순차감축 알고리즘과 지연감축 알고리즘을 이용한 효과적인 지배자 트리의 구현

Implementation of Effective Dominater Trees Using Eager Reduction Algorithm and Delay Reduction Algorithm

이 대 식*
Dae Sik Lee

요 약

지배자 트리란 유향그래프에서 지배관계를 트리로 표현한 것이다. 임의의 유향그래프로부터 지배자 트리를 구성하기 위한 효과적인 알고리즘을 제시한다. 감축 가능한 흐름그래프는 지배자 계산을 한 후 지배자 트리로 감축된다. 감축 불가능한 흐름그래프는 정보 테이블의 연결가지 정보를 이용하여 지배자 연결그래프로 구성된다. 지배자 연결그래프에서 지배자 트리로 감축하기 위하여 효과적인 순차감축 알고리즘과 지연감축 알고리즘을 구현한다. 구현한 결과 순차감축 알고리즘 보다 지연감축 알고리즘의 실행시간이 빠르다. 따라서 흐름그래프에서 효과적인 지배자 트리로 감축된다.

Abstract

The dominator tree presents the dominance frontier from directed graph to the tree. we present the effective algorithm for constructing the dominator tree from arbitrary directed graph. The reducible flow graph was reduced to dominator tree after dominator calculation. And the irreducible flow graph was constructed to dominator-join graph using join-edge information of information table. For reducing the dominator tree from dominator-join graph, we implement the effective sequency reducible algorithm and delay reducible algorithm. As a result of implementation, we can see that the delay reducible algorithm takes less execution time than the sequency reducible algorithm. Therefore, we can reduce the flow graph to dominator tree effectively.

☞ Keyword : Flow Garph, sequency reducible algorithm, delay reducible algorithm, Dominator Tree

1. 서 론

최적화 컴파일러의 기본이 되는 철저한 자료흐름 분석(data-flow analysis)을 통해 프로그램의 여러 곳에서 정보와 관련시키는 자료 흐름 방정식을 세우고 이것을 풀어서 변수 생성과 소멸을 확인하고, 제어 흐름 정보에 따라 입력과 출력을 계산하여 코드 최적화가 이루어진다[1].

자료흐름 그래프는 자료흐름 분석을 위하여 지배자 트리로 감축하여 분석을 한다. 자료흐름 분석은 지배자 트리가 항상 정확하게 유지되어야 프로그램의 의미 변화가 일어나지 않는다[2, 3].

자료흐름 분석에 대한 기존의 연구는 Allen과 Cocke의 간격으로 분할하는 흐름그래프[4], Tarjan의 간격분석[5] 등이 있다. 이러한 연구는 goto문이 없는 감축 가능한 흐름그래프에서만 자료흐름 분석이 가능하다는 단점을 갖고 있다.

Sreedhar에 의해 연구된 자료흐름은 지배자 연결그래프를 이용하여 goto문이 있는 감축 불가능한 흐름그래프에서도 자료흐름 분석을 가능하게 하였다. 그러나 지배자 연결그래프의 연결가지를 제거하고 지배자 트리를 구하기 위하여 자료흐름 정보를 자료흐름 방정식에 적용하여 복잡한 지배영역 계산이라는 단점을 갖고 있다[6-8].

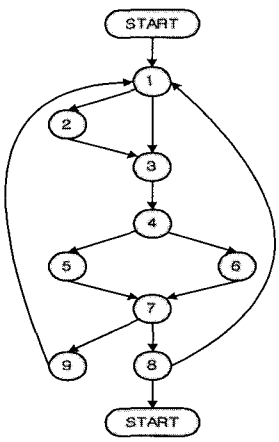
* 정 회 원 : 안동과학대학 사이버테러대응과 전임강사
dslee@andong-c.ac.kr(제1저자)
[2005/11/14 투고 - 2005/11/18 심사 - 2005/12/05 심사완료]

본 논문에서는 흐름그래프의 자료흐름 분석을 통해 지배자 트리 감축을 위하여 감축 가능한 흐름그래프와 감축 불가능한 흐름그래프를 판단하여 지배자 트리로 감축하는 알고리즘을 제시한다. 감축 가능한 흐름그래프는 지배자 계산을 한 후 바로 지배자 트리로 감축한다. 기존 연구의 단점을 극복하기 위하여 감축 불가능한 흐름그래프는 흐름그래프 가지에서 지배자 가지를 제거한 정보 테이블의 연결가지 정보를 이용하여 지배자 연결그래프로 구성한다[9]. 지배자 연결그래프에서 정보 테이블의 레벨 정보와 연결가지 정보를 이용하여 지배자 트리로 감축한다. 특히, 효율적으로 감축하기 위하여 순차감축 알고리즘과 지연감축 알고리즘을 실제로 구현하여 비교 분석해 보고자 한다.

2. 흐름그래프와 지배자 트리

2.1 흐름그래프

흐름그래프(flow graph)는 G = (N, E, START, END)인 유향그래프(directed graph)를 구성하여 기본 블록 집합에 제어 흐름 정보를 나타낸 것으로 노드의 기본은 블록을 의미한다. 여기서, N은 노드의 집합이고, E는 흐름 가지의 집합이며, START는 N에 속하는 시작 노드이고, END는 N에 속하는 끝 노드이다. 감축 가능한 흐름그래프는 (그림 1)과 같다[1, 6-8].



〈그림 1〉 감축 가능한 흐름그래프

2.2 지배자 트리

흐름그래프에서 초기 노드로부터 x를 통과하여 y에 이르는 연결이 있다면 노드 x는 노드 y를 '지배한다'라고 표현하고 'x dom y'라 표기한다. 그리고 'x dom y'이고 $x \neq y$이면, x는 y를 '직접 지배한다'라고 하며, 'x idom y'라고 표기한다. x가 y를 지배하지 못하면 'x !dom y'라고 표기한다. 이러한 지배관계를 트리로 표현한 것을 지배자 트리라 한다[1, 6-8].

지배자 계산은 식 (1)과 같다[1].

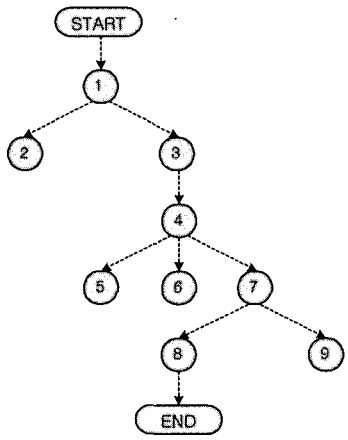
$$Dom(n) := \{n\} \cup (\cap \quad Dom(p)) \qquad (1)$$

p는 n의 선행노드

(그림 1)의 감축 가능한 흐름그래프를 지배자 트리로 표현하기 위해서는 흐름그래프의 각 노드를 식 (1)에 의해 지배자 계산을 하고 지배자 트리로 감축한다.

Dom(START) = {START}
Dom(1) = {1} ∪ Dom(START) = {START, 1}
Dom(2) = {2} ∪ Dom(1) = {START, 1, 2}
Dom(3) = {3} ∪ { Dom(1) ∩ Dom(2) } = {START, 1, 3}
Dom(4) = {4} ∪ Dom(3) = {START, 1, 3, 4}
Dom(5) = {5} ∪ Dom(4) = {START, 1, 3, 4, 5}
Dom(6) = {6} ∪ Dom(4) = {START, 1, 3, 4, 6}
Dom(7) = {7} ∪ { Dom(5) ∩ Dom(6) } = {START, 1, 3, 4, 7}
Dom(8) = {8} ∪ Dom(7) = {START, 1, 3, 4, 7, 8}
Dom(9) = {9} ∪ Dom(7) = {START, 1, 3, 4, 7, 9}
Dom(10) = {END} ∪ Dom(8) = {START, 1, 3, 4, 7, 8, END}

식 (1)에 의해 계산된 각 노드를 지배자 트리로 표현하면 (그림 2)와 같다.



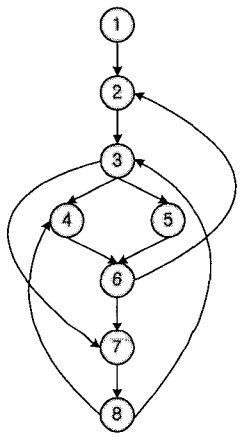
〈그림 2〉 지배자 트리

(그림 2)의 지배자 트리를 보면 START 노드는 노드 1, 2, 3, 4, 5, 6, 7, 8, 9, END를 지배한다. 또한 START 노드는 노드 1을 직접 지배한다. 이것을 'START idom 1'이라 표기한다. 또한 노드 1은 노드 2, 3을 직접 지배하므로 '1 idom 2', '1 idom 3'이다. 노드 3은 노드 4를 직접 지배하지만, 노드 2는 노드 3과 같은 레벨이면서 노드 4를 지배하지 못하기 때문에 '2 !dom 4'로 표기한다.

2.3 지배자 연결그래프

자료흐름 분석을 통한 흐름그래프를 지배자 트리로 감축하는데 있어 감축 가능한 흐름그래프와 감축 불가능한 흐름그래프를 판별하는 것은 매우 중요하다. 흐름그래프의 지배관계를 이해하면 역 연결선을 쉽게 찾을 수 있다. 역 연결선이란 (그림 1)의 goto문이 없는 역 연결선 9→1, 8→1과 같은 경우를 일컫는데, 이 역 연결선을 제거하여 남아있는 순차 연결선이 비순환이면 감축 가능한 흐름그래프이다. 실제로 형성되는 흐름그래프중 많은 것이 감축 가능한 흐름그래프에 속하게 된다. 만약 (그림 3)에서 goto문 같은 감축 불가능한 제어 흐름 문장을 사용한다면 지배자 트리로 감축할 때 프로그램의 의미나 제어형태가 바뀐다. 이러한 문제점을 극복하기 위해 지배자 연결그래프를 이용하여 감축 불가능한 흐름그래프를 안전하게 지배자 트리로 감축한다. 이 처럼 감축 불가능한 흐름그래프의 정의는 첫째 루프의 유일한 입구가 하나 이상이고, 둘째 사이클 밖에서 중간으로 들어오는 점프가 있어야 한다[1].

감축 불가능한 흐름그래프는 (그림 3)과 같다.



〈그림 3〉 감축 불가능한 흐름그래프

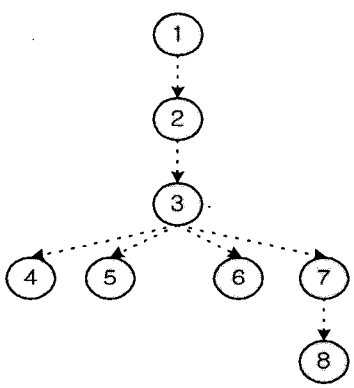
(그림 3)은 노드 4-6-7-8 사이클 중간에 goto문 3→7이 점프하여 들어오고, 노드 2-3-5-6 사이클 중간에 goto문 8→3이 점프하여 들어올 경우 사이클 밖에서 중간으로 점프하여 들어오는 3→7, 8→3에 의해 (그림 3)은 감축 불가능한 흐름그래프이다.

{연결가지 정보} = {흐름그래프 가지 정보}-
{지배자 트리의 가지 정보}
(2)

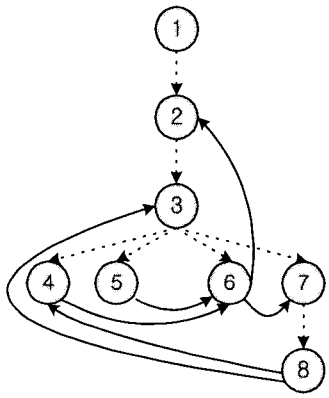
감축 불가능한 흐름그래프는 식 (1)을 이용하여 감축 가능한 흐름그래프와 같은 방법으로 지배자 계산을 한 후 (그림 4(a))의 지배자 트리로 감축한다. 지배자 트리로 감축하였을 때 감축 불가능한 흐름그래프는 감축 가능한 흐름그래프와

〈표 1〉 정보 테이블

흐름그래프의 가지 정보		지배자 트리의 가지 정보		레벨 정보	연결가지 정보
시작노드	도착노드	시작노드	도착노드		
노드 ①	②	노드 ①	②	레벨 0	null
노드 ②	③	노드 ②	③	레벨 1	null
노드 ③	④, ⑤, ⑦	노드 ③	④, ⑤, ⑥, ⑦	레벨 2	null
노드 ④	⑥	노드 ④	null	레벨 3	④→⑥
노드 ⑤	⑥	노드 ⑤	null	레벨 3	⑤→⑥
노드 ⑥	②, ⑦	노드 ⑥	null	레벨 3	⑥→②, ⑥→⑦
노드 ⑦	⑧	노드 ⑦	⑧	레벨 4	null
노드 ⑧	③, ④	노드 ⑧	null	레벨 3	⑧→③, ⑧→④



(a) 지배자 트리



(b) 지배자 연결그래프

〈그림 4〉 지배자 트리와 지배자 연결그래프

달리 본래 프로그램의 의미나 제어형태가 달라질 수 있다. 따라서 감축 불가능한 흐름그래프에서는 식 (2)를 이용하여 (그림 3)의 감축 불가능한 흐름그래프의 가지 정보에서 (그림 4(a))의 지배자 트리 가지 정보를 제거한 정보 테이블의 연결가지 정보를 추가하여 (그림 4(b))의 지배자 연결 그래프를 구성한다[9]. 지배자 연결그래프는 모든 노드의 의미와 흐름 정보를 알 수 있으므로 정보 테이블의 레벨 정보와 연결가지 정보를 이용한 순차감축 알고리즘과 지연감축 알고리즘으로 연결가지 정보를 안전하게 제거하여 의미 변화없는 지배자 트리로 감축할 수 있다.

3. 지배자 트리 변환 알고리즘

3.1 흐름그래프에서 지배자 트리로의 변환 알고리즘

흐름그래프를 지배자 트리로 변환하기 위하여 (알고리즘 1)을 제시한다. 감축 가능한 그래프는 지배자 계산을 한 후 바로 지배자 트리로 감축한다. 그러나 감축 불가능한 흐름그래프는 모든 노드의 정보를 알 수 있는 지배자 연결그래프를 구성하고, 연결가지를 제거하여 안전하게 지배자 트리로 감축한다.

```
procedure maindom
begin
  inputdata(); /*흐름그래프를 입력받는 함수*/
  compute_dom(); /*흐름그래프를 지배자 트리로 계산*/
  chk_cycle(int level, nodeptr curnode)
     begin
     for i=0 to tmp[i] do
     begin
        if (iscycle(tmp[i])) then
        /*흐름그래프의 사이클 존재 여부 검사*/
          if (isinnerlink(level, tmp[i]))
          /*사이클안으로 점프하는 연결선의 존재 여부 검사*/
          begin
             process_jedges() /* 연결가지 생성 */
             make_dj() /* DJ 그래프 형성 */
             del_jedges() /* 연결가지 제거 */
          end;
              else if( chk_cycle(level+1, tmp[i])
     end;
     trnode[level] := NULL;
  end;
  make_dom; /*지배자 트리 완성*/
end;
```

(알고리즘 1) 지배자 트리로의 변환 알고리즘

3.2 지배자 연결그래프로부터 연결가지 제거 알고리즘

지배영역이라는 계산 과정없이 <표 1> 정보 테이블의 연결가지 정보를 지배자 트리에 추가하여 지배자 연결그래프로 구성하고, 정보 테이블의 레벨 정보와 연결가지 정보를 이용하여 지배자 연결그래프에서 지배자 트리로 의미 변화없이 안전하게 감축한다.

(1) 순차감축

지배자 연결그래프의 연결가지를 순차적으로 제거하고 프로그램의 의미나 흐름의 변화없이 안전하게 지배자 트리로 감축하기 위해 지배 영역 계산이라는 복잡한 계산과정을 거치지 않고 상향식으로 감축하는 순차감축 알고리즘을 제시하면 (알고리즘 2)와 같다.

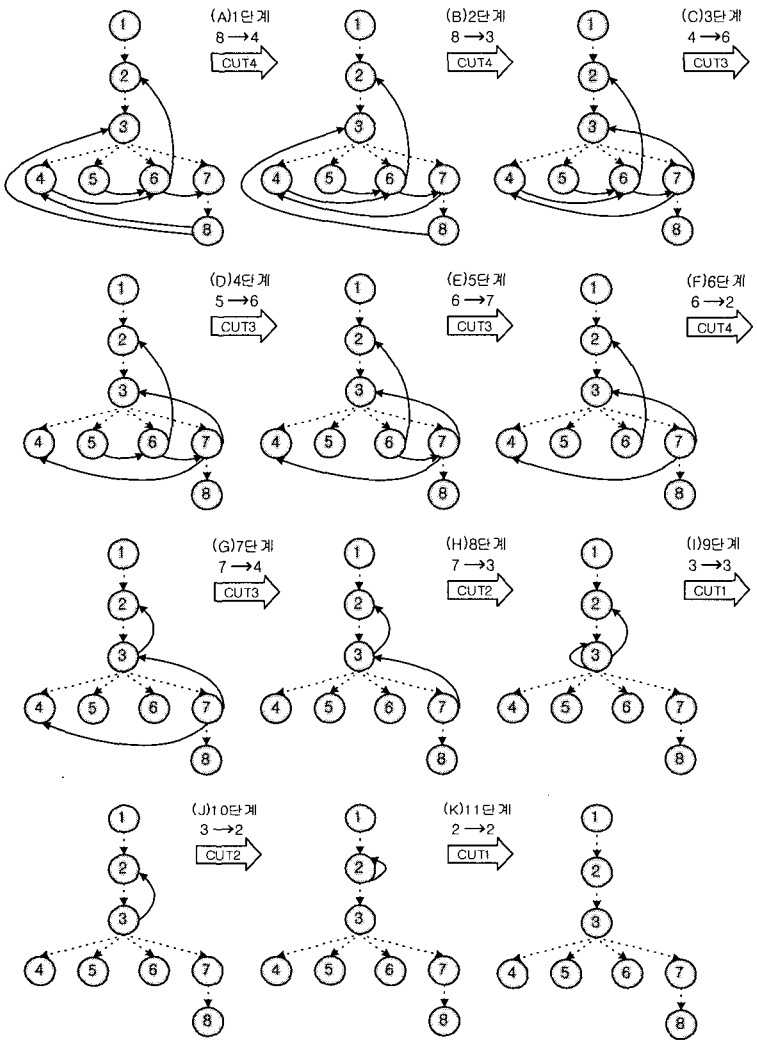
순차감축 알고리즘의 시간 복잡도를 구하면 O(|N|×|E|)이다. 여기서 |N|은 지배자 트리의 노드 갯수이고, |E|는 연결가지 갯수이다.

(알고리즘 2)의 CUT1, CUT2, CUT3, CUT4를 자세히 설명하면 다음과 같다.

```
procedure del_jedges_sequency()
begin
   for i:=0 to DJ_tree[i] do
   begin
      if DJ_tree[i]↑jedges then
         for j=0 to tmp[j] do
         begin
            j_sequency(DJ_tree[i], tmp[j]);
         end;
   end;
end;
procedure j_sequency(nodeptr from_n, nodeptr to_n)
begin
   if from_n = to_n then                              /* CUT1 */
      deljedge(from_n, to_n);
      else if from_n_parent = to_n then
      begin                                           /* CUT2 */
         to_n↑data = to_n↑data + from_n↑data;
         deljedge(from_n, to_n);
         addjedge(to_n, to_n);
      end;
      else if from_n_level = to_n_level then
      begin                                           /* CUT3 */
         to_n↑data = to_n↑data + from_n↑data;
         deljedege(from_n, to_n);
      end;
         else
         begin                                        /* CUT4 */
            from_n_parent↑data = from_n_parent↑dat + from_n↑data;
            deljedege(from_n, to_n);
            addjedge(from_n_parent, to_n);
            j_sequency(from_n_parent, to_n);
         end;
end;
```

(알고리즘 2) 순차감축 알고리즘

순차감축 알고리즘에서 CUT1은 시작노드(from_n)와 도착노드(to_n)가 일치하는 노드로서 자기 자신으로 도는 루프이므로 연결가지를 제거한다. CUT2는 시작노드가 바로 부모노드(from_n_parent)를 가리키는 연결가지로서 시작노드의 레벨 정보와 연결가지 정보를 부모노드에게 주고 연결가지를 제거하여 도착노드 자기 자신으로 도는 루프를 생성한다. CUT3은 시작노드와 도착노드가 같은 레벨에 있는 연결가지로서 시작노드의 레벨 정보와 연결가지 정보를 도착노드에게 주고 연결가지를 제거한다. CUT4는 시작노드와 도착노드가 서로 다른 레벨로 시작노드의 레벨 정보와 연결가지 정보를 부모노드에게 주고 연결가지를 제거한다. 또한 부모노드와 도착노드 사이에 새로운 연결가지를 생성한다.



〈그림 5〉 순차감축 알고리즘에 의한 지배자 트리로의 변환과정

(알고리즘 2)의 순차감축 알고리즘을 적용하여 (그림 4(b))의 지배자 연결그래프를 지배자 트리로 감축하는 과정은 (그림 5)와 같다.

(그림 5)의 과정을 간단히 설명하면 다음과 같다.

1단계 CUT4의 시작노드 8과 도착노드 4는 서로 다른 레벨로 시작노드 8의 레벨 정보와 연결가지 정보를 부모노드 7에 주고 연결가지를 제거한다. 부모노드 7과 도착노드 4 사이에 새로운 연결가지를 생성한다. 3단계 CUT3의 시작노드 4와 도착노드 6은 같은 레벨에 있는 연결가지로서 시작노드 4의 레벨 정보와 연결가지 정보를 도착노드 6에 주고 연결가지를 제거한다. 8단계 CUT2는 시작노드 7이 바로 부모노드를 가르키는 연결가지로서 시작노드 7의 레벨 정보와 연결가지 정보를 부모노드 3에 주고 연결가지를 제거하여 도착노드 3 자기 자신으로 도는 루프를 생성한다. 9단계 CUT1은 시작노드 3과 도착노드 3이 일치하는 노드로서 자기 자신으로 도는 루프이므로 연결가지를 제거한다.

(2) 지연감축

지연감축도 순차감축과 마찬가지로 상향식으로 지배자 연결그래프를 감축하는데 지연감축 알고리즘을 제시하면 (알고리즘 3)과 같다.

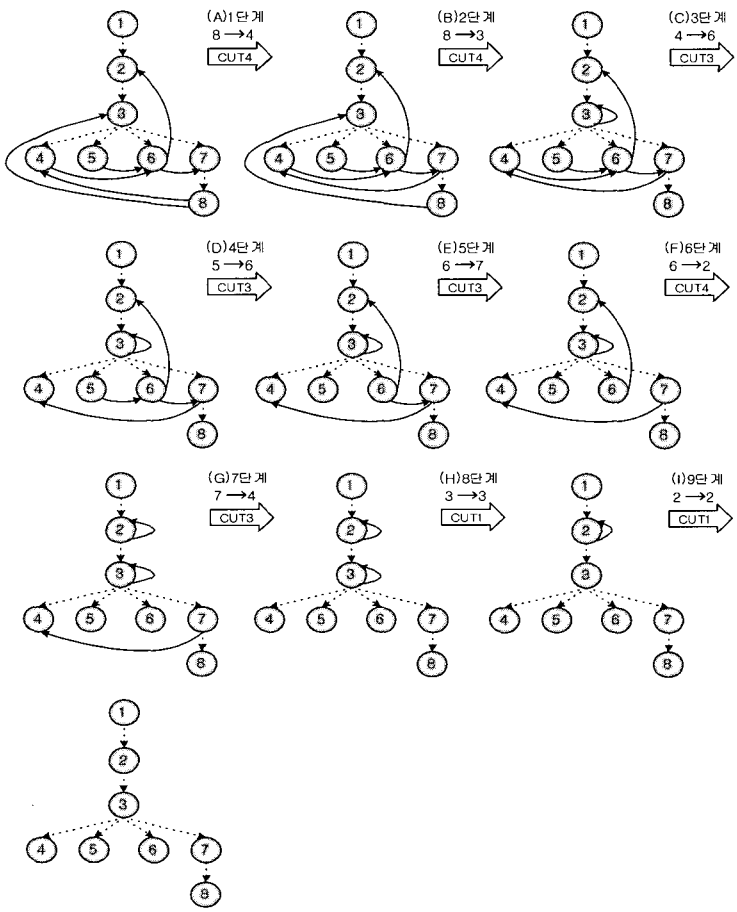
지연감축 알고리즘의 시간 복잡도를 구하면 O(log|N|×|E|)이다. 여기서 |N|은 지배자 트리의 노드 갯수이고, |E|는 연결가지 갯수이다.

```
begin                                                    /* CUT4 */
    find_ancestor(nodeptr from_n_ancestor)
    from_n_ancestor↑data = from_n_ancestor↑data + from_n↑data;
    deljedege(from_n, to_n);
    addjedge(from_n_ancestor, to_n);
    j_delay(from_n_ancestor, to_n);
end;
```

(알고리즘 3) 지연감축 알고리즘

지연감축 알고리즘에서 CUT1, CUT2, CUT3는 순차감축 알고리즘의 연결가지 제거방법과 같고, CUT4는 시작노드와 도착노드가 서로 다른 레벨로서 시작노드의 레벨 정보와 연결가지 정보를 부모노드가 아닌 시작노드의 선행노드이면서 도착노드와 같은 레벨인 조상노드에게 주고 연결가지를 제거한다. 또한 조상노드와 도착노드 사이에 새로운 연결가지를 생성하므로 순차감축 알고리즘보다 효과적으로 사용할 수 있다.

(알고리즘 3)의 지연감축 알고리즘을 적용하여 (그림 4(b))의 지배자 연결그래프를 지배자 트리로 감축하는 과정은 (그림 6)과 같다.



〈그림 6〉 지연감축 알고리즘에 의한 지배자 트리로의 변환과정

(그림 6)의 과정을 간단히 설명하면 다음과 같다.

(그림 6)에서 지연감축 알고리즘의 CUT1, CUT2, CUT3는 순차감축 알고리즘과 같은 방법으로 연결가지를 제거한다. 2단계 CUT4의 시작노드 8과 도착노드 3은 서로 다른 레벨로서 시작노드 8의 레벨 정보와 연결가지 정보를 부모노드 7이 아닌 시작노드 8의 선행노드이면서 도착노드 3과 같은 레벨인 조상노드 3에 주고 연결가지를 제거한다. 또한 조상노드 3과 도착노드 3사이에 새로운 연결가지를 생성한다. 이처럼 2단계와 6단계 감축과정이 (그림 5)보다 수행단계가 9단계 과정으로 감소하고, 의미 변화없이 안전하게 지배자 트리로 감축된다.

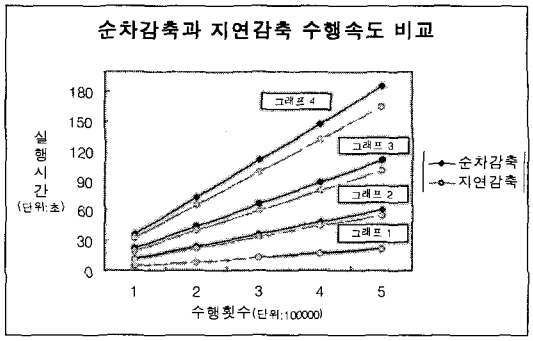
(3) 성능분석

(그림 5)와 (그림 6)은 (그림 4(b))의 지배자 연결그래프에 대하여 순차감축 알고리즘과 지연감축 알고리즘을 이용한 지배자 트리로의 변환 과정을 보여준다.

따라서, 본 논문에서 제시한 순차감축 알고리즘과 지연감축 알고리즘의 타당성을 검증하기 위해 구현 실험해 본 결과 <표 2>에서 알 수 있듯이 지연감축 알고리즘이 순차감축 알고리즘보다 실행시간과 수행단계가 단축되는 것을 알 수 있다.

〈표 2〉 순차감축과 지연감축의 수행속도

종류 / 시간 / 수행 횟수	그래프 1		그래프 2		그래프 3		그래프 4	
	순차	지연	순차	지연	순차	지연	순차	지연
100000	4.45	4.34	12.34	11.39	22.3	20.17	37.11	33.10
200000	8.90	8.62	24.63	22.51	44.57	40.32	73.98	66.02
300000	13.29	12.96	36.98	34.10	66.83	60.49	111.35	99.13
400000	17.74	17.25	49.35	45.42	89.13	80.65	147.10	132.11
500000	22.25	21.59	61.72	56.11	111.39	100.83	185.89	165.53



4. 결 론

감축 불가능한 흐름그래프에서 지배자 트리로 감축할 때 지배자 연결그래프를 구성하고, 각 노드의 지배영역을 계산하여 지배자 트리로 감축하는 기존의 연구와 비교하여, 본 논문에서는 흐름그래프 가지정보에서 지배자 트리 가지정보를 제거한 연결가지 정보를 지배자 트리에 추가하여 지배자 연결그래프를 구성하고, 또한 효과적인 지배자 트리 감축을 위하여 지배자 트리 변환 알고리즘을 제시하였다.

효과적인 자료흐름 분석을 하기 위하여 임의의 흐름그래프를 지배자 계산을 한 후 감축 가능한지를 판단하여 감축 가능한 흐름그래프는 지배자 계산을 한 후 바로 지배자 트리로 감축하고, 감축 불가능한 흐름그래프는 지배자 트리로 감축한다. 감축 불가능한 흐름그래프의 지배자 트리는 지배관계만 형성되므로 제어흐름 형태가 변할 수 있다. 따라서 정보 테이블의 연결가지 정보를 이용하여 지배자 트리에 연결가지를 추가함으로써 모든 노드의 흐름정보를 알 수 있는 지배자 연결그래프를 구성하였다. 지배자 연결그래프를 프로그램의 의미 변화없이 안전하게 지배자 트리로 감축하기 위하여 순차감축 알고리즘과 지연감축 알고리즘을 제시하였다. 순차감축 알고리즘과 지연감축 알고리즘은 자료흐름 분석을 효과적으로 하기 위해 정보 테이블의 레벨 정보와 연결가지 정보를 이용하여 안전하게

감축한다. 특히, 순차감축 알고리즘과 지연감축 알고리즘의 타당성을 검증하기 위해 구현해 본 결과 수행 단계, 수행 속도 면에서 지연감축 알고리즘이 우수하다는 타당성이 검증되었다.

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] Carroll, M. and Ryder, B. G. "Incremental data flow update via attribute and dominator updates," *In ACM SIGPLAN-SIGACT Symposium on the Principles of Programming Languages,* pp.274-284, January 1988.

[3] Ramalingam, G. and Reps, T. "An incremental algorithm for maintaining the dominator tree of a reducible flowgraph," *In ACM SIGPLAN-SIGACT Symposium on the Principles of programming Languages,* pp.314-325, January 1994.

[4] Allen, F. E., and Cocke, J. "A program data flow analysis procedure," *Communication of ACM,* Vol.19, No.3, pp. 137-147, 1977.

[5] Tarjan, R. E. "A unified approach to path problems," *Journal of ACM,* Vol.28, No.3, pp.577-593. 1981b.

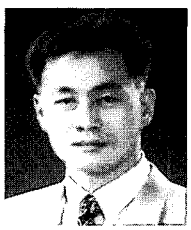
[6] Sreedhar, V. C., Gao, G. R., and Lee, Y. F. "Incremental Computation of Dominator Trees," *ACM SIGPLAN Notices,* Vol.30, No.4, pp.1-12, 1995.

[7] Sreedhar, V. C., and Gao, G. R., and Lee, Y. F. "Identifying Loops Using DJ Graphs," *ACM Transactions on Programming Languages and Systems,* Vol.18, No.6, pp.649-658, 1996.

[8] Sreedhar, V. C., and Gao, G. R., and Lee, Y. F. "A New Framework for Elimination-Based Data Flow Analysis Using DJ Graphs," *ACM Transactions on Programming Languages and Systems,* Vol.20, No.2, pp.388-435, 1998.

[9] 심손권, 장재춘, 안희학, "자료 흐름 분석을 위한 효과적인 지배자 트리 알고리즘", 한국정보처리학회 '98 춘계 학술 발표 논문집, 1998.

◐ 저 자 소 개 ◑



이 대 식 (Dae Sik Lee)

1995년 관동대학교 전자계산공학과(공학사)
1999년 관동대학교 전자계산공학과(공학석사)
2004년 관동대학교 전자계산공학과(공학박사)
2005년~현재 안동과학대학 사이버테러대응학과 전임강사
E-mail : dslee@andong-c.ac.kr