

파스 트리를 이용한 불완전 XML 문서의 완성

조용윤^o, 박용관, 유재우
승실대학교 컴퓨터학과

{yycho^o, psiver}@amin.ssu.ac.kr, cwyo0@comp.ssu.ac.kr

Completion of Incomplete XML Document Using Parse Tree

Yong-Yoon Cho^o, Ho-Byung Park, Chae-Woo Yoo
School of Computer, SoongSil University

요 약

프로그래머는 프로그램 작성 중 문법적으로 올바르지 않은 문장을 입력할 수 있다. 문법적으로 불완전한 문장 입력은 사용자의 계속적인 편집 상태를 종료하고, 정상적인 파싱을 보장하지 않는 원인이 된다. 따라서, 사용자는 편집기가 제공하는 적절한 오류 처리 루틴을 통해 문법적으로 잘못된 문장에 대해 수정 작업을 실행해야 한다. 에디터 환경에서의 사용자 입력은 입력 부분의 왼쪽, 오른쪽에서 완전하지 못한 형태로 입력될 수 있다. 따라서, 에디터가 문법적으로 빠져있는 부분의 심볼이 무엇인지 정확히 인식 가능하고, 주어진 문법에 따라 부분적인 파스 트리를 완성한다면, 사용자의 프로그래밍 편집 상태를 종료하지 않고 계속적인 편집과 성공적인 파싱을 보장할 수 있을 것이다. 본 논문은 파서가 문법적으로 올바르지 않은 문장의 입력에 대해 해당 문법에 따라 빠진 부분을 올바로 인식하고, 누락된 문법 심벌을 찾아 부족한 부분 파스 트리를 완성함으로써 성공적인 편집을 보장할 수 있는 파싱 방법을 제안하고 실험한다. 제안된 파싱 방법을 통해 사용자는 프로그래밍 편집 중 문법 오류에 대한 부담을 줄일 수 있으며, 불완전 입력에 대해 계속적인 파싱을 보장받아 편집 효율을 높일 수 있다.

1. 서론

프로그래밍 편집기의 파서에 의해 프로그램은 문법적인 파싱을 거쳐, 결과로 파스 트리로 표현된다. 텍스트 기반의 프로그램 편집 환경에서 사용될 수 있는 상향식 파싱 기법에 있어서 파스 트리가 상향식으로 생성되고 트리의 노드는 축약이 이루어질 때 생성되어지기 때문에 정확하게 파스 트리의 누락된 부분을 찾는다는 것은 어렵다. 상향식 파싱 과정에서 사용되는 자료 구조인 스택은 파스 트리의 각 부분 트리의 루트를 의미하는 비단말 기호를 포함한다. 그러나, 입력이 불완전할 경우, 파싱 스택은 완전히 축약되지 못하여 여러 개의 비단말기호를 가진 문장형을 이루게 되며, 더 이상의 표준 하향식 파싱 작업이 이루어질 수 없다. 편집기의 입력은 프로그램의 편집 위치에 따라 문장의 왼쪽 혹은 오른쪽이 부분적으로만 존재할 수 있다. 다음의 [그림 1]은 완전하지 않은 XML 프로그램의 일부분이다. 불완전 입력 문장을 인식하고 완성하기 위해 제안되는 파서는 불완전 입력 문장이 문법에 대해 유효한 경우에 한해서만 부족한 부분을 완성해야 하기 때문에, 본 논문에서 실험되어지는 불완전 입력 문장의 범위는 문법에 어긋나지 않지만 구성 성분이 빠져있는 문장만을 그 논의의 범위로 한다.

```
<name>
  <firstname> Cho </firstname>
  yong yoon </lastname>
</name>
```

[그림 1] 불완전한 XML 입력

위의 완전하지 않은 문장에 대해 파서는 해당 부분 파스 트리를 완성하지 않고 파싱을 종료한다. 그러나, 파서는 불완전 입력에 대한 적절한 문법 정보를 이용해 빠진 부분을 인식하고 그 부분에 대한 부분 트리를 구성하여 계속적인 입력과 성공적인 파싱을 보장하는 것이 바람직하다. 본 논문에서는 불완전 입력에 대해 부분적인 파스 트리를 생성하고 전체 파스 트리의 정확한 위치에 고정시킬 수 있는 파싱 방법을 제안한다. 제안된 파싱 방법은 부분 파스 트리로부터 정해진 문법에 대해 문법적 유효여부를 확인하고 불완전한 입력에 대해 문법적으로 빠져있는 입력 심벌을 생성하기 위한 일반적인 LR파싱 테이블의 확장을

포함한다. 본 논문에서 제안된 알고리즘은 불완전한 XML 입력에 대해 실험을 수행했다.

2. 본 문

편집 환경에서 입력되는 구문은 부주의로 인해 완전한 문장으로 입력되기보다는 부분적으로 누락된 문장의 일부가 입력될 수 있다. 따라서 본 논문은 부분적인 입력 문장을 문법적으로 인식하고 이를 통해 부족한 문장을 완성할 수 있는 방법을 제시한다.

2.1 불완전 입력의 인식

하향식 형태의 서브스트링을 인식하는 방법은 기본적으로 모든 가능한 prefix에 대하여 파서의 동작 상태를 따라 인식 가능한 형태의 파스 트리를 구하는 것이다. 서브 스트링의 인식을 위해서는 스택의 움직임에 따르지만 푸시다운 오토마타가 파스 경로에 따라 다양한 형태의 동작을 가지도록 하여야 한다. 따라서 스택의 여러 가지 동작 과정을 모두 표현할 수 있는 방법이 필요하다. 이것은 Graph Structured Stack을 이용하여 해결할 수 있다[5]. 불완전 입력을 인식하기 위한 알고리즘은 다음과 같다.

[알고리즘 1] 서브 스트링 인식

서브 스트링을 인식하기 위해 LR파서의 동작을 reduce 항목에 대해 다음과 같이 수정한다.

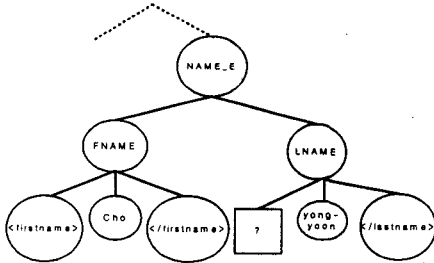
reduce A := aβ 일 때

case |aβ| + 1 > |stack| : 정상적인 LR 방법으로 파싱

case |β| = |stack| : placeholder로서 a에 대한 노드를 생성;
goto(A)에 대한 다음 상태를 조사;
A로 축약;

case |aβ| = |stack| : goto(A)에 대한 다음 상태를 조사;
A로 축약;

제한된 알고리즘을 이용하여 [그림 1] 과 같은 불완전 입력에 대해 파싱이 성공적으로 끝나면 [그림 2]와 같이 불완전 노드를 포함하는 파스 트리가 구성된다.



[그림 2] 불완전 입력 인식

루트노드에서 단말노드까지의 모든 경로는 입력된 부분 문맥의 가능한 해석으로 하향식 파서 스택의 top 부분에 해당하며, 이 경우 경로의 root에 해당되는 노드의 상태는 스택의 top에 해당하게 된다.

2.2 불완전 입력에 대한 부분 트리 완성

파서가 누락된 것으로 인식된 심벌을 적절한 위치에 생성시키고 파싱을 계속하기 위해서는 푸시다운 오토마타가 파스 경로에 따라 다양한 형태의 동작을 가지도록 스택의 여러 가지 동작 과정을 모두 표현할 수 있는 방법이 필요하다. 따라서, 파싱 테이블은 [알고리즘 1]에 따라 축약 부분에 대해 추가적인 정보를 제공할 수 있도록 확장되어야 한다. 확장된 LR 파싱 테이블을 얻기 위한 알고리즘은 다음과 같다.

[알고리즘 2] 불완전 축약을 위한 파싱 테이블 확장

입력 : 문법 G와 canonical collection C={I₁, I₂, ..., I_n}

출력 : n개의 state를 갖는 ACTION 함수로 구성된 확장형 파싱표

방법 :

- (1) [알고리즘 2]에 의해 파싱 테이블을 구성한다.
- (2) ACTION[S_i, \$] = "error" 항목을 다음과 같이 수정한다.

(2.1) 간일[A → α · β]의 형태가 I_i에 있으면,

$$ACTION[S_i, \$] = \text{"reduce } \sim A \rightarrow \alpha \beta \text{"}$$

(2.2) ACTION[S₀, \$] = "error"

파싱 과정에서 여러 개의 경로가 나올 경우 각각의 트리는 별개로 구성한다. 트리가 구성될 수 있으면 유효한 서브스트링으로 간주된다. 트리가 구성될 수 없다면 해당 입력을 무시하고 트리의 단말에서 루트까지의 모든 경로를 삭제한다. 부분 문맥의 모든 입력이 이루어지고 유효한 파스 경로가 하나 이상 생성된다면 주어진 부분 문맥은 유효한 부분 문맥이다. 따라서, 파서가 각 유효한 파스 경로에 대해 파싱 결정을 위한 정보를 이용하기 위해 확장된 파싱 테이블 정보가 필요하다.

3. 실험 및 평가

본 논문은 불완전한 XML 입력 문장에 대해 제안된 알고리즘을 이용해 올바르게 인식하고 부분적인 파스 트리를 정확히 완성하는 과정을 보여준다. 파싱의 결과는 불완전 입력에 대해 문법적으로 완전한 파스 트리이다.

3.1 XML DTD 에 대한 CFG 표현

불완전 입력에 대한 인식과 완성을 위해, 본 논문은 불완전 XML 입력을 LR 파서가 인식할 수 있는 CFG 형태로 표현한다. 따라서, XML DTD 의 일반적인 문법은 다음과 같은 변환 규칙에 따라 CFG 구성 요소로 바꾸어 표현 될 수 있다.

XML DTD 문법에 대한 CFG문법 G' = {E, T, P, S}

- 1. E : set of elements
- 2. T : set of tags and PCDATA/CDATA
- 3. P : set of DTD rules
- 4. S : start symbol

일반적으로, 언어를 표현하기 위한 CFG G = {N, T, P, S} 로 표현된다.[1] 본 논문은 XML DTD를 위한 CFG G'를 G' = {E, T, P, S}로 표현한다. G'과 G의 각 튜플은 일대일 매핑 된다. 따라서, G'의 E는 G의 nonterminal 집합과 같으며, G'의 T, P와 S는 G의 T, P 그리고 S와 대응되어진다. 즉, G'의 E는 non-terminal 역할을 하는 XML DTD의 각 element 집합을 나타내며, 파싱 과정에 있어서 부분 트리의 루트에 해당한다. 또한 G'의 T는 terminal 들의 집합을 의미하며, XML DTD의 각 tag와 PCDATA/CDATA에 해당한다. G'의 P는 각 XML DTD element에 적용되는 문법 규칙이다. G'의 S는 문법 표현에 있어서 시작 심벌을 의미한다. [표 1]은 CFG로 표현된 XML DTD의 예제이다.

1.<ELEMENT name (firstname?, lastname)>	
2.<ELEMENT firstname (#PCDATA)>	
3.<ELEMENT lastname (#PCDATA)>	
1. Start	-> NAME
2. NAME	-> <name> NAME_E </name>
3. NAME_E	-> LNAME
4. NAME_E	-> FNAME LNAME
5. FNAME	-> <firstname> PCDATA </firstname>
6. LNAME	-> <lastname> PCDATA </lastname>
7. PCDATA	-> id

[표 1] XML DTD 에 대해 변환된 CFG

파서는 파서가 문법적으로 유효한 상태에 대해서만 파싱을 할 수 있기 위해, 문법 G'에 대해 [알고리즘 2]를 적용해 [표 2]과 같은 확장된 파싱 테이블을 얻는다. 각 Nonterminal 심벌의 파싱 동작에 대해 파서는 서로 다른 파싱 과정을 독립적으로 수행하게 된다. 따라서, 파서는 독립된 파싱 과정에서 생성된 부분 트리를 문법적으로 유효한지를 판단하여, 유효하지 않은 부분 트리를 유도한 파싱 동작은 삭제하고 유효한 부분 트리를 유도한 파싱 동작에 대해서만 계속적인 파싱을 수행한다.

	<f>	</f>	<l>	</l>	id	<n>	</n>	\$
0						s1		
1	s7		s6					
2								acc
3							s 8	
4							r3	
5			s6					
6					s11			
7					s11			
8								r2
9							r4	
10					s13			
11		r7		r7				
12		s14						
13							r6	
14			r5					

[표 2] 문법 G'에 대한 확장된 파싱 테이블

이것은 불완전한 입력에 대해 문법적으로 모호한 파싱 동작을 해결할 수 있는 방법을 제공한다.

3.2 불완전 XML 입력 문장에 대한 인식과 완성

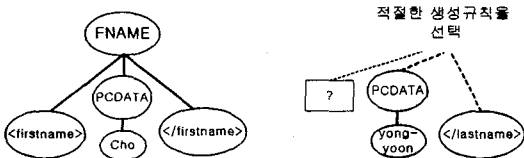
문법 G'에 대해 [그림 1]에 나타난 불완전 XML 입력 <name><firstname>yong-yeon</firstname><cho</lastname></name>의 파싱 과정은 [표 3]과 같다.

STACK	INPUT	ACTION
1 0	<n> </> cho </> yy </></n>\$	s 1
2 0 <n> 1	</> cho </> </> yy </></n>\$	s 7
3 0 <n> 1 </> 7	cho </> </> yy </></n>\$	s 11
4 0 <n> 1 </> 7 yy 11	</> </> yy </></n>\$	r 7
5 0 <n> 1 </> 7 PC	</> </> yy </></n>\$	goto 12
6 0 <n> 1 </> 7 PC 12	</> </> yy </></n>\$	s 14
7 0 <n> 1 </> 7 PC </> 14	yy </></n>\$	r 5
8 0 <n> 1 F	yy </></n>\$	goto 5
9 0 <n> 1 F 5	yy </></n>\$	s 6
10 0 <n> 1 F 5 </> 6	yy </></n>\$	s 11
11 0 <n> 1 F 5 </> 6 cho 11	</></n>\$	r 7
12 0 <n> 1 F 5 </> 6 PC	</></n>\$	goto 10
13 0 <n> 1 F 5 </> 6 PC 10	</></n>\$	s 13
14 0 <n> 1 F 5 </> 6 PC 10 </> 13	</n>\$	r 6
15 0 <n> 1 F 5 L	</n>\$	goto 9
16 0 <n> 1 F 5 L 9	</n>\$	r 4
17 0 <n> 1 NE	</n>\$	goto 3
18 0 <n> 1 NE 3	</n>\$	s 8
19 0 <n> 1 NE 3 </n> 8	\$	r 2
20 0 N	\$	goto 2
21 0 N 2	\$	accept

[표 3] 불완전 XML 입력에 대한 파싱 과정

파서는 초기 상태를 결정하기 위해서 [표 2]의 파싱 테이블을 참조한다. 파서는 파싱표에서 첫 번째 입력인 <firstname>에 해당하는 열에서 모든 시프트 동작을 찾는다. 축약 동작은 이전의 입력에 적용되는 것이므로 여기서는 시프트 동작만을 선택하여 그 상태집합을 초기 상태로 한다.

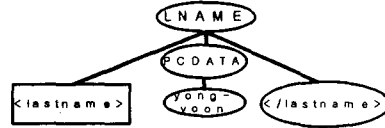
문법 G'에 대한 [그림 1]의 불완전한 XML 입력은 문법적으로 빠져있는 심벌인 <firstname>에 대한 시프트 동작이 모두 상태7로 전이된다. 따라서 초기 상태는 7이 된다. 초기 상태에서부터 상태 10과 11에서 생성된 파스 포레스트는 [그림 3]과 같다.



[그림 3] 불완전 입력에 대한 부분 트리 인식

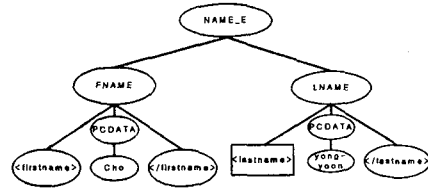
이 때 축약에 적용된 생성 규칙인 "LASTNAME → <lastname>PCDATA </lastname>"를 만족하기에는 스택의 내용이 부족하다.

따라서 부족한 기호인 '<lastname>'에 해당하는 기호노드를 생성하고 축약을 수행한다. 축약 후 상태는 goto 표를 참조하여 <lastname>에 대한 전이 상태를 결정한다.



[그림 4] 부분 트리의 노드 생성

실험에서 보이고 있는 예에서는 4와 9의 두 가지 shift 상태가 결정된다. 4와 9중 유효한 파스 트리를 선택하기 위해 두 개의 파스 트리를 모두 구해 보아야 한다. 따라서, 이 상태들에 대해 각각 파싱을 수행하기 위해 스택을 두 가지 상태로 분리한 후 계속된 파싱 과정에서 유효하지 않은 파스 트리인 9번은 삭제하고 유효한 파스 트리인 4번으로의 shift 동작을 갖는 파스 트리를 선택한다. 파싱 결과는 [그림 5]와 같다.



[그림 5] 파스 트리 인식과 완성

4. 결론

본 논문은 사용자가 프로그램을 입력하는 과정에서 불완전한 입력을 입력할 경우 발생할 수 있는 파싱 중단을 개선하기 위해 불완전한 입력을 인식하고 완성하는 파싱 방법을 제안하고 실험하였다. 제안된 파싱 방법은 부족한 심벌을 생성할 수 있으며, 그 심벌을 루트로 하는 부분 트리를 파싱 하기 위해 부가 정보를 가지는 확장 파싱 테이블을 이용한다. 따라서, 사용자는 프로그램 편집에 있어서 불완전한 입력에 대해 문법적인 오류로 인한 파싱 작업의 중단에 대한 부담을 줄일 수 있고, 중단 없는 편집을 통해 편집 효율을 증대시킬 수 있다.

참고 문헌

- [1] Aho, A.V., Sethi R., and Ullman J. D., *Compilers: Principles, Techniques and Tools*, Addison-Wesley, 1986.
- [2] Bates, J. and Lavie A., "Recognizing Substring of LR(K) Languages in Linear Time", *ACM TOPLAS*, Vol.16, No.3, pp.1051-1077, 1994.
- [3] Grune D., Jacobs C., *Parsing Techniques: A Practical Guide*, Ellis Horwood Limited, 1998.
- [4] Jalili F. and Gallier J., "Building Friendly Parsers", *Proceedings of 9th ACM POPL*, pp.196-206, 1982.
- [5] Larchéveque J.M., "Optimal Incremental Parsing", *ACM TOPLAS*, Vol.17, No.1, pp.1-15, 1995.
- [6] Nazhoor-Parshi R., "GLR parsing for ε-grammars", *Generalized LR Parsing*, pp.61-75, Kluwer Academic Publishers, 1991.
- [7] Reckers J. and Koom W., Substring parsing for arbitrary context-free grammars. *ACM SIGPLAN Notices*, 26(5), pp.59-66, 1991.
- [8] Snelting G., "How to build LR parsers which accept incomplete input", *ACM SIGPLAN Notices*, vol. 25, no. 4, pp.83-89, 1990.
- [9] Wagner A., and Graham S., "Incremental analysis of real programming languages.", *Proceedings of ACM PLDI '97*, 1997.