프로그램 유사도 평가 알고리즘☆

A Program Similarity Evaluation Algorithm

김 영 철* 황 석 찬** 최 재 영***
Young-Chul Kim Seog-Chan Hwang Jaeyoung Choi

요 약

본 논문에서는 서로 다른 두 개의 C 프로그램의 구문트리를 이용하여 유사도를 평가하는 시스템을 제시한다. 구문 트리를 이용하는 방법은 기존의 유사도 평가 방법과는 달리 들여쓰기, 여백, 설명문 등 프로그램과 무관한 프로그램 스타일의 변화에 민감하지 않으며, 문장, 코드 블록, 함수 등의 순서 바꾸기 같은 제어 구조의 변경에 민감하지 않은 특징을 가지고 있다. 그리고 프로그램을 파싱함으로써 구문 오류도 함께 검사할 수 있는 장점을 제공한다. 논문에서는 유사도를 평가하기 위한 알고리즘과 함께 프로그램의 비교횟수를 줄이기 위한 그룹 짓기 알고리즘도 같이 제공한다. 실험부분에서는 구문트리 비교방법을 이용한 프로그램의 유사도 평가 결과와, 그룹 짓기를 수행한 후에 많은 비교 횟수를 줄일 수 있다는 것을 보여준다.

Abstract

In this paper, we introduce a system for evaluating similarity of C program source code using method which compares syntax-trees each others. This method supposes two characteristic features as against other systems. It is not sensitive for program style such as indentation, white space, and comments, and changing order of control structure like sentences, code block, procedures, and so on. Another is that it can detect a syntax-error cause of using paring technique. We introduce algorithms for similarity evaluation method and grouping method that reduces the number of comparison. In the examination section, we show a test result of program similarity evaluation and its reduced iteration by grouping algorithm.

Keyword : program plagiarism, evaluation algorithm, AST, program parsing, grouping

1. 서 론

오늘날 프로그래밍에 관한 많은 서적과 인터넷 매체의 발달로 인하여 프로그램을 연구하는 개발자나 프로그래밍 언어를 공부하는 학생들은 언제 어디서든 쉽게 원하는 프로그램 예제를 찾아 사용할 수 있다. 이러한 매체의 발달은 프로그램을 공부하는 많은 사용자에게 도움을 주지만, 다른 한편으로는 공유된 프로그램을 이용함으로써 프로그래밍 언어의 연구를 소홀히 하거나 쉽게 과제물을 작성할 수 있는 단점도 있다. 이렇게 작성된 과제물들은 서로 엇비슷하여 검수자가 제출된 과제물을 정확히 비교, 평가하기도 어렵게 만든다. 특히, 약간의 코드 수정 및 스타일이 변경된 과제물은 평가를 더욱 어렵게 만들며, 과제물을 평가하는 검수자마다 평가 결과가 각기 다를 수 있다[1,2,3]. 본 연구에서는 제출된 프로그램의 구문 트리를 이용하여 서로 다른 프로그램 사이의 유사도를 평가하는 시스템을 제시한다. 이 시스템은 파싱과정에서 생성된 구문트리를 이용하여 두 프로그램 사이의 유사도를 판정한다. 구문트리를 이용하여 유사도를 평가하면 프로그램 소스 코드의 변형에 관계없이 유사 여부를 구조적으로 검사할 수가 있으며, 구문 오류 검사도 수행할 수

* 정 회 원 : 숭실대학교 정보미디어기술연구소
전임연구원
yckim@ss.ssu.ac.kr(제 1저자)
** 정 회 원 : 한국과학기술정보연구원 슈퍼컴퓨팅센터
선임연구원
seogchan@kisti.re.kr(공동저자)
*** 종신회원 : 숭실대학교 컴퓨터학부 부교수
choi@ssu.ac.kr(공동저자)
[2003/12/01 투고 - 2003/12/22 심사 - 2004/09/22 심사 완료]
☆ 본 연구는 숭실대학교 교내연구비 지원으로 이루어졌음.

있는 장점을 가지고 있다. 본 연구는 다음과 같은 환경에서 학생들이 과제물로 제출한 프로그램 소스 코드 평가의 어려움 때문에 연구를 시작하게 되었다.

1. 강사 수에 비해 학생 수가 너무 많아 과제물 평가가 여의치 않다.
2. 많은 참고문헌과 인터넷으로 인한 정보 공유 때문에 과제물이 유사하다.
3. 프로그램 과제물 대행(복제) 업무 사이트가 활황을 이루고 있다[조선일보 2003.2, 2002.2.1, 1999.7.13 기사].

프로그램의 복제는 갈수록 더 다양해지고 있다. 심지어 몇몇의 학생들은 과제물을 수행하는 시간보다 프로그램 소스 코드를 편집하는 과정에 더 시간을 투자하고 있다는 기사가 나와[조선일보 2003. 2] 눈길을 끌고 있다. 이처럼 여러 가지 현실적/환경적 요인으로 볼 때 프로그램의 소스 코드의 유사 여부를 판단하는 것은 아주 중요한 요소가 된다.

Hamblen[3]은 프로그램 소스 코드의 복제(plagiarism in software)를 "한 프로그램에 약간의 루틴의 변형을 가해서 만들어내는 프로그램"이라고 정의하였다. 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.

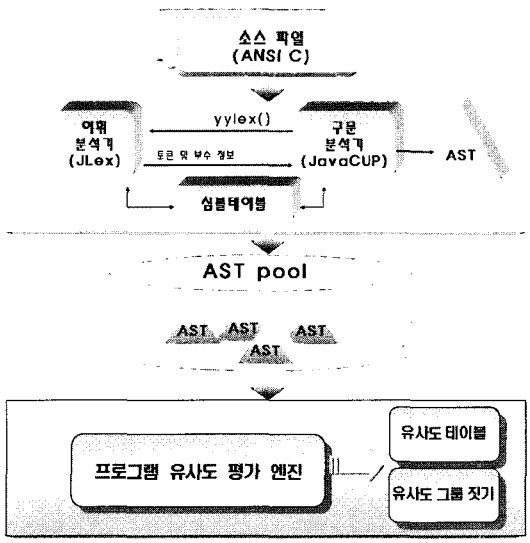
이러한 여러 가지 이유로, 본 논문에서는 이러한 프로그램의 유사성을 판단하는 시스템을 설계하고 구현한다. 본 논문에서 제시한 프로그램 유사도 평가 시스템 모델은 다음 그림 1과 같다.

그림 1과 같이 본 시스템은 프로그램 유사도 평가를 수행하기 위하여 1989년에 제정된 ANSI C 언어로 작성된 프로그램을 대상으로 하며, 다양한 시스템에서 활용하기 위하여 Java 언어로 구현하였다. 또한 C 언어의 어휘 분석 및 구문 분석을 수행하기 위하여 컴파일러 보조도구인 JLex[4]와 JavaCUP[5]를 이용하였다. 또한 구문 분석이 끝난 후에는 프로그램에 적합한 구문트리(AST : Abstract Syntax Tree, 이하 AST로 칭함)를 생성하여 유사도를 평가한다. 특히, 많은 프로그램을 비교하기 위해 AST pool과 같은 저장소를 이용하였다. AST pool은 파싱과정에서 생성된 구문트리의 집합이다. 이 외에 프로그램 유사도 평가 엔진 및 유사도 테이블, 유사도 그룹 짓기에 관한 사항은 제 3장에서 설명하기로 한다.

본 논문은 다음과 같이 구성되었다. 제2장에서는 관련연구에 대해서 기술하였으며, 제3장에서는 프로그램 유사도 평가 알고리즘 및 그룹 짓기에 대해서 기술하였다. 제4장에서는 실험 및 평가를 기술하였으며, 마지막으로 제5장에서는 결론 및 향후 연구 과제에 대해서 기술하였다.

2. 관련 연구

프로그램 유사도와 관련된 연구는 20-30년 전부



〈그림 1〉 프로그램 유사도 평가 시스템 모델

터 진행되어 왔다. 연구 초기에는 문서의 유사성과 관련된 연구가 수행되었으며, 이를 토대로 프로그램 유사도 검사를 수행하는 방향으로 연구되었다.

2.1 유사도 평가 시스템

문서의 복제는 글을 쓸 때 원본 자료의 출처를 분명히 밝히지 않고 자신의 의견처럼 서술하는 것을 말한다[6]. 문서는 선형 구조를 이루고 있으나 문서의 크기에 따라 선형 구조의 길이가 변하기 때문에 구조적인 특징보다는 통계학적인 특징을 이용하여 유사도를 판정한다. 이처럼 문서의 유사도 판정 기준에 통계학적인 특징을 이용하는 것을 지문법(fingerprint)[7]이라고 한다. 지문법은 두 개의 문서에서 사용된 단어들의 유사성을 살피거나 사용된 단어의 빈도수 등을 비교하는 방법이다. 초기의 프로그램 유사도 평가 방법은 지문법을 이용한 프로그램 유사도 평가 방법이다. 즉, 프로그램의 통계적인 부분을 이용하여 두 프로그램 사이의 유사도를 평가한다. 지문법을 기반으로 만들어진 최초의 시스템은 Halstead 매트릭스[8] 시스템이며, 이 매트릭스를 이용하였거나 확장해서 만든 시스템은 Ottenstein[9], Donaldson [10], Berghel[11] 등이다. 이 중 Donaldson이 제시한 복제 방법은 Halstead 매트릭스 외에, 반복문의 수, 프로시저 문의 수와 같은 프로그램 구조를 혼합하여 이용했다는 점도 주목할 만하다.

가장 최근에 개발된 프로그램 유사도 평가 시스템들은 어휘 분석 과정에서 생성한 "토큰"을 이용하여 유사도를 평가한다. 프로그램 "토큰"을 이용하면 이전 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. 토큰 스트링을 이용하는 방법 중 생물 정보학을 이용한 연구가 있었다. 즉, 유전체의 서열들을 비교하여 유사한 영역을 찾아내는 서열 분석 방법[17, 20]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.

2.2 프로그램 유사도 평가 방법

M.J. Wise에 의해 개발된 YAP3[12]는 구조적 매트릭스 방법을 이용한 유사도 평가 시스템이다. YAP3에서는 유사도를 평가하기 위하여 프로그램을 재설정한다. 즉, 설명문과 스트링 상수 제거하기, 대문자를 소문자로 바꾸기, 소스 프로그램을 똑같은 혹은 유사한 동의어로 변환하기, 함수 호출 순서를 재 정렬하기 등을 수행한다. 또한 제어 구조, 블럭문, 서브프로그램 및 라이브러리 함수 등과 같은 프로그램 구조를 가리키는 토큰을 유지한다는 특징을 가지고 있다. YAP 시스템에서 이용되고 있는 유사도 평가 값은 0 ~ 1사이의 값을 나타낸다. 유사도 값이 0일 경우에는 두 프로그램이 완전히 다른 프로그램을 의미하며, 유사도 값이 1일 경우에는 두 프로그램은 완전히 일치한다는 것을 의미한다. YAP에서 사용되는 유사도 평가의 기준 Match는 다음과 같다.

$$\frac{(same - diff)}{\min file} - \frac{(\max file - \min file)}{\max file} \quad (1)$$

식(1)에서 maxfile과 minfile은 각각 두 파일 중 큰 파일과 작은 파일의 길이를 나타낸다. 또한 변수 "same"은 두 파일에서 공통적으로 나타나는 토큰의 수를 나타내며, "diff"는 블록 내부에서 다른 줄 수를 나타낸다.

Sim[15]에서도 역시 두 토큰 스트링 사이의 유사도를 평가하기 위한 값으로 0 ~ 1 사이의 실수 값을 이용하고 있다. Sim에서 사용되는 유사도 평가 수식은 다음 식(2)와 같다.

$$S = 2 * \frac{score(s,t)}{score(s,s) + score(t,t)} \quad (2)$$

식(2)에서, score(s, t)는 두 프로그램 토큰(s 와 t)의 일치 여부와 관련되어 있는 점수를 나타낸다. 즉, sim에서도 유전자 염기 서열 기법을 이용하기 때문에 이 서열에 대한 가중치를 score로 표현하고 있다.

SID[16]는 Kolmogorov 복잡도를 기반으로 하고 있다. SID 역시 Sim과 마찬가지로 0 ~ 1 사이의 유사도 값을 나타내었으며, 다음 식(3)과 같이 유사도를 측정한다.

$$R(s, t) = \frac{K(s) - K(s|t)}{K(st)} \quad (3)$$

식(3)에서 K(s|t)는 Kolmogorov 복잡도를 나타내며, t가 없다(empty)면, Kolmogorov 복잡도는 K(s)가 된다. 또한 K(s)-K(s|t)는 상호 배제 정보(mutual information)로 K(s)와 K(t)의 다른 부분을 나타낸다. 반대로 K(st)는 공통된 정보를 나타낸다.

I. D. Baxter[18] 등은 두 프로그램 사이에 중복된 코드 조각을 찾는 연구를 하였다. 이 연구에서는 중복된 코드를 "clone"이라 칭하였으며, 중복된 코드를 통하여 유사도를 평가하는 것을 제시하였다. [18]에서 제시한 유사도 평가 수식은 다음 식(4)와 같다.

$$Similarity = 2 * \frac{S}{2 * S + L + R} \quad (4)$$

식(4)에서 S는 공유 노드의 수를 나타내며, L은 원본 소스에 있는 트리 중 대상 소스에 없는 노드의 수를 나타낸다. 또한 R은 다른 소스에 있는 트리 중 원본 소스와 다른 노드의 수를 나타낸다.

식(1)부터 식(4)는 프로그램의 유사성 여부를 판단하는데 이용된다. 이러한 수식은 프로그램 유사도 평가 방법에 따라 적용된다. 예를 들면, YAP3에서 사용된 식(1)은 토큰과 블록 내부의 줄 수를 이용하여 기술하였고, Sim이나 SID에서 사용된 식(2)와 식(3)은 두 프로그램의 토큰과 공유 정보를 이용한다.

본 시스템의 유사도 측정에 이용된 수식은 식(4)와 개념 면에서 일치한다. 다만 사용되는 수식만 다를 뿐 의미 자체는 일치한다. 왜냐하면 두 프로그램 사이의 일치된 부분(S)과 일치되지 않는 부분(L) 사이에 유사도를 판단하기 때문이다. 본 논문에서 이용되는 유사도 평가 수식은 다음 제3장에서 기술한다.

3. 유사도 평가 알고리즘 및 그룹 짓기

3.1 유사도 평가 알고리즘

파싱 과정에서 생성된 구문 트리는 언파서에 의해서 노드스트링으로 변환된다. 노드스트링이란 구문 트리의 노드를 포스트오더(post order) 방법으로 검색하여 선형적으로 나열한 스트링을 말한다.

정의 1. 프로그램 복제 검사할 두 프로그램은 각각 P1과 P2로 정의하며, P1은 원본 프로그램으로, P2는 검사될 대상 프로그램으로 정의한다.

정의 2. 노드스트링 A와 B는 각각 언파서에 의해 만들어진 원본 소스 프로그램(P1)과 복제 대상 소스 프로그램(P2)의 노드스트링으로 정의한다.

정의 2의 노드스트링은 이전 장에서 언급한 구문 트리를 선형적으로 나열한 것을 말한다. 즉, 파싱과정에서 생성된 구문 트리를 언파서가 노드스트링으로 바꾼 문자열을 의미한다. 따라서 본 시스템은 복제 검사될 두 프로그램의 노드스트링을 입력받아 유사도를 측정한다. 또한 유사도 평가 알고리즘은 크게 3단계로 수행되며, 최종적으로 0과 1사이의 유사도 값을 반환한다.

정의 3. 프로그램 유사도 평가 알고리즘에 의해 수행된 두 프로그램 사이의 유사도 값은 다음과 같이 0보다는 같거나 크고, 1보다는 같거나 작은 값을 나타낸다.

$$0 \leq \text{유사도 } sim(A, B) \leq 1 \quad (5)$$

본 시스템의 유사도 알고리즘은 식(5)와 같이 0보다는 같거나 크고, 1보다는 작거나 같은 값을 반환하여 주는데, 이 값의 의미는 다음과 같다. 유사도 값이 0인 경우에는 검사될 두 노드스트링 중 하나는 공 노드스트링(empty NodeString)이거나 minlength 값이 두 노드스트링 중 일치하는 스트링이 minlength 길이보다 작을 경우에 해당된다. 유사도 값이 1인 경우는 검사될 두 노드스트링 A와 B는 완전히 일치함을 의미한다. 유사도 값이 0보다 크고 1보다 작은 실수인 경우는 복제 검사될 두 프로그램은 부분적으로 일치함을 의미한다.

정의 4. 두 프로그램 사이의 유사도 결과 값은 **완전 복제, 강한 유사, 중간 유사, 약한 유사** 중 하나로 정의한다[19]. 완전 복제는 유사도 값이 정확히 1 값으로 정의한다. 강한 유사는 유사도 값이 0.9 ~ 1 사이의 실수 값일 경우로 정의한다. 또한 중간유사는 유사도 값이 0.7 ~ 0.9 사이 값을 경우로 정의하며, 약한 유사는 유사도 값이 0.7 이하일 경우로 정의한다.

정의 4에서 이용된 유사도 결과 값은 [19]에서 평가된 값으로 이용되었다. 정의 4에서 완전 복사는 두 프로그램 사이의 노드스트링 A, B가 완전히 일치함을 나타낸다. 또한 강한 유사는 0.9보다 크고 1보다 작은 경우에 해당하며, 두 프로그램 P1, P2 사이에는 거의 복제되었을 가능성이 크다는 것을 의미한다. 또한 중간 유사는 유사도가 0.7보다 크고 0.9보다 작은 경우로 두 프로그램 P1, P2 사이에는 많은 부분이 일치하는 것을 의미한다. 중간 유사일 경우에는 보다 세심한 복제 검사가 필요하다. 약한 유사는 두 프로그램 사이에 복제되지 않았음을 의미한다. 즉, 0.7이라는 값은 전체 코드의 70%가 같지만 프로그램 언어 특성상 같은 키워드나 문장 혹은 함수 등이 어느 정도 일치되는 요소가 있으므로 같지 않다고 할 수 있다.

(1) 두 노드스트링에서 일치하는 스트링 찾기

알고리즘 1에서는 입력된 두 개의 노드스트링(A, B)에서 첫 번째로 일치하는 서브스트링을 검색한다. 이때 가장 길게 일치되는 스트링(MMS : Maximum Match String)을 찾는 과정을 수행한다. 따라서 알고리즘 1의 주 목적은 두 노드스트링 중 일치되는 가장 긴 서브스트링을 찾는 것이다.

알고리즘 1. 두 노드스트링(A, B)에서 매칭 스트링 찾기

```
string MatchString(NodeString A, NodeString B) {
    long int i, j;
    long int matchsize, maxmatch;
    matchstring = "";

    for each unmark(Ai) in A {
      for each unmark(Bj) in B {
        matchsize = 0;
        while(Ai+matchsize == Bj+matchsize &&
        Ai+matchsize && Bj+matchsize)
            matchsize++;
        if (matchsize > maxmatch) {
            matchstring = match(Ai, Bj,
            matchsize);
            maxmatch = matchsize;
        }
      } end for
    } end for
    Deleting(A, B, i, j, matchsize);
    return matchstring;
}
```

알고리즘 1에서, minlength는 복제 대상 노드 스트링 중에서 최소로 일치하는 서브노드스트링의 개수를 말한다. 예를 들면, minlength가 3일 경우, 노드스트링 A, B 중 적어도 3개 이상의 노드가 일치할 경우에 그 서브스트링을 찾아준다. 본 시스템에서는 minlength를 5로 설정하였다. 왜냐하면 가장 간단한 C 언어의 문장은 노드 수가 5개가 되기 때문이다. 이처럼 minlength 값은 검사하는 대상 프로그램에 따라 값을 변경하여 유사도를 측정할 수 있다. 또한 maxmatch는 일치된 스트링의 개수를 말하며, minlength 보다 클 동안 계속 스트링을 비교한다.

서브스트링이란 노드스트링 중 특정한 일부분의 스트링으로 정의하였으며, X_i 형태로 표현하였다. 여기서 X는 노드스트링을 말하며, i는 X 스트링의 인덱스로 말한다. 또한 matchsize는 두 노드스트링 중 일치하는 서브스트링의 개수를 말한다. 즉, matchsize는 두 노드스트링에서 일치하는 서브스트링의 개수를 일컫는다. 예를 들면, 두 노드스트링 A와 B가 각각 표 1과 같다면 matchsize 값은 3(즉, 34, 25, 54)과 8(즉, 81, 83, 84, 22, 55, 44, 33, 90)이 된다.

〈표 1〉 노드스트링 A와 B

A(P1nodestring) :	23 34 25 54 44 45 49 81 83 84 22 55 44 33 90 68
B(P2nodestring) :	34 25 54 46 47 81 83 84 22 55 44 33 90 93 92 95 34 35

또한 match(A_i, B_j, matchsize)는 두 노드스트링 A와 B에 대해서 각각 인덱스 i, j부터 시작하여 matchsize 만큼 일치되는 스트링으로 match 함수가 반환하는 값은 일치되는 두 노드스트링의 서브스트링이다. 예를 들면, 표 1에서 두 노드스트링 A와 B 중 일치되는 서브스트링 { 34, 25, 54 }가 있을 때, i, j의 값은 각각 1과 0이 되며, matchsize는 3이 된다. 따라서 "34, 25, 54"가 반환된다.

알고리즘 1은 노드스트링 A에서 각각의 서브스트링 A_i에 대해서 동일한 B_j를 찾는다. 동일한 서브 노드스트링을 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 반환되며, 동시에 노드스트링 A, B에서 삭제되기 위하여 알고리즘 2를 호출한다.

(2) 일치하는 스트링의 삭제

유사도 검사 알고리즘의 알고리즘 2는 일치된 노드스트링 A, B의 서브스트링을 찾아 삭제하고 마크 비트를 삽입하는 단계이다. 즉, 알고리즘 1에서 찾은 서브스트링을 나중에 중복 검사되지 않기 위하여 수행되는 단계이다.

알고리즘 2. 일치된 서브스트링을 노드스트링에서 삭제

```
void Deleting(NodeString A, NodeString B, long int i, long int j, long int matchsize) {
    long int k;
    for k = 0 to matchsize -1 {
        Delete(Ai+k);
        Delete(Bj+k);
    } end for
}
```

알고리즘 2의 Delete($X_{i+matchsize}$) 함수는 노드스트링 X의 "i+matchsize"번째의 스트링을 삭제하는 함수이다. 알고리즘 2에서는 두 노드스트링 A, B에서 일치되는 서브스트링을 Delete 함수를 이용하여 노드스트링에서 삭제한다. 일치하는 서브스트링을 삭제하는 이유는 두 노드스트링이 비교된 후에, 또다시 중복 검사되는 것을 막기 위함이다.

(3) 두 노드스트링의 유사도 검사

알고리즘 3은 노드스트링 A, B에서 min-length보다 큰 일치가 발생할 때까지 알고리즘 1, 2를 반복한다. 또한 모든 일치 스트링을 찾은 후에는 두 노드스트링의 유사도 값을 계산하고 반환한다.

알고리즘 3. 유사도 검사 알고리즘

```
double Sim(NodeString A, NodeString B,
long int minlength) {
    String matchstring, totalmatch-
    string;
    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(to-
       talmatchstring)
          matchlength = matchlength +
          Length(matchstring);
       end for
       return (2 * matchlength / (Length(A)+Length(B)));
}
```

알고리즘 3에서, Set(totalmatchstring)는 두 노드스트링에서 찾은 모든 서브스트링을 보관하는 함수이다. 예를 들면, 표 1과 같은 두 노드스트링이 있다면, Set(totalmatchstring) 함수는 최종적으로 { {34, 25, 54}, {81, 83, 84, 22, 55, 44, 33, 90} } 값을 갖는다.

또한 Length(X)는 노드스트링 X의 길이를 구하는 함수로 알고리즘 1, 2에서 구한 일치된 서브스트링의 길이와 입력으로 들어온 노드스트링의 길이를 구해주며, 최종적으로 유사도 수식에서 이용된다.

본 알고리즘 설명하기 위하여 표 1과 같은 노드스트링 A와 B가 있다고 가정하자. 알고리즘 1에서는 두 노드스트링의 일치된 부분을 검사한다. 즉, 예에서 { {34, 25, 54}, {81, 83, 84, 22, 55, 44, 33, 90} }을 찾아낸다. 알고리즘 2에서는 알고리즘 1에서 찾은 부분 스트링을 matchstring에 추가한 후, A, B 노드스트링에서 이를 마크한다. 왜냐하면, 찾은 스트링은 다음에 일치하는 스트링과 다시 비교하는 것을 피하기 위함이다. 알고리즘 3에서는 알고리즘 1, 2를 이용하여 유사도를 측정한다. 즉, 노드스트링 A, B 유사도는 최종적으로 다음과 같이 0.647이 된다.

$$\text{sim(A, B)} = 2 * \frac{matchlength}{Length(P1)+Length(P2)} = 2 * \frac{(3+8)}{16+18} = 0.647$$

본 알고리즘의 시간 복잡도와 관련하여 최악의 경우 시간 복잡도는 $O(n^3)$이다. 즉, 알고리즘 1에서는 2개의 반복 구조를 취하므로 시간 복잡도는 $O(n^2)$이고, 알고리즘 3에서 한번의 반복문을 취하므로 전체 시간 복잡도는 $O(n^3)$을 나타내게 된다. 또한 최상의 경우 시간 복잡도는 $O(n^2)$이다. 왜냐하면, 두 노드스트링이 완전히 일치한다면, 알고리즘 3의 반복문은 1번만 수행된다. 따라서 최상의 경우 시간 복잡도는 $O(n^2)$이 된다.

3.2 유사 프로그램 그룹 짓기

본 논문에서 제시한 프로그램 유형 복제 검사 시스템은 n개의 프로그램에 대해서 모든 프로그램을 비교하는데 걸리는 횟수는 n(n-1)/2 번이다. 예를 들어 100개의 프로그램을 복제 검사할 경우 총 4950(100*99/2)번의 비교를 수행한다. 따라서

복제 검사 알고리즘의 시간과 프로그램 수, 프로그램의 라인을 모두 비교할 경우에는 아주 많은 시간 비용이 든다. 따라서 프로그램의 유사성이 높은 집단을 그룹으로 관리하면 보다 적은 횟수로 비교할 수 있다. 그룹 짓기(clustering)란 유사한 집단의 데이터를 분류해서 하나의 그룹을 만드는 것이다. 즉, 동일한 그룹에 있는 데이터들은 서로 유사한 집단으로 분류되며, 서로 다른 그룹에 속한 데이터들은 상이하다. 따라서 가장 좋은 그룹 짓기는 한 그룹 내에서 데이터 간의 유사도는 높고, 다른 그룹에 존재하는 데이터는 낮은 유사도를 갖는 것이다. 그룹 짓기에서 가장 중요한 요소는 데이터 사이의 나누는 기준이 어떤가에 따라 달라진다. 본 논문에서는 그룹 간에 전역 유사도를 기준으로 나누어 그룹 짓기를 수행한다.

정의 5. 그룹 짓기는 전역 유사도 S에 의해 설정된 그룹 G로 정의하며, 다음과 같은 유사도를 갖는 프로그램들이다.

$$\mathbf{sim(A, B) \geq S, \forall_A \in G, \forall_B \in G} \quad (6)$$

식(6)과 같이 그룹 짓기는 특정한 유사도가 모두 S 값 이상인 프로그램들을 말한다. 본 논문에서 이용한 그룹 짓기 알고리즘은 다음과 같다.

```
file *P;
int g;
boolean addgroup = false;
add P to G(1);
i =1;
Set G(1) = Ø
while( not eof) {
        input P;
        for each i in G(i)
                if Sim(P, G(i)) > g then
                {
                        add P to G(i);
                        addgroup = true;
                }
        end for
        if (not addgroup) then {
                        i = i + 1;
                        Set G(i) = Ø
                        add P to G(i);
        }
}
```

위에서 제시한 그룹 짓기를 이용하면, 많은 비교 횟수를 줄일 수 있다. 예를 들면, n개의 프로그램에 대해서 모두 비교한다면 정상적으로 n(n-1)/2번이다. 그러나 그룹 짓기 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. 또한 모든 프로그램 비교 횟수는 전역 유사 G값에 비례한다. 즉, 전역 유사성 G값이 크면 클수록 그룹이 많아지므로 비교횟수가 많아지며, 작을수록 그룹수가 적어지므로 모든 프로그램의 비교횟수는 적어진다. 그룹 짓기에 관한 실험은 제4장에서 보여준다.

4. 실험 및 평가

4.1 실험

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[4]와 Java CUP[5] 유틸리티를 사용하였다.

〈표 2〉 원본 프로그램(source.C)

```
#include <stdio.h>
void main(void) {
 long result; int index=0;
 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;
 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[in-
    dex++]=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〉 유사한 프로그램(sim.C)

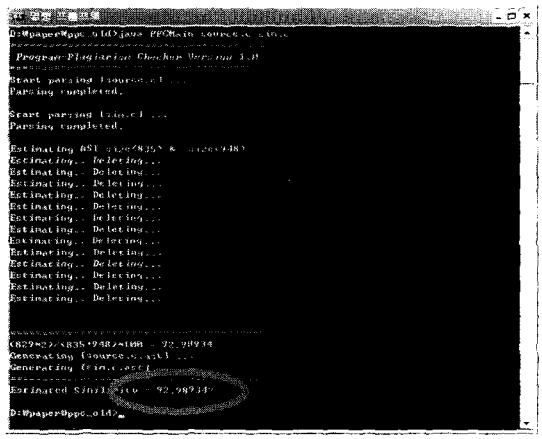
```
#include <stdio.h>
void main(void) { /* 스타일 변화 */
long result; int index=0; /* 문장 위치 바꿈 */
int n1, n2, m1, m2, divs[100], lnum, i,
    flag;
printf("Input 2 numbers for calculating
    GCM...\n");
scanf("%d %d", &n1, &n2); m1= n1; m2 = n2;
for(;;) { /* 제어 구조 변환 while ==> for */
flag = 0; lnum = (m1>=m2?m1:m2); /* 문장 바꾸
    기 */
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); }
}
```

실험 1. 두 프로그램의 유사도 평가

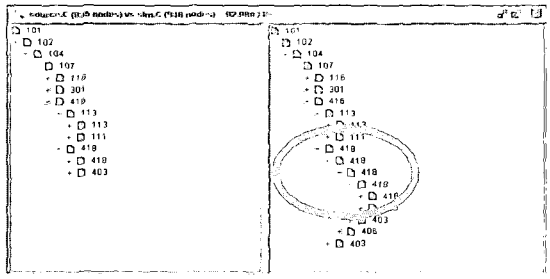
실험을 위하여 다음 표 2, 3과 같은 최소 공배수를 구하는 두개의 프로그램이 있다고 가정하자. 표 3은 2에서 스타일이 변경되거나, 유사한 제어문으로 변경된 것이다.

두 프로그램의 비교 결과는 다음 그림 2와 같다. 그림 2에서처럼 두 프로그램의 유사도 값은 92.98934 값이다. 실험에서와 같이 같은 프로그램인데도 유사도 값이 적게 나온 이유는 위의 표 3에서처럼 중복 코드(dummy code)가 많이 들어갔기 때문이다.

따라서 본 시스템은 중복 코드(dummy code)가 많이 있다면 본 시스템의 유사도는 현저하게 떨어질 수도 있다. 다음 그림 3은 위에서 비교한 source.C 파일과 sim.C파일을 구문트리를 보여준다. 그림에서 정수는 각각 구문트리의 노드를 나타낸다.



〈그림 2〉 source.C와 sim.C 프로그램의 유사도 평가

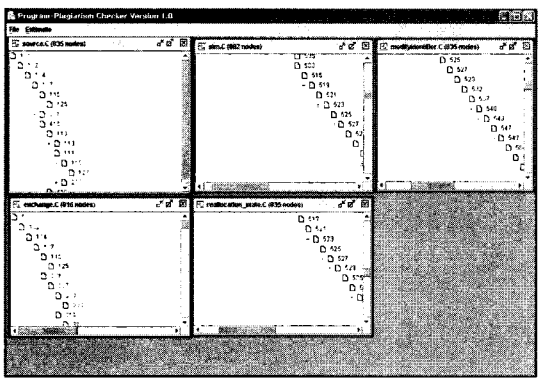


〈그림 3〉 source.C와 sim.C 프로그램 구문트리

〈표 4〉 전체 프로그램의 유사도 결과

File Name	129.c	035.c	039.c	078.c	180.c	469.c	486.c	123.c	456.c
160.c	0.666908	0.855566	0.929878	0.898305	0.752316	0.822222	-1	0.848864	0.872826
129.c		0.786918	0.706133	0.663991	0.52589	0.808895	-1	0.604669	0.643149
035.c			0.890583	0.84	0.691122	0.917197	-1	0.76459	0.813511
039.c				0.909465	0.761791	0.822203	-1	0.835903	0.864979
078.c					0.812539	0.799641	-1	0.891705	0.906388
180.c						0.657128	-1	0.854806	0.844215
469.c							-1	0.731173	0.778696
486.c								-1	-1
123.c									0.913507

그림 3은 두 프로그램에서 다른 부분을 잘 보여주고 있다. 즉, 두 프로그램에서 일치되는 코드는 붉은(■)표시로 나타나며, 흰색(□) 표시는 두 프로그램에서 일치되지 않는 부분을 나타내고 있다. 따라서 본 시스템은 중복 코드가 많이 들어가면 갈수록 유사도 값이 현저하게 떨어짐을 볼 수 있다. 다음 그림 4는 그림 1에서 언급하고 있는 AST pool을 보여준다.



〈그림 4〉 AST pool

실험 2. 유사도 테이블

본 절에서는 S대 컴퓨터 학부 학생들이 제출한 과제물에 대해서 유사도 평가를 수행하고 분석한다. 과제물은 정렬(sort)에 관한 C 프로그램 예제이며, 분석을 위하여 10명의 학생들의 과제를 추출하여 수행하였다. 다음 표 4는 전체 프로그램을 모두 비교하여 유사도 테이블로 나타낸 결과이다. 표에서 -1 값은 비교하지 않은 경우를 나타내며, 나머지 값들은 유사도를 나타낸다. 따라서 "486.c" 파일은 구문 오류가 있기 때문에 다른 프로그램과의 유사도 검사 대상에서 제외되었다.

표 4에서 알 수 있듯이 모든 프로그램들은 0.5 ~ 1 사이의 유사성을 잘 나타내고 있다. 특히 이 프로그램 사이에는 완전 복제가 없었으며, 강한 유사는 많이 나타났다. 만일 이 프로그램을 앞서 논의했던 그룹 짓기를 이용하여 수행하면 다음 실험 3과 같다. 다음 표 5는 유사도를 0.9로 설정하여 그룹 짓기를 수행한 실험 결과이다.

실험 3. 전역 유사도 0.9로 설정하여 그룹 짓기 수행

Grouping...
Group0 : 160.c, 039.c
Group1 : 129.c
Group2 : 035.c, 469.c
Group3 : 078.c, 456.c
Group4 : 180.c
Group5 : 486.c
Group6 : 123.c

〈표 5〉 전역 유사도 0.9로 설정하여 그룹 짓기를 수행한 결과

Group Name	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							

표 5에서 눈여겨 볼 점은 전역 유사성 값을 0.9(강한 유사) 전체를 비교한 결과보다 훨씬 적게 비교가 이루어졌다는 점이다. 본 실험의 신뢰성을 알아보기 위하여 전역 유사성 0.9에 대한 그룹 짓기 결과를 오프라인으로 검사해보았다. 즉 그룹 짓기 결과 그룹을 형성한 그룹에 대해서 조사한 결과 완전 복제는 없었으며, 비슷한 부분이 상당히 많았다. 이는 그룹이 형성된 프로그램은 실제 0.9 유사성을 갖는다는 것을 증명해주었다.

4.2 평가

구문트리를 이용한 프로그램 복제 검사 방법을 평가하기 위하여 기존의 프로그램 복제 검사 기법의 장단점을 먼저 알아보자. 다음 표 6은 프로그램의 다양한 복제 유형에 대해서 본 시스템과 관련 연구와 비교한 것이다.

표 6에서 나타난 바와 같이 프로그램 복제 검사를 수행하는 많은 기법들은 대체적으로 많은 장단점을 가지고 있다. 이 중 특히, 단점으로 지적되고 있는 문장 코드의 위치 변화, 오퍼랜드/오퍼레이터의 위치 변화, 프로그램 오류 판단 등은 본 시스템을 이용하면 쉽게 해결할 수 있다는 것을 실험에서 보여주었다.

이처럼 본 시스템을 이용하여 복제 검사를 수행하면 다음과 같은 장점을 가지고 있다. 첫째, 구문트리는 방대한 프로그램에서 작은 프로그램까지 유사도 검사에 적용될 수 있다. 둘째, 복제

〈표 6〉 복제 유형에 대한 기존 검사 기법의 비교
(O : 복제 판정, △ : 작은 영향, × : 아주 민감)

복제 대상 프로그램의 변화	매트릭스 카운팅	토큰 패턴 매칭	유전자 서열 분석	AST
정확히 원본 프로그램 복제	O	O	O	O
단순 설명문 변경	O	O	×	O
공백(White Space) 및 형식 변환	O	O	O	O
변수나 함수 이름 바꾸기	O	O	O	O
불필요한 변수 및 문장 추가	△	△	△	△
제어구조 바꾸기	×	×	×	△
타입 바꾸기	×	×	×	△
코드 블록, 문장, 함수 바꾸기	×	×	×	O
오퍼랜드/오퍼레이터의 바꾸기	×	×	×	O

를 한 당사자가 약간의 프로그램을 수정했을 경우에도 유사도 측정이 가능하다. 셋째, 컴퓨터가 정확히 판별하기 어려운 프로그램도 검사자에 의해서 유사도 측정이 가능하다. 넷째, AST는 프로그램에 오류가 있는 경우 트리를 생성할 수 없으므로 에러를 쉽게 판별해 낼 수 있다. 다섯째, 구문트리는 함수 각각에 대해서도 유사도를 검사할 수 있다. 여섯째, 복제를 한 당사자가 프로그램의 변수나, 함수 이름, 변수에 할당된 값 등을 바꾸었을 경우 유사도를 정확히 판별해 낼 수 있다. 일곱째, while이나 if, switch문 등의 안에 복제자가 간단한 변수나, 수치를 추가했을 경우 구문트리에 대한 전체적인 트리는 영향을 받지 않으므로 유사도를 검사할 수 있다. 여덟째, 함수나 순환문, if then, switch 등의 순서를 바꾸었을 경우, 즉 문장의 순서를 바꾸었을 경우에도 쉽게 유사도를 검사할 수 있다.

그러나 이러한 장점과는 달리 본 시스템은 몇 가지 단점도 가지고 있다. 프로그래머가 의도적으로 여분 코드(dummy code)를 삽입할 시 많은 수의 노드가 발생하여 유사도가 많이 떨어진다. 또한 많은 노드로 인하여 복제 검사 시간도 많이 걸린다. 특히 본 시스템은 프로그램의 크기가 크면 클수록 복제 검사하는 시간이 기하급수적으로 늘어날 수 있기 때문에 1000라인 이하에서 가능하다는 단점을 가지고 있다.

5. 결 론

본 논문에서는 서로 다른 두 프로그램의 유사도를 평가하는 시스템을 제시하고 구현하였다. 또한 프로그램의 구문 트리를 비교, 평가하기 위하여 유사도 평가 알고리즘을 제시하였다. 본 연구에서 제시한 프로그램 유사도 평가 알고리즘은 최악의 경우 $O(n^3)$와 최상의 경우 $O(n^2)$의 시간 복잡도를 갖는다. 또한 모든 프로그램의 비교 횟수를 줄이기 위하여 그룹 짓기를 수행하였다. 그룹 짓기를 수행한 결과 n개의 프로그램에 대해서 모든 프로그램의 유사도를 측정할 경우, 기존에 n(n-1)/2 번의 비교 횟수가 최소 n-1번의 비교 횟수로 줄일 수 있다는 것을 보여주었다.

본 시스템의 평가 및 실험 부분에서는 구문트리를 이용하여 프로그램 유사도 평가와 그룹 짓기를 실험하였다. 본 시스템은 불필요한 변수나 문장 등을 추가한 코드(dummy code)에 문제점을 드러내고 있다. 따라서 이 문제를 해결하기 위해서는 소프트웨어 공학 분야에서 연구되고 있는 코드 최적화기나 중복 코드 검사기(dummy code checker)를 수행함으로써 해결할 수 있을 것이다.

본 연구의 향후 과제로는 이전에 언급한 중복 코드 검사기외에도 수행 속도에 관한 연구가 필요하다. 본 시스템을 실험한 결과 약 500라인 이상인 프로그램에 대해서는 속도가 현저하게 떨어진다는 것을 알 수 있었다. 따라서 알고리즘의 속도 향상 측면에서 연구가 이루어져야 할 것이다. 이외에도 XML 문서의 유사도 측정, 코드 생성 및 최적화를 이용한 유사도 검사, 웹 기반의 프로그램 복제 검사 시스템 활용 등이 있다. 본 연구에서는 이용된 알고리즘을 이용하여 다양한 프로그램 복제나 문서 복제에 활용할 수 있다.

참 고 문 헌

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

[2] P. Cunnignham & A. N. Mikpyan, "Using CBR Techniques to Detect Plagiarism in Programming Assignments", available at Department of Computer Science, Trinity College, Dublin. 1993.

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

[4] J. Lin & JLex Tutorial, available at ber-

keley.edu/courseware/cs164/spring98/proj/jlex/ tutorial.html.

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

[6] 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.

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

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

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

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

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

[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 SCG CSE 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] 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.

[19] Y. C. Kim, "A Program-Plagiarism Checker", PhD. Thesis. Dept. of Computing, Soongsil Univ. 2003.

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

◐ 저 자 소 개 ◑

김 영 철(Kim Young-Chul)
1990년 한남대학교 전자계산학과 졸업(학사)
1996년 숭실대학교 전자계산학과 석사
2003년 숭실대학교 컴퓨터학과 공학박사
현재 : (주)뉴스텍시스템즈 이사, 명지전문대학 겸임교수
관심분야 : 프로그래밍 언어, 컴파일러, XML, 컴퓨터 통신, 소프트웨어공학
E-mail : yckim@ss.ssu.ac.kr

황 석 찬(Seogchan Hwang)
1996년 청주대학교 전자계산학과 졸업(학사)
1998년 숭실대학교 대학원 컴퓨터학과 졸업(석사)
2003년 숭실대학교 대학원 컴퓨터학과 졸업(박사)
2004~현재 한국과학기술정보연구원 선임연구원
관심분야 : 시스템 소프트웨어, 병렬/분산처리, 미들웨어, 소프트웨어 아키텍트
E-mail : seogchan@kisti.re.kr

최 재 영(Jaeyoung Choi)
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) 초빙 연구원
1995년~현재 숭실대학교 컴퓨터학부 부교수
관심분야 : 시스템 소프트웨어, 고성능컴퓨팅(HPC), 병렬/분산처리,
E-mail : choi@ssu.ac.kr