구문트리에서 키워드 추출을 이용한 프로그램 유사도 평가

김 영 철[†] · 최 재 영[††]

요 약

본 논문에서는 프로그램의 분석 과정에서 생성된 구문트리에서 키워드만을 추출하여 유사도 평가하는 방법을 소개한다. 이 방법은 기존의 구조 기반 방법과 같이 프로그램 구조적 특징에 상관없이 유사도를 평가할 수 있으며, 구문트리의 키워드만을 평가에 이용함으로써 기존 시스템의 단점이었던 속도를 개선할 수 있었다. 따라서 본 논문에서는 유사도 평가 모델을 제시하고, 생성된 구문트리에서 키워드를 추출하는 방법을 제시하였다. 본 논문의 평가 부분에서는 기존 시스템에 비해 본 시스템이 구조적 특징이나 속도 면에서 많이 개선되었다는 것을 보여주었다. 따라서 본 시스템은 향후에 텍스트 위주의 문서의 유사도나 XML과 같은 전자 문서의 유사도 평가에 지대한 영향을 줄 것으로 기대된다.

A Program Similarity Evaluation using Keyword Extraction on Abstract Syntax Tree

Kim Young-Chul[†] · Jaeyoung Choi[††]

ABSTRACT

In this paper, we introduce the method that a user analyses the similarity of the two programs by using keyword from the syntactic tree, created after the syntax analysis, and its implementation. The main advantage of the method is the performance improvement through using only keyword of syntax tree. In the paper, we propose the similarity evaluation model and how we extract keyword from syntax tree. In addition, we also show the improvement in the performance in analysis and in the system's structure. We expect that our system will be utilized in the similarity evaluation in text and XML documents.

키워드 : 프로그램 유사도 평가(Program Similarity Evaluation), 키워드 추출(Keyword Extract), 그룹 짓기(Grouping)

1. 서 론

인터넷 매체의 발달과 더불어 많은 자료 및 정보의 공유로 문서의 복제 및 프로그램 코드의 복제는 그 어느 때 보다도 쉽게 자행되고 있다. 이와 대조적으로 복제 방지 대책은 아직도 미흡한 실정이다. 얼마 전 모 신문에서 학생들의 과제물에 대한 비평이 나와 눈길을 끌고 있다. 즉, 학생들이 과제물을 수행하는데 있어서 몇몇의 학생들은 과제물을 수행하는 시간보다 문서나 프로그램 소스 코드를 편집하는 과정에 더 시간을 투자하고 있다는 기사이다[조선일보 2003. 2]. 이러한 기사는 문서나 프로그램 복제가 현재 공공연히 만행되고 있다는 것을 단적으로 보여준다. 특히, 오늘날 정보 공유를 목적으로 한 인터넷이나 다양한 매체의 활성화는 복제를 더욱 쉽게 만들고 있다. 이처럼 여러 가지 현실적/환경적 요인으로 볼 때 문서나 프로그램의 소스 코드의 복제 여부를 판단은 아주 중요한 요소가 되었다. 특히, 두 프로그램의 복제 여부를 자동으로 판단하는 일은 그 어느 때 보다도 절실해지고 있으나 그 대비책은 여전히 미흡한 상태이다. 또한 프로그램의 복제 여부를 판별하는 일은 단순히 복제 방지 차원에서 뿐만 아니라 정보의 유용성 및 소프트웨어 공학적 도구로 이용할 가치도 있다[1,2].

J. R. Edlun[3]은 자신의 연구 논문에서 프로그램 복제를 "타인의 프로그램 소스코드를 복사해서 간단한 편집과정을 거치거나 이미 만들어진 원저자의 참조 사항을 달지 않고 이용하는 것"이라고 정의하였다. 또한 Hamblen[4]은 프로그램 소스 코드의 복제(plagiarism in software)를 "한 프로그램에 약간의 루틴의 변형을 가해서 만들어내는 프로그램"이라고 정의하고 있다. 여기서 변형이라는 의미는 원본 소스 코드의 주석문을 바꾸거나 변수의 데이터 타입, 식별자(identifier)를 수정하는 것과 중복된 구문이나 불필요한 변수를 삽입하는 것을 말한다. 또한 원본 소스 코드를 논리적인 흐름에 영향을 주지 않도록 단순하게 구조를 뒤섞는 일도 이러한 변형에 포함된다.

프로그램이나 문서의 복제 여부를 판단하는 연구는 크게

※ 본 연구는 2004년도 한국학술진흥재단의 지원으로 연구되었음. (KRF-2004-005-D00172)
† 정 회 원 : 숭실대학교 정보미디어기술연구소 연구원
†† 종신회원 : 숭실대학교 컴퓨터학부 교수
논문접수 : 2004년 10월 21일, 심사완료 : 2005년 3월 7일

2가지 부류 즉, 지문법(fingerprint)[6]과 구조 기반(structure-based) 방법으로 나뉘어 진다. 지문법은 단어나 키워드 혹은 특수문자, 공백, 토큰 등이 포함된 횟수를 통계적으로 표현하여 비교검사를 수행하는 방법이며, 구조 기반 방법은 프로그램의 구조를 기반으로 하여 검사하는 방법이다. 이 두 가지 방식은 서로 상반된 장단점(trade off)을 가지고 있다. 즉, 지문법은 프로그램의 길이에 민감하지 않고 유사도를 평가할 수 있는 반면에 프로그램의 구조를 평가할 수 없는 단점을 가지고 있다. 이와 반대로 프로그램 구조 기반 방법은 프로그램의 길이에 민감하지만, 프로그램의 구조적인 유사도 평가를 할 수 있다. 따라서 본 논문에서는 이 두 방식의 장점만을 혼용한 새로운 방식을 제시하고자 한다. 즉, 본 논문에서 제시한 유사도 평가방법은 구문트리를 이용함으로써 프로그램 구조를 평가할 수 있으며, 단점인 프로그램의 길이에 민감하지 않고 키워드만을 추출하여 비교하고 평가할 수 있다.

본 논문은 다음과 같이 구성되었다. 제2장에서는 관련연구에 대해서 기술하였으며, 제3장은 본론 부분으로 본 논문에서 제시된 유사도 평가 시스템 모델, 프로그램 유사도 평가 및 그룹 짓기, 구문 트리 추출 과정에 대해서 기술하였다. 제4장에서는 실험 및 평가를 기술하였으며, 마지막으로 제5장에서는 결론에 대해서 기술하였다.

2. 관련 연구

초기의 프로그램 유사도 평가 방법은 지문법을 이용한 시스템이 많이 있다. 지문법을 기반으로 만들어진 최초의 시스템은 Halstead 매트릭스[7] 시스템이 있으며, 이 매트릭스를 이용하였거나 확장해서 만든 시스템은 Ottenstein[8], Donaldson[9], Berghel[10] 시스템 등이 있다. 특히, Donaldson이 제시한 복제 방법은 Halstead 매트릭스 외에, 반복문의 수, 프로시저문의 수와 같은 프로그램 구조를 혼합하여 이용했다는 점도 주목할 만하다. 이러한 지문법은 프로그램이나 문서의 길이에 영향을 받지 않는다는 특징이 있으며, 지문을 추출해내기만 하면 이 지문들끼리 비교를 통해 복제 여부를 판단하기 때문에 시간 복잡도도 낮다. 그러나 지문법은 부분적인 복제 여부를 찾아내기가 어렵고, 구조적인 특징을 가지는 프로그램 코드의 검사에 적용하기에는 신뢰성이 부족하다. 또한 문서나 프로그램의 지문은 추출해 낼 수는 있지만, 구조적인 분석은 어려워진다. 특히, 문서와는 달리 프로그램은 제어 흐름의 변경이 어려우며, 프로그래밍 언어의 문법이 정해져 있기 때문에 구조적인 특성이 잘 나타난다. 따라서 구조 기반 복제 검사 방법은 프로그램 소스 코드의 복제 검사에 많이 이용된다[11].

구조 기반 복제 검사 방법은 대부분 언어의 예약어(토큰)나 함수의 호출 순서 등을 이용하여 복제 검사를 수행한다. 프로그램 "토큰"을 이용하면 지문법을 이용한 시스템과는 달리 프로그램 스타일이나 설명문, 들여쓰기 등의 프로그램 구문과 상관없는 요소에 민감하지 않다는 특징을 가지고 있다. 프로그램의 토큰을 이용하여 유사도를 평가하는 시스템은 대표적으로 YAP3[12], MOSS[13], Jplag[14], Sim[15], SID[16] 등이 있다. 다음 <표 1>은 지문법을 이용하는 시스템과 구조 기반 방법을 이용한 시스템의 비교한 표이다.

〈표 1〉 지문법과 구조 기반 방법의 비교(출처:[11])

설명	지문법	구조기반 방법
문서에서 얻어내는 정보	유동적	고정적
문서의 길이	영향을 받지 않음	길이에 따라 얻어내는 정보가 비례함
부분적인 복제 탐지	어려움	쉬움
시스템	Ottenstein, Donaldson, Berghel	CHECK, Plaque, YAP, MOSS, JPlag, SIM, SID

또한 토큰 스트링을 이용하는 방법 중 생물 정보학을 이용한 연구가 있었다. 즉, 유전체의 서열들을 비교하여 유사한 영역을 찾아내는 서열 분석 방법[11, 17]이다. 유전체 서열 분석 연구는 DNA 서열이나 단백질 서열들을 비교해서 유사한 영역을 찾아내는 것인데, 이것은 동일한 서열을 가지는 유전자는 같은 기능을 가진다는 가정 하에 동종이나 타종의 유전체 서열의 기능 분석에 사용된다. 즉, 프로그램의 소스코드의 구조와 특성을 나타내는 정보를 추출하여 생물학에서 이용되는 여러 가지 정렬 시스템을 사용하여 비교한다. 다음 <표 2>는 소스코드를 아미노산 서열로 대응시키는 과정을 보여준다.

〈표 2〉 소스코드에서 키워드 추출한 후 유전체 서열로 바꾸는 과정

소스코드의 예	추출되는 키워드	대응되는 염기서열

#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;
}
	void void {
	long
	int
	int =
	
	= =
	}
		V V D
		L
		I
		I S
		
		S S
		W

표 2에서처럼 소스 코드에서 추출되는 부분은 "{ void void { long int int = = = } }"이다. 따라서 이를 유전체 서열로 바꾸면 "VVDLIISSSW"이 된다.

프로그램과 관련된 국내의 연구는 [5], [11] 등이 있다. 특히, [5]에서 제시한 트리비교 알고리즘은 어휘 분석과 구문 분석 과정을 통하여 생성된 구문트리를 이용한다. 구문 트리를 이용하는 방법은 기존의 매트릭스나, 토큰 스트링을 이용한 시스템과 달리 같은 많은 장점을 가질 수 있다는 점을 보여주었다. 그러나 만일 복제 검사될 프로그램 크기가 커지면 커질수록 비교될 구문트리의 노드 수가 증가하므로

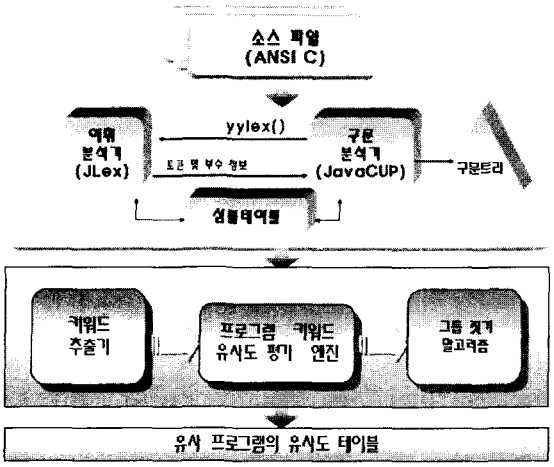
복제 검사하는데 많은 시간이 소요된다. 또한 많은 수의 프로그램이 있는 경우, 모두 일일이 비교하는데 필요한 비교 횟수는 n개의 프로그램에 대하여 n(n-1)/2 번의 비교횟수가 소용된다. 이처럼 [5]에서 제시한 방법은 프로그램이 크거나 비교될 프로그램의 개수가 커질 경우 단점을 가지고 있다.

[11]에서는 프로그램의 호출 순서를 고려한 복제 여부 판정을 하는 시스템이다. 이 논문에서는 소스 프로그램에 나타난 호출 순서대로 프로그램을 재구성하여 CodeDNA를 구성하는 함수 호출 선형화 방법을 사용하였다. 즉, 프로그램 소스 코드에 나타나는 함수 순서를 재배치하여도 프로그램에서 함수 호출 순서가 바뀌지 않으므로 최종 추출되는 CodeDNA에는 영향을 미치지 않는다는 점을 이용한 것이다. 또한 프로그램 소스코드에는 있는 함수이지만, 실제로 수행 중에 한 번도 호출되지 않는 함수일 경우에는 함수 호출 순서에서 제외되므로 CodeDNA에는 포함되지 않는다. 따라서 프로그램 수행 중에 호출되지 않는 불필요한 함수의 시퀀스가 제거된다. 함수 호출 선형화 방법을 사용함으로써 의미 있는 코드들이 의미있는 순서대로 배치되어서 최종 CodeDNA가 구성되는 점을 이용하였다.

3. 본 론

3.1 유사도 평가 시스템 모델

본 논문에서 제시한 프로그램 복제 검사 시스템 모델은 다음 그림 1과 같다. 그림 1은 크게 2 부분 즉, 구문트리 생성단계와 유사도 평가 단계로 나누어진다. 구문트리 생성단계는 기존의 컴파일 단계의 앞부분(front-end)에 해당되며, 어휘 분석 및 구문 분석에 해당된다. 구문 분석이 끝나면 프로그램의 구문 트리가 생성된다. 따라서 본 시스템은 이와 같은 파싱과정을 거치므로 프로그램 오류를 평가할 수 있는 장점이 있다. (그림 1)의 시스템 모델에서 두 번째 단계인 유사도 평가 단계는 키워드 추출기, 프로그램 유사도 평가 엔진, 유사도 그룹짓기 단계로 이루어져있다. 키워드 추출기는 파싱 과정에서 생성된 구문트리(AST)에서 키워드



(그림 1) 키워드 추출을 이용한 유사도 평가 시스템 모델

를 추출하는 단계로 유사도 평가 엔진에 입력된다. 따라서 유사도 평가 엔진은 추출된 키워드만을 가지고 유사도를 평가하기 때문에 구문트리의 노드 수가 줄어들어 구조 기반 시스템의 단점이었던 속도 부분을 현저하게 줄일 수 있다. 또한 유사도 그룹짓기는 프로그램 유사도가 높은 프로그램을 서로 묶을 수 있는 단계로 [18]에서 제시된 알고리즘을 이용한다. 따라서 본 논문의 최종 단계는 많은 프로그램이 입력되어 유사도가 높은 프로그램을 테이블로 가질 수 있도록 설계되었다.

3.2 유사도 평가 알고리즘과 그룹 짓기

본 시스템은 [5]에서 제시한 유사도 평가 알고리즘을 이용하였다. [5]에서 제시한 알고리즘은 크게 3 단계를 거친다. 즉, 첫 번째 단계는 입력된 두 개의 노드스트링에서 일치하는 스트링을 검색하며, 가장 길게 일치되는 스트링을 찾는다. 따라서 가장 긴 스트링을 찾았다면, 찾은 스트링을 보관하고 노드 스트링에서 제거하는 과정이 두 번째 단계이다. 마지막으로 세 번째 단계에서는 일치된 스트링을 이용하여 유사도를 평가하는 단계이다. 다음 알고리즘은 이 과정을 개략적으로 보여준다.

```
double Sim(NodeString A, NodeString B, long int minlength) {
        String matchstring, totalmatchstring;
        int maxmatch = 0;
        long int matchlength = 0;
        Set(totalmatchstring) = {}

        do {
           matchstring = "";
           matchstring = MatchString(A, B);
           Set(totalmatchstring) = Set(totalmatchstring) + matchstring;
         } while (maxmatch > minlength);
    for each matchstring in Set(totalmatchstring)
        matchlength = matchlength + Length(matchstring);
    end for
return (2 * matchlength / (Length(A) + Length(B)))
}
```

위에서 제시한 알고리즘은 프로그램의 사이즈가 크면 클수록 노드 스트링 수가 증가하여 유사도를 평가하는데 시간이 많이 필요하다. 그러나 본 시스템은 [5]에서 제시한 방법을 수정하여 평가하였다. 즉, 노드 스트링 중 키워드에 해당되는 부분을 추출하여 유사도 평가에 이용하였다. 따라서 노드 수가 현저하게 줄기 때문에 유사도를 평가하는 시간이 많이 단축되었다는 것을 평가에서 보여준다.

또한 본 논문에서는 많은 프로그램을 대상으로 유사한 프로그램을 묶는 그룹 짓기를 수행하였다. 본 시스템이 이용한 그룹 짓기 알고리즘은 [18]에서 제시한 알고리즘을 이용하였으며, 다음과 같다.

```
file *P; /* 비교할 프로그램 입력 */
int  g;  /* 전역 유사성 값 입력 */
boolean addgroup = false;  /* 그룹에 추가되었는지에 대한 flag */
```

```
add P to G(1);      /* 첫 번째 비교 프로그램은 무조건 그룹에 추가 */

i =1;  /* 그룹 개수 나타내는 계수 */
Set G(1) = ∅        /* 그룹 초기화 */
while( not eof) {
        input P;                /* 비교 대상 프로그램 입력 */
        for each i in G(i)      /* 모든 그룹에 대해서 수행 */
             if Sim(P, G(i)) > g then   /* 그룹에 포함되는 경우 */ {
                  add P to G(i); /* 그룹에 추가 */
                  addgroup = true;
             }
        end for

        /* 그룹에 추가가 안 되었으면 그룹 생성 */
        if (not addgroup) then {
             i = i + 1;  /* 그룹 카운터 추가 */
             Set G(i) = ∅    /* 그룹 초기화 */
             add P to G(i);   /* 모두다 포함이 안 될 경우 */
        }
}
```

위의 그룹 짓기 알고리즘은 비교할 대상 프로그램이 입력되면 이미 그룹화되어 있는 프로그램과 비교를 수행한다. 비교한 유사성이 지정된 유사도보다 낮으면 다른 그룹과 비교하며, 만일 지정된 유사도보다 프로그램 유사도가 높다면 프로그램을 해당 그룹에 포함시킨다. 또한 모든 그룹의 프로그램보다 유사도가 낮다면 새로운 그룹으로 포함시킨다. 이 과정은 프로그램 입력이 끝날 때까지 반복적으로 수행된다. 일반적으로 n개의 프로그램에 대해서 모두 비교하는데 걸리는 시간은 n(n-1)/2 번이다. 그러나 위에서 기술된 알고리즘을 이용하면, n-1번 ~ n(n-1)/2 번까지 줄일 수 있다. 즉, n-1번이란 n개의 프로그램이 모두 복제되었을 경우에 해당되며, n(n-1)/2는 모두 복제가 되지 않았음을 의미한다. 따라서 복제된 프로그램이 많으면 많을수록 그룹 수가 적어지므로 n개의 프로그램을 모두 비교할 수 있는 횟수는 크게 줄어든다. 반면, 복제되지 않은 프로그램이 많으면 많을수록 그룹 수가 증가하여 비교할 횟수가 많아진다.

3.3 구문트리에서 키워드 추출하기

본 논문은 [5]에서 제시하고 있는 방법과는 달리 파싱과정에서 생성된 구문트리에서 키워드에 해당되는 부분을 추출하여 유사도를 평가한다. 즉, [5]에서는 파싱과정에서 생성된 구문트리를 역파서에 의해 노드 스트링으로 변환한 후, 그 스트링을 가지고 유사도를 평가한다. 그러나 프로그램에서 한 문장이 약 20-30 여개의 노드를 생성된다면, 전체 프로그램의 크기가 크면 클수록 유사도를 평가하기가 어렵다. 이처럼 프로그램의 길이가 길어지면 길수록 많은 노드 스트링이 발생하여 유사도를 평가하는데 많은 시간을 갖는 단점을 가지고 있다. 예를 들어 표 2와 같은 프로그램 조각은 [5] 알고리즘을 이용하면 다음과 같은 많은 수의 노드 스트링을 발생한다. 다음 알고리즘은 본 논문에서 이용된 키워드 추출 알고리즘은 다음과 같이 아주 간단하다.

〈표 2〉 키워드 추출을 이용한 노드 스트링의 예

[5]의 입력 노드 스트링	본 시스템의 입력 노드 스트링
[■NProgram]	[■NProgram]
[■NTransUnit]	[■NTypeSpecVoid] // void
[■NExterDeclFuncDef]	[■NDirectDeclIdent] // main
[■NFuncDefDeclCompStmt]	[■NTypeSpecVoid] // void
[■NDeclSpecTypeSpec]	[■NTypeSpecLong] // long
[■NTypeSpecVoid]	[■NDirectDeclIdent] // result
[■NDeclDirectDecl]	[■NTypeSpecInt] // int
[■NDirectDeclParamTypeList]	[■NDirectDeclIdent] // n1
[■NDirectDeclIdent]	[■NDirectDeclIdent] // n2
[■NSTR] main	[■NDirectDeclIdent] // m1
[■NParamTypeList]	[■NDirectDeclIdent] // m2
[■NParamList]	[■NDirectDeclIdent] // divs
[■NParamDeclDeclSpec]	[■NConstInt] // 100
[■NDeclSpecTypeSpec]	[■NDirectDeclIdent] // lnum
[■NTypeSpecVoid]	[■NDirectDeclIdent] // i
[■NCompStmtDeclListStmtList]	[■NDirectDeclIdent] // flag
[■NDeclListList]	[■NTypeSpecInt] // int
[■NDeclListList]	[■NDirectDeclIdent] // index
[■NDeclList]	[■NEqlExpr] // =
[■NDeclDeclSpecList]	[■NConstInt] // 0
[■NDeclSpecTypeSpec]	[■NPriExprIdent] // printf
[■NTypeSpecLong]	[■NSTR]// "Input 2 numbers for calculating GCM...\n"
[■NInitDeclList]	[■NPriExprIdent] // scanf
[■NInitDecl]	[■NSTR] // "%d %d"
[■NDeclDirectDecl]	[■NPriExprIdent] // m1
[■NDirectDeclIdent]	[■NEqlExpr] // =
[■NSTR] result	[■NPriExprIdent] // n1
... (이하 생략)	[■NPriExprIdent] // m2
	[■NEqlExpr] // =
	[■NPriExprIdent] // n2

```
file *t; /* 노드 스트링의 포인터 */
Set G(n) = ∅ /* 추출된 노드 스트링 초기화 */
while( not null) {
        if (G(k) ∈ *t) then add *t to G(n);
        t++;
}
```

위의 알고리즘에서 집합 G(k)는 프로그램의 키워드 집합을 의미한다. 또한 G(n)은 추출된 구문트리 노드를 갖는 집합이다. 따라서 위 알고리즘은 이미 정의되어 있는 G(k)에 노드 t가 있다면 유사도 평가될 노드 스트링의 집합 G(n)에 추가하기만 하면 된다. 이 과정은 노드 스트링이 끝날 때까지 반복 수행된다.

<표 3>은 <표 2>의 예문을 평가하기 위한 구문트리 노드 스트링이다. <표 3>의 왼쪽은 생성된 노드 스트링이며, 오른쪽은 본 시스템에서 제시한 키워드 추출 방법을 이용한 노드 스트링이다. <표 3>은 본 시스템의 장점을 잘 보여주고 있다. 즉, 왼쪽 영역은 <표 2>의 노드 스트링에 해당되며, 오른쪽 스트링은 본 시스템에서 제시한 키워드 추출 방법을 이용한 노드 스트링이다. <표 2>와 같은 아주 조그만 프로그램 조각이라도 한 문장에 20-30개의 노드가 발생되어 1000라인 이상되면, 무수히 많은 노드가 발생되어 비교하는데 많은 시간을 소비한다. 또한 1만 라인 이상되면, 알고리즘은 치명적이다. 그러나 본 시스템에서 제시한 키워드 추출방법을 이용하면, 1/20~1/30 정도의 노드수가 감소하므로 알고리즘을 효율적으로 이용할 수 있다. 키워드를 추출하는 방법은 프로그램의 구조에 영향을 미치지 않는다. 왜냐하면, 구문 트리에서 중간 노드들은 의미 분석에서 속성의 값을 전달하기 위해 이용되는 노드들이기 때문에 키워드를 추출한다해도 프로그램의 제어구조에는 별다른 영향을 미치지 않는다. 따라서 이 노드를 그대로 이용하지 않고 프로그램의 속성과 관련된 노드만 이용하여 유사도를 평가하면 대단히 경제적이고 빠르게 유사도를 평가할 수 있다. 따라서 본 논문에서는 생성된 구문트리에서 프로그램 소스 코드와 직접 관련있는 키워드, 문장, 수식, 오퍼레이터 등을 추출하여 이용한다. 본 논문의 평가 부분에서는 노드 수와 시간적 관점에서 실험한 부분을 잘 보여준다.

4. 실험 및 평가

본 시스템은 Java 언어로 구현되었으며, jdk1.3.1을 이용하였다. 따라서 윈도우 체제나 UNIX 환경에서 모두 활용가능하며, 1989년에 제정된 ANSI C 언어로 작성된 프로그램을 복제 검사할 수 있다. 또한 C 언어의 어휘 분석과 구문 분석을 수행하기 위하여 본 시스템에서는 JLex[19]와 Java CUP[20] 유틸리티를 사용하였다. 본 논문의 평가 부분에서는 [18]에서 실험된 10개의 프로그램에 대해서 평가되었다.

4.1 노드 수와 시간에 따른 분석 실험

본 시스템은 [5]에 비해 노드 스트링 수를 많이 줄여서 빠르게 수행되도록 제안되었다. 따라서 본 절에서는 실제로 노드가 몇 개까지 줄일 수 있는지 평가해 보았다. 본 절에서 평가된 프로그램은 [18]에서 제시한 프로그램을 가지고 평가했다. 따라서 본 절에서 나타낸 노드는 프로그램의 구문트리를 선형적으로 정렬하여 나타낸 노드 스트링이며, 실제로 유사도를 측정하기 위한 노드의 수를 나타낸다. 또한 평가된 시간은 평가 알고리즘의 시작 부분과 끝부분에 타이머(timer)를 두어 계산된 시간으로 단위는 초(second)이다. 다음 <표 4>는 이렇게 평가된 노드와 시간을 보여준다. 표 4를 보면 본 시스템의 장점을 잘 볼 수 있다. 즉, 노드 수는 보통 1/10~1/30 정도 감소하였으며, 이에 따른 시간도 많이 단축되었다는 것을 알 수 있다. 또 하나는 특이한 것은 노드 수에 비해 시간이 정확히 비례하지 않는다는 점이다. 이것은 일치되는 노드 스트링이 많으면 많을수록 시간이 더 단축된다는 사실이다. 따라서 본 실험에서 나타나듯이 키워드를 추출하여 평가한 결과 노드 수와 시간이 많이 단축된 것을 볼 수 있다.

<표 4> 프로그램 노드 수와 속도 비교

프로그램	노드 수(count.h)		평가된 시간(time.h) (단위 : Sec)		비고
	[18]에서 제안된 시스템	본 시스템	[18]에서 제안된 시스템	본 시스템	
1234.c	1120	75	129.834	8.746	
2486.c	?	?	?	?	구문 오류
3469.c	987	70	101.734	5.450	
4035.c	994	71	98.935	5.089	
4039.c	912	60	78.864	6.840	
4078.c	1220	82	135.396	9.606	
4129.c	1328	95	138.345	9.706	
4160.c	890	64	70.849	7.847	
4560.c	1009	72	108.592	8.652	
7180.c	1323	94	149.47	9.748	

4.2 그룹 짓기 수행

본 절에서는 [18]에서 평가되었던 유사 프로그램의 그룹 짓기를 수행하여 비교 분석해 보았다. 실험에 이용된 프로그램은 모두 10개였으며, 전역 유사도는 0.9로 설정하여 그룹 짓기를 수행해본 결과 다음 <표 5>와 같았다.

<표 5> 전역 유사도 0.9로 설정하여 그룹 짓기를 수행한 결과

Group	Group0	Group1	Group2	Group3	Group4	Group5
Group0		0.68676	0.834566	0.894335	0.7676	-1
Group1			0.794643	0.68555	0.555234	-1
Group2				0.833433	0.693454	-1
Group3					0.833535	-1
Group4						-1
Group5						

(Group0 : 4160.c, 4039.c Group1 : 4129.c Group2 : 4035.c, 3469.c
Group3 : 4078.c, 4560.c, 1234.c Group4 : 7180.c Group5 : 2486.c)

위의 결과는 다음 <표 6>과 비교해 보면 거의 유사하다는 것을 알 수 있다. <표 6>은 [18]에서 평가하였던 그룹 짓기 결과이다.

〈표 6〉 전역 유사도 0.9로 설정하여 그룹 짓기를 수행한 결과

Group	Group0	Group1	Group2	Group3	Group4	Group5	Group6
Group0		0.666908	0.855566	0.898305	0.752316	-1	0.848864
Group1			0.786918	0.663991	0.52589	-1	0.604669
Group2				0.84	0.691122	-1	0.76459
Group3					0.812539	-1	0.891705
Group4						-1	0.854806
Group5							-1
Group6							

(Group0 : 4160.c, 4039.c Group1: 4129.c Group2 : 4035.c, 3469.c
Group3 : 4078.c, 4560.c Group4 : 7180.c Group5 : 2486.c Group6 : 1234.c)

<표 5>와 <표 6>은 약간의 차이를 보이고 있다. 즉, 표 4에서는 프로그램 10개에 대하여 유사한 그룹이 6개로 되었다. 이는 표 5에서 보는 그룹 6이 그룹 3에 포함되었기 때문이다. 그러나 나머지는 거의 유사한 형태의 그룹 결과를 보이고 있다. 따라서 본 시스템은 이전 시스템에 비해 거의 유사한 결과를 가지며, 대신 비교할 노드 스트링을 1/15 만큼 줄이며, 많은 속도 향상을 이루었다는 것을 알 수 있었다.

4.3 평 가

본 논문에서는 기존의 구문트리를 이용하여 유사도를 평가하는 방법의 단점을 보완하기 위하여 키워드 추출 방법을 이용하였다. 즉, 파싱과정에서 생성된 노드 스트링을 유사도 평가에 그대로 이용하지 않고, 프로그램 키워드만을 추출하여 유사도 평가에 이용하였다. 따라서 기존에 단점이었던 노드 스트링 개수에 비례하는 시간을 많이 줄일 수 있다는 것을 4.1절에서 보여주었다. 이처럼 본 시스템은 기존의 지문법이나 프로그램 구조 기반 방법보다 많은 장점을 가지고 있다. 다음 <표 7>은 지문법과 프로그램 구조 기반 방법 및 본 시스템의 특징 및 장단점을 잘 보여준다. 특히, 구조 기반 방법에 비해 속도 개선, 구문 오류 탐지, 프로그램 구조에 대해서 민감하지 않고 찾아낼 수 있다는 장점을 보여준다. 속도 면에서 살펴보면, 본 시스템은 기본적으로 구조 기반 방법을 택하고, 유사도 평가 시에는 지문법과 같은 방법을 이용함으로써 프로그램의 길이에 상관없이 빠르게 유사도를 평가할 수 있다. 또한 본 시스템은 파싱과정에서 생성된 구문 트리를 이용하기 때문에 프로그램이 오류가 있는지 없는지 여부를 검사할 수 있다는 장점을 가지고 있다. 또한 복제 유형 중에서 함수나 문장의 재배치에 장점을 보이고 있다. 이것은 본 시스템이 파싱과정을 거치므로 함수의 위치에 상관없이 최종적으로 단하나의 구문트리가 완성되므로 위치와는 전혀 무관하게 유사도를 판별할 수 있기 때문이다. 그러나 이러한 장점과는 <표 7>에서 보여주듯이 단점도 가지고 있다. 프로그래머가 의도적으로 여분 코드(dummy code)를 삽입할 시 많은 수의 노드가 발생하여 유사도가 많이 떨어진다. 따라서 향후에 여분 코드 검사기(dummy code checker)를 통해 여분 코드를 제거한 후에 유사도를 검사하여야 할 것이다.

5. 결 론

본 논문에서는 서로 다른 두 프로그램의 유사도를 평가하는 시스템을 제시하고 구현하였다. 본 시스템은 파싱과정에

〈표 7〉 프로그램 유사도 평가 방법과의 비교

특 징		지문법	프로그램 구조 기반	본 시스템
장 점		- 프로그램의 길이에 큰 영향이 없음	- 부분 탐지 쉬움 - 구조적 분석 가능	- 부분 탐지 쉬움 - 구조적 분석 가능
단 점		- 신뢰성 부족 - 구조적인 분석이 어려움 - 부분적임 복제 탐지 어려움	- 프로그램 길이에 따라 속도가 비례함 - 여분 코드에 미약 - 제어 구조 분석의 어려움	- 여분 코드에 미약
속 도		- 빠름	- 프로그램 길이에 따라 비례함	- 빠름
복제 유형	원본 복제,여백(white space, indent) 변화	- 대체로 정확함	- 정확히 찾음	- 정확히 찾음
	설명문 추가, 변수, 함수 이름 바꾸기	- 보통	- 정확히 찾음	- 정확히 찾음
	불필요한 문장추가	- 취약	- 취약	- 취약
	함수 재배치, 문장 재배치	- 취약	- 취약	- 정확히 찾음
	제어 구조 바꾸기	- 취약	- 유사도 판별	- 유사도 판별
특징		지문 추출 쉬움		구문 오류 자동 검사

서 생성된 구문트리를 분석하여 프로그램에 이용되는 키워드를 추출해서 유사도를 평가하였다. 그 결과 기존의 단점이었던 프로그램 구조 기반 방법의 속도 개선을 이루었다는 것을 실험에서 보여주었다. 따라서 본 논문은 기존의 지문법과 구조 기반 방법의 장점만을 이용한 방법으로 볼 수 있다. 즉, 프로그램 길이에 상관없이 지문을 추출하여 유사도를 평가하는 지문법의 장점을 갖고, 구조 기반 방법에서 취한 프로그램 구조를 이용하여 유사도를 평가하는 방법을 혼용함으로써 두 가지 방법의 단점을 해결할 수가 있었고, 기존의 다른 방식에 비해 많은 특징을 가진다는 것을 평가에서 보여주었다. 또한 [18]에서 제시한 그룹 짓기를 수행함으로써 유사도를 평가하는데 이용되는 노드 수와 속도를 많이 개선할 수 있었다는 것을 보여주었다.

본 연구의 향후 과제로는 평가에서 언급한 중보 코드 검사기의 수행 및 문서의 유사도를 들 수 있다. 즉, 불필요한 변수나 문장 등을 추가한 여분 코드(dummy code)에 문제점을 드러내고 있으므로, 이를 줄일 수 있는 방법을 연구해야 할 것이다. 또한 본 연구를 기반으로 프로그램의 유사도 평가 외에 일반 문서나 XML 문서와 같은 다른 분야에 응용하여 유사도를 평가하는 것이 필요할 것이다. 또한 본 시스템을 더 확장하면, 프로그램 스타일 분석기나 웹 기반의 평가시스템 등에 활용할 수 있을 것이다.

참 고 문 헌

[1] J. K. Harris, "Plagiarism in Computer Science Courses", *In proc. Ethics in Computer Age*, pp. 133-135, 1994.

[2] M. Joy & M. Luck, "Plagiarism in Programming Assignments", *IEEE Transaction in Education*, 42(2), pp. 129-133. 1999.

[3] J. R. Edlun, "What is "Plagiarism" and why do people do it?", available at http://www.calstatela.edu/centers/write_cn/plagiarism.htm, University Writing Centre Director, California State University, LA, 1998.

[4] J. O. Hamblen & Parker, "Computer Algorithm for Plagiarism Detection", *IEEE Transactions on Education* 32(2), pp. 94-99, May., 1989.

[5] 김영철, 유재우, "유사도 평가를 위한 트리 비교 알고리즘", 한국정보처리학회 논문지, 11-A(2), Mar. 2004.

[6] J. H. Jonson, "Identifying Rddundancy in Source Code using Fingerprints"., *In proc. of CASCON* 93, pp. 171-183, 1993.

[7] M. Howard & Halstead, *Elements of Software Science*, Elsevier, 1977.

[8] K. J. Ottenstein, "an Algorithmic Approach to the Detection and Prevention of Plagiarism". *ACM SIGSCE Bulletin*, 8(4), pp. 30-41, 1976.

[9] J. L. Donaldson & A. M. Lancaster "A Plagiarism Detection System", *ASM SIGSCE Bulletin(proc. of 12th SIGSCE Technical Symp.)*, 13(1), pp. 21-25, Feb., 1981.

[10] H. L. Berghel & D. L. Sallach, "Measurements of Program Similarity in Identical Task Environments". *ACM SIGPLAN Notices*, 19(8), pp. 65-76, Aug., 1984.

[11] 황미녕, 강은미, 조환규. "유전체 서열의 정렬 기법을 이용한 소스 코드 표절 검사", 제21회 정보과학논문경진대회 입상작, 2002.

[12] M. J. Wise, "Detection of Similarities in Student Programs: YAP'ing may be Preferable to Plague'ing". *ACM SIGSCE Bulletin(proc. of 23rd SIGCSE Technical Symp.)*, 24(1), pp. 268-271, Mar., 1992.

[13] A. Aiken, "MOSS(Measure Of Software Similarity) Plagiarism detection system", available at http://www.cs.berkeley.edu/~moss/, University of Berkeley, CA, Apr., 2000.

[14] L. Prechelt, G. Malpohl & M. Philppsen, "JPlag: Finding Plagiarism Among a Set of Programs", available at http://wwwipd.ira.uka.de/EIR/ *D-76128 Karlsruhe, Germany, Technical Report 2000-1*, Mar., 2000.

[15] D. Gitchell & N. Tran, "Sim: A Utility For Detecting Similarity in Computer Programs", available at ftp://ftp.cs.vu.nl/pub/dick/similarity_tester/, *In proc. of 30th SCGCSE Technical Symp.*, pp. 266-270, New Orleans, USA, 1998.

[16] X. Chen, M. Li, B. Mckinnon & A. Seker, "A Theory of Uncheatable Program Plagiarism Detection and Its Practical Implementation", University of California, Santa-Barbara, May., 2002.

[17] X. Chen, S. Kwong & M. Li, "A Compression Algorithm for DNA Sequence and its Applications in Genome Comparion", *In Proc. of the20th Workshop on Genome Information*, pp. 52-61, 1999.

[18] 유재우, 김영철. "프로그램 유사도 평가를 이용한 유사 프로그램 그룹짓기". 한국정보과학회 논문지, 31(1), Jan. 2004.

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

[20] 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@ss.ssu.ac.kr

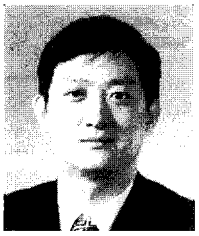
1990년 한남대학교 전자계산학과 졸업 (학사)

1996년 숭실대학교 전자계산학과 석사

2003년 숭실대학교 컴퓨터학과 공학박사

현재 숭실대학교 정보미디어기술연구소 연구원, 명지전문대학 겸임교수

관심분야 : 프로그래밍 언어, 컴파일러, XML, 컴퓨터 통신, 소프트웨어공학



최 재 영

e-mail : choi@ssu.ac.kr

1984년 서울대학교 제어계측공학과(학사)

1986년 미국 남가주대학교 전기공학과(석사)

1991년 미국 코넬대학교 전기공학부(박사)

1992년~1994년 미국 국립 오크리지연구소 연구원

1994년~1995년 미국 테네시 주립대학교 연구교수

2001년~2002년 미국 국립 수퍼컴퓨팅 응용센터(NCSA) 초빙연구원

1995년~현재 숭실대학교 컴퓨터학부 부교수

관심분야 : 시스템 소프트웨어, 고성능컴퓨팅(HPC), 병렬/분산처리