← one(e1)
      case (a0 = INDCAT_NODE) :
      /* ti의 트리노드 a0처리 */
        e0 ← search_tagNameTable(a0)
        e1 ← search_tagNameTable(a1)
        if (a2가 존재하면) then
          e2 ← search_tagNameTable(a2)
        if (a0 = seq를 나타내는 INDCAT_NODE ) then
          pi의 RHS ← pair(e1 e2)
        else /* a0 = choice를 나타내는 INDCAT_NODE일때, 규칙 2개 생성*/
          pi의 RHS ← one(e1)
          pi+1의 RHS ← one(e2)
    od
```

알고리즘 2 그래프 EDG를 이용한 문법 생성의 초기화

### 3.2 변환처리기

변환처리기는 엘리먼트방향성그래프구조(EDG)로 표현된 구문정보를 테이블구조의 문맥자유형태인 XML 추상문법으로 변환한다. 변환된 추상문법의 문서규칙은 $e_0\{attr_1; attr_2; \cdots\}_{opt} \rightarrow op(e_1\ e_2\ \ldots\ e_n)$ 형태로서, 이것은 4개의 필드로 구성되는 테이블에 $e_0$ 부분, 속성 부분, $op$ 부분, 그리고 괄호안의 $e_1$, $e_2$, .. 부분들이 구분 저장한다. $op$는 지시자와 반복자의 변환 과정 중 변환되는 오퍼레이터이다.

1) 초기 단계

EDG G로부터 추상문서규칙을 생성하는 첫 번째 단계로, 그래프 $G$의 루트노드를 시작으로 preorder 방식으로 순회하면서 각 노드별로 깊이가 1인 트리 $t_i$를 대상으로 다음과 같은 초기 추상문서규칙 형태를 생성한다.

$$e_0 \rightarrow op(e_1\ e_2)$$

2진트리형태인 $t_i$에서, 상위 노드는 규칙의 왼쪽부분(LHS) $e_0$에, 하위 노드는 오른쪽부분(RHS) $e_1$와 $e_2$에 대응된다. 만일 $t_i$의 상위 노드가 엘리먼트노드라면 추상문서규칙의 왼쪽 $e_0$에는 해당 엘리먼트명을 사용하지만, 반복노드나 지시노드인 경우라면 임시노드명을 정의한다. 트리의 루트가 지시노드인 경우는 하위 노드가 2진형태이므로 자식노드 왼쪽과 오른쪽을 모두 처리한다. $t_i$의 하위 노드는 추상문서규칙의 오른쪽에 적용되는 것으로, 만일 하나의 엘리먼트노드만이 연결되어 있다면 $op$는 one이 되고, $e_1$로는 해당 엘리먼트명을 사용한다. 반복노드, 지시노드인 경우 $op$는 해당 노드 정보에 따라 pair, or, option, star, plus 중 선택하고 $e_1$에는 임시명을 사용한다. 이때 사용되는 모든 엘리먼트명은 별도의 테이블을 생성하여 관리한다. 문법생성의 초기화 알고리즘은 다음과 같다<알고리즘-2>.

알고리즘-2에 의해 XML 문법의 구성요소인 비단말집합 $N$과 추상문서규칙 $P$가 생성된다. 비단말집합 $N$은 tagNameTable에 저장되어 있는 모든 엘리먼트명 $\{e_0, e_1, e_2, \ldots\}$이고, PCDATA의 엘리먼트 $e_{pcdata}$은 단말기호 $T$가 되며 알고리즘-3에 의해 관리된다.

생성된 추상문서규칙 $P=\{p_0, p_1, p_2, \ldots\}$은 그래프의 노드별 깊이 1인 트리를 이용하여 만들어진 것으로 규칙 개수가 적지 않다. 그러므로 이 추상문서규칙의 최적화 처리과정이 필요하다.

2) 최적화 1 단계 : 단일 추상문서규칙 결합처리

추상문서규칙 최적화의 첫 번째 단계로는, 추상문서규칙 중 오른쪽에 한 개의 엘리먼트명만을 갖는 규칙의 결합으로 불필요한 규칙의 수를 제거하는 것이다<알고리즘-4>. 먼저, 추상문서규칙 중 오른쪽이 one($e_1$) 형태

```
input : t_i 트리의 노드 a
output : tagNameTable의 엘리먼트 e_i
        tagNameTable = 엘리먼트명을 관리하기
                       위한 테이블 배열
let
    find = 엘리먼트 발견여부 확인 boolean
in
    find ← false
    repeat
        for tagNameTable에 저장된 모든 엘리먼트 e do
        if (노드 a의 이름) = (tagNameTable의 e_i) then
                find ← true
    until find = false
    if (find = false) then
            if (노드 a = ELEM_NODE) then
                    tagNameTable e_i ← 노드 a의 이름
            else
                    tagNameTable e_i ← 노드 a의
                    적당한 순차번호를 이용한 임시명
    retrun (tagNameTable의 e_i)
```

알고리즘 3 search_tagNameTable(a) : 엘리먼트 테이블관리

인 규칙 $p_i$를 찾는다. 그리고 $e_1$이 추상문서규칙 왼쪽에 기술되어 있는 추상문서규칙 $q_i$를 찾아서 $p_i$의 왼쪽과 $q_i$의 오른쪽을 결합하여 새롭게 수정된 추상문서규칙 $p_i'$를 생성한다.

$p_i$ : $e_0 \rightarrow$ **one**( $e_1$ ),

$q_i$ : $e_1 \rightarrow op(\ e_{11}\ e_{12}\ \ldots\ e_{1n}\ )$

$\Rightarrow$ 수정된 $p_i'$ : $e_0 \rightarrow op(\ e_{11}\ e_{12}\ \ldots\ e_{1n}\ )$

3) 최적화 2 단계 : pair 추상문서규칙 처리

추상문서규칙 최적화의 두 번째 단계는 추상문서규칙 오른쪽의 $op$가 $pair$인 것에 대한 처리다<알고리즘-5>.

```
input : <알고리즘-2>에 의해 생성된 추상문서규칙 P
output : 최적화된 추상문서규칙 P'
let
    p_i, q_i = P의 추상문서규칙
    p_i'     = 수정된 추상문서규칙 p_i
in
    for P의 모든 추상문서규칙 pi에 대하여 do
        op가 one인 추상문서규칙 pi 탐색
        if p_i가 존재하면 then
        p_i의 RHS의 e_1이 다른 추상규칙 LHS에 정의된
        추상규칙 q_i를 찾아 다음과 같은 결합 처리(⊕)를
        한다.
         p_i' ← (p_i의 LHS) ⊕ (q_i의 RHS)
od
```

알고리즘 4 단일(one) 추상문서규칙 결합하기

*op*가 *pair*인 추상문서규칙은 엘리먼트들이 열거된 순서대로 차례로 발생됨을 나타내는 seq에 대한 것으로, 이는 추상문서규칙 오른쪽에 엘리먼트명들이 2개 이상 열거 가능한 규칙이 될 수 있다. 그러나 그래프 EDG를 변환한 규칙은 오른쪽에 2개 엘리먼트명만이 명시되므로 한 엘리먼트 선언에서 유도된 pair 규칙 모두를 찾아 결합한다.

$p_i$ : $e_0$ → **pair**( $e_1$ $e_{\$tmp\_n}$ )

$q_i$ : $e_{\$tmp\_n}$ → **pair**( $e_2$ $e_3$ )

⇒ 수정된 $p_i'$ : $e_0$ → **pair**( $e_1$ $e_2$ $e_3$ )

pair로 명시된 추상문서규칙 $p_i$의 오른쪽에 열거되어 있는 엘리먼트명 중에 임시명을 포함하는지를 확인한다. 만일 임시 엘리먼트명을 포함한다면, 이 임시명이 추상문서규칙 왼쪽에 기술된 추상규칙 $q_i$을 찾는다. 그리고 찾은 추상규칙 $q_i$의 *op* 또한 pair라면 $q_i$의 오른쪽에 열거되어 있는 엘리먼트명들을 규칙 $p_i$의 해당 임시명 위치에 대치시켜 추상문서규칙을 최적화시킨다.

input : <알고리즘-4>에 의해 생성된 추상문서규칙 $P$
output : 최적화된 추상문서규칙 $P'$
**let**
  $p_i$, $q_i$ = $P$의 추상문서규칙
  $p_i'$ = 수정된 추상문서규칙 $p_i$
**in**
  **for** 모든 추상문서규칙 P에 대하여 **do**
    *op*가 pair인 추상규칙 $p_i$ 탐색
    **if** (규칙 $p_i$ 있다) **and** ($p_i$의 RHS에 임시엘리먼트 $e_{\$tmp\_n}$ 있다면) then
      $p_i$의 RHS의 임시명을 LHS으로 갖는 어떤 추상규칙 $q_i$ 탐색
      **if** $q_i$의 *op* = pair **then**
        $p_i' \leftarrow (p_i$의 RHS) ⊕ ($q_i$의 RHS)
  **od**

알고리즘 5 pair 추상생성규칙의 재 형성

4) 최적화 3 단계 : plus, star 추상문서규칙의 변환

이 단계는 *op*가 plus와 star인 추상문서규칙에 대한 처리로서, [12]에서 언급한 "오른쪽 순환적인" 해석을 이용한다. plus는 한번 이상 반복을, star는 반복이 없거나 아니면 한번 이상을 의미하는 것에 따라 추상문서규칙을 재구성한다. 따라서 *op*가 plus인 추상문서규칙은 *op*가 one과 pair인 2개의 추상문서규칙으로, 그리고 *op*가 star 추상규칙은 *op*가 null과 pair 인 2개의 추상문서규칙으로 분리된다<알고리즘-6>.

input : <알고리즘-5>에 의해 생성된 추상문서규칙 $P$
output : 최적화된 추상문서규칙 $P'$
**let**
  $p_i$, $p_i'$ = $P$의 추상문서규칙
**in**
  **for** 모든 추상문서규칙 $P$에 대하여 **do**
    **case** *op*가 **plus**인 추상문서규칙 $p_i$에 대하여
      새로운 임시엘리먼트명 $e_{\$tmp_n}$생성
      2개 새로운 추상문서규칙 $p_i'$, $p_{i+1}'$생성
      $p_i'$ : $e_{\$tmp\_n}$ → **one**($p_i$의 RHS)
      $p_{i+1}'$ : $e_{\$tmp\_n}$ → **pair**($p_i$의 RHS, $e_{\$tmp\_n}$)
    **case** *op*가 **star**인 추상규칙 $p_i$에 대하여
      새로운 임시엘리먼트명 $e_{\$tmp_n}$생성
      2개 새로운 추상문서규칙 $p_i'$, $p_{i+1}'$생성
      $p_i'$ : $e_{\$tmp\_n}$ → **null**()
      $p_{i+1}'$: $e_{\$tmp\_n}$ → **pair**($p_i$의 오른쪽부분, $e_{\$tmp\_n}$)
    $p_i$ 추상문서규칙의 오른쪽부분 변경
    $p_i$ : $e_0$ → **one**($e_{\$tmp\_n}$)
  **od**

알고리즘 6 plus, star 추상문서규칙의 변형

알고리즘-6에서는 추상문서규칙의 개수를 줄이기보다는 함축되어 있는 *op*의 종류와 의미를 단순화시킨다.

## 4. XML 추상문법 실험 결과

모든 DTD를 대상으로 XML 추상문법을 생성하기 위하여 그림 1 구조의 자동생성기를 구현하였다. 이 생성기는 다른 응용프로그램에 임베디드되는 형태로 편집기나 브라우저에 포함된다. XML 추상문법을 자동생성하는 생성기는 DTD파서와 그래프변환기로 구성된다. DTD파서는 LALR방식으로 입력 DTD가 표준규칙에 따라 작성되었는지 구문적 오류를 점검한 후 알고리즘-1을 이용하여 엘리먼트그래프 EDG를 생성한다. 그리고 그래프변환기는 알고리즘-2에서부터 알고리즘-6을 이용하여 입력된 그래프 EDG를 테이블구조로 변환하는 처리기로, 변환된 테이블 구조는 속성값을 갖는 문맥자유문법형태인 r-attribute XML 추상문법이 된다.

간단한 예제 mail DTD 예로 생성기에 의해 추상문법 생성 결과와 그 사용예를 살펴본다.

```
<!DOCTYPE mail [
<!ELEMENT mail      (sender, receiver+, cc?, title, content) >
<!ELEMENT sender    (email)>
<!ELEMENT receiver  (email)>
<!ELEMENT cc        (email)>
<!ELEMENT title     (#PCDATA)>
<!ELEMENT content   (p)+>
```

```
<!ELEMENT email     (#PCDATA)>
<!ELEMENT p         (#PCDATA)>
<!ATTRLIST  sender  name CDATA  #REQUIRED>
<!ATTRLIST  receiver name CDATA  #REQUIRED>
 ]>
```

예제 DTD는 DTD 파서의 파싱 결과 표준 규칙에 따라 올바르게 작성된 것으로 검증되면 알고리즘-1을 이용하여 그림 2와 같은 그래프 EDG를 생성한다. 이 그래프 구조는 엘리먼트노드, 반복노드, 지시노드 3가지 종류의 노드들이 단 방향성을 갖는 에지로 연결되는 2진형태를 갖는다. 일반적으로 하나의 루트노드에서 시작하여 하나의 단말노드(PCDATA)로 마무리되는 그래프 EDG는 엘리먼트노드와 반복노드는 한개의 자식노드를 갖는 반면에 seq나 choice를 나타내는 지시노드는 자식노드를 2개까지 가능하고, 같은 지시자들로 연결되는 엘리먼트 리스트는 우측유도방식을 취한다.

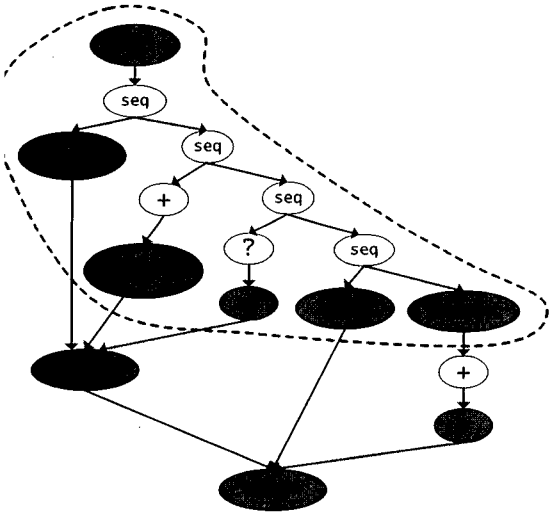


그림 2 그래프 구조의 mail DTD

그림 2의 그래프 EDG는 9개의 엘리먼트노드와 3개의 반복노드 그리고 4개의 지시노드들이 방향성을 갖는 에지로 연결되는 형태이고, 특히 sender노드와 receiver 노드는 속성정보를 포함하고 있다.

생성된 그래프 EDG는 그래프변환기를 이용하여 구문편집환경에서 좀 더 용이하여 사용할 수 있는 테이블 구조로 변환한다. 이 변환은 초기단계, 최적화 1단계, 최적화 2단계 그리고 최적화 3단계를 거치게 된다. 다음 표 1은 그림 2의 DTD 그래프에서 mail 엘리먼트선언에 해당하는 부분에 대한 변환 결과를 구문규칙형태로 보인다.

①은 초기단계에서 알고리즘-2와 알고리즘-3에 의해 생성한 결과이다. 그래프에서 노드간에 깊이가 1인 서브트리를 대상으로 상위노드는 규칙 왼쪽에 하위노드는 규칙 오른쪽에 열거된다. 이때 엘리먼트 노드를 제외한 노드들은 순차적인 번호를 부여받는 임시명으로 처리한다. 그리고 규칙 *op*에는 one, pair, plus, star, null들이 열거된다. ②는 최적화1단계인 단일 추상문서규칙 결합처리결과로서, 알고리즘-4에 의해 mail → one($e1)와 $e1 → pair(sender $e2) 규칙은 하나의 mail → pair(sender $e2)규칙으로 재 표현된다. ③은 *op*가 pair인 추상구문규칙들을 하나로 결합하는 최적화 2단계 처리결과로서, 알고리즘-5에 의한다. 따라서 mail → pair(sender $e2), $e2 → pair($e3 $e4), $e4 → pair ($e5 $e6) 그리고 $e6 →pair(title content) 5개 추상구문규칙은 하나의 추상구문규칙 mail → pair(sender $e3 $e5 title content)으로 결합된다. mail 엘리먼트선언의 *contentspec*이 +,*로 끝나지 않으므로 최적화3단계는 통과한다. 결과적으로 변환처리기에 의한 테이블 구조의 예제 DTD는 표 2와 같다. 이 생성된 테이블은 LHS, attribute, op, RHS 4개의 필드로 이루어진다.

테이블 구조의 예제 DTD는 XML 추상문법의 4개 튜플로 표현하면 다음과 같다. 먼저 비단말기호 N=

표 1 ① 테이블초기화결과 ② 최적화1단계결과 ③ 최적화2단계결과

| ① | ② | ③ |
|---|---|---|
| mail → one($e1)<br>$e1 → pair(sender $e2)<br>$e2 → pair($e3 $e4)<br>$e3 → plus(receiver)<br>$e4 → pair($e5 $e6)<br>$e5 → null()<br>$e5 → one(cc)<br>$e6 → pair(title content) | mail → pair(sender $e2)<br>$e2 → pair($e3 $e4)<br>$e3 → plus(receiver)<br>$e4 → pair($e5 $e6)<br>$e5 → null()<br>$e5 → one(cc)<br>$e6 →pair(title content) | mail → pair(sender $e3 $e5 title content)<br>$e3 → plus(receiver)<br>$e5 → null()<br>$e5 → one(cc) |

표 2 테이블 구조의 mail DTD

| LHS | attribute | op | RHS |
|---|---|---|---|
| mail | | pair | sender $e3 $e5 title content |
| sender | {name:(CDATA,#REQUIRED);} | one | email |
| $e3 | | one | receiver |
| $e3 | | pair | receiver $e3 |
| receiver | {name:(CDATA,#REQUIRED);} | one | email |
| $e5 | | null | ε |
| $e5 | | one | cc |
| cc | | one | email |
| email | | one | PCDATA |
| title | | one | PCDATA |
| content | | one | $e9 |
| $e9 | | one | p |
| $e9 | | pair | p $e9 |
| p | | one | PCDATA |

{mail, sender, $e3, receiver, $e5, cc, email, title, content, $e9, p}으로서, 특히 $로 시작하는 비단말기호는 변환과정에서 생긴 임시기호로 작성된 XML 문서의 트리를 유지하기 위한 감춰진 내부 비단말기호이다. 단말기호 Σ는{PCDATA}이고, 시작기호 S는 mail이며, 추상문서규칙 P는 테이블 정보에 의한 것으로 LHS{*attribute*}$_{opt}$ → *op*(*RHS*)의 문맥자유문법 형태를 취한다. 이때 *op*는 오직 pair, one, null 3가지만 존재하는데 pair는 *RHS*에 2개 이상 열거된 엘리먼트의 순차적 처리되는 규칙을 나타내고, one은 *RHS*에 처리할 엘리먼트가 오직 하나인 규칙 그리고 null은 처리할 엘리먼트가 없는 규칙이다. 비단말기호 중 sender와 receiver는 선언된 속성정보를 갖는다.

생성된 XML 추상문법은 구문지향적 편집기 생성시 이용할 수 있는 XML문서문법이다. 그림 3은 mail 추상문법을 이용한 구문편집기 처리결과로서, 왼쪽은 텍스트 기반이고 오른쪽은 트리기반이다.

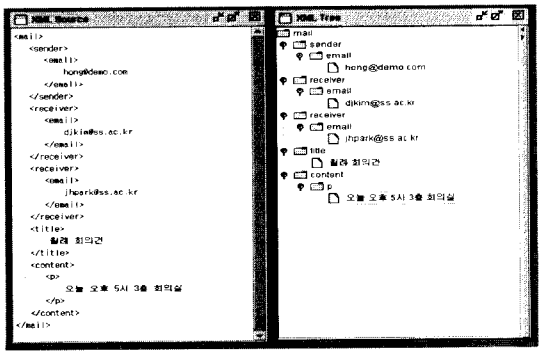

그림 3 mail 추상문법을 이용한 XML 문서 구문편집 예

XML 추상문법은 voiceXML, HTML, DOCBOOK와 같이 복잡한 문서타입정의에서도 생성이 가능하므로 다양한 문서타입의 XML 구문편집이 가능하다[8,14]. 그리고 테이블구조의 XML 추상문법은 구문지향편집기 생성 도구인 Synthesizer Generator[9]의 SSL(Synthesizer Specification Language)로 변환하여 그림 4와 같은 XML 구문지향편집기를 자동생성 할 수 있다.

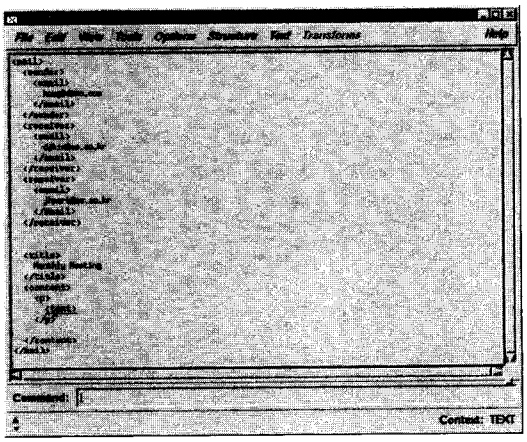

그림 4 Synthesizer Generator를 이용한 mail DTD의 구문편집기

## 5. 관련 연구

XML 문서를 처리하기 위한 평문 형태의 DTD는 문맥자유문법과 유사하지만, 규칙은 확장적이고 제한적인 정규표현으로 되어 좀 복잡하다[1]. [2]에서는 DTD에서 정의된 구문규칙은 정규표현(regular expression)으

로 변환한다. 이 변환된 구조의 정규표현은 DTD 인식기와 구문을 따르는 XML문서인지를 처리하는 유한오토마타 시스템 구축에 이용된다. [3]은 DTD를 확장된 문맥자유문법으로 표현한다. 확장된 문맥자유문법 G는 구문규칙 오른쪽이 정규표현으로 표기되는 문맥자유문법으로, 4개의 튜플(N,Σ,P,S)로 기술된다. 그 중 규칙 P는 $A \rightarrow E$ 형태로, A는 비단말기호($N$)이고 E는 $V(=N \cup \Sigma)$로 표기되는 정규표현이다. [3]에서는 XML 문서를 확장된 문맥자유문법 G를 따르는 언어 L(G)로서, G의 문장기호에서 유도되는 단말스트링의 집합으로 다룬다:$L(G)=\{w \in \Sigma^* \mid S \Rightarrow^+ w\}$. [4]의 XML-grammar는 DTD를 문맥자유문법으로 표현하고 XML 문서를 XML-language로 다룬다. XML-grammar는 XML문서가 기본적으로 시작태그와 끝태그가 쌍으로 잘 정의(well-formed)되어 있다는 특성을 이용하여 균형적인(balanced)문법으로 정의된다. 따라서 XML문서를 시작태그 $A$와 끝태그 $\bar{A}$의 집합으로 간주하는 XML-grammar G의 구성은 다음과 같다. 단말 T는 엘리먼트명 V에 의해 생성되는 *axiom*특성을 갖는 태그의 집합으로, $T=A \cup \bar{A}$이다. 구문규칙은 $X_a \rightarrow aR_a\bar{a}$ 형태로, 정규집합 $R_a \subset V^*$은 다음에 유도되는 엘리먼트들의 정규표현이다. 구문규칙의 왼쪽부분에 나타나는 비단말기호 $X_a$는 A에 속하는 원소 a, $a \in A$를 유도하는 구문기호이다. 이러한 연구들은 XML문서를 DTD 규칙에 따라 작성되는 언어로 간주하여, 이 언어 인식에 적합한 형태로 DTD를 문법표현하였다.

이에 비해 본 논문에서 제시하는 XML 추상문법은 DTD를 문맥자유문법으로 표현하였다. 이 XML 추상문법 G는 구문편집환경에서 문서 편집을 유도하기 위한 구문정보로 사용하기 위한 것으로 문법 구성은 다음과 같다. 먼저, 단말 $\Sigma$는 실제문서내용을 입력하는 PCDATA이고, 비단말 N은 엘리먼트명의 집합이며, 그리고 구문규칙 P는 $N \rightarrow \alpha$ , $\alpha \subset (N \cup \Sigma)^*$ 형태이다. 문서의 시작태그와 끝태그는 엘리먼트명에 의해 발생되는 것으로서 이것은 비단말기호의 내포정보로 간주하여 편집할 때 구문편집기에 의해 자동태깅 되도록 한다. 추상문법 G는 DTD의 어트리뷰트선언을 이용하여 속성정의하였다. 어트리뷰트는 엘리먼트에 대한 자체 정의와 이미 정의된 것을 참조하는 것으로, 추상문법 G의 구문규칙 왼쪽 비단말기호에 $N\{attributes\}_{opt} \rightarrow \alpha$ , $\alpha \subset (N \cup \Sigma)^*$와 같은 형태로 정의된다.

## 6. 결 론

DTD는 사용자에 의해 임의로 작성되는 텍스트형의 문서 구조로서 표준규칙에 따라 올바르게 작성된것만이 XML 문서문법으로 유효하다. 이에 본 논문에서는 적합한 XML 문서 편집을 위하여 필요한 DTD 구문을 표현한 알고리즘과 그 결과 생성된 문맥자유문법형태의 문서문법에 대하여 설명하였다.

문서구문을 표현한 알고리즘은 문자열의 엘리먼트 선언정보를 트리의 집합체인 포리스트형태의 그래프구조로 나타낸 것과 그래프구조를 테이블구조로 표현한 것으로 구분된다. 특히 테이블구조에 저장된 구문 정보는 4개의 튜플(N, Σ, S, P)로 구성되는 문맥자유문법형태로서, 단계별 알고리즘 처리로 테이블 정보를 최적화 하였다. 이 최적화된 구문정보는 구문편집환경에서 XML 문서편집시 이용되는 것으로 본 논문에서는 이것을 XML 추상문법이라 한다. XML 추상문법은 엘리먼트와 관련있는 어트리뷰트선언에 의해 참조속성값을 갖는 추상문법(r-attribute XML 추상문법)으로도 표현한다. 이러한 XML 추상문법을 생성하기 위하여 DTD파서와 변환처리기를 구현하였다. DTD에 따른 XML 추상문법의 생성으로 구문편집기의 자동생성[14]은 물론 여러유형의 유저인터페이스와의 결합으로 동일한 문서를 다양한 편집화면으로 출력해 볼 수 있다[8]. 그리고 구문정보를 여러구조로 표현할 수 있으므로서 응용 분야가 다양하다.

DTD의 구문정보를 그래프형태로 표현한 엘리먼트방향성그래프(EDG)는 트리의 집합체인 포리스트형태로서 트리문법에 대한 연구를 기반으로 엘리먼트방향성그래프에 대한 문법적인 정의와 그래프기반의 구문편집에 대한 연구가 이루어져야 한다.

## 참 고 문 헌

[1] XML, http : //www.w3.org/TR/1998/REC-xml-19980210.

[2] R.W. Matzen, K.M. George, and G.E. Hedrick, "A Formal Language Model for Parsing SGML", *Journal of Systems and Software, v. 36, #2, February*, 1997.

[3] P. Kilpeläinen and D. Wood, "SGML and XML Document Grammars and Exceptions", HKUST-TCSC-99-01, 1999.

[4] J. Berstel and L. Boasson, "XML Grammars", *MFCS'2000, Vol. 1893 of LNCS*, 2000.

[5] XML, http : //www.xml.com/axml/target.html#determinism

[6] A. Bruggemann-Klein and D. Wood, "Deterministic Regular Language", STACS 1992, 1991.

[7] J. Warmer and S.Van Egmond, "The Implementation of the Amsterdam SGML Parser", *Elecl*

*tornic Publishing, VOL 2(July 1989)*, 1989.
[8] 신경희, 유재우, "다중뷰를 지원하는 구조적 XML 에디터 생성", 프로그래밍언어학회 논문지, 2001년 11월.
[9] T. W. Reps and T. Teitelbaum, *The Synthesizer Generator:a system for constructing language-based editors*, Springer-Verlag, 1989.
[10] T. Reps, *Generating Language-Based Environment*, The MIT Press, 1986.
[11] J. Knudsen, M. Lofgren, O. Massen and B. Magnusson, *Object-Oriented Environments:The MJOLNER APPROACH*, Prentice Hall, 1993.
[12] D. Grune and C. Jacobs, *Parsing Techniques: A Practical Guide*, Prentice Hall, 1990.
[13] H. Albals and B. Melichar, Attribute Grammars, *Applications and Systems*, Springer-Verlag, 1991.
[14] K. H. Shin and C. W. Yoo, "The Validated XML Editor", *IC'2001 Conference, 2001*. 7.
[15] A. Aho, J. Ullman, *The Theory of Parsing, Translation, and Compiling*, Prentice-Hall, 1972.

신 경 희
1985년 인하대학교 전자계산학과 학사. 1995년 숭실대학교 전자계산학과 석사. 현재 숭실대학교 컴퓨터학과 박사과정 수료. 관심분야는 프로그래밍 언어, 프로그래밍 환경, 시각 프로그래밍 언어, XML

최 종 명
1992년 숭실대학교 전자계산학과 학사. 1996년 숭실대학교 전자계산학과 석사. 1997년 ~ 현재 숭실대학교 컴퓨터학과 박사과정. 관심분야는 시각 프로그래밍, 멀티패러다임 시스템, XML

유 재 우
1976년 학사 숭실대학교 전자계산학과. 1985년 박사 한국과학기술원 전산학과. 1983년 ~ 현재 숭실대학교 컴퓨터학부 교수. 1986년 ~ 1987년, 1996년 ~ 1997년, 코넬대학교, 피츠버그대학교 객원교수. 1999년 ~ 2000년 한국정보과학회 프로그래밍언어 연구회 위원장. 관심분야는 프로그래밍 언어, 컴파일러, 인간과 컴퓨터 상호작용