점진적 속성문법을 위한 효과적인 최적화 알고리즘에 관한 연구

장 재 춘† · 안 희 학††

요 약

복잡한 언어 처리에 점진적 속성 문법을 적용하기 위해서는 최적화 알고리즘을 사용하는 것이 효과적이다. 점진적 속성문법의 최적화 알고리즘에서는 새로운 입력 속성 트리가 기존 입력 속성 트리와 정확히 비교되어서 새로운 속성 트리를 구성할 때 기존 속성 트리의 어떤 서브 트리를 사용해야 하는가를 결정한다. 본 논문에서는 Carle과 Pollock에 의해 제안된 알고리즘을 분석하여 효과적인 최적화 알고리즘으로 재구성하고, 새로운 속성 트리 d'copy의 생성 과정과, 최적화된 속성트리 d'copy의 새로운 최적화 알고리즘을 추가하였다. 이 논문에서 제안한 매칭 알고리즘의 성능평가를 통하여 기존의 알고리즘 보다 제안한 최적화 알고리즘의 실행 시간을 약 19.5% 향상시킬 수 있었다.

A study on the effectively optimized algorithm for an incremental attribute grammar

Jae Chun Jang† · Heui Hak Ahn††

ABSTRACT

The effective way to apply incremental attribute grammar to a complex language process is the use of optimized algorithm. In optimized algorithm for incremental attribute grammar, the new input attribute tree should be exactly compared with the previous input attribute tree, in order to determine which subtrees from the old should be used in constructing the new one. In this paper the new optimized algorithm was reconstructed by analyzing the algorithm suggested by Carle and Pollock, and a generation process of new attribute tree d'copy was added. Through the performance evaluation for the suggested matching algorithm, the run time is approximately improved by 19.5%, compared to the result of existing algorithm.

키워드 : 점진적속성문법(incremental attribute grammar), 최적화(optimized), 알고리즘(algorithm), 속성트리(attribute tree)

1. 서 론

속성 문법은 모든 방향의 자료전달을 표현할 수 있으며, 비 단말간의 자료전달이 명확히 기술된다. 이러한 자료를 속성이라 하고 자료의 흐름은 속성의 전달에 의해서 이루어진다. 그러나 모든 방향으로의 속성전달이 가능하기 때문에 상황에 따라 순환구조를 이루기도 하고 속성전달이 파싱 방향과 관계없이 전달되므로 한번 이상의 파싱이 필요한 경우도 발생한다. 이러한 문제 때문에 속성 평가라는 방식이 도입되었고 여러 가지의 속성 평가 방법들이 제안되었다[1-3].

속성에 할당된 새로운 조건을 통해 평가를 수행할 때 이미 산출된 부분을 재사용하기 위해서는 새로운 평가방법이 필요하다.

점진적 속성문법에서의 효과적인 최적화 알고리즘은 속성화된 조건값을 트리 형태가 아닌 대그(dag) 형태로 표현하여 대그 d를 새로운 대그 d'에 연결하고, 속성 트리 d_{copy}를 새로운 속성 트리 d'_{copy}에 연결하여 평가한다.

기존의 Teitelbaum 과 Chapman의 알고리즘은 기존 속성 트리 d_{copy}의 최대 서브 트리를 재사용하는 최적화된 알고리즘을 만들어 내지 못하며 최적의 실행 시간인 $O(|d_{copy} - d'_{copy}| + |d'_{copy} - d_{copy}|)$를 초과할 수도 있다. 따라서, 이런 문제점을 개선한 알고리즘이 Carle과 Pollock에 의해 제안되었다[2-4].

따라서 본 논문에서는 Carle과 Pollock의 알고리즘을 분석하여 최적화된 알고리즘으로 재구성하여 새로운 속성 트리 d'_{copy}의 생성 과정을 나타내고, 속성 트리 d_{copy}의 구성요소를 새로운 속성 트리 d'_{copy}에 적용하여 최적의 d'_{copy}를 구성하기 위한 최적화 알고리즘을 제안하고 평가한다.

† 종신회원 : 영동전문대학 전자계산과 교수
†† 종신회원 : 관동대학교 컴퓨터공학과 교수
논문접수 : 2000년 8월 29일, 심사완료 : 2001년 6월 15일

2. 관련 연구

2.1 속성문법과 점진적 속성평가

속성 문법은 문법에서 각각의 기호와 관련된 속성 집합을 갖는 문맥 자유 문법이다. 문맥 자유 문법은 4개의 요소(N, T, P, S)로 구성된다. 여기서, N은 비단말 기호의 집합이고, T는 단말 기호의 집합으로 ($N \bigcap T = \varnothing$), S는 시작 기호로 $S \in N$이며, P는 생성규칙의 집합이다. 생성규칙 P에는 생성규칙에 관계되는 속성 값들을 계산해 주기 위한 의미 방정식과 출력 속성 및 입력 속성들이 추가된다. 여기서, 입력 속성은 생성규칙의 우측에 있는 문법 기호에 속하는 속성이고 출력 속성은 생성규칙의 좌측에 있는 비단말 기호의 속성이다. 의미 규칙에 의해 생성된 종속 함수는 종속 그래프로 표현될 수 있으며 종속 그래프에서 파스 트리의 속성 평가 순서가 유도될 수 있다[1, 8].

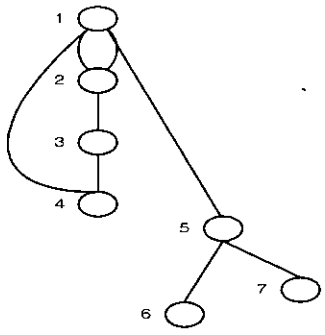
점진적 속성 평가는 구문 트리에서 임의의 속성에 변화가 일어났을 때 그 임의의 속성으로부터 영향을 받은 속성 값들을 점진적으로 재평가하는 것이다. 즉, 프로그램의 어느 부분을 수정했을 때 프로그램 전체를 처음부터 평가하지 않고 변경된 부분 즉, 삽입, 삭제, 대체 등이 이루어진 부분에 의하여 영향을 받는 최소한의 부분만 재평가하는 것을 의미한다. 구문 트리상에서 임의의 한 노드의 속성 값에 변화가 발생했을 때 영향을 받는 속성 값들은 점진적으로 재평가되고 이때, 재평가의 범위를 최소화함으로서 작업의 효율을 높인다.

2.2 Teitelbaum과 Chapman 알고리즘

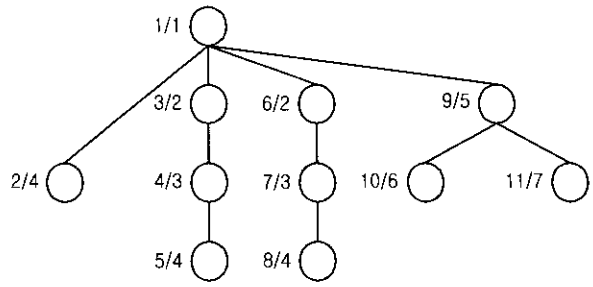
2.2.1 대그 d와 속성 트리 d_{copy}

Teitelbaum과 Chapman 은 하나의 모듈 인스턴스에 의해 속성화된 가장 효율적인 조건 값을 트리 형태가 아닌 대그 형태로 표현한다.

그래프 형태의 표현 중 대그는 공통되는 부분식들이 식별되기 때문에 같은 내용을 나타내지만 속성 트리 d_{copy}는 원시 프로그램의 자연스러운 계층 구조를 묘사하고, 대그는 속성 트리 d_{copy}보다 간결한 표현 방법이다.



(그림 1) 대그 d



(그림 2) dcopy 트리

대그 d의 구성은 (그림 1)과 같으며 대그 d는 노드 1, 2, 3, 4, 5, 6, 7과 노드간의 연결 가지로 구성된다. 대그 d의 구성요소들은 대그 d'를 구성할 때 사용한다.

대그 d를 d_{copy} 트리로 구성하면 (그림 2)와 같으며, 대그 노드는 각각 하나의 고유한 이름이 부여된다. d_{copy} 트리의 각 노드들은 X/Y의 라벨이 정해지며 X는 속성 트리 노드 자신을 나타내는 고유 식별자이며 Y는 트리 노드에 의해 표현되는 대그 d의 노드와 연계된 고유한 식별자이다.

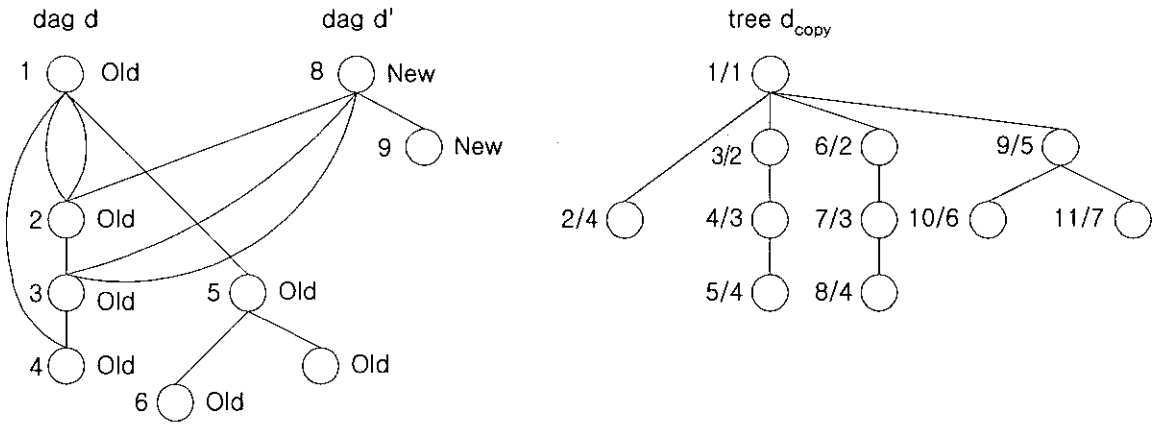
2.2.2 속성 트리 d'_{copy}의 구성

Teitelbaum과 Chapman의 알고리즘은 속성 트리 d_{copy}의 구성요소에서 새로운 속성 트리 d'_{copy}를 생성한다[5-7].

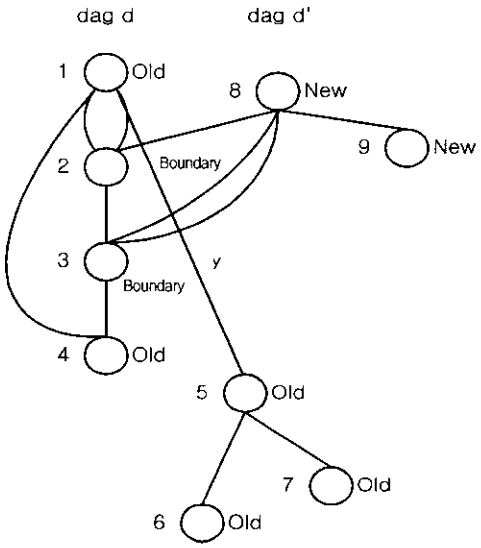
(알고리즘 1)은 대그 d와 d'속성 트리 d_{copy}를 입력받아 새로운 속성 트리 d'_{copy}를 생성하는 Teitelbaum과 Chapman의 알고리즘이다.

```
Procedure Make_D'copy(Dag_D[], Dag_D'[], Tree_Dcopy[])
  begin
    for dag1 := Dag_D[Min] to Dag_D[Max] do
      begin
        Dag_Value[] := Old;
      • /*입력받은 대그 노드에 Old 레이블 표시*/
        for dag2 := Dag_D'[Min] to Dag_D'[Max] do
          begin
          New_Dag_Value(Dag_D'[]) := New;
          /* 새로운 대그 노드에 New 레이블 표시 */
          Desc_Dag_Value(Dag_D'[]) := Boundary;
          /* New 레이블의 직접 상속 노드에 Boundary 레이블
             표시 */
          Desc_Dag_Value(Dag_D[], Dag_D'[]) := Root
          /* Boundary 레이블 중 대그 (d-d')와 대그 (d'-d)의
             직접 상속 노드에 Root 레이블 표시 */
             end ;
          if (Dag_D'[] = Root) then
          /* Root 레이블을 가진 대그 노드 */
             ASS_entry(r, f) := Tree_Dcopy[];
             /* 연관 사상 엔트리 구성 */
             SUB_Tree_Dcopy[] := ASS_entry(r, f)
             /* 연관 사상 엔트리를 서브 트리로 구성 */
        end ;
      Tree_D'copy[] := Dag_D'[];
      /* New 레이블로 구성된 트리 D'copy */
      Tree_D'copy[] := SUB_Tree_Dcopy[]
      /* 트리 Dcopy에서 재 사용되는 서브트리 */
  end ;
```

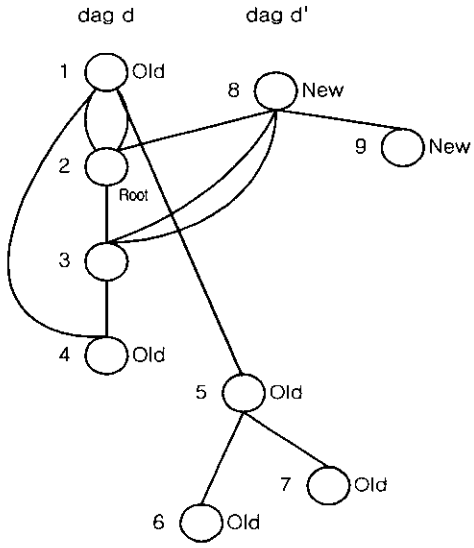
(알고리즘 1) Teitelbaum과 Chapman의 알고리즘



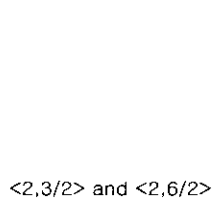
(그림 3) 대그 d와 d' 그리고 속성 트리 dcopy



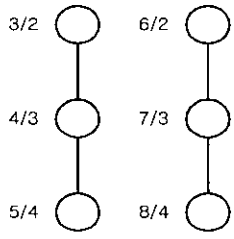
(a) 1단계 : 대그 d와 d'



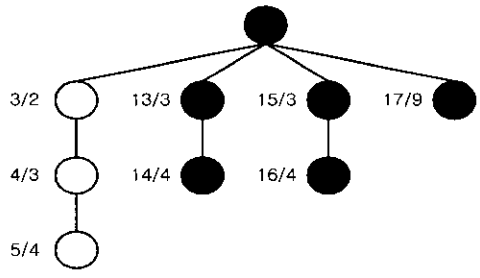
(b) 2단계 : 대그 d와 d'



(c) 3단계 : 연관 사상 엔트리



(d) 4단계 : d_{copy}의 서부트리



(e) 5단계 : d'_{copy} 트리

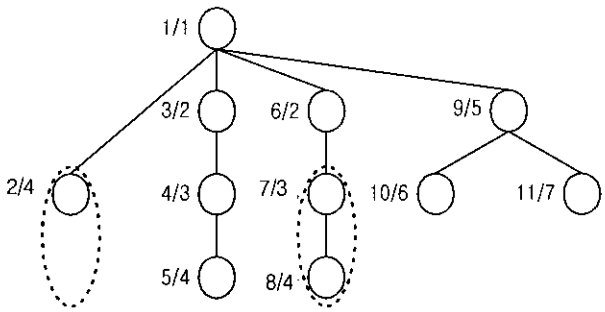
(그림 4) Teitelbaum과 Chapman의 속성 트리 d'copy 구성과정

Teitelbaum과 Chapman 알고리즘의 점진적 평가는 대그 d를 새로운 대그 d'에 연결한다. 대그 d와 d'의 연결 및 속성트리 d'_{copy}의 구성과정은 (그림 3)과 같다.

(그림 4)의 1단계는 기존 대그 d에는 old 라벨을 부여하고, d'-d의 직접 상속자인 노드 2, 3은 boundary를, 새로운 대그 d는 new 라벨을 부여한다. 2단계는 1단계의 old, new 라벨은 같고 d-d'와 d'-d의 직접 상속자인 boundary 노드 2의 라벨을 root로 바꾼다. 3단계는 대그 d와 대그 d'가 연결되는 root 라벨을 (그림 3)에 있는 속성 트리 d_{copy}에서 연관 사상 엔트리로 구성한다. 4단계는 연관 사상 엔트리를 속성 트리 d_{copy}의 서브 트리로 나타낸다. 5단계는 속성 트리 d_{copy}의 서브 트리를 사용해 속성 트리 d'_{copy}를 구성한

다. 또한 채워진 원은 새로운 속성 트리 노드이고, 채워지지 않은 노드는 기존의 속성 트리 노드이다.

Teitelbaum과 Chapman의 알고리즘은 속성 트리 d_{copy}에 있는 노드 2/4의 서브 트리와 노드 7/3의 서브 트리((그림 5)의 점선부분)를 재사용하지 못한다.



(그림 5) 재사용하지 못한 서브

(그림 5)에서 Teitelbaum과 Chapman의 알고리즘은 서브 트리 7/3과 서브 트리 2/4에 의해 고정된 d_{copy}에서 2개 서브 트리를 재사용하지 못하는 것은 첫 번째로 d'_{copy}를 구성할 때 루트라는 라벨을 붙인 $d' \cap d$의 서브 트리를 나타내는 d_{copy}의 하부 트리만 d'_{copy}를 구성할 때 사용되어야 한다는 것과 두 번째로 $d' \cap d$ 의 한 노드가 $d'-d$의 어떤 노드의 직접적 피상속자이며 $d-d'$의 어떤 노드의 직접적 피상속자라면 Root라는 라벨을 붙이는 전제조건으로 한다.

이 조건을 보면 서브 트리 2/4에 의해 고정된 d_{copy}의 서브 트리는 d'_{copy}를 구성할 때 사용될 수 있지만, 대그 노드 4 ($d-d'$)와 ($d'-d$)의 직접적 피상속자가 아니다.

서브 트리 7/3에 의해 고정된 d_{copy}의 서브 트리는 d'_{copy}를 구성할 때 사용될 수 있지만, 대그 노드4 ($d'-d$)의 직접적 피상속자이지만, ($d-d'$)의 직접적 피상속자가 아니다. 그러므로, 서브 트리 7/3과 서브 트리 2/4는 재사용하지 못한다.

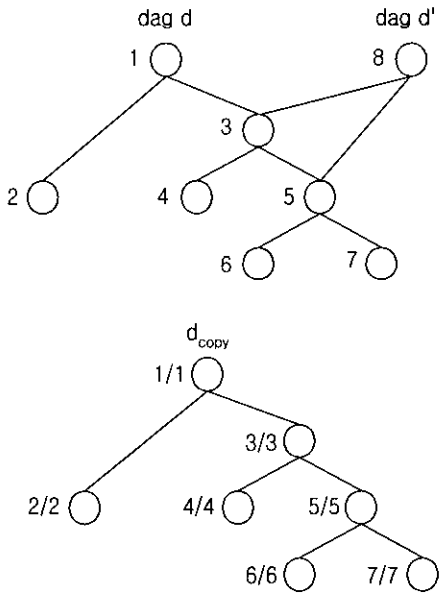
따라서 Teitelbaum 과 Chapman의 알고리즘은 최적화된 점진적인 매칭 알고리즘을 만들어내지 못하며 실행 시간인 $O(|d_{copy} - d'_{copy}| + |d'_{copy} - d_{copy}|)$을 초과할 수 있다는 문제점이 나타난다.

3. 점진적 속성 문법을 위한 효과적인 최적화 알고리즘

본 논문의 기본 알고리즘은 위상 매칭(TM : Topological Matching) 알고리즘이다. 위상 매칭 알고리즘은 기존 속성 트리 d_{copy}의 최대 구성요소로부터 새로운 속성 트리 d'_{copy}를 구성한다. 효과적인 최적화 알고리즘은 기존 대그 d와 새로운 대그 d'와 대그 d에 대한 속성 트리 d_{copy}를 입력받아 속성 트리 d'_{copy}를 구성한다.

최적의 실행시간은 $|d_{copy} - d'_{copy}| = |d_{copy}| - |d_{copy} \cap d'_{copy}|$ 와 $|d'_{copy} - d_{copy}| = |d'_{copy}| - |d_{copy} \cap d'_{copy}|$이므로 $|d_{copy} - d'_{copy}| + |d'_{copy} - d_{copy}| = |d_{copy}| - |d_{copy} \cap d'_{copy}| + |d'_{copy}| - |d_{copy} \cap d'_{copy}| = |d_{copy}| + |d'_{copy}| - 2 * |d_{copy} \cap d'_{copy}|$ 이다. 따라서 d_{copy}에서 나온 최대 서브 트리를 재사용하여 $|d_{copy} \cap d'_{copy}|$를 최대화시킴으로써 최적의 속성 트리 d'_{copy}를 구성한다. 속성 트리 d'_{copy}를 구성하기 위해 각 대그 노드에 대한 PROVIDED 집합과 NEEDED 집합을 이용한다. 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. PROVIDED는 속성 트리 d_{copy}를 구성하는 각 서브 트리들의 루트 노드이며, 이것은 속성 트리 d'_{copy}를 구성할 때 속성 트리 d_{copy}의 구성요소를 서브 트리 단위로 사용할 수 있게 한다. NEEDED는 속성 트리 d'_{copy}를 구성하는 새로운 노드에 속성 트리 d_{copy}의 기존 서브 트리를 삽입하거나, 새로 삽입되어야 할 노드의 레벨과 위치를 결정한다.

기존 대그 d와 새로운 대그 d' 그리고 대그 d에 대한 속성 트리 d_{copy}를 (그림 6)과 같이 구성한다.



(그림 6) 대그 d와 d' 그리고 d_{copy} 트리

3.1 초기화 단계 알고리즘

(알고리즘 2)에서는 대그 d와 d'의 모든 대그 노드와 대그 d에 대한 속성 트리 d_{copy}를 입력받는다. 입력받은 대그 노드는 위상 탐색 순서대로 정렬한다. 기존 속성 트리 d_{copy}는 속성 트리 d'_{copy}를 구성하는데 유용하므로 속성 트리 d_{copy}를 PROVIDED 집합에 삽입한다. 또한, 속성 트리 d'_{copy}를 구성하는 임시 루트 노드 ⊤를 생성하고, 이 루트 노드를 NEEDED 집합의 초기 값으로 설정한다.

```
Procedure Initialize(dagnode, Tree_Dcopy)
begin
    Dag_Buffer[] := Topol(dagnode);
    /* 입력받은 대그 노드를 위상 정렬하여 버퍼에 삽입 */
    ROOT_PROV[] := Tree_Dcopy;
    /* 입력받은 속성 트리 Dcopy를 ROOT_PROV 집합에 삽입 */
    Tree_D'copy[] := Create T();
    /* 트리 D'copy의 임시 루트 노드 생성 */
    ROOT_NEED := T
    /* NEEDED 집합의 초기 값 설정 */
end;
```

(알고리즘 2) 초기화 단계 알고리즘

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
    /* 위상 정렬한 대그 노드로 Need와 Prov 집합을 구성 */
      begin
        Need[dag] := NEEDED(dag);
        Prov[dag] := ROOT_PROV[dag]
      end
    if (Need <> EMPTY) then
    /* Need 노드 구성이 empty이면 Dag_NEED 집합 구성 */
        for dag := Dag_Buffer[Min] to Dag_Buffer[Max] do
          Dag_NEED[dag] := <Need[dag],i> ;
    if (Prov <> EMPTY) then
    /* Prov 노드 구성이 empty가 아니면 Dag_PROV 집합 구성 */
        for dag := Dag_Buffer[Min] to Dag_Buffer[Max] do
          Dag_PROV[dag] := Prov[dag]
 end ;
```

(알고리즘 3) 유용한 대그 노드와 새로운 노드 추가

(알고리즘 3)에서는 위상 탐색 순서대로 정렬된 대그 노드를 입력받아서 각 대그 노드에 대한 NEEDED 집합과 PROVIDED 집합을 계산한다. 계산된 NEEDED와 PROVIDED가 공집합이 아니면 즉, 대그 d와 d'가 서로 연관되어있고 속성 트리 dcopy의 기존 노드가 속성 트리 d'copy를 구성하는데 유용하다면 Need와 Prov를 각 대그 노드에 대한 NEEDED 집합과 PROVIDED 집합에 추가한다.

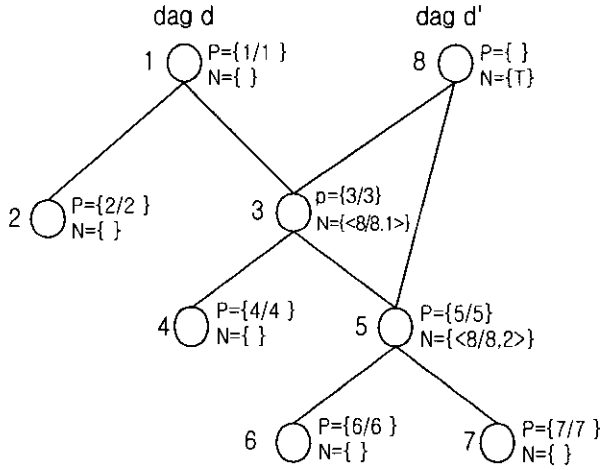
예제에 대한 NEEDED 집합 Dag_NEED와 PROVIDED 집합 Dag_PROV를 (알고리즘 3)에 의해 계산한 결과는 <표 1>과 같다.

<표 1> NEEDED 집합과 PROVIDED 집합의 계산결과

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

<표 1>의 결과를 그림으로 나타내면 (그림 7)과 같다.

(그림 7)에서 P는 Dag_PROV의 값을 나타내며 N은 Dag_NEED의 값을 나타낸다. 또한, 대그 노드 3의 P값은 대그 노드 3이 속성 트리 d_{copy}의 노드 3/3으로 사용되고 있음을 나타내는 것이며, N값은 대그 노드 3이 속성 트리 d'_{copy}에 새로 추가될 노드 8/8의 첫 번째 직접 상속자임을 나타낸다. 또한, 대그 노드 8의 N 값이 {T}인 것은 (알고리즘 1) 초기화 단계에서 ROOT_NEED : = T로 초기화하였기 때문이다.



(그림 7) Dag_NEED와 Dag_PROV

이 결과에서 N의 값이 공집합인 경우는 해당 대그 노드가 속성 트리 d'_{copy}를 구성하는데 필요 없는 노드임을 나타내며, P의 값이 공집합인 경우는 해당 대그 노드가 속성 트리 d'_{copy}에 새로운 노드로 삽입되어야 함을 의미한다.

3.3 최적화된 d'_{copy} 구성 알고리즘

(알고리즘 4)에서는 (알고리즘 3)에서 계산된 Dag_NEED와 Dag_PROV 그리고 (알고리즘 2)에서 초기화된 Tree_D'copy와 대그 노드를 입력받는다. 입력받은 대그 노드는 대그 d'의 루트 노드를 시작으로 위상 탐색 순서대로 정렬된다.

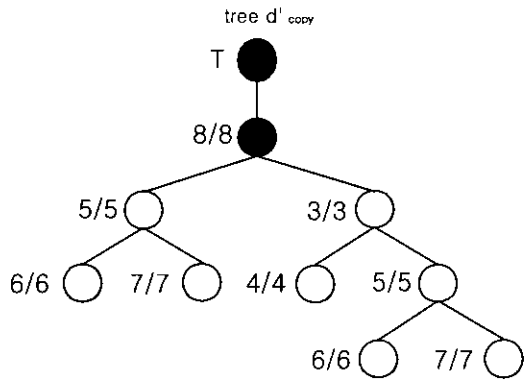
각 대그 노드에 대한 Dag_PROV가 공집합이면 새로운 노드를 생성하여 Tree_D'copy의 Dag_NEED에 추가하고, 공집합이 아니면 해당 대그 노드에 대한 Dag_PROV를 Tree_D'copy의 Dag_NEED에 삽입한다.

```
Procedure Make_D'copy(Dag_Buffer[], Dag_NEED[], Dag_PROV[],
Tree_D'copy[])    /* 속성 트리 d'copy 구성 */
 begin
     Dag_Buffer[] := Topol(d'_root(dagnode)) ;
    /* 새로운 대그 노드를 위상 정렬하여 Dag_Buffer에 삽입 */
     for dag := Dag_Buffer[Min] to Dag_Buffer[Max] do
    /* 새로운 대그 노드로 속성 트리 d'copy 구성 */
       begin
          if (Dag_PROV[dag] = EMPTY) then
         /* Dag_PROV 노드의 구성이 empty이면 새로운 대그
            노드로 속성 트리 d'copy 구성 */
          begin
              New_Dag := NEW(dag);
```

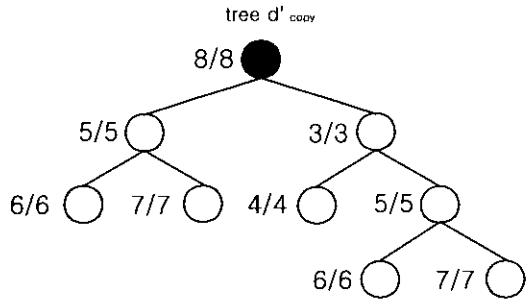
```
            Tree_D'copy[Dag_NEED[dag]] := New_Dag
        end
    if (Dag_NEED[dag] <> EMPTY) then
    /* Dag_NEED 노드의 구성이 empty가 아니면 Dag_
       PROV의 노드로 속성 트리 d'copy 구성 */
       Tree_D'copy[Dag_NEED[dag]] := Dag_PROV[dag] ;
    end;
      Remove ⊤(Tree_D'copy[])
      /* 임시 루트 노드 ⊤를 제거하여 최적화 속성 트리
         d'copy 구성 */
end;
```

(알고리즘 4) 최적화된 속성 트리 d'_{copy} 구성

(그림 7)의 결과에 (알고리즘 4)를 적용하면 (그림 8)과 같다.



(a)

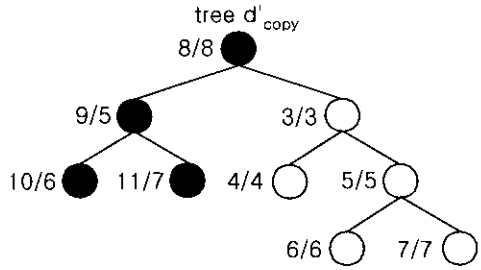


(b)

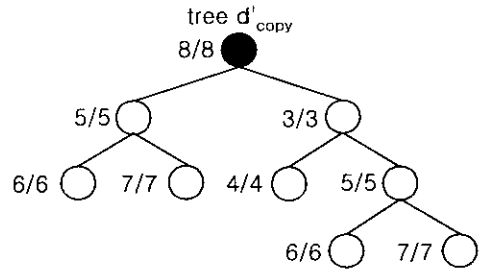
(그림 8) 최적화된 d'_{copy} 구성

(그림 8) (a)는 초기화 단계에서 Tree_D'copy[] := Create ⊤()에 의해 추가된 임시 루트 노드 ⊤에 대그 d'의 노드가 추가된 것이고, (그림 8) (b)는 임시 루트 노드 ⊤를 제거한 것이다.

(그림 9)에서 채워진 원은 새로운 속성 트리 노드이고, 채워지지 않은 노드는 기존의 속성 트리 노드를 재사용 한 것으로 (a)는 Teitelbaum과 Chapman의 알고리즘을 적용한 결과이며, (b)는 최적화 알고리즘이 적용된 결과로서 더 많은 d_{copy}의 서브 트리를 재사용 할 수 있으므로 최적화가 향상되었음을 알 수 있다.



(a) 기존 알고리즘의 수행결과

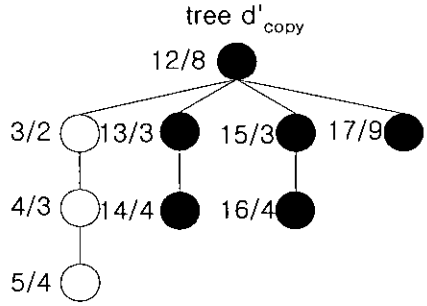


(b) 최적화 알고리즘의 수행결과

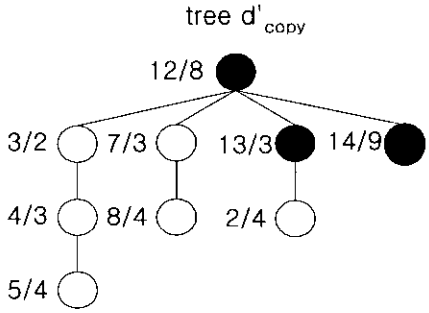
(그림 9) 알고리즘 수행결과의 비교

3. 성능평가

이 논문에서의 최적화 알고리즘과 기존의 평가 알고리즘을 비교하기 위하여 SUN ENTERPRISE 3500 시스템을 사용하여 평가하였다.



(a) 기존 알고리즘의 수행결과



(b) 최적화알고리즘의 수행결과

(그림 10) 입력 대그 1의 수행 결과

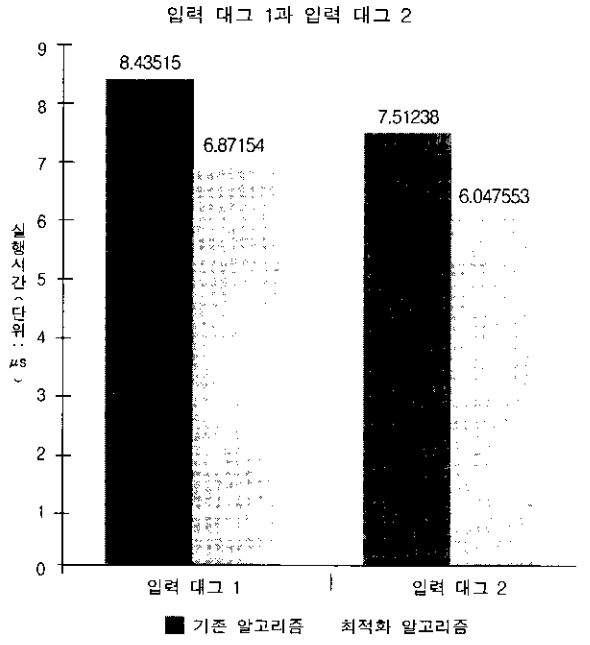
(그림 10)은 본문에서 제시한 (그림 5)의 결과를 기존 알고리즘과 비교하여 수행한 결과를 나타낸 입력 대그 1이고, 입력 대그 2는 본문에서 제시한 (그림 9)의 결과를 입력 대그 2로 나타내었다.

입력 대그 1과 입력 대그 2의 수행 결과에 대한 성능 평가를 <표 2>와 (그림 11)에 나타내었다. 표의 결과로부터 제안한 최적화 알고리즘이 실행 시간 측정에서는 기존 알고리즘 보다 입력 대그 1에서는 약 1.56μs, 입력 대그 2에서는 약 1.46μs 가 향상되었고, 재사용 노드수 에서도 전체 9개의 노드 중 기존 알고리즘에서는 입력 대그 1에서는 3개의 노드, 입력 대그 2에서는 5개의 노드를 재사용 하였으며, 제안한 최적화 알고리즘에서는 입력 대그 1에서는 6개의 노드, 입력 대그 2에서는 8개의 노드를 재사용 할 수 있었다.

〈표 2〉 입력 대그 1과 2의 알고리즘 수행 결과

(단위 : μs)

종 류 / 구 분	입력 대그 1		입력 대그 2	
	실행시간	재사용 노드 수 (전체 노드 수)	실행시간	재사용 노드 수 (전체 노드 수)
기존 알고리즘	8.435149	3(9)	7.512378	5(9)
최적화 알고리즘	6.871543	6(9)	6.047553	8(9)



(그림 11) 입력 대그 1과 입력 대그 2의 수행 분석 결과

4. 결 론

점진적 속성 문법을 위한 최적화 알고리즘에서는 새로운 입력 트리가 기존 입력 트리와 정확히 비교되어서 새로운 트리를 구성할 때 기존 속성 트리의 어떤 서브 트리를 사용해야 하는가를 결정한다. 따라서, 이 논문에서의 최적화 알고리즘의 동작 과정에서도 새로운 속성 트리의 생성 과정을 나타냈으며, Carle과 Pollock의 알고리즘을 분석하여 최적화된 알고리즘으로 재구성하고, 속성 트리 d_{copy}의 구성요소를 새로운 속성 트리 d'_{copy}에 적용하여 최적화된 속성 트리 d'_{copy}의 점진적 최적화 알고리즘을 구성하였다.

이 논문에서의 최적화 알고리즘의 성능평가를 통하여 노드의 재사용을 고려한 실험에서 기존 알고리즘 보다 최적화 알고리즘의 실행 시간이 입력 대그 1의 경우 약 18.5% , 입력 대그 2경우 약 19.5%가 향상되었다.

이와 같은 결과를 토대로 이 논문에서의 최적화 알고리즘은 기존의 알고리즘 보다 속성 트리 노드의 재사용과 실행시간이 향상되었음을 알 수 있었다.

최적화된 알고리즘은 제거, 단순화, 프로그램의 실행 시간과 공간을 줄이기 위한 시도로서 생성된 속성들의 재정렬을 포함하는 다양한 변환들을 수행한다. 따라서 이들 환경내에서 변환된 속성과 복잡한 재계산을 피하고 앞서 수행된 재사용 정보를 효율적으로 대체하기 위한 공간 최적화로의 응용이 앞으로의 연구과제다.

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. 1983. "Incremental context-dependent analysis for language-based editors," *ACM Trans. Program. Lang. Syst.* 5, 3(July), pp.449-477.

[3] Teitelbaum, T. and Chamman, R. 1990. "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.

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

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

[6] Carle, A. and Pollock, L. 1995a. "A context-based incremental evaluators for hierachical attribute grammars," *J. Program. Lang.* 3, pp.1-29.

[7] Carle, A. and Pollock, L. 1995b. "Maching-based incremental evaluators for hierachical attribute grammar dialects," *ACM Trans. Program. Lang. Syst.* 17, 2, pp.394-429

[8] Jia, A. and Qian, J. 1985. "Incremental evaluation of attributed grammars for incremental programming environments," *In IEEE COMPSAC'85(Chicago, III)*, IEEE, New York, 342-349.

장 재 춘

e-mail : jcjang@yeongdong.ac.kr

1992년 관동대학교 정보처리학과 졸업 (공학사)

1994년 관동대학교 교육대학원 전자계산공학과 졸업(교육학석사)

2001년 관동대학교 대학원 전자계산공학과 졸업(공학박사)

1994년~현재 영동전문대학 전자계산과 조교수

관심분야 : 컴파일러, 병렬 컴파일러, 웹 프로그래밍, 코드 최적화

안 희 학

e-mail : hhahn@mail.kwandong.ac.kr

1981년 숭실대학교 전자계산학과 졸업 (공학사)

1983년 숭실대학교 대학원 전자계산학과 졸업(공학석사)

1994년 숭실대학교 대학원 전자계산학과 졸업(공학박사)

1994년~1996년 관동대학교 전자계산소 소장

1984년~현재 관동대학교 컴퓨터공학과 교수

2001년~현재 관동대학교 전자계산소 소장

관심분야 : 컴파일러, 병렬컴파일러, 프로그래밍 언어, 함수언어, 오토마타 등