논문 2010-47CI-4-6

X-treeDiff+ 기반의 프로그램 복제 탐지

(Program Plagiarism Detection based on X-treeDiff+)

이 석 균*

(Suk Kyoon Lee)

요 약

컴퓨터 프로그래밍 교육에서 프로그램 복제는 프로그래밍 학습 효율을 저해하는 심각한 요인이다. 본 논문에서는 학생들이 프로그래밍 과제의 무분별한 복제를 방지하기 위해 유사 또는 동일 프로그램을 탐지하는 기법을 제안한다. 지문법이나 스트링 매칭을 기반으로 하는 기존의 탐지 기법과는 달리, 우선 C 프로그램을 파싱하여 문법요소를 엘리먼트로 하는 XML 문서로 변화시킨 후 XML 문서의 변화탐지 알고리즘인 X-treeDiff+를 실행시켜 그 차이를 분석하는 방법을 취한다. 이때 대응의 정도를 나타내는 유사도와 두 문서의 차이로 제시되는 일련의 편집연산인 편집스크립트를 프로그램 복제의 관점에서 분석하여 복제 여부에 대한 판단을 하게 된다. 편집스크립트의 분석은 두 프로그램 간의 변환 과정을 유추할 수 있게 하여 기존 방법들과는 달리 사용자는 과제의 성격이나 복제의 정도를 고려한 정성적인 판단이 가능하다는 장점이 있다.

Abstract

Program plagiarism is a significant factor to reduce the quality of education in computer programming. In this paper, we propose the technique of identifying similar or identical programs in order to prevent students from reckless copying their programming assignments. Existing approaches for identifying similar programs are mainly based on fingerprints or pattern matching for text documents. Different from those existing approaches, we propose an approach based on the program structur. Using paring progrmas, we first transform programs into XML documents by representing syntactic components in the programs with elements in XML document, then run X-tree Diff+, which is the change detection algorithm for XML documents, and produce an edit script as a change. The decision of similar or identical programs is made on the analysis of edit scripts in terms of program plagiarism. Analysis of edit scripts allows users to understand the process of conversion between two programs so that users can make qualitative judgement considering the characteristics of program assignment and the degree of plagiarism.

Keywords: 프로그램 표절, X-tree Diff+, 편집스크립트, XML

I. 서 론

최근 많은 분야에서 디지털 데이터의 사용은 필수적이다. 그러나 책, 종이 등 물리적인 매체에 존재하던 기존의 아날로그 데이터와는 달리 디지털 데이터는 복사, 수정이 쉽게 이루어지고 있어 디지털 데이터의 표절(plagiarism)은 사회적 문제로 등장하고 있다. 컴퓨터 프로그램 또한 디지털 데이터의 유형으로 대학의 컴퓨터 프로그래밍 과목에서 학생들은 프로그래밍 과제를 종종 상대방의 협조 하에 또는 무단으로 복제, 부분 복제 및 수정해서 과제 결과로 제출하곤 한다. 대학의 컴퓨터 교육 환경에서 이러한 문제는 대부분 상대방의 협조 하에 이루어지므로, 엄밀한 의미에서 문학 작품, 그림, 상표 등에서의 표절의 의미로 해석하기에는 어려우나 대개의 논문들에서는 이러한 프로그램의 복제 또는 부분 복제를 표절(plagiarism)로 표현한다[1~3]. 그러나 본 논문에서는 교육 과정이라는 배경과 학생들의 상호 협조 및 묵인을 전제로 할 때 이를 부분 복제 또는 더

* 정회원, 단국대학교 공과대학 컴퓨터학부
(Department of Computer Science and Engineering, College of Engineering, Dankook University)
※ 이 연구는 2009년도 단국대학교 대학연구비의 지원으로 연구되었음
접수일자: 2010년6월1일, 수정완료일: 2010년7월7일

간단히 복제라고 부르기로 한다.

대개 학생들이 과제를 복제할 때는 프로그램에 대한 논리적 이해 없이 단순한 편집 과정을 거쳐 제출한다. 따라서 프로그램의 복제는 프로그램에 대한 상세한 이해 없이 간단한 일련의 텍스트 편집 연산들을 통해 다른 프로그램으로부터 생성된다[1]. 프로그램 복제에 관한 연구들은 지문법 (fingerprint)과 구조 기반 (structured-based) 방법으로 구분된다[4]. 지문법이란 키워드 등 특정 단어의 빈도를 통계적으로 계산해서 비교하는 방법이고 구조기반은 프로그램 텍스트를 어휘 분석기를 통해 토큰들의 시퀀스를 생성한 후 이들 토큰들의 시퀀스를 서로 비교하는 방법이다. 이 과정에서 다양한 매칭 기법을 사용해서 적절한 메트릭을 계산하는데 메트릭의 질이 성능을 결정하게 된다[2,4~5].

본 논문에서는 프로그램을 XML 문서로 변환한 후 XML 문서의 변화 탐지 알고리즘을 사용하여 차이의 정도와 변화 내용을 파악하고자 한다. 우선 프로그램을 EBNF의 문법에 기반 한 파스 트리로 변환한 후 각 문법적 구성요소의 앞뒤에 태그를 붙이면 쉽게 XML 엘리먼트로 표현된다. 이 방법을 통해 분석 대상의 프로그램들을 트리구조의 XML 문서로 변화시킨 후 이들에 대해 XML 문서의 변화 탐지 알고리즘인 X-tree Diff+[6]를 실행시켜 문서의 차이 즉 변화 내용을 발견한다. 문서의 차이는 일련의 편집연산들, 즉 편집 스크립트로 표현되는데 편집 스크립트의 내용을 분석해서 프로그램의 복제 여부를 평가할 수 있다. 본 논문의 제안 방법은 유사도를 통한 정량적인 분석과 프로그램의 차이, 즉 편집스크립트를 분석하여 사용자에게 어떠한 수정 과정이 가능한지의 해석을 제시하는 정성적인 분석을 제공한다.

기존의 방법들은 정량적인 수치만을 제공하므로 실제 사용자가 수작업으로 복제여부를 확인하는 절차가 필요했다. 특히 프로그래밍 교육과정의 학생들의 제출 과제는 강의 자료를 참고하고 참고문헌을 공유하므로 상당부분 비슷하여 복제하지 않았더라도 유사한 결과가 나올 수 있다[4]. 이러한 경우 단순히 유사도를 나타내는 수치만으로 평가하기는 어렵다. 이에 비해 본 제안 방법은 프로그램의 차이가 어떠한 변화를 통해 가능한지를 확인할 수 있으므로 과제의 성격이나 복제 정도를 고려하여 복제여부의 판단을 할 수 있고 이에 따른 교육적 효과도 기대된다. 본 논문의 구성은 다음과 같다. Ⅱ장에서는 X-tree Diff+의 알고리즘의 소개와 편집 스크립트에 사용되는 연산자들을 설명하며, Ⅲ장에서는 본 논문의 핵심인 프로그램 복제 탐지 시스템의 구조와 사용되는 기법들, 그리고 Ⅳ장에서는 프로토타입 시스템과 성능 비교를 소개하고 Ⅴ장에서는 결론과 추후 연구 방향을 논의한다.

Ⅱ. 기존 연구 소개

1. X-treeDiff+:트리구조 문서의 변화탐지알고리즘

두 문서 사이의 차이 또는 변화 내용을 계산하는 작업은 컴퓨터 과학 분야의 오랜 연구 분야이다. 그러나 XML/HTML과 같은 트리 구조의 데이터에 대한 변화 탐지 연구는 70년대부터 진행되었다[7~18]. 일반적인 트리 구조 데이터에 대한 변화 탐지는 NP-Hard 문제로 이들 알고리즘들은 실시간 사용에는 적합하지 못하여[3], 최근의 연구들은 해시 등의 휴리스틱 기법을 통해 실시간 사용 가능한 알고리즘 개발에 집중되었다. XyDiff[19]는 대용량의 XML 문서를 저장 관리를 위한 XML 데이터 웨어하우스에서 버전 관리의 목적으로 고안되었다. XyDiff는 서브트리의 구조와 내용을 해시로 통해 표현하고 대응 과정에서 가중치를 사용하는 등의 방법을 통해 대응의 효율성을 제고하여 실행시간 복잡도에 있어서 O(nlogn)의 성능을 보이고 있다. 한편, X-Diff+[17]는 두 문서간의 편집이격도(edit distance)라는 척도를 통해 적절한 수준의 편집 스크립트를 계산해낸다.

본 논문에서 사용되는 X-treeDiff+는 해커의 웹 사이트 공격 즉 해커에 의한 웹 사이트 변조를 실시간으로 탐지하기 위해 개발되었던 X-treeDiff[6, 20]을 확장한 것으로 XML/HTML과 같은 트리 구조의 문서들의 차이를 실시간으로 탐지하고 그 차이를 나타내는 편집스크립트를 생성한다. X-tree Diff+는 O(n)의 성능을 보이면서 X-tree Diff의 대응의 질을 높이기 위해 대응과정에서 XML의 ID 속성을 사용하고 복사 연산을 추가하고 대응 결정 후 튜닝 과정을 포함하여 국소적인 대응 개선 효과를 제공하고 있다[21].

2. X-treeDiff+의 기본 자료 구조 및 편집연산

X-tree Diff+에서는 XML 문서는 X-tree로 표현된다. X-tree는 X-tree 노드를 통해 정의되는데[6, 20~21] 본 논문에서는 X-tree의 노드 중 필요한 부분만 설명한다. Type필드는 엘리먼트 노드와 텍스트 노드를 구별하며, Value필드에는 엘리먼트 노드의 경우 레이블 이름이,

```
<source>
 <fn>
  <fID><ID>main</ID>()</fID>
  <csm>{ <ep><ID>printf</ID>("Hello");</ep>} </csm>
 </fn>
</source>
```

그림 1. 간단한 XML 문서의 예
Fig. 1. Example of an simple XML document.

그리고 텍스트 노드의 경우 텍스트 값이 저장된다. Index 필드는 같은 부모노드 내에 동일한 레이블을 갖는 자식노드들 간의 순서를 나타내는데 같은 레이블을 갖는 형제노드들을 식별을 위해 사용된다. 형제노드들 중 동일한 레이블을 갖는 노드가 하나일 경우는 Index 필드는 0의 값을 갖는다.

그림 1은 "main() { printf("Hello"); }"의 매우 간단한 C 프로그램을 XML 문서로 표현한 것으로 문법적 구성 요소들을 엘리먼트로 표현하고 있다. <fn>, <ID>, <csm>, <ep>는 각각 함수, 식별자, 복합문(compound statement), 수식(expression)을 나타내는 엘리먼트들을 의미한다. main, (), printf에서 알 수 있듯이 함수 명, 변수 명, 수식이나 상수는 텍스트 노드의 Value 필드에 저장된다.

한편, 동일 부모 내의 형제 노드들은 Value 필드과 Index 필드의 값을 통해 표현된다. 예를 들어 <fID>는 <fn> 내에서 경로 fID[0]으로 표현된다. 트리 전체에서 각 노드의 식별은 루트로부터의 노드까지의 경로, 즉 XPath로 표시하는데 이를 nID라고 한다. 그림 2에서 main함수의 바디는 복합문으로 이 엘리먼트의 nID는 /source[0]/fn[0] /csm[0]이고 함수 명 printf는 /source[0]/fn[0]/csm[0] /ID[0]/text()[0]로 표현된다.

X-tree Diff+에서는 두 X-tree들에 대해 대응과정을 통해 차이를 발견하고 이에 대한 편집 스크립트를 생성한다. 편집스크립트는 편집연산들의 집합으로 X-tree Diff+에서는 삽입, 삭제, 갱신연산, 그리고 이동연산과 복사연산을 지원한다. 이들을 논리적 수준에서 간단히 정의하면 다음과 같다. X-tree에 대해 삽입연산 *Ins*(m, n, *k*)은 노드 m을 노드 n의 *k*번째 자식노드로 삽입하고, 삭제연산 *Del*(n)은 노드 n을 삭제하며, 갱신연산 *Upd*(n, v)은 텍스트 노드 n의 Value 필드의 값을 v로 갱신한다. 이동연산 *Mov*(m, n, *k*)은 노드 m을 노드 n의 *k*번째 자식노드로 이동시키고 복사연산 Copy(m, n,*k*)은 노드 m을 노드 n의 *k*번째 자식노드로 복사시킨다.

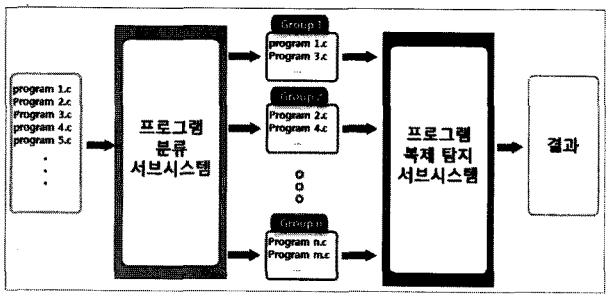

그림 2. 프로그램 복제 탐지 시스템의 처리과정의 개요
Fig. 2. Overview of program plagiarism detection system.

III. 프로그램 복제 탐지 시스템

본 절에서는 제안하는 프로그램 복제 탐지 시스템을 소개한다. 전체 시스템 구조, 프로그램 분류 서브시스템, 프로그램 복제분석 서브시스템 순으로 설명한다.

1. 시스템 기본 구조

본 논문의 제안 시스템에서는 학생들이 제출한 프로그램들의 복제 여부의 확인을 효율적으로 처리하기 위해 일련의 프로그램 리스트에 대해 그림 2에서 제시한 바와 같이 두 단계의 처리 절차를 거친다. 우선 모든 프로그램들을 구조적인 특성에 따라 다수의 그룹들로 분류하고 복제 여부의 분석은 각 그룹에 속한 프로그램들에 대해 수행한다. '프로그램 분류 서브시스템'은 프로그램들을 분류하는 작업을 '프로그램 복제 분석 서브시스템'은 동일 그룹에 속한 두 프로그램에 대해 복제 여부를 분석하는 작업을 수행한다.

그림 3에 제시되는 프로그램 분류 서브시스템은 복제 여부의 분석 작업의 효율성을 위해 필요하다. 즉 n

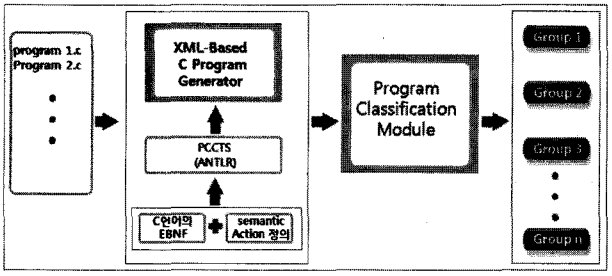

그림 3. 프로그램 분류 서브시스템의 기능 구조
Fig. 3. Structure of program classification subsystem.

개의 프로그램들에 대한 복제여부의 분석은 모든 서로 다른 프로그램들의 조합 즉 n(n-1)/2에 대한 작업이 요구되므로 시간 비용이 적지 않을 뿐 아니라 유사정도의 가능성이 매우 낮은 프로그램들에 대해 복제여부 분석하는 것은 그다지 효과적이지도 않다.

한편, 그림 6의 프로그램 복제탐지 서브시스템은 두 프로그램 간의 복제 여부 즉 동일 또는 유사 여부의 확인을 위해 두 프로그램 사이의 차이(변화)를 발견하고 그 차이의 정도를 분석한다. 이 시스템에서는 문자열간의 대응관계를 기반으로 하는 기존의 복제 탐지 시스템들과는 달리 두 프로그램의 차이를 발견하는 과정에 문법적인 구조를 반영한다. 이를 위해 파서를 사용하여 프로그램을 구문 분석하고 분석과정에서 각 문법적 요소 단위를 XML 문서의 엘리먼트로 변환시켜 XML 문서를 생성한다. 비교 대상의 프로그램들로부터 생성된 XML 문서들을 X-tree Diff+로 변화내용을 편집스크립트로 출력시킨 후, 편집스크립트를 분석하여 복제 여부를 결정한다.

2. 프로그램 분류 서브시스템

프로그램 분류서브시스템의 핵심은 XML-based C Program Generator이다. XML-based C Program Generator는 C 프로그램이 입력될 때 이를 EBNF의 문법 기반의 XML 문서를 생성한다. 이 과정에는 C언어를 위한 파서가 필요한데 본 시스템에서는 파서 자동생성 툴인 PCCTS[22]를 사용한다.

PCCTS는 프로그래밍 언어 인식과 번역에 사용되는 퍼블릭 도메인의 툴로 ANTLR, DLG, SORCERER의 세 가지 소프트웨어로 구성되는데, 논문에서는 ANTLR(ANother Tool for Language Recognition)을 사용한다. ANTLR은 EBNF 등의 문법적 정의를 입력받아 명령문들을 구문 분석하는 프로그램을 생성하는 자동 파서 생성기로 yacc보다는 다양한 기능을 지원하고 유연하고 사용하기 편리하다. ANTLR에서 사용자는 EBNF의 문법적 구성요소에 시맨틱 액션(semantic action)을 정의하여 파싱 과정에서 파스트리를 어떻게 생성할 지를, 그리고 번역 과정을 통제할 수 있다. 본 논문에서는 이를 통해 파싱과정에서 XML 엘리먼트들을 생성하고 있다. 다음에 EBNF의 문법적 구성요소에 시맨택 액션이 정의된 예를 보인다.

```
statement
  : plain_label_statement
  | << strOutput+= "<clsm>";>>
    case_label_statement << strOutput+=
      "</clsm>"; >>
  |  .... /* 생략*/;
plain_label_statement
  : id:IDENTIFIER  COLON
    << strOutput += "<ID>";
       sprintf(tStr, "%s : ", id->getText());
       strOutput += std::string(tStr);
       strOutput += "</ID>"; >>
  statement ;
```

위의 내용은 명령문(statement)에 관한 EBNF 규칙들의 일부를 보인 것으로, IDENTIFIER, COLON 등 대문자로 표시된 것은 어휘 분석 후 생성된 토큰이다. 첫 번째 EBNF 규칙은 statement가 plain_label_statement, case_label_ statement 등 일련의 세부 statement들로 구성됨을, 두 번째 규칙은 plain_label_statement가 ID, COLON. 그리고 statement로 구성됨을 보인다. 시맨틱 액션은 <<와 >> 내부에 정의하는데, 위의 예에서는 strOutput이라는 문자열 변수에 XML 문서로 변화될 내용을 저장한다. 예를 들면 statement에 대한 규칙 중 case_label_statement에 대응되는 경우 case_label_ statement 처리 전 <clsm>을 출력하고 case_label_ statement 처리 후 </clsm>을 출력하여 엘리먼트를 만든다.

PCCTS는 위와 같은 문법 파일을 입력으로 받아 파서를 생성한다. 따라서 본 시스템의 파서를 실행하면, 입력된 프로그램을 파싱할 때 각 문법적 구성요소에 정의된 시맨틱 액션에 의해 엘리먼트들을 생성하여 최종적으로 XML 파일을 출력한다. 그런데 C의 EBNF 문법이 statement가 expression으로 표현되는 등 그다지 직관적이지 못한 부분이 있고 모든 EBNF 규칙에 대해 엘리먼트를 생성하는 것은 XML 문서의 엘리먼트의 중첩도를 상당히 심화시키므로 본 시스템의 목적인 유사도 분석에 도움이 되지 못한다. 따라서 본 논문에서는 다음의 원칙에 입각해서 문법 파일, 즉 시맨틱 액션을 정의한다.

(a) #define, #include와 같은 명령문들은 컴파일 과정에는 필요하나 C 언어의 문법에는 속하지 않기 때

문에 어휘분석 단계에서 삭제한다.

(b) 모든 EBNF 규칙에 대해 엘리먼트를 생성하지는 않는다. 재귀적인 규칙들이 많고 모든 규칙에 대해 엘리먼트를 생성하는 것이 본 시스템의 목적에 도움이 되지 않기 때문이다.

(c) 상수와 연산자는 텍스트 노드로 처리한다. 연산자의 텍스트 표현으로 수식(expression)의 분석이 가능하며 또한 불필요한 트리의 중첩도를 줄이기 위함이다.

(d) 함수의 매개변수의 선언부와 변수의 선언부를 포함시키지 않는다.

(d)의 내용의 존재는 분석의 편의성 때문으로 함수의 매개변수의 선언부와 변수의 선언부도 XML화 시킬 수는 있으나 본 시스템의 프로그램 복제여부 분석 과정에서 사용되고 있지 않아 생략하였다. 학생들은 흔히 프로그램의 논리에 대한 이해 없이도 함수의 선언 순서, 함수의 매개변수의 선언 순서, 또는 변수의 선언 순서들을 수정하거나 매개변수나 변수의 타입을 보다 유연한 타입으

```
<source><fn>
  int <fID><ID>main</ID>(void )</fID>
  <csm>{
    <ism>for (<ID>i</ID> = 1<semicol/><ID>i</ID>
        &lt;=11<semicol/><ID>i</ID>++)
      <csm>{<ep><ID>result</ID> = <ID>i</ID>
          <semicol/></ep>}</csm>
    </ism>
    <jsm>return0;</jsm>
  }</csm>
</fn></source>
```

그림 4. 대상 프로그램1
Fig. 4. Target program1.

```
<source><fn>
  int <fID><ID>main</ID>(void )</fID>
  <csm>{
    <ism>for (<ID>i</ID> = 0<semicol/><ID>i</ID>
        &lt;=10<semicol/><ID>i</ID>++)
      <csm>{<ep><ID>result</ID> = <ID>i</ID>
          <semicol/></ep>}</csm>
    </ism>
    <jsm>return0;</jsm>
  }</csm>
</fn></source>
```

그림 5. 대상 프로그램2
Fig. 5. Target program2.

로 변경하여 다른 프로그램처럼 보이게 할 수 있다. 그림 4와 그림 5는 각각 main 함수 안에 for 문이 있는 예제를 XML 문서로 변경한 것인데 두 프로그램의 차이는 for 문의 초기화와 비교에 관한 부분이다.

프로그램 분류서브시스템의 Program Classification Module은 입력된 프로그램을 XML 문서로 변화시킨 후 프로그램의 구조적 특성에 기초하여 프로그램을 다양한 그룹으로 분류한다. 여기에서는 다양한 알고리즘이 사용될 수 있는데 본 시스템에서 C 프로그램의 복제여부 분석에서는 함수의 개수와 생성된 XML 문서의 노드 수를 기준으로 분류하였다.

3. 프로그램 복제분석 서브시스템

그림 6에서 제시하듯이 프로그램 복제분석 서브시스템에서는 XML-Based C Program Generator가 두 프로그램을 XML 문서로 변환하고 X-treeDiff+는 이들 문서 즉 원본(source) 문서와 목표(target)문서에 대해 편집스크립트를 생성한다. 편집스크립트도 X-treeDiff+에서는 XML 문서로 표현되며 원본문서에 편집스크립트가 적용되면 목표문서가 생성되는 특징이 있다. 그림 7은 그림 4와 그림 5에서 제시된 XML로 변환된 두 프로그램에 대해 X-treeDiff+가 생성한 편집스크립트이다.

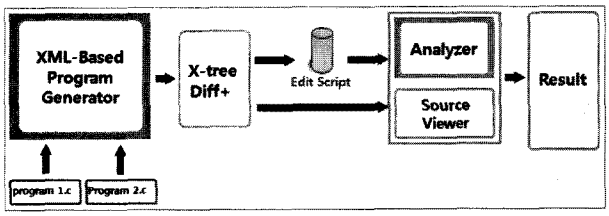

그림 6. 프로그램 복제분석 서브시스템의 기능구조
Fig. 6. Structure of program plagiarism analysis subsystem.

가. 프로그램 복제탐지에 있어서 편집스크립트의 의미

그림 7의 편집 스크립트는 두 개의 update문으로 구성되고 있다. 실제 X-treeDiff+가 생성하는 편집스크립트에서 갱신연산 명령문에는 snid 속성은 갱신 대상 노드의 XPath, sv 속성은, 갱신 전의 텍스트 값, tv 속성은 갱신 후의 텍스트 값을 나타낸다. 편집스크립트의 분석에 앞서 우선 편집연산들의 특성을 설명한다. 삽입연산과 삭제연산은 두 문서들 사이에 대응되지 않는 부분을 의미한다. 삽입연산은 원본문서에는 없으나 목표문서에 존재하는 노드를 삽입하는 편집연산이고, 삭제

```
<dblab:es xmlns:dblab="http://dbwebapplicationlab.dankook.ac.kr" dc="March 25, 2010">
 <UPDATE_TEXT snid="/source[0]/fn[0]/csm[0]/ism[0]/text()[2]" sv="&lt;=11" spos="5"  tv="&lt;=10"/>
 <UPDATE_TEXT snid="/source[0]/fn[0]/csm[0]/ism[0]/text()[1]" sv=" = 1" spos="2" tv=" = 0"/>
</dblab:es>
```

그림 7. 편집 스크립트
Fig. 7. Editscript.

연산은 그 반대의 경우에 해당된다. 갱신연산은 두 문서간의 대응되는 텍스트 노드의 값이 다를 때, 사용된다. 이동연산은 원본문서의 서브트리가 다른 부모의 자식으로 이동될 때 사용되며, 복사연산은 원본의 노드가 목표문서의 여러 곳에서 발견될 때 사용된다.

프로그램의 복제탐지에서 편집스크립트에 삽입연산과 삭제연산이 많음은 두 프로그램이 유사하지 않다는 의미로 해석될 수 있으며 이 경우 복제의 가능성은 매우 낮다.

한편, 이동연산은 프로그램에서 함수 정의의 위치 변경, 명령문의 위치 변경, 수식에서 변수의 사용 위치 변경 등 프로그램의 구조 변환 시도를 해석된다. 따라서 이동연산은 프로그램 복제탐지의 분석에 있어서 매우 중요하다. 학생들이 프로그램 복제 과정에서 흔히 프로그램의 논리에 영향을 주지 않는 범위 내에서 함수 정의의 위치 이동, 명령문의 위치 이동을 시도하기 때문이다. 갱신연산은 텍스트 노드에 대해 적용되는데, 텍스트 노드는 변수 명, 함수 명 등과 같은 식별자(identifier)이거나 연산자 또는 상수로 구성되는데, 연산자나 상수 값의 갱신은 프로그램 논리의 변화로 종종 해석된다. 반면, 변수 명, 함수 명의 갱신은 학생들이 복제 과정에서 흔히 시도하는 방법인데, 이는 프로그램 내에서의 통용범위(scope)를 고려해서 분석해야 한다.

나. 프로그램 복제탐지를 위한 분석 방법

복제 탐지의 분석모듈에서는 정량적인 분석과 정성적인 분석을 수행한다. 정량적인 분석은 XML화된 두 프로그램 간의 유사도를 유사도와 대응정도의 수치로 표현하는 작업이고 정성적인 분석은 편집스크립트를 분석해서 어떤 변화가 발생했는가를 판단하는 작업이다. 정성적인 분석은 정량적인 분석의 결과에 따라 복제 가능성이 높을 때 수행된다.

유사도 기반의 정량적 분석 기법

원본문서의 대응된 노드 수를 m1, 전체 노드 수를 n1, 목표문서의 대응된 노드 수를 m2, 전체 노드 수를 n2라 할 때 문서의 유사도와 대응정도는 각각 다음과 같이 정의된다.

$$\text{유사도} = (m1 + m2) / (n1 + n2)$$
$$\text{원본문서의 대응정도} = m1 / n1$$
$$\text{목표문서의 대응정도} = m2 / n2$$

XML 문서로 표현된 프로그램에서 노드들은 C 프로그램의 EBNF의 규칙을 기반으로 생성되었기 때문에 위의 유사도는 C 프로그램의 문법을 반영한 두 프로그램 간의 유사도라 할 수 있다. 두 프로그램에 속한 노드들 간의 대응은 다음의 경우로 구분된다. 두 프로그램 간에 동일한 부분, 즉 아무런 변화 없이 1-1로 대응된 경우, 이때는 편집연산이 필요가 없다. 그러나 1-1 대응의 경우에도 내용이 변경된 경우는 갱신연산으로, 구조가 변경된 경우는 이동연산으로 표현된다. 복사연산은 내용이 같으나 1-n으로 대응된 경우를 의미한다. 갱신연산과 이동연산은 1-1 대응의 경우를 의미하기 때문에 단지 유사도가 1이라 해서 두 프로그램이 완전 동일하다고는 할 수 없다. 물론 완전 복제한 프로그램들의 유사도는 1이 된다.

한편, 문서의 대응정도가 1보다 적은 경우는 삽입연산이나 삭제연산이 존재하는 경우를 의미한다. 프로그램 복제에서 복사연산은 거의 발생하지 않기 때문에 이에 대한 분석은 고려하지 않는다. 한편 편집스크립트가 빈(empty) 경우, 즉 어떤 편집연산도 포함하지 않는 경우는 두 문서가 완전히 동일한 경우를 의미한다. 위의 내용을 기반으로 한 정량적 분석의 알고리즘은 그림 8에 제시된다. 이때 유사도가 정성적 분석 여부를 결정하는데 그 비교 수치는 프로그램의 규모 및 그 용도에 따라 달라진다.

```
if Empty(편집스크립트) then 완전복제 판정;
elsif 유사도 = 1    then 두 프로그램의 모든 부분이
                    대응됨. 유사 가능성 매우 높음 판정;
elsif 유사도 >= 0.9 then 두 프로그램의 유사 가능성이
                    매우 높은 판정;
elsif 유사도 > 0.8 then  두 프로그램의 유사 가능성이
                    높음 판정;
else 두 프로그램은 유사하지 않다고 판정;

if 원본문서 대응정도 = 1 then
    원본 프로그램의 모든 내용이 목표 프로그램에
    대응 판정;
if 목표문서 대응정도 = 1 then
    목표 프로그램의 모든 내용이 원본 프로그램에
    대응 판정;
if 완전복제가 아니고 유사도가 0.8보다 큰 경우 then
    정성적 분석 모듈 호출
```

그림 8. 프로그램 복제 탐지를 위한 정량적 알고리즘
Fig. 8. Quantitative algorism for program plagiarism detection.

편집스크립트 분석을 통한 정성적 분석 방법

완전복제 판정이 아닐 경우 유사도가 일정 수치보다 클 경우 정성적 분석을 수행하는데, 이는 편집스크립트를 분석하여 어떤 변화가 발생했는가를 사용자에게 제시하여 사용자가 직접 복제여부를 판단하는 방법이다. 다음의 순서대로 진행한다.

(1) 식별자들의 변경여부의 확인

함수 명, 상수 명, 그리고 변수 명은 식별자로 표현되는데, 다수의 갱신연산들이 존재할 때 갱신연산이 적용되는 통용범위를 작은 단위부터 큰 단위로 확장하면서 식별자들에 대한 갱신 현황을 제시한다. 이때 갱신되는 노드의 조상노드들의 분석을 통해 이들이 식별자의 유형 정보(예, 함수 명, 상수 명 또는 변수 명인지의 구분여부)도 같이 제공한다.

(2) 식별자가 아닌 수식 및 연산의 변경여부의 확인

수식에서 단순히 값의 변경 또는 적용 연산을 변경하는 복제시도를 탐지하기 위해서 사용자에게 수식 및 연산의 변경여부를 제시한다. 이때 분석 순서는 문서의 깊이우선탐색 순으로 진행하는데, 갱신연산의 nID를 분석해서 텍스트 노드의 상위 엘리먼트 노드에 속한 모든 텍스트 노드들을 통합한 결과, 즉 sv 값을 포함한 프로그램 세그먼트와 sv 값, tv값을 제시한다. 대부분의 경우 명령문 단위로 보이므로 분석이 가능하다.

(3) 명령문 또는 함수의 이동의 확인

이동연산이 다수 존재할 때 문서의 넓이우선탐색 순으로 즉 가장 큰 통용범위의 이동연산부터 분석한다. 이때 이동의 원본문서에서의 위치와 목표문서의 위치를 nID의 엘리먼트 이름들을 분석하여 같이 제공한다. 학생들이 흔히 프로그램의 논리에 영향을 주지 않는 범위 내에서 함수 정의의 위치 이동, 명령문의 위치 이동을 시도하기 때문에 분석에서 중요한 역할을 한다.

(4) 삭제 및 삽입에 대한 분석

삭제 및 삽입연산도 nID를 분석해서 함수단위의 삭제 및 삽입은 간단히 함수의 삭제 또는 삽입이 발생했음을 제시한다. 명령문 단위의 삭제 및 삽입은 그 대상 내용을 제시함으로 어떤 변화가 발생했는지를 보인다. 수식 단위의 삭제 및 삽입은 그 상위 노드에 속한 모든 텍스트 노드들의 통합 결과를 같이 제시하여 사용자한테 어떤 변화가 발생하는지를 알 수 있게 한다. 이는 변화의 내용을 이해하기 위함이고 삭제 또는 동일 부모 노드에 대한 삽입연산이 많은 경우는 복제 또는 부분복제의 가능성이 거의 없다고 할 수 있다.

Ⅳ. 프로토타입 시스템 및 성능 비교

앞에서 설명한 복제 탐지 방법을 사용한 SCDC (Smart Cheat Detector for C program) 프로토타입 시스템을 구현하고 다양한 경우에 테스트 중에 있다. 이는 PCCTS 1.33과 MS Visual Studio 6.0과 MS Visual Studio2005가 사용되었는데 프로그램 복제 분석서브시스템의 실행 결과 화면이 그림 9와 그림 10에 제시되었다. 그림 9의 상단은 유사도를 제시하고 왼쪽 하단은 갱신연산의 의미들을 분석하고 우측 화면은 이동연산의 의미를 제시하고 있다. 그림 10은 상세보기를 통해 소스 프로그램의 내용을 확인할 수 있도록 한다.

본 시스템의 성능을 확인하기 위해 1학년 C 프로그래밍 과목의 과제로 제출된 C 프로그램 22개를 가장 보편적으로 사용되는 JPlag[23]의 복제 탐지 시스템과 본 논문의 SCDC를 통해 분석했다. 프로그램은 각각 1.c, 2.c ..,

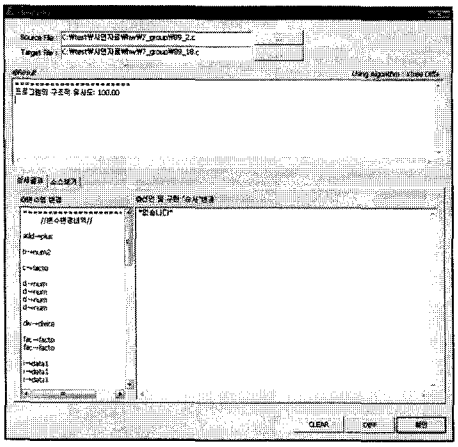

그림 9. SCDC의 실행화면1
Fig. 9. View1 of running SCDC.

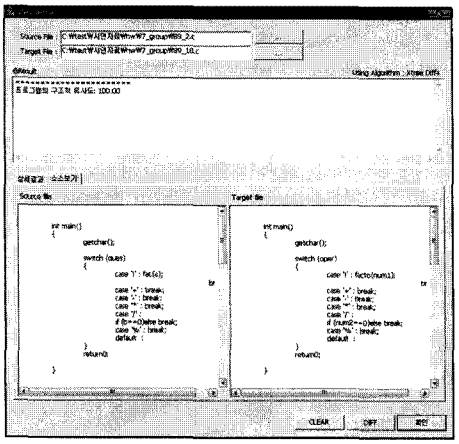

그림 10. SCDC의 실행화면2
Fig. 10. View2 of running SCDC.

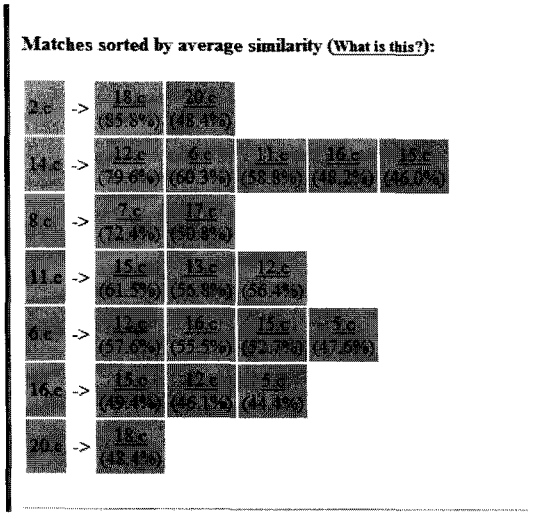

그림 11. JPlag의 C프로그램 테스트 결과
Fig. 11. Test result of JPlag for C programs.

22.c로 이름으로 제시되어 있고 JPlag으로 분석한 결과가 그림 11에 그리고 SCDC로 분석한 결과가 표 1에 제시되어 있다. JPlag의 분석 결과는 유사도(similarity) 44%를 임계치(threshold)로 설정되어 그 이상의 유사도를 갖는 프로그램들에 대해 분석 결과를 제시하고 있다.

표 1. SCDC의 C 프로그램의 테스트 결과
Table 1. Test result of SCDC for C programs.

함수 수	소스1	소스2	유사도	복제 여부
1	46_21.c	44_22.c	54.13	X
2	46_4.c	54_5.c	62.5	X
		59_6.c	67.29	X
		42_13.c	41.33	X
		65_15.c	48.26	X
		63_16.c	58.99	X
	54_5.c	59_6.c	86.2	X
		42_13.c	실행 오류	
		65_15.c	81.65	O
		63_16.c	83.01	X
	59_6.c	42_13.c	60.9	X
		65_15.c	74.77	X
		63_16.c	78.37	X
	42_13.c	65_15.c	65.96	X
		63_16.c	55.64	X
	65_15.c	63_16.c	66.86	X
3	62_11.c	87_12.c	81.11	O
		81_14.c	84.51	O
	81_14.c	87_12.c	94.43	◎
6	113_3.c	67_8.c	26.61	X
		85_9.c	31.92	X
		69_10.c	19.25	X
		94_17.c	34.26	X
		125_19.c	38.97	X
	67_8.c	85_9.c	54.45	X
		69_10.c	36.73	X
		94_17.c	79.16	X
		125_19.c	45.96	X
	85_9.c	69_10.c	41.55	X
		94_17.c	49.77	X
		125_19.c	40.38	X
	69_10.c	94_17.c	51.4	X
		125_19.c	35.94	X
	94_17.c	125_19.c	57.25	X
7	89_2.c	89_18.c	100	◎

SCDC는 총 22개의 프로그램들을 프로그램 분류 서브시스템에 의해 함수 수를 기준으로 총 7개의 그룹으로 구분하였고 각 그룹에 속한 프로그램들에 대해 프로그램 복제분석 서브시스템의 분석결과를 제시하였다. 소스1과 소스2의 열의 프로그램 이름에서 밑줄 앞의 숫자는 프로그램의 노드 수, 밑줄 뒤의 숫자는 원 프로그램 이름을 나타낸다. 즉 46_21.c는 노드 수가 46이고 프로그램 이름은 22.c를 의미한다. 또한 '복제 여부' 열은 해당 과목의 교수자가 직접 확인했을 때 복제로 확신하는 경우는 ◎,

복제 가능성이 높다고 판단하는 경우는 O로 표시하였다.

다음에서 두 시스템의 분석 결과를 비교 분석한다. 가장 유사도가 높은 프로그램 쌍은 프로그램 2.c와 18.c인데 JPlag의 유사도는 85.8%, 그리고 SCDC의 유사도는 100%로 평가되었다. 그러나 두 프로그램의 실제적인 차이는 함수 명, 변수 명의 변경과 일부 함수의 위치 변경으로 인한 차이로 이를 두 문서간의 대응관계로 나타내면 때 갱신과 이동 연산으로 표현되는 경우이다. SCDC의 유사도는 이러한 경우를 정확히 탐지해내는 특성이 있다. 두 번째로 유사도가 높은 프로그램 쌍은 14.c와 12.c인데 이 경우도 변수 명과 함수 명의 수정과 위치 이동이 이루어져있으며 일부 명령문들을 다른 표현으로 표현한 경우로 두 프로그램을 실제로 확인하면 거의 프로그램 복제한 것으로 확신할 수 있다.

SCDC의 유사도가 80% 이상의 경우들을 보면 교수자가 복제 가능성 있다고 판단한 프로그램 쌍들은 모두 이 범위에 포함된다. 반면 JPlag의 경우에는 SCDC의 유사도 100%의 경우도 단지 85.8%로 나타내고 있으며 특히 교수자가 복제 가능성있다고 판정한 경우들 즉 (5.c, 15.c), (11.c, 12.c), (11.c, 14.c), (12.c, 14.c), (2.c, 18.c) 들 중 SCDC는 모두 유사도 80%이상으로 판정했지만 JPLAG는 이들 중 유사도 70%이상으로 판정한 경우는 단 하나밖에 없으며 (5.c, 15.c)의 경우는 임계치 밖으로 평가되어 리스트에 나타나지도 못하는 문제가 발생하였다.

단 SCDC의 유사도는 두 프로그램의 대응 노드 수를 전체 노드의 수로 나눈 결과이기 때문에 대응되지 않은 명령문들이 많은 경우, 즉 프로그램을 복제한 후 의미 없는 명령문들을 상당량 삽입했을 경우에는 거짓음성(false negative)의 결과가 발생할 수 있다. 그러나 프로그램 크기가 일정 수준을 넘어서는 경우 이로 인해 잘못 판별하는 경우는 거의 없다. 또한 거짓양성(false positive)으로 판별하는 경우인데 이 경우는 정성적인 판단이 이루어지므로 아무런 문제가 없다.

SCDC의 유사도의 성능과 별도로 본 시스템은 두 프로그램의 차이를 편집스크립트를 통해 분석하므로 사용자는 확인하고자 하는 대상의 프로그램들의 차이를 편집연산 관점에서 이해할 수 있게 하며 구체적으로 어느 부분이 어떻게 변화해서 복제된 것인지를 제시한다는 장점이 있다.

V. 결 론

본 논문은 프로그램 복제 탐지의 영역에 트리 구조 문서들에 대한 변화 탐지 기술을 적용하는 새로운 접근 방법을 제시한 논문이다. 프로그램을 토큰들의 시퀀스로 보는 기존의 논문들과는 달리 트리 구조로 접근하여 프로그램을 XML 문서로 변화시키고 이에 대해 X-tree Diff+ 알고리즘을 적용하였다. 이 과정에서 유사도와 편집 스크립트를 생성하고 이를 통해 프로그램 복제 탐지 분석을 수행하였다. 복제 여부 판정 과정은 유사도를 통한 정량적 분석과 편집 스크립트를 분석을 통한 정성적인 분석 과정을 통해 이루어지는데, SCDC의 유사도는 JPlag의 유사도에 비해 월등한 성능을 보이고 있으며 또한 정성적인 분석과정은 본 논문에서 새로이 제안하는 방법이다.

현재 개발된 프로토타입은 아직 사용자의 인터페이스의 개선의 여지가 많다. 특히 컴퓨터 프로그래밍 교육이라는 관점에서 어떤 사용자 인터페이스가 적당할지는 추후 연구 주제이다. 앞으로 효과적인 프로그램 복제 탐지를 기존의 연구 방법들과 통합하여 다양한 시도를 계획 중에 있다. 또한 현재는 단일 파일의 C 프로그램들에 대한 복제 탐지를 수행하고 있으나 이를 복수의 파일로 구성된 프로그램으로, 그리고 JAVA나 C++과 같은 객체지향 프로그래밍 언어로 확장할 계획이다. 자동 파서 생성기인 PCCTS의 특성 상 다른 언어로 확장하는 것이 그다지 어렵지 않다.

참 고 문 헌

[1] X. Chen, B. Francia, M. Li, B. Mckinnon, A. Sker, "Shared Information and Program Plagiarism Detection," IEEE Transactions on Information Theory, v. 50 n.7, 1545-1551, 2004.

[2] S. Schleimer, D. Wilkerson, A. Aiken, "Winnowing: Local Algorithms for Document Fingerprinting," SIGMOD'03, June 2003.

[3] D. Saccol, N. Edelweiss, R. Galante, C. Zaniolo, "XML Version Detection," DocEng'07, Aug. 2007.

[4] 김영철, 곽동규, 문현주, 최종명, 유재우, "프로그램 문장 결합 및 제어 구조를 이용한 유사도 평가," 정보과학회논문지: 기술교육 제 2권 제 1호, 2005.

[5] 한소정, "오픈 소스코드 표절 탐지 기법," 이화여자대학교 대학원 컴퓨터정보통신공학과 석사학위 논문, 2009.

[6] S.K. Lee, D.A Kim, "Efficient Change Detection

In Tree-Structured Data," Lecture Notes in Computer Science (LNCS2713) pp675-681, 2003.
[7] A. Haake, "CoVer: A Contextual Version Server for Hypertext Applications," In Proc. of 4th ACM Conf., Hypertext, pp.43-52, Milan. Italy, Nov. 1992.
[8] K. Osterbye, "Structural and Cognitive Problems in Providing Version Control for Hypertext," In Proc. of 4th ACM Conf., Hypertext, pp33-42, Milan. Italy, Nov. 1992.
[9] W. Labio and H. G. Molina, "Efficient snapshot differential algorithms for data warehousing," In Proc. of 20th Conf. VLDB, pp.63-74, Bombay. India, Sep. 1996.
[10] J. Widom and S. Ceri, Active Database System: Triggers and Rules for Advanced Database Processing, Morgan Kaufmann, 1996.
[11] E. W. Myers, "An O(ND) Difference Algorithm and Its Variations," Algorithmica, 1(2), pp.251-266, 1986.
[12] "Concurrent Versions System(CVS)," Free Software Foundation, http://www.gnu.org/manual/cvs-1.9.
[13] S. Chawathe, A. Rajaraman, H. G. Molina and J. Widom, "Change Detection in Hierarchically Structured Information," In Proc. of ACM SIGMOD Int'l Conf. on Management of Data, Montreal, June 1996.
[14] S. M. Selkow, "The tree-to-tree editing problem," Information Proc. Letters, 6, pp.184-186, 1977.
[15] K. Tai, "The tree-to-tree correction problem," Journal of the ACM, 26(3), pp.422-433, July 1979.
[16] S. Lu, "A tree-to-tree distance and its application to cluster analysis," IEEE TPAMI, 1(2), pp.219-224, 1979.
[17] J. T. Wang and K. Zhang, "A System for Approximate Tree Matching," IEEE TKDE, 6(4), pp.559-571, August 1994.
[18] S. Chawathe and H. G. Molina. "Meaningful Change Detection in Structured Data," In Proc. of ACM SIGMOD '97, pp.26-37, 1997.
[19] G. Cobéna, S. Abiteboul and A. Marian, "Detecting Changes in XML Documents," the 18th ICDE, 2002.
[20] 이석균, 김동아, "X-tree Diff: 트리 기반 데이터를 위한 효율적인 변화 탐지 알고리즘," 정보처리학회 논문지 10-C권 6호 pp683-694 2003.
[21] S.K. Lee, D.A Kim, "X-Tree Diff+:Efficient Change Detection Algorithm in XML Documents," Lecture Notes in Computer Science(LNCS4096), Springer Verlag, pp1037-1046, 2006.
[22] PCCTS, http://www.antlr2.org/pccts133.html.
[23] JPlag, https://www.ipd.uni-karlsruhe.de/jplag/

저 자 소 개

이 석 균(정회원)
1982년 서울대학교 경제학 학사.
1990년 University of Iowa 전산과학 석사.
1992년 IEEE 8th International Conference on Data Engineering 최우수 논문상 수상.
1993년 University of Iowa 전산과학 박사.
1993년~1997년 세종대학교 정보처리학과 전임강사.
1997년~현재 단국대학교 컴퓨터학부 교수.
<주관심분야 : 데이타 모델링, 데이타베이스에서 불완전 정보관리, 객체지향 데이타베이스 시스템, 데이타베이스 질의어, 시각질의어, 다중처리기에서의 실시간 스케줄링, 데이터웨어하우스, 데이터마이닝, 데이터베이스 시각 질의어, 문서의 Diff 알고리즘, XML, Change Detection, Version Control>