유사도 평가를 위한 트리 비교 알고리즘

김 영 철[†] · 유 재 우[††]

요 약

기존의 트리 비교에 관한 연구는 대부분 노드에 가중치가 있거나 레이블이 있는 트리(장식이 있는 트리)에 대해서 연구되었다. 그러나 본 연구에서는 장식이 없는 서로 다른 두개의 트리를 비교하여 유사도를 평가하는 알고리즘을 제시하고 구현한다. 본 시스템에서 제시한 트리 유사도 평가 알고리즘은 비교할 두 개의 트리를 언파서에 의해 노드 스트링으로 변환된 후, 유사도 알고리즘에 의해서 평가되며, 0.0~1.0 사이의 유사 값을 돌려준다. 본 논문의 실험 부분에서는 여러 형태의 트리를 비교 분석하였으며, 두 트리 사이에 일치되는 노드와 불일치되는 노드를 시각적으로 표현하였다. 본 연구를 활용하면, 특정한 프로그램이나 문서의 유사도 및 중복 코드 발견 등에 활용할 수가 있다.

A Tree-Compare Algorithm for Similarity Evaluation

Young-Chul Kim[†] · Chae-Woo Yoo[††]

ABSTRACT

In the previous researches, tree comparison methods are almost studied in comparing weighted or labeled tree(decorated tree). But in this paper, we propose a tree comparison and similarity evaluation algorithm can be applied to comparison of two normal trees. The algorithm converts two trees into node string using unparser, evaluates similarity and finally return similarity value from 0.0 to 1.0. In the experiment part of this paper, we visually presented matched nodes and unmatched nodes between two trees. By using this tree similarity algorithm, we can not only evaluate similarity between two specific programs or documents but also detect duplicated code.

키워드 : 유사도 평가(Similarity Evaluation), 트리 비교(Tree Compare), 언파서(Unparser), 노드 스트링(Node String)

1. 서 론

많은 응용 프로그램의 개발환경에 사용되는 자료구조 중 트리는 표준화되고 일관된 스타일로 데이터 타입을 표현할 수 있기 때문에 많이 이용되고 있다. 또한 트리 자료구조는 일반적인 프로그래밍 개발환경에서 핵심 데이터를 쉽게 표현할 수 있으며, 다양한 형태로 변형되어 사용되고 있다. 트리를 변형하여 이용하는 것은 기억 공간의 절약 측면에서 수행되고 있다. 즉, 깊이가 크면 클수록 저장 공간을 많이 이용되므로 트리의 자식 노드를 많게 함으로써 깊이를 줄일 수 있다. 이처럼 트리는 많은 응용 프로그램의 내부 구조로 많이 표현되고 있다[1].

서로 다른 두 개의 트리를 비교하여 유사도를 측정하는 것은 아주 중요한 분야이다[2, 3]. 이 분야는 인공지능, 소프트웨어 공학, 프로그래밍 언어 분야에서 많이 연구되고 있다. 특히 트리의 유사성 표현은 거의 연구가 되지 않았으며, 유틸리티 이론 부분이나 인공지능 분야에서 연구가 이루어진 적이 있었다. 특히, Lisp, Prolog 및 XML 분야에서도 레이블을 가진 트리나 가중치가 있는 트리에 대한 유사성 연구가 있었다[4, 5].

서로 다른 두 개의 트리 T1, T2를 비교하는 방법은 크게 두 부류로 구분되어 연구되어왔다[6]. 첫 번째 방법은 트리를 변환할 때 이용되는 오퍼레이션 수를 이용하여 비교하는 방법으로, 트리 T1에서 트리 T2로 변환될 때 요구되는 최소의 오퍼레이션 수를 계산하는 방법이다. 두 번째 방법은 트리를 보다 간단한 다른 구조로 변환한 후 유사도를 측정한다. 두 번째 방법에서 유사도에 이용되는 트리의 표현은 클러스터나 4분열 트리(quartets)을 이용한다.

본 논문에서는 앞서 언급한 두 번째 방법으로, 비교하고자하는 트리를 언파서(unparser)를 이용하여 노드의 스트링으로 변환한 후, 유사도를 측정하는 방법을 제시한다. 본 시스템은 서로 다른 두 개의 트리를 비교하여 유사성을 값으로 표현한다. 트리의 유사도 값은 0과 같거나 크고, 1보다 작거나 같은 값을 반환한다. 다음 (그림 1)은 트리 유사도 평가 모델을 보여준다.

(그림 1)에서 검사될 두 트리는 각각 P1과 P2로 정의하며, P1은 원본 트리이며, P2는 비교될 대상 트리로 정의한다. 노드 스트링 A와 B는 각각 언파서에 의해 만들어진 트리 P1과 P2의 노드 스트링으로 정의한다.

※ 본 연구는 숭실대학교 교내연구비 지원으로 이루어졌음.
† 준 회 원 : (주)뉴스텍시스템즈 이사, 명지전문대학 조교수
†† 정 회 원 : 숭실대학교 컴퓨터학부 교수
논문접수 : 2003년 9월 22일, 심사완료 : 2004년 3월 26일

(그림 1) 트리 유사도 평가 모델

(그림 1)과 같이 비교될 두 개의 트리는 유사도 평가 알고리즘에 입력되어 유사도를 평가한다. 그림에서 보는 것처럼 비교될 두 개의 트리는 트리 유사도 평가 엔진에 의하여 비교하기 전에 언파서(Unparser)에 의해서 노드 스트링 형태로 변환된다. 변환된 트리 노드 스트링은 트리 유사도 평가 엔진에 의해 유사도를 평가한 후, 유사도 값을 반환한다. 트리 유사도 평가 모델에서 반환되는 유사도 값은 식 (1-1)과 같다.

$$0 \leq \text{유사도 } sim(P1, P2) \leq 1 \qquad (1\text{-}1)$$

식 (1-1)에서 유사도 값이 0인 경우에는 검사될 두 트리 중 하나는 공 트리(empty Tree)이거나 비교하고자 하는 최소의 노드 수보다 작은 트리의 노드 수일 경우에 해당된다. 유사도 값이 1인 경우는 검사될 두 트리는 완전히 일치함을 의미한다. 유사도 값이 0보다 크고 1보다 작은 실수인 경우는 두 트리가 부분적으로 일치함을 의미한다.

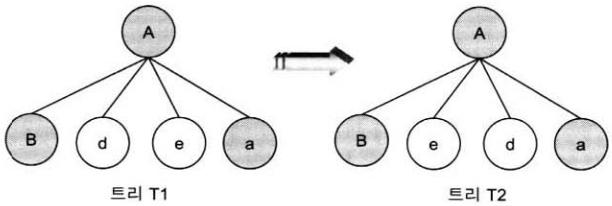
본 논문은 다음과 같이 구성되었다. 제2장에서는 관련연구에 대해서 기술하였으며, 제3장에서는 트리 유사도 평가 알고리즘에 대해서 기술하였으며, 제4장에서는 실험 및 평가를 기술하였다. 마지막으로 제5장에서는 결론을 기술하였다.

2. 관련 연구

기존의 트리 비교 방법은 다양한 분야에서 진행되었다. 특히 Bledsoe와 Raikow[7]은 생물학에서 종의 분류학 일치 개념을 트리와 연관하여 비교하였다. 이외에도 참조 트리(reference tree)를 가진 부트스트랩 트리를 비교하는 방법이나 분할된 트리 찾는 방법에서 트리 비교가 수행되어 왔다[6]. 또한 앞장에서 언급했듯이 트리 비교 방법은 크게 두 가지 방법 즉, 트리 변환과정에서 이용된 오퍼레이션 수를 이용하거나 트리를 다른 표현으로 바꾸어 유사도를 평가하는 방법들이 연구되어 왔다.

트리를 변환하여 비교하는 방법은 Waterman과 Smith[8]가 연구한 방법으로 NNI(가장 이웃한 단말 노드 교환) 행렬을 이용한다. 예를 들면 다음 (그림 2)에서 트리 T1에서 트리 T2로 변환될 때 요구되는 최소의 NNI수는 1이므로 트리 T1과 T2 사이의 거리(distance) $D_{NNI(T1, T2)} = 1$이 된다. 트리 사이의 거리가 1이라는 의미는 트리 T1을 트리 T2로 변환할 때 한번이면 된다는 의미이다.

트리를 다른 표현으로 바꾸어 비교하는 방법은 4분열(quartet) 이라는 측정방법이다. 4분열은 특정한 트리의 서브트리로 다양한 형태가 존재하며, 일치되는 4분열 트리와 불일치되는 4분열 트리로 구분하여 유사도를 평가한다.



(그림 2) 트리의 변환

이외에도 트리에 관한 유사성 연구는 Ira D.B axter[9]와 A.W. cole[10]을 들 수 있다. Ira D.B axter는 두 트리 사이에 중복된 노드를 찾는 연구를 수행하였다. 이 연구에서는 중복된 노드의 집합을 "clone"이라 칭하였으며, 중복된 노드를 통하여 유사도를 평가하는 방법을 제시하였다. 이 논문에서 제시한 유사도 평가 수식은 다음 식 (2-1)과 같다.

$$\text{Similarity} = 2 \times \frac{S}{2 \times S + L + R} \qquad (2\text{-}1)$$

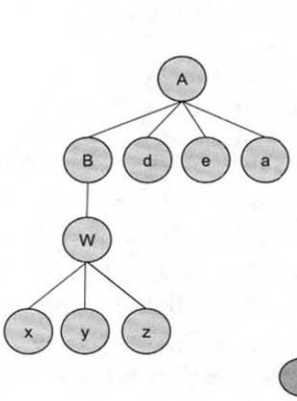
위의 식 (2-1)을 설명하기 위하여 비교될 두 트리를 각각 T_a, T_b라 하면, 식 (2-1)에서 S는 두 트리 T_a, T_b에서 공통적으로 존재하는 공유 노드(Shared Node)의 수를 나타낸다. 또한 L은 트리 T_a에는 있고 트리 T_b에는 없는 노드의 수를 나타내며, R은 트리 T_b에는 있고 트리 Ta에는 없는 노드의 수를 나타낸다.

A.W. Cole[10]은 자신의 연구논문에서 "tromp"라는 트리 비교 도구 시스템을 제시하였다. tromp란 "tree compare" 약어로 서로 다른 두개의 트리를 비교하는 유틸리티이다. tromp의 입력은 2개의 트리이며, 출력은 0~1까지의 유사도를 나타낸다. tromp의 값은 다음과 같다.

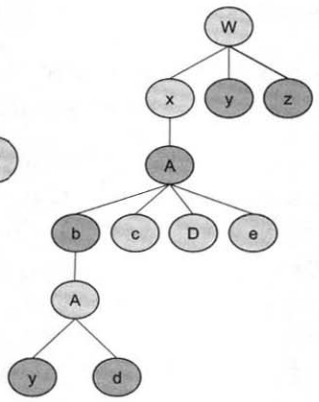
① **tromp(t(A), t(A))™ = 1** : 같은 트리의 비교는 1값을 반환한다.
② **tromp(t(A), t(empty))™ = 0** : 하나의 트리(t(A)와 공 트리(t(empty))를 비교하였을 경우의 유사도 값이다.
③ **tromp(t(A), t(B))™ = 0.0~1.0** : 서로 다른 두개의 트리를 비교하였을 경우의 트리이다.

예를 들면 다음 (그림 3), (그림 4)와 같은 두 개의 트리 t(A)와 t(W)가 있다고 가정하자.

A.W. Cole이 제시한 시스템에서는 다음과 같이 유사도를 측정한다. tromp에서 정규화된 간선(edge) 빈도수는 정규화된 트리의 루트에서 단말노드(leaf)까지 경로의 평균값이다. 따라서 tromp는 트리 t(A)와 t(W)로부터 정렬된 간선을 비교한다. 순서화된 간선들의 각 리스트가 압축되고 정규화된다. 예를 들면, 두 트리 t(A)와 t(W)에서 간선들의 수는 다음과 같다.



(그림 3) 트리 t(A)



(그림 4) 트리 t(W)

t(A) : [aa1], [ab1], [ad1], [ae1], [bw1], [wx1], [wy1], [wz1]
t(W) : [ab1], [ac1], [ad2], [ay1], [ae1], [da1], [wx1], [wy1], [wz1], [za1]
정규화된 값 : 0, 1/2, 1, 1/2, 1, 1, 1, 1, 0, 0, 0 = 6

따라서 총합 11, 정규화된 합이 6이므로 tromp에서 두 트리 사이의 유사도는 6/11 혹은 0.5454 값이다.

이 외에도 트리 비교에 관한 연구는 가중치가 있는 트리나 레이블이 있는 트리와 같은 특정 분야에서 많이 연구되고 있다. 특히 V.C. Bhavsar[4]는 가중치가 있는 트리 유사도 알고리즘을 연구하였다. 이 연구는 전자상거래 환경에서 이용되는 다중 에이전트 시스템을 위한 트리 유사도 알고리즘이다. 즉, 소비자와 판매자와의 관계를 가중치를 두어 트리로 표현하여 유사도를 측정하였다.

3. 트리 유사도 평가 알고리즘

본 논문에서 제시한 서로 다른 두 트리의 유사도를 평가하기 위한 알고리즘은 크게 2 단계로 이루어져 있다. 첫 번째 알고리즘에서는 일치되는 스트링을 찾기 위한 단계이며, 두 번째 단계는 유사도를 측정하기 위한 단계이다.

(알고리즘 1)에서 서브스트링이란 노드 스트링 중 특정한 일부분의 스트링을 말하며, X 측정하기 위한 단계이다. Xi 형태로 표현하였다. 여기서 X는 노드 스트링을 말하며, i는 X 스트링의 인덱스이다. 또한 maxmatch와 matchsize는 각각 일치된 스트링의 최대 개수와 두 노드 스트링 중 일치되는 서브스트링의 개수이다.

함수 match(A_i, B_j, matchsize)는 두 노드 스트링 A와 B에 대해서 각각 인덱스 i, j부터 시작하여 matchsize 만큼 일치되는 스트링을 말하며, match 함수가 반환하는 값은 일치되는 두 노드 스트링의 서브스트링이다.

함수 Deleting($X_{i+matchsize}$)는 노드 스트링 X의 "i+matchsize"번째의 스트링을 삭제하는 함수이다. 이 함수는 두 노드 스트링 A, B에서 일치되는 서브스트링을 노드 스트링에서 삭제하는데 이용된다. 노드 스트링에서 일치되는 서브스트링을 삭제하는 이유는 두 노드 스트링이 비교된 후에, 또 다시 중복 검사되는 것을 막기 위함이다.

```
NodeString MatchString(NodeString A, NodeString B) {
  long int i, j, k ; /* 각각 A, B에서 일치되는 스트링의 인덱스 */
  long int matchsize, maxmatch ; /* 각각 일치되는 스트링의 크기 */
  matchstring = "" ; /* 가장 길게 일치되는 스트링 */
  for Ai in A {
    for Bj in B {
      matchsize = 0 ; /* 일치되는 스트링의 크기 */
      while(Ai+matchsize == Bj+matchsize && Ai+matchsize && Bj+matchsize)
          matchsize++ ;
      if (matchsize > maxmatch) {
          /* 찾은 스트링의 개수가 이전에 찾은 스트링의 개수보
             다 클 때 수행 */
          matchstring = match(Ai, Bj, matchsize) ;
          /* 가장 큰 스트링을 matchstring에 할당 */
          maxmatch = matchsize ;
      }
    } end for
  } end for
  Ai = Bj = "NULL" ; /* 잘못 검사되는 것을 방지하기 위한 플래
                        그 삽입 */
  for k = 1 to matchsize -1 {
    Deleting(Ai+k) ;
    Deleting(Bj+k) ;
  } end for

  return matchstring ; /* 찾은 스트링을 반환 */
}
```

(알고리즘 1) 두 노드 스트링(A, B)에서 매칭 스트링 찾기

(알고리즘 1)은 노드 스트링 A에서 각각의 서브스트링 A_i에 대해서 동일한 B_j를 찾는다. 동일한 서브 노드 스트링을 찾으면 가능한 최대 크기의 토큰 스트링을 찾기 위해서 계속 비교된다. 두 토큰($A_{i+matchsize}$와 $B_{j+matchsize}$)이 다르다면, 찾은 서브스트링을 추가할 것인지 검사한다. 만일 matchsize가 maxmatch 보다 크다면, matchstring에 새롭게 추가된다. 추가된 스트링은 중복 검사를 방지하기 위하여 노드 스트링 A, B에서 삭제된다.

(알고리즘 2)에서 입력으로 이용되는 minlength는 노드 스트링 중 최소로 일치될 서브스트링 개수이다. minlength는 대상 노드 스트링 중에서 최소로 일치하는 서브노드 스트링의 개수를 말한다. 예를 들면, minlength가 3일 경우, 노드 스트링 A, B 중 적어도 3개 이상의 노드 스트링이 일치될 경우에 그 서브스트링을 찾아준다.

함수 Set(totalmatchstring)는 두 노드 스트링에서 찾은 모든 서브스트링을 보관하는 함수이며, Length(X)는 노드 스트링 X의 길이를 구하는 함수이다. Length(X) 함수는 노드 스트링의 길이를 구하는 함수로 유사도 측정에 이용된다. 즉, 이 함수는 (알고리즘 1)에서 구한 일치된 서브 스트링의 길이와 입력으로 들어온 노드 스트링의 길이를 구해주며, 최종적으로 유사도 수식에서 이용된다.

본 알고리즘의 시간 복잡도와 관련하여 최악의 경우 시간 복잡도는 $O(n^3)$이다. 즉, (알고리즘 1)에서는 2개의 반복 구조를 취하므로 시간 복잡도는 $O(n^2)$이고, (알고리즘 2)

에서 한번의 반복문을 취하므로 전체 시간 복잡도는 $O(n^3)$을 나타내게 된다. 또한 최상의 경우 시간 복잡도는 $O(n^2)$이다. 왜냐하면, 두 노드 스트링이 완전히 일치한다면, (알고리즘 2)의 반복문은 1번만 수행된다. 따라서 최상의 경우 시간 복잡도는 $O(n^2)$이 된다.

```
double Sim(NodeString A, NodeString B, long int minlength) {
    String matchstring, totalmatchstring ;  /* 일치된 스트링 */
    int maxmatch = 0 ; /* 일치된 스트링의 개수 초기화 */
    long int matchlength = 0 ;  /* 일치된 스트링의 전체 개수
                                   초기화 */
    Set(totalmatchstring) = {} ;  /* 일치되는 전체 스트링 집합*/

    /* 일치되는 스트링을 찾을 때까지 알고리즘 1 반복 */
    do {
        matchstring = "" ;  /* 일치되는 스트링 */
        matchstring = MatchString(A, B) ;  /* (알고리즘 1)
                                              호출 */
        Set(totalmatchstring) = Set(totalmatchstring)+
                                matchstring ;
    } while (maxmatch > minlength) ;
    /* 일치되는 스트링의 총 개수 계산*/
    for each matchstring in Set(totalmatchstring)
        matchlength = matchlength + Length(matchstring) ;
    end for
    return (2 × matchlength / (Length(A) + Length(B))) ;
                                /* 유사도 값 계산 및 반환 */
}
```

(알고리즘 2) 트리 유사도 평가 알고리즘

4. 실 험

본 시스템은 Java 언어로 구현되었으며, jdk1.3.1을 이용하였다. 따라서 윈도우 체제나 UNIX 환경에서 모두 활용 가능하며, 프로그램 트리를 구성하기 위하여 구문 분석과 어휘 분석이 가능한 JLex[11, 12]와 Java CUP[13] 유틸리티를 사용하였다.

[실험 1] 서로 다른 두 트리의 유사성 비교

본 알고리즘 설명하기 위하여 (그림 1), (그림 2)의 트리를 비교하면 다음과 같다. (그림 1), (그림 2)는 언파서에 의해서 다음과 같은 노드 스트링 A, W로 변환된다.

t(A) 노드 스트링 A : **x W y z B A d e a**
t(W) 노드 스트링 W : **y A d b A c D e x W y z**

minlength 값이 2로 지정되었다고 가정하고, 유사도를 측정해보면 다음과 같다. 즉, (알고리즘 1)에서는 두 노드 스트링의 일치된 부분을 검사하며, {{A d}, {x W y z}}을 찾아낸다. (알고리즘 2)에서는 (알고리즘 1)에서 찾은 부분 스트링을 matchstring에 추가한 후, A, B 노드 스트링에서 제거된다. (알고리즘 2)에서는 (알고리즘 1)를 이용하여 유사도를 측정한다. 즉, 알고리즘에 의해서 트리 t(A)와 t(W)는 최종적으로 다음과 같은 유사성을 갖는다.

$$sim(A, B) = 2 * \frac{matchlength}{Length(A) + Length(W)} = 2 * \frac{(2+4)}{9+12} = 0.57142$$

[실험 2] 변환된 트리의 유사도 측정

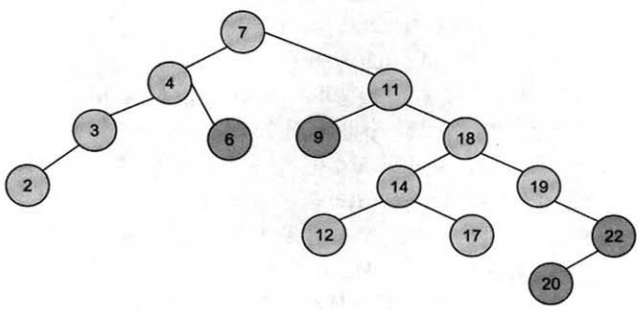
Red-Black 트리[1]과 같이 같은 속성을 유지하면서 다른 형태의 모양을 갖는 트리 역시 본 시스템을 이용하면 유사도 측정이 가능하다. 예를 들면, (그림 5), (그림 6)과 같은 트리 A, B가 있다고 가정하자. 트리 A는 트리 B와 같은 속성을 갖는다. 즉, 트리 B는 트리 A에서 균형(balance)을 유지하기 위하여 왼쪽으로 회전한 트리이다.

트리 A, B는 본 시스템의 언파서에 의해서 다음과 같은 노드 스트링으로 변환된다.

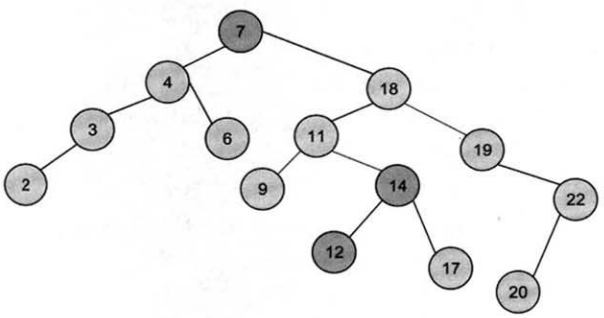
트리 A 노드 스트링 : **2 3 4 7 6 9 11 12 14 17 18 19 20 22**
트리 B 노드 스트링 : **2 3 4 7 6 9 11 12 14 17 18 19 20 22**

minlength 값이 2로 지정되었다고 가정하자. (알고리즘 1)에서는 두 노드 스트링의 일치된 부분을 검사하기 때문에 { **2 3 4 7 6 9 11 12 14 17 18 19 20 22** }을 찾아낸다. 따라서 두 트리 A, B의 유사도 값은 다음과 같다.

$$sim(A, B) = 2 \times \frac{matchlength}{Length(A) + Length(B)} = 2 \times \frac{(14)}{14+14} = 1$$



(그림 5) 원본 트리 A

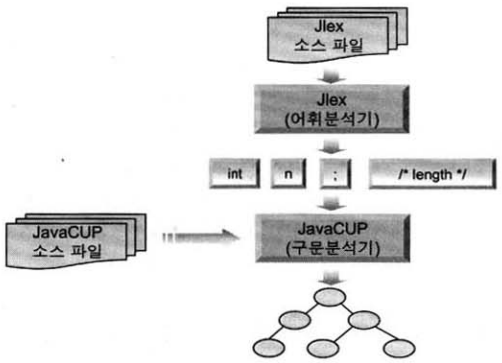


(그림 6) 변환된 트리 B

위의 두 트리는 같은 속성을 유지하며, 다른 형태를 지니고 있기 때문에 변환된 노드 스트링은 같다. 따라서 두 트리의 유사도 값은 1이 되며, 같은 트리임을 나타낸다.

[실험 3] 프로그램 트리의 유사성 비교

본 실험에서는 특정한 프로그램을 트리로 변환하여 유사도 측정을 수행한다. 따라서 본 연구에서는 프로그램을 트리로 변환하는 과정을 수행한다. 즉, 프로그램 언어를 트리로 표현하기 위하여 본 연구에서는 어휘 분석 및 구문 분석을 수행하였다. 어휘 분석은 소스 프로그램을 입력으로 하여 일련의 토큰들로 나누는 일을 한다. 어휘 분석기는 구문 분석기에 의해 필요에 따라 수시로 호출되는데, 이때 한 개의 토큰과 부수적인 정보(토큰 값, 원시 프로그램의 파일명, 원시 프로그램에서의 토큰의 위치를 나타내는 줄의 수 등)를 호출한 구문분석기로 전달한다. 다음 (그림 7)은 프로그램을 트리로 변환하기 위한 과정을 보여준다.

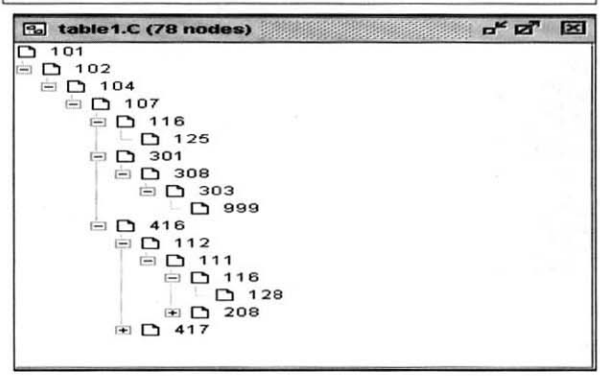


(그림 7) 트리의 생성

예를 들면 다음 표와 같은 C 프로그램 소스 코드가 있다고 가정하자. 이 프로그램의 구문 트리는 다음 (그림 8)과 같이 표현할 수 있다.

〈표 1〉 간단한 C 프로그램

```
void main() {
    int i = 50 ;
    printf("변수 i 값은 %d입니다.", i) ;
}
```



(그림 8) 〈표 1〉의 트리 표현

(그림 8)에서 정수는 트리의 노드 상수이며, 전체 노드 수는 78개의 노드를 가지고 있음을 알 수 있다. 또한 이 트리를 자세히 보기 위해서는 트리의 + 모양(⊞)을 클릭하면 해당되는 하위 트리가 펼쳐지며, - 모양(⊟)을 누르면 해당 트리를 감출 수 있다.

본 시스템의 실험을 위해서 다음 <표 2>, <표 3>과 같은 최소 공배수를 구하는 C 언어 프로그램이 있다고 가정하자. 두 프로그램의 결과는 똑같다. 그러나 두 프로그램은 스타일, 프로그램 구조 등이 다르다. 이 경우 본 시스템에서 제시한 알고리즘은 다음과 같이 효과적으로 유사도를 측정할 수 있다.

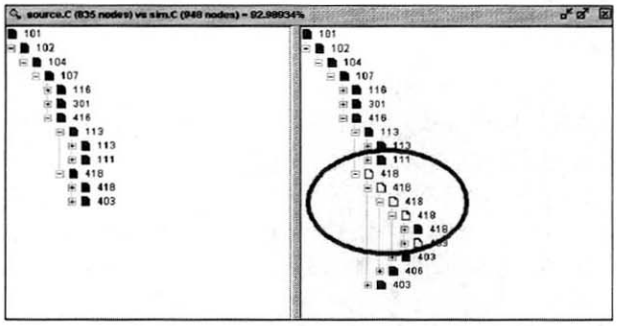
〈표 2〉 solurce.C 프로그램

```
#include < stdio.h >
void main(void) {
        long result ;
        int n1, n2, m1, m2, divs[100], lnum, i, flag ; int index = 0 ;
        printf("Input 2 numbers for calculating GCM…\n") ;
        scanf("%d %d", &n1, &n2) ;
            m1 = n1 ; m2 = n2 ;
        while(1) {
                lnum = (m1 > = m2?m1 : m2) ; flag = 0 ;
                for(i = 2 ; i < = lnum ; i++) {
                        if(m1%i ==0 && m2%i == 0) {
                                m1 / = i ; m2/ = i ; divs[index++]=i ;
                                flag = 1 ; break ;
                        }
                }
                if(flag == 0) break ;
        }
        result = m1 * m2 ;
        for(i = 0 ; i < index ; i++) result * = divs[i] ;
        printf("LCM of %d and %d is %ld.\n", n1, n2, result) ;
}
```

〈표 3〉 sim.C 프로그램

```
#include <stdio.h>
void main(void) {/* 스타일 변화 */
long result; int index=0 ; /* 문장 위치 바꿈 */
int n1, n2, m1, m2, divs[100], lnum, i, flag ;
printf("Input 2 numbers for calculating GCM…\n") ;
scanf("%d %d", &n1, &n2) ; m1 = n1; m2 = n2 ;
for(;;) {/* 제어 구조 변환 while == > for */
flag = 0 ; lnum = (m1 > = m2?m1 : m2) ; /* 문장 바꾸기 */
for(i = 2 ; i < = lnum ; i++) { /* 스타일 바꾸기 */
if(m1%i == 0 && m2%i == 0) {
m1 / = i ; m2 / = i ; divs[index++] = i ; flag = 1 ; break ; }} if(flag
== 0) break ; }
result = m2 * m1 ; /* 오퍼랜드 바꾸기 */ result = m2 * m1 ;
        /* 중복 코드 추가 */
result = m2 * m1 ; /* 중복 코드 추가 */ result = m2 * m1 ; /*
        중복 코드 추가 */
result = m2 * m1 ; /* 중복 코드 추가 */ for(i = 0 ; i < index ;
        i++) result * = divs[i] ;
printf("LCM of %d and %d is %ld.\n", n1, n2, result) ; }
}
```

본 시스템에서 <표 2>와 <표 3>의 프로그램을 비교하면 다음 (그림 9)와 같다. (그림 9)에서 나타난 것처럼 트리의 총 노드 수는 source.C 프로그램이 835개이며, sim.C 프로그램이 948개이다.



(그림 9) source.C와 sim.C의 트리 비교 결과

또한 그림에서 트리의 일치되는 부분은 붉은(■) 노드로 표시 되었으며, 흰색(□) 노드는 두 트리에서 일치되지 않는 것을 나타내고 있다. 두 프로그램 트리의 유사도는 다음과 같다.

$$sim(source.C,\ Sim.C) = 2 * \frac{matchlength}{Length(source.C) + Length(sim.C)}$$
$$= 2 * \frac{(829)}{835+948} = 0.9298934 \text{ 이다.}$$

따라서 두 프로그램(source.C와 sim.C)은 유사도 값 0.9298934 값을 갖는다.

5. 결 론

본 논문에서는 서로 다른 두 트리의 유사도를 평가하는 시스템을 제시하고 구현하였다. 서로 다른 두 개의 트리를 비교하기 위하여 언파서를 이용하였으며, 유사성을 평가하기 위하여 트리 유사도 평가 알고리즘을 제시하였다. 본 논문의 실험 부분에서는 두 개의 트리를 비교하여, 유사도 0과 같거나 크고, 1보다 작거나 같은 값이 나온다는 것을 실험에서 보여주었다. 본 시스템에서 제시한 트리 유사도 평가 알고리즘은 최악의 경우 $O(n^3)$와 최상의 경우 $O(n^2)$의 시간 복잡도를 갖는다.

본 연구는 다양한 형태로 응용이 가능하다. 즉, 트리를 기반으로 하는 개발 환경에서 유사도를 측정하는 연구에 활용이 가능하다. 또한 XML 문서의 유사도 측정, 코드 생성 및 최적화를 이용한 유사도 검사, 웹 기반의 프로그램 복제 검사 시스템 활용 등에 이용할 수 있다. 본 시스템은 노드 10000개 이상인 트리에 대해서 속도가 현저하게 떨어진다는 것을 알 수 있었다. 따라서 알고리즘의 속도 향상 측면에서 연구가 이루어져야 할 것이다.

참 고 문 헌

[1] T. H. Cormen, C. E. Leiserson and R. L. Rivest, "*Introduction to Algorithms*," The MIT Press, 1994.

[2] D. Bryant, "Building Trees, Hunting for Trees, and Comparing Trees," phD thesis, University of Caterbury, 1997.

[3] D. Robinson and L. Foulds, "Comparison of Weighted Labelled Trees," In lecture notes in mathematics, pp. 119-126, Springer-Verlag, Germany, 1979.

[4] V. C. Bhavsar, H. Boley and L. Yang, "A Weighted-Tree Similarity Algorithm for Multi-Agent Systems in E-Business Environments," available at http://www.ruleml.org/indoo/BhavsarBoleyYang-Final.pdf.

[5] M. Steel and D. Penny, "Distributions of tree comparison metrics-some new results," syst. biol., 42(2), pp.126-141, 1993.

[6] Component Web Site available at http://taxonomy.zoology.gla.ac.uk/rod/cpw.html

[7] A. H. B. ledsoe and F. H. Sheldon, "Molecular homology and DNA hybridization," J. Mol. Evol. 30, pp.425-433, 1990.

[8] M. S. Waterman and T. F. Smith "On the Similarity of Dendrograms," journal of theoretical biology, 73, pp.789-800, 1978.

[9] I. D. Baxter, A. Yahin, L. Moura, M. Sant'Anna & L. Bier, "Clone Detection using Abstract Syntax Trees," In proc. of the international Conference on Software Maintenance, Bethesda, Maryland, pp.368-378, Nov., 1998.

[10] A. W. Cole, "Prototype Tree Comparison Tools for Describing File Systems," available at http://www.ldc.upenn.edu/Staff/acole/research/thesis.html, 2002.

[11] E. Berk, "Jlex : A Lexical Analyzer Generator for Java TM," available at http://www.cs.princeton.edy/~appel/ modern/java/JLex/.

[12] J. Lin, "JLex Tutorial," available at http://bmrc.berkeley.edu/courseware/ cs164/spring98/proj/jlex/tutorial.html.

[13] S. E. Hudson, "CUP Parser Generator for Java," available at http://www.cs.princeton.edy/~appel/modern/java/CUP/.

김 영 철

e-mail : yckim@amin.ssu.ac.kr

1990년 한남대학교 전자계산학과(학사)

1996년 숭실대학교 대학원 전자계산학과 (공학석사)

2003년 숭실대학교 대학원 컴퓨터학과 (공학박사)

현재 (주)뉴스텍시스템즈 이사, 명지전문대학 겸임조교수

관심분야 : 프로그래밍 언어, 컴파일러, 망관리, XML

유 재 우

e-mail : cwyoo@computing.soongsil.ac.kr

1976년 학사, 숭실대학교 전자계산학과

1985년 박사, 한국과학기술원 전산학과

1986년~1987년, 1996년~1997년 코넬대학교, 피츠버그대학교 객원교수

1999년~2000년 한국정보과학회 프로그래밍 언어 연구회 위원장

1983년~현재 숭실대학교 컴퓨터학부 교수

관심분야 : 프로그래밍언어, 컴파일러, 인간과 컴퓨터 상호작용