2000년 한국정보처리학회 추계 학술발표논문집 제7권제2호

속성문법의 점진적 평가 알고리즘

장재춘*, 이대식**, 신현덕**, 안희학**
*영동전문대학 전자계산과
**관동대학교 컴퓨터공학과
e-mail:jcjang@yeongdong.ac.kr

An Incremental Evaluation Algorithm of Attribute Grammar

Jael-Chun Jang*, Dae-Sik Lee**, Hyun-Deok Shin**, Heui-Hak Ahn**
*Dept of Computer Science, Yeongdong University
**Dept of Computer Science, Kwandong University

요약

프로그래밍 환경이 단순구조 편집환경에서 보다 복잡한 환경으로 진보되고, 언어 기반 편집 환경의 비중이 확대되면서 속성 문법의 점진적 평가의 이용이 효과적이다. 점진적 평가는 새로운 속성 트리가 기존의 속성 트리와 정확히 비교되어 기존 속성 트리를 사용하여 새로운 속성 트리를 구성한다. 본 논문에서는 Carle의 알고리즘을 분석하고 새로운 점진적 평가 알고리즘으로 재구성한다. 특히, 새로운 속성 트리 d'_{copy}의 생성 과정과, 최적화된 속성트리의 새로운 점진적 평가 알고리즘을 추가한다.

1. 서론

속성문법(attribute grammar)은 언어 변환기의 각 단계와 상호 호환적 편집환경을 정의하는데 효과적이다. 따라서, 프로그래밍 언어를 점진적 평가하는데 속성문법을 이용한다.[1-3].

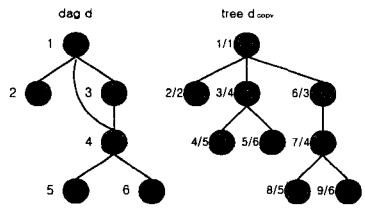
속성문법의 점진적 평가에 대한 Teitelbaum 과 Chapman의 알고리즘은 기존 속성 트리 d_{copy}를 재사용 하는데 있어 최대의 서브 트리를 사용하지 못하며 최적의 실행 시간을 초과할 수도 있다[2-3].

본 논문에서는 이러한 문제점을 개선한 Carle과 Pollock의 알고리즘을 분석하여 점진적 평가 알고리즘으로 재구성하고 최적화된 속성 트리 d'_{copy}의 구성 알고리즘을 추가하여 보완한다. 점진적 평가 알고리즘에서는 속성 트리 d_{copy}의 구성요소를 새로운 속성 트리 d'_{copy}에 적용하여 최적의 d'_{copy}를 구성하기 위한 점진적 평가 알고리즘을 제안하고자 한다[4-8].

2. Teitelbaum과 Chapman 알고리즘

2.1. 대그 d와 속성 트리 d_{copy}

그래프 형태의 표현 중 속성 트리 d_{copy}는 원시 프로그램의 자연스러운 계층구조를 묘사하고, 대그는 공통되는 부분식들이 식별되기 때문에 같은 내용을 나타내지만 속성 트리 d_{copy}보다 간결한 표현 방법이다[1]. 대그 d와 속성 트리 d_{copy}의 구성은 (그림 1)과 같다.

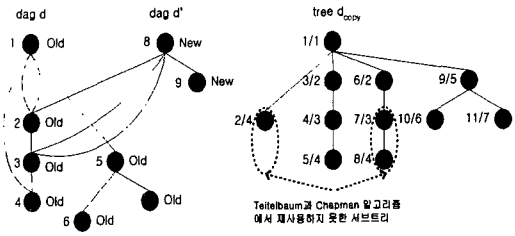


(그림 1) 대그 d와 속성 트리 d_{copy}

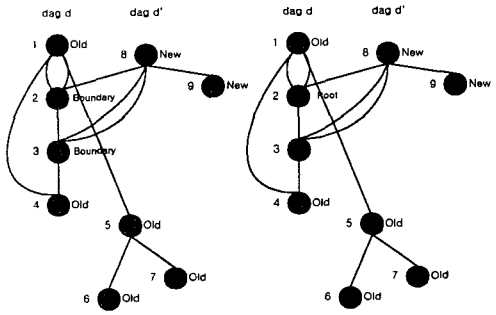
(그림 1)에서 대그 d는 노드 1, 2, 3, 4, 5, 6과 노드간의 연결가지로 구성된다. 속성 트리 d_{copy}의 노드들은 X/Y의 레이블이 정해지며 X는 속성 트리 d_{copy} 노드를 나타내는 고유 식별자이며 Y는 대그 d의 노드 식별자이다.

2.2 Teitelbaum과 Chapman의 속성 트리 d'_{copy}의 구성

Teitelbaum과 Chapman의 알고리즘은 일반적으로 트리형태가 아닌 대그형태로서 표현되는데 속성 트리 d_{copy}의 구성요소에서 새로운 속성 트리 d'_{copy}를 구성하는 과정은 (그림 2)와 같다[2-9].

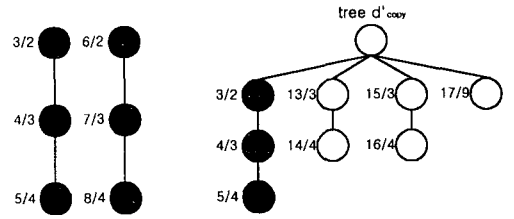


(그림 2) 대그 d와 d' 그리고 속성 트리 d_{copy}



(a) 1단계 : 대그 d와 d' (b) 2 단계 : 대그 d와 d'

<2,3/2> and <2,6/2>
(c) 3 단계 : 연관 사상 엔트리



(d) 4단계 : dcopy의 서브트리 e) 단계 5 : d'copy트리

(그림 3)의 단계 1에서 기존 대그 d에는 old, d'-d의 직접 상속자인 노드 2, 3에는 boundary, 새로운 대그 d에는 new 레이블을 부여한다. 단계 2에서 1단계의 old, new 레이블은 같고 d-d'와 d'-d의 직접 상속자인 boundary 노드 2의 레이블을 루트(root)로 바꾼다. 단계 3에서 대그 d와 대그 d'가 연결되는 루트 레이블을 (그림 2)에 있는 속성 트리 d_{copy}를 이용하여 연관 사상 엔트리로 구성한다. 단계 4에서 연관 사상 엔트리를 속성 트리 d_{copy}의 서브 트리로 나타낸다. 단계 5에서 속성 트리 d_{copy}의 서브 트리를 사용해 속성 트리 d'_{copy}를 구성한다. 채워진 원은 기존의 속성 트리 노드이고, 채워지지 않은 노드는 새로운 속성 트리 노드이다.

Teitelbaum과 Chapman의 알고리즘은 속성 트리 d_{copy}에 있는 노드 2/4의 서브 트리와 노드 7/3의 서브 트리((그림 2)의 점선부분)를 재 사용하지 못하므로 최적의 실행시간인 $O(|d_{copy} - d'_{copy}| + |d'_{copy} - d_{copy}|)$를 초과할 수 있다[2-8].

3. 속성 문법을 위한 점진적 평가 알고리즘

본 논문의 점진적 평가 알고리즘은 기존 대그 d의 속성 트리 d_{copy}와 새로운 대그 d'의 속성 트리 d'_{copy}를 비교하여 기존 속성 트리 d_{copy}의 최대 구성요소로부터 새로운 속성 트리 d'_{copy}를 구성한다.

점진적 평가 알고리즘의 최적 실행시간은 $|d_{copy} - d'_{copy}| + |d'_{copy} - d_{copy}| = |d_{copy}| - |d_{copy} \cap d'_{copy}| + |d'_{copy}| - |d_{copy} \cap d'_{copy}| = |d_{copy}| + |d'_{copy}| - 2 * |d_{copy} \cap d'_{copy}|$ 이다. 따라서 d_{copy}에서 나온 최대 서브 트리를 재사용하여 $|d_{copy} \cap d'_{copy}|$를 최대화시킴으로써 최적의 속성 트리 d'_{copy}를 구성한다. PROVIDED 집합은 속성 트리 d_{copy}에서 사용된 대그 노드의 집합이며, NEEDED 집합은 속성 트리 d'_{copy}를 구성할 때 새로이 추가되는 노드의 직접 상속자 집합이다.

3.1 초기화 단계 알고리즘

```
Procedure Initialize(dagnode, Tree_Dcopy)
begin
        Dag_Buffer[] := Topol(dagnode)
        ROOT_PROV[] := Tree_Dcopy;
        Tree_D'copy[] := Create ⊤();
        ROOT_NEED := ⊤
end;
```

(알고리즘 1) 초기화 단계

(알고리즘 1)에서는 대그 d와 d'의 모든 대그 노드와 대그 d에 대한 속성 트리 d_{copy}를 입력받는다. 입력받은 d_{copy}는 속성 트리 d'_{copy}를 구성한다. 또한, 속성 트리 d'_{copy}를 구성하는 임시 루트 노드 ⊤를 생성하고, 이 루트 노드를 NEEDED 집합의 초기 값으로 설정한다.

3.2 유용한 대그 노드와 새로운 노드 추가 알고리즘

```
Procedure Com_Prov_Need(Dag_Buffer[])
begin
  for dag := Dag_Buffer[Min] to Dag_Buffer[Max]
  do begin
    Need[dag] := NEEDED(dag);
    Prov[dag] := ROOT_PROV[dag]
  end;
  if (Need == !EMPTY) then
    for dag := Dag_Buffer[Min] to Dag_Buffer[Max]
    do
      Dag_NEED[dag] := <Need[dag],i>;
    if (Prov == !EMPTY) then
      for dag := Dag_Buffer[Min] to Dag_Buffer[Max]
      do
        Dag_PROV[dag] := Prov[dag]
end;
```

(알고리즘 2) 유용한 대그 노드와 새로운 노드 추가

(알고리즘 2)에서는 대그 노드를 입력받아서 각 대그 노드에 대한 NEEDED 집합과 PROVIDED 집합을 계산한다. 계산된 NEEDED와 PROVIDED가 공집합이 아니면 즉, 대그 d와 d'가 서로 연관되어 있고 속성 트리 d_{copy}의 기존 노드가 속성 트리 d'_{copy}를 구성하는데 유용하다면 Need와 Prov를 각 대그 노드에 대한 NEEDED 집합과 PROVIDED 집합에 추가한다.

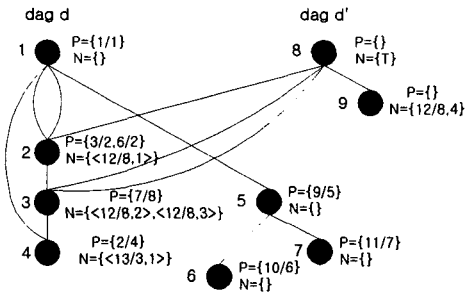
(그림 2)의 대그 노드에 대한 NEEDED 집합 Dag_NEED와 PROVIDED 집합 Dag_PROV를 (알고리즘 2)에 의해 계산한 결과는 (표 1)과 같다.

(표 1) NEEDED 집합과 PROVIDED 집합의 계산결과

대그노드	Dag_NEED	Dag_PROV
1	∅	{1/1}
2	{<12/8,1>}	{3/2, 6/2}
3	{<12/8,2>, <12/8,3>}	{7/3}
4	{<13/3,1>}	{2/4}
5	∅	{9/5}

대그노드	Dag_NEED	Dag_PROV
6	∅	{10/6}
7	∅	{11/7}
8	{⊤}	∅
9	{<12/8,4>}	∅

(표 1)의 결과를 그림으로 나타내면 (그림 5)와 같다.



(그림 5) Dag_NEED와 Dag_PROV

(그림 5)에서 P는 Dag_PROV의 값을 나타내며 N은 Dag_NEED의 값을 나타낸다. 또한, 대그 노드 2의 P값은 대그 노드 2가 속성 트리 d_{copy}의 노드 3/2와 6/2로 사용되고 있음을 나타내는 것이며, N값은 대그 노드 2가 속성 트리 d'_{copy}에 새로 추가될 노드 12/8의 첫 번째 직접 상속자임을 나타낸다. 또한, 대그 노드 8의 N 값이 {⊤}인 것은 (알고리즘 1) 초기화 단계에서 ROOT_NEED := ⊤로 초기화하였기 때문이다.

(그림 5)의 결과에서 N이 공집합이면 속성 트리 d'_{copy}를 구성하는데 필요 없는 노드이다. 또한 P의 값이 공집합이면 속성 트리 d'_{copy}에 새로운 노드로 삽입되어야 함을 의미한다.

3.3 최적화된 d'_{copy} 구성 알고리즘

```
Procedure Make_D'copy (Dag_Buffer[], Dag_NEED[],
Dag_PROV[], Tree_D'copy[])
begin
  Dag_Buffer[] := Topol(d'_root(dagnode));
  for dag := Dag_Buffer[Min] to Dag_Buffer[Max] do
  begin
    if (Dag_PROV[dag] == EMPTY) then
    begin
      New_Dag := NEW(dag);
      Tree_D'copy[Dag_NEED[dag]] := New_Dag
    end;
    if (Dag_PROV[dag] == !EMPTY) then
     Tree_D'copy[Dag_NEED[dag]] := Dag_PROV[dag]
     Remove ⊤(Tree_D'copy[])
end;
```

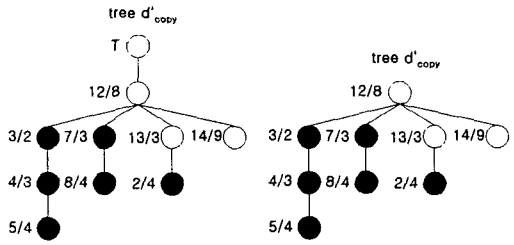
(알고리즘 3) 최적화된 d'_{copy} 구성

(알고리즘 3)에서는 (알고리즘 2)에서 계산된

Dag_NEED와 Dag_PROV 그리고 (알고리즘 1)에서 초기화된 Tree_D'copy와 대그 노드를 입력받는다. 입력받은 대그 노드는 대그 d'의 루트 노드를 시작으로 위상 탐색 순서대로 정렬된다.

각 대그 노드에 대한 Dag_PROV가 공집합이면 새로운 노드를 생성하여 Tree_D'copy의 Dag_NEED에 추가하고, 공집합이 아니면 해당 대그 노드에 대한 Dag_PROV를 Tree_D'copy의 Dag_NEED에 삽입한다.

(그림 5)의 결과에 (알고리즘 3)을 적용하면 (그림 6)과 같다.



(그림 6) 최적화된 d'copy 구성

(그림 6) (a)는 초기화 단계에서 Tree_D'copy[] := Create T()에 의해 추가된 임시 루트 노드 T에 대그 d'의 노드가 추가된 것이고, (그림 6) (b)는 임시 루트 노드 T를 제거한 것이다.

Teitelbaum과 Chapman의 d'_{copy} 구성과정의 결과인 (그림 3) (e)와 본 논문에서 제안한 (그림 6) (b)를 비교하면 더 많은 d_{copy}의 서브 트리를 재사용 할 수 있다. 따라서, 점진적 평가 알고리즘에 의해 최적화가 향상 되었음을 알 수 있다.

4. 결론

속성 문법의 점진적 평가를 효율적으로 하기 위해 점진적 평가 알고리즘을 이용하였으며 점진적 평가 알고리즘에서는 새로운 입력 트리가 기존 입력 트리와 정확히 비교된다. 따라서, 새로운 트리를 구성할 때 기존 속성 트리의 재 사용 가능한 모든 서브 트리를 재 사용한다.

기존의 Teitelbaum과 Chapman의 알고리즘은 속성 트리 d_{copy}의 최대 서브 트리를 재사용하는 점진적 평가 알고리즘을 만들어 내지 못하였으며 최적의 실행 시간을 초과할 수도 있다. 이러한 문제점을 개선한 Carle과 Pollock의 알고리즘을 분석하여 점진적 평가 알고리즘으로 재 구성하고, 알고리즘의 동작과정을 구체적으로 제시하여 새로운 속성 트리 d'_{copy}의 생성 과정을 나타내었으며, 속성 트리 d_{copy}의 구성요소를 새로운 속성 트리 d'_{copy}에 적용하여 최적화된 속성트리 d'_{copy}의 점진적 평가 알고리즘을 새로이 추가하였다.

점진적 평가 알고리즘은 변화된 속성과 복잡한 재 계산을 피하고 기존의 속성 트리를 재 사용하는 공간 최적화로의 응용이 가능해야 한다.

참 고 문 헌

[1] Aho, A. V., Sethi, R., and Ullman. J. D. "Compilers Principles, Techniques, and Tools" , Addison-wesley publishing Co., 1986.

[2] Teitelbaum, T., and Demers, A. "Incremental context-dependent analysis for language-based editors", ACM TOPLAS, Vol. 5, No. 3, pp 449-477, 1983.

[3] Teitelbaum, T. and Chamman, R. "Higher-order attribute grammars and editing environments", In processing of the SIGPLAN'90 Symposium on Programming Language Design and Implementation. ACM, New York, pp 197-208, 1990.

[4] Carle, A. and Pollock, L. "Incremental evaluators for hierachical attribute grammars", Tech. Rep. TR90-103 Rice Univ., Houston, Tex. Jan, 1990.

[5] Carle, A. "Hierarchical attribute grammars: Dialects, applications and evaluation algorithms" Ph.D. thesis, Dept. of Computer Science, Rice Univ., Houston, tex, 1992.

[6] Carle, A. and Pollock, L. "A context-based incremental evaluators for hierachical attribute grammars", J. Program. Lang, No 3, pp 1-29, 1995a.

[7] Carle, A. and Pollock, L. "Maching-based incremental evaluators for hierachical attribute grammar dialects", ACM TOPLAS, Vol. 17, No. 2, pp 394-429, 1995b.

[8] Carle, A. and Pollock, L. "On the Optimality of Change Propagation for Incremental Evaluation of Hierarchical Attribute Grammars", ACM TOPLAS, Vol. 18, No. 1, pp 16-29, 1996.