구문트리 비교를 통한 프로그램 유형 복제 검사
(A Program-Plagiarism Checker using Abstract Syntax Tree)

김 영 철 † 김 성 근 †† 염 세 훈 ††† 최 종 명 †††† 유 재 우 †††††
(Young-Chul Kim) (Sung-Keun Kim) (Se-Hoon Yeom) (Jong-Myung Choi) (Chae-Woo Yoo)

요 약 기존의 프로그램 유형 복제 검사 시스템들은 단순한 텍스트 기반의 프로그램 복제 검사나, 속성 및 토큰 스트링을 이용하여 복제 검사를 수행한다. 이 시스템들은 들여쓰기, 여백, 설명문과 같은 프로그램의 구문과 상관없는 프로그램 스타일에 어려움을 갖고 있다. 본 연구에서는 서로 다른 두 프로그램의 구문트리를 이용하여 복제 검사를 수행하는 모델을 제시한다. 구문트리를 이용한 프로그램 유형 복제 검사는 프로그램 스타일에 취약한 기존의 복제 검사 시스템의 단점을 극복할 수 있으며, 구문분석과 의미분석을 통해 프로그램의 구조적인 검사까지 수행할 수 있다는 장점을 가지고 있다. 또한 본 시스템은 인터넷이나 사이버 교육 체제에서 대량의 C/C++ 언어의 프로그램 복제 검사를 수행하기 위하여 AST 생성, 역파서 및 유사도 검사 알고리즘을 제시하며, 프로그램 복제 유형에 대해서 평가한다.

키워드 : 유사도, 복제, 프로그램 복제 검사, 구문트리, 역파서

Abstract Earlier program plagiarism check systems are performed by using simple text, attribute or token string base on match techniques. They have difficulties in checking program styles which have nothing to do with program syntax such as *indentation, spacing and comments*. This paper introduces a plagiarism check model which compares syntax-trees for the given programs. By using syntax-trees, this system can overcome the weekness of filtering program styles and have advantage of comparing the structure of programs by syntax and semantic analysis. Our study introduces syntactic tree creation, unparsing and similarity check algorithms about C/C++ program plagiarism checking for internet cyber education and estimate plagiarism pattern.

Key words : similarity, plagiarism, program plagiarism checker, syntax tree, unparser

1. 서 론

많은 학생들을 대상으로 과제물의 복제 여부를 판단하는 일은 결코 쉬운 일이 아니다. 특히, 수강하는 학생들이 많은 경우, 강사는 학생들이 인터넷(internet)이나 전자우편(e-mail) 등을 통하여 대량으로 제출한 과제물을 정확히 비교, 평가하기가 어렵다. 또한 많은 프로그램 예제들이 존재하기 때문에 학생들이 제출한 과제가 복제되었다는 것을 판단하기가 어렵다. 프로그램 복제는 요즘 문제가 되고 있는 불법 복제에 의한 소프트웨어 사용의 근간이 될 수 있으며, 이는 교육적인 측면에서도 지양되어야 할 문제로 인식되고 있다.

프로그램 복제란 원본 프로그램을 똑같이 복사하거나, 원본 프로그램을 표 1과 같이 다양한 형태로 변환한 것을 말한다[1]. 복제된 프로그램을 찾는 연구들은 대부분 두 프로그램이 얼마나 밀접한 연관이 있는지를 검사하는 연구이다. Whale[2]은 자신의 연구논문에서 프로그램 복제를 감추기 위해 이용되는 은닉 기술들을 제시하였으며, 이 제시된 프로그램 은닉 기술들은 대부분 설명문 바꾸기, 식별자나 프로그램 스타일 바꾸기, 독립 코드 조각 변경하기 혹은 인라인(in-line) 코드를 프로시저 호출문 등으로 바꾸기 등에 해당된다. 그러나 이러한 은닉 기술에도 불구하고, 많은 프로그램들은 어느 정도의 프로그램 유사성을 피할 수가 없다. 왜냐하면, 프로

· 본 연구는 숭실대학교 교내연구비 지원으로 이루어졌음
† 비 회 원 : 숭실대학교 컴퓨터학과
yckim@amin.ssu.ac.kr
†† 비 회 원 : 가톨릭상지대학 컴퓨터정보계열 교수
skkim@amin.ssu.ac.kr
††† 비 회 원 : 동서울대학 전자계산학과 교수
shyeom@amin.ssu.ac.kr
†††† 학생회원 : 숭실대학교 컴퓨터학과
jmchoi@comp.ssu.ac.kr
††††† 종신회원 : 숭실대학교 컴퓨터학부 교수
cwyoo@comp.ssu.ac.kr
논문접수 : 2002년 12월 2일
심사완료 : 2003년 4월 1일

표 1 다양한 형태의 프로그램 변환

1. 정확히 복사하기
2. 설명문 변경하기
3. 공백(white Space)이나 형식 변환하기
4. 식별자 이름 바꾸기
5. 코드 블록 재배치하기
6. 코드 블록 내에서 문장들 재배치하기
7. 수식에서 오퍼랜드/오퍼레이터의 순서 바꾸기
8. 데이터 타입 변환하기
9. 필요없는 문장이나 변수 추가하기
10. 똑같은 구조를 갖는 제어 구조로 바꾸기

그램 소스들이 워낙 많이 산재되어 있으며, 이를 토대로 프로그램을 구현하는 사례가 많기 때문이다.

본 연구에서 제시한 프로그램 유형 복제 검사 시스템은 구문분석 과정을 통하여 구문트리를 생성하는데, 유사도를 검사하기 위하여 추상 구문트리(Abstract Syntax Tree ; AST)를 이용한다. 구문 트리를 이용하는 방법은 기존의 매트릭스나, 토큰 스트링을 이용한 시스템과 달리 다음과 같은 많은 장점을 가질 수 있는 것으로 평가되고 있다. 첫째, AST는 방대한 프로그램에서 작은 프로그램까지 유사도 검사에 적용될 수 있다. 둘째, 복제를 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, AST는 함수 각각에 대해서도 유사도를 검사할 수 있다. 여섯째, 복제를 한 당사자가 프로그램의 변수나, 함수 이름, 변수에 할당된 값 등을 바꾸었을 경우 유사도 검사를 정확히 판별해 낼 수 있다. 일곱째, while이나 if, swich문 등의 안에 복제자가 간단한 변수나, 수치를 추가했을 경우 AST에 대한 전체 적인 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.

본 연구에서 제시한 프로그램 복제 유형 검사 모델은 그림 1과 같다. 그림 1의 모델은 크게 2 단계로 나누어진다. 첫 번째 단계는 AST를 생성하는 단계로 원시 프로그램을 파싱하여 구문 검사와 동시에 AST를 생성한다. 두 번째 단계는 생성된 AST를 복제 유형 검사 알고리즘을 이용하여 유사도를 검사한다. 본 시스템에서는 C/C++ 언어 프로그램의 검색을 위해 컴파일러 보조도구인 lex와 yacc을 이용하여 구현되었다[3,4].

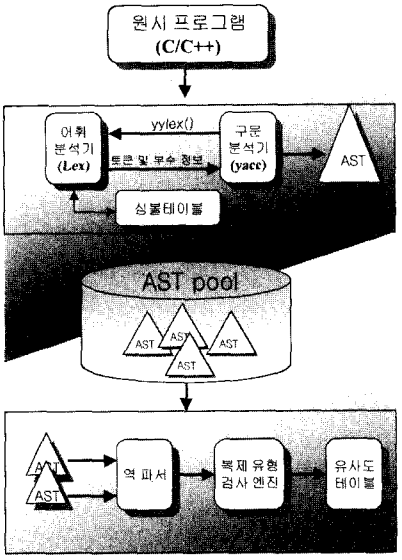

그림 1 프로그램 유형 복제 검사 모델

본 논문의 구성은 다음과 같다. 제 2장에서는 관련 연구에 대해서 기술하며, 제 3장에서는 본 논문에서 제시한 프로그램 유형 복제 검사 모델과 프로그램 복제 알고리즘에 대해서 설명한다. 제 4장에서는 실험에 대해서 기술하며, 마지막으로 제 5장에서는 결론을 기술한다.

2. 관련 연구

프로그램 복제 연구와 관련된 연구는 예전부터 진행되어 왔는데 주로 코드 중복 검사 연구에 관한 것이다. 특히, 프로그램의 중복된 코드를 찾는 방법은 소프트웨어 공학에서 많은 연구가 진행되어 왔으며, 이 분야의 연구들은 프로그램에서 중복된 코드가 보통 7-23%에 달한다는 연구 결과가 나왔다[5,6]. 그러나 이러한 연구들은 프로그램 복제 판결에 목적이 있는 것이 아니라 중복된 코드를 찾는데 목적을 가지고 있다. 중복된 코드는 프로그램의 분석, 수정 및 유지보수를 어렵게 만들기 때문에 중복된 코드를 없애는데 목적이 있다.

프로그램 복제와 관련된 연구는 지금까지 20-30년간 진행되어 왔다. 초기의 프로그램 복제 감지 방법은 두 프로그램의 단순한 텍스트 파일의 비교이다. 이 시스템들은 각각의 프로그램에 대해서 다양한 소프트웨어 특징을 계산하고, n-차원으로 매핑한다. 가장 초기에 만들어진 시스템은 두 프로그램 사이에 유사성 정도를 검사하기 위한 Halstead 매트릭스[7]를 이용한 시스템이다. Halstead 매트릭스는 4가지 벡터 즉, $H = (\mu_1, \mu_2, N_1, N_2)$로 구성된다. μ_1은 유일한 오퍼레이터 수를 말하며, μ

$_2$는 유일한 오퍼랜드 수를 지칭한다. 또한 N_1는 오퍼레이터의 출현 수에 해당되며, N_2는 오퍼랜드의 출현 수를 의미한다. Halstead가 제안한 매트릭스를 이용해 만든 최초의 복제 검색 시스템은 Ottenstein[8]이 개발한 시스템이다. 이 시스템은 Fortran 프로그램을 비교하기 위하여 만들어졌다. 이 시스템에서 두 프로그램이 Halstead 매트릭스인 μ_1, μ_2, N_1, N_2가 같으면, 유사한 프로그램이라 판결하고 세밀한 조사를 한다.

Donaldson[9], Berghel[10] 등은 Halstead의 매트릭스를 확장하여 이용하였다. 특히 Donaldson이 제시한 복제 방법은 Halstead 매트릭스 외에, 반복문의 수, 프로시저 문의 수와 같은 프로그램 구조를 혼합하여 이용했다는 점도 주목할 만하다. 이러한 시스템들을 모두 총칭하여 '속성 카운팅 매트릭스 시스템'이라 한다.

가장 최근에 개발된 시스템들은 토큰 스트링 등 프로그램 구조를 이용하여 평가하는 시스템이다. 이전 시스템과는 달리 토큰 스트링을 이용함으로써 프로그램 스타일이나 설명문, 들여쓰기 등의 프로그램 구문과 상관없는 요소에 민감하지 않다는 특징을 가지고 있다. 프로그램 구조를 이용한 대표적인 시스템은 YAP3, MOSS, Jplag이다.

M.J. Wise에 의해 개발된 YAP3[11]는 구조적 매트릭스 방법을 이용한 복제 검사 시스템으로 다중 파일을 지원한다. YAP3는 이전 버전인 YAP1과 YAP2의 새로운 버전으로 크게 2가지 단계로 수행된다. 첫 번째 단계는 어휘 분석기가 소스 프로그램을 토큰 스트링으로 바꾸는 단계이다. 이 과정에서 수행되는 작업은 설명문과 스트링 상수를 제거하기, 대문자를 소문자로 바꾸기, 소스 프로그램을 똑같은 혹은 유사한 동의어로 매핑하기, 함수 호출 순서를 재정렬하기 등을 수행한다. 또한 제어구조, 블록문, 서브프로그램 및 라이브러리 함수 등과 같은 프로그램 구조를 가리키는 토큰을 유지한다는 특징을 가지고 있다. 두 번째 단계에서는 첫 번째 단계에서 변환된 토큰 스트링을 비교 검사하는 단계로 GST(Greedy-String-Tiling) 알고리즘을 이용한다.

Alex Aiken이 개발한 MOSS(Measure Of Software Similarity)[12]는 프로그램 복제 검사 시스템 중 가장 많은 언어를 지원한다는 특징을 가지고 있다. MOSS는 총 8가지 언어(Java, C, C++, Pascal, Ada, ML, Lisp, Scheme)를 지원하며, Unix에서 구현되었다. 또한 Web 상에서 학생들의 보고서를 받을 수도 있으며, 파싱되지 않은 파일을 비교 검사한다. MOSS 시스템의 매칭 알고리즘은 토큰이나 라인 매칭을 이용하여 검사되며 다중 파일을 지원한다.

Guido Malpohl에 의해 개발된 JPlag[13]는 4가지 언어(Java, C++, C, Scheme)를 지원하며, 토큰 패턴 매칭 알고리즘을 이용한다는 특징을 가지고 있다. 또한 자기 유효성 검사(Self Validation)도 수행되며, 다양한 형태의 플랫폼을 지원한다. JPlag 역시 YAP3 시스템과 똑같은 토큰 패턴 매칭 알고리즘을 사용한다.

이 외에도 프로그램 복제 검사에 관련된 연구는 David Gitchell이 개발한 Sim[14]과 Xin Chen 등이 개발한 SID[15]가 있다. Sim은 C++와 Tcl/Tk로 구현되었으며, 서로 다른 두개의 C 프로그램을 비교한다. 또한 SID(Software Integrity Diagnosis system)는 Kolmogorov 복잡도를 기반으로 하고 있으며, 특히 두 프로그램 사이의 "공유 정보(Shared information)"의 양을 측정하여 유사도를 측정한다.

대부분 이들 시스템이 이용한 방법은 어휘분석에서 사용된 토큰 스트링이다. 이러한 접근은 같거나 유사한 코드 조각을 찾기 위하여 프로그램을 토큰 스트링으로 바꾸어 비교한다. 이전 방법에 비해서 이러한 접근은 복제자가 쉽게 바꿀 수 있는 들여쓰기, 줄바꾸기, 설명문과 같은 쉽게 변경할 수 있는 정보를 무시하고 복제를 검사할 수 있다는 장점을 가지고 있다.

프로그램 유사도 복제 검사는 국내에서도 연구가 진행되어 왔다. 특히 [16, 17]의 연구는 본 연구의 이전 단계로 수행되었으며, 구문트리(AST)를 이용한 프로그램 복제 검사방법을 처음으로 제시하였다. 또한 [16]에서도 AST를 이용한 유사도 측정 방법을 기술했다. 특히, [18]의 연구에서는 C 언어뿐만 아니라 Java, Scheme, nML로도 프로그램 유사도를 측정하였으며, 학생들이 제출한 과제물에 대해서 유사도 그룹짓기를 수행하여 비교시간을 단축한다는 특징을 가지고 있다.

가장 최근에 연구된 [19]는 프로그램의 소스 코드로부터 추출한 키워드들의 서열을 이용하여 두 프로그램간의 유사성 및 부분 표절 검출에 관한 방법을 제시하였다. 특히, 이 연구의 특징은 생명체에서 DNA염기서열의 배치순서가 비슷할수록 동일 종에 가깝다는 점에 착안하여, 프로그램 복제 검사에 이용했다는 특징을 가지고 있다. 즉, 프로그램 소스 코드의 배치순서가 유사할수록 프로그램 표절 가능성이 높다는 점을 인식하여 연구되었다.

3. 복제 검사 알고리즘

3.1 AST와 역파서

AST를 생성하는 과정은 일반적인 컴파일러 구현의 전단부와 유사하다. 구문분석 과정은 어휘분석 과정에서

수행된 일련의 토큰들을 입력으로 하여 원시 프로그램을 C/C++ 언어의 문법 구조에 적합한가를 검사하고, 문법적 구조를 나타내는 AST를 만든다. AST는 파스 트리와 같이 모든 심볼에 대하여 노드를 만들지 않고 최소한의 필요한 노드를 만들게 되어 이후의 처리 단계에서 시간(propagation time)을 절약할 수 있다. 본 시스템에서 적용된 C 언어의 AST 정의는 [부록 A]에 첨부하였다.

파싱 과정에서 생성된 AST는 AST_pool이라는 저장소에 저장된다. AST_pool이란 비교할 프로그램의 AST를 보관하는 장소로 이용된다. 본 시스템에서는 비교하고자하는 프로그램의 AST를 AST_pool에 저장한 후, 프로그램 복제 검사 엔진에 의해서 유사도를 검사한다. 이 AST는 역 파서에 의해 노드 스트링 형태로 변환되어 유사도를 검사한다. 본 시스템의 AST_pool은 그림 2와 같은 형태로 구성되었다.

역파서(unparser)는 AST 형태로 저장된 자료구조를 노드 스트링 형태로 변환해준다. 즉, 서로 다른 두개의 AST를 비교하기 위해서 선형구조의 노드 스트링으로 변환하는 역할을 한다. 역파서는 AST_pool에 저장된 AST를 입력받아 노드 스트링으로 바꾼 후, 복제 유형 검사 엔진의 입력 스트링으로 전달된다. 다음 그림 3은 역파서가 AST를 AST 노드 스트링으로 바꿔주는 과정을 설명한 그림이다. 역파서는 단말노드를 생략하고, 비 단말노드만을 가지고 AST 노드 스트링으로 바꿔준다. 즉, 역파서가 비 단말노드만을 취급함으로써, 변수나 함수 이름을 바꾸어도 노드 스트링이 변하지 않기 때문에 쉽게 복제를 발견할 수 있다.

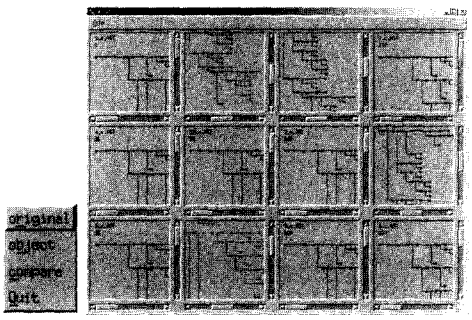

그림 2 AST_pool

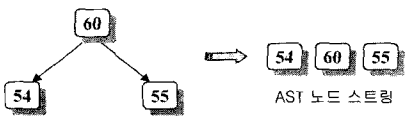

그림 3 AST와 노드 스트링

3.2 유사도 검사 알고리즘

파싱 과정에서 생성된 AST는 역파서에 의해서 선형적인 연속된 노드들의 순으로 배치된다. 따라서 두 프로그램은 복제 검사를 하기 위해 최종적으로 노드들의 스트링으로 이루어져 비교된다. 본 시스템에서는 두 노드 스트링을 비교하기 위하여 다음과 같은 복제 검사 알고리즘을 사용한다. 이 알고리즘은 두 노드 스트링을 입력으로 받아 유사도($0 < sim(P1, P2) \leq 1$)를 출력한다. 유사도가 0일 경우 두 프로그램이 전혀 다른 경우를 뜻하며, 1인 경우는 두 프로그램이 동일한 경우를 말한다. 본 시스템의 프로그램 복제 검사 알고리즘은 다음과 같이 크게 3부분으로 구성되었다. [알고리즘 1]에서는 입력된 두 개의 노드스트링(A, B)에서 첫 번째로 일치하는 스트링을 검색한다. 이때 가장 길게 매칭되는 스트링(MMS : Maximum Match String)을 찾는 과정을 수행한다. 따라서 [알고리즘 1]의 주 목적은 두 노드 스트링 중 최초로 일치되는 가장 긴 서브스트링을 찾는 것이다.

[알고리즘 1]. 두 노드 스트링(A, B)에서 매칭 스트링 찾기

```
MatchString(NodeString A, NodeString B) {
  for each SubString Aa in NodeString(A)
    for each  SubString Bb in NodeString(B)
          MatchSize = 0 ;
          while ((Aa+MatchSize == Bb+MatchSize && Aa+MatchSize)
                && Bb+MatchSize) MatchSize++;
          if (MatchSize == maxmatch)
                Set(MatchString) = Set(MatchString) +
                match(a, b, MatchSize);
          else if (MatchSize > maxmatch) {
                Set(MatchString) = match(a, b, MatchSize);
                maxmatch = MatchSize; }
    end for
  end for
}
```

[알고리즘 1]은 노드 스트링 A에서 각각의 서브스트링 Aa에 대해서 동일한 Bb를 찾는다. 동일한 서브 노드 스트링을 찾으면 가능한 최대 크기의 토큰 스트링을 찾기 위해서 계속 비교된다. 두 토큰(Aa+MatchSize와 Bb+MatchSize)이 다르다면, 찾은 서브스트링을 추가할 것인지 체크한다. 만일 MatchSize가 maxmatch 보다 크다면, Set(MatchString)는 다시 초기화되고, 새로운 최대길이 MatchSize 값이 maxmatch에 저장된다.

또한 알고리즘의 "Set(MatchString) = Set(MatchString) + match(a, b, MatchSize);" 부분은 현재 스트링이 일치되고, Set(MatchString)에 없다면 추가시키는

역할을 한다. match(a, b, MatchSize) 함수는 스트링 A와 B의 동일한 서브스트링 사이의 연관 관계를 말하는 것으로, 동일한 서브스트링이 A_a와 B_b 시점에서 같고 그 길이는 MatchSize와 같음을 의미한다.

[알고리즘 2]. 일치된 스트링을 Set(matchString)에 추가하고, 노드 스트링에서 제거

```
SetOfMatchString(NodeString A, NodeStirng B) {
  for each match(a, b, maxmatch) ∈ Set(MatchString)
    Set(matchString) = Set(matchString) + match(a, b,
    maxmatch);
    for MatchSize = 0 to (maxmatch - 1)
        Delete(Aa+MatchSize);
        Delete(Bb+MatchSize);
    end for
  end for
}
```

유사도 검사 알고리즘의 [알고리즘 2]는 집합 matches가 검사되고, 일치된 노드 스트링은 Set(matchString)에 추가된 후, 두 노드 스트링 A, B에서 제거된다. [알고리즘 2]에서는 두 노드 스트링에서 일치된 서브스트링을 Set(matchString)에 저장한 후, 노드 스트링에서 제외시킨다. 왜냐하면 일치된 서브스트링은 후에, [알고리즘 1]을 반복하므로 다시 반복 검사되지 않게 하기 위한 것이다.

[알고리즘 3]. 유사도 검사

```
Procedure Sim(NodeString A, NodeString B, int MinLength) {
        Set(matchString) = {};
        matches = {};
        LengthA = Length(A);
        LengthB = Length(B);
        do {
            matches = {};
            MatchString(A, B);   /* 알고리즘1 호출 */
            SetOfMatchString(A, B); /* 알고리즘2 호출*/
        } while (maxmatch > MinLength);

        for each matchString in Set(matchString)
            MatchLength = MatchLength + Length(matchString);
        end for

        return ( sim(A, B) = 2*MatchLength / LengthA + LengthB );
}
```

[알고리즘 3]은 길이 2보다 큰 match가 발견될 때까지 알고리즘 1, 2를 반복하며, 알고리즘이 끝난 후에는 두 노드 스트링의 유사도 값을 리턴한다. 입력 값으로 전달된 MinLength 값은 최소 매칭되는 크기를 말한다. 즉, 일치되는 서브스트링 중, MinLength 이하 값은 비교에서 제외된다.

프로그램 유형 복제 검사 알고리즘은 두 프로그램의 유사도 값으로 $0 < Sim(A, B) \leq 1$ 의 범위를 갖는다. 만일 서로 다른 두 프로그램이 전혀 다르다(즉, Set(matchString)이 공집합인 경우)면, 유사도 검사 값 Sim(A, B)는 0이다. 반면에, 서로 다른 두 프로그램이 정확히 일치한다면 Sim(A, B) 값은 1 값을 갖게 된다.

본 알고리즘 설명하기 위하여 다음과 같은 노드 스트링 P1과 P2가 있다고 가정하자(단, 다음의 숫자는 노드를 나타내는 숫자이다).

$P1_{nodestring}$: 23 34 25 54 44 45 49 81 83 84 22 55 44 33 90 68

$P2_{nodestring}$: 34 25 54 46 47 81 83 84 22 55 44 33 90 93 92 95 34 35

[알고리즘 1]에서는 두 노드 스트링의 일치된 부분을 검사한다. 즉, 예에서 { {34, 25, 54}, {81, 83, 84, 22, 55, 44, 33, 90} }을 찾아낸다. [알고리즘 2]에서는 [알고리즘 1]에서 찾은 부분 스트링을 Set(matchString) 집합에 추가한 후, P1, P2 노드 스트링에서 이를 제거한다. 왜냐하면, 찾은 스트링은 다음에 일치하는 스트링과 다시 비교하는 것을 피하기 위함이다. [알고리즘 3]에서는 앞선 알고리즘을 이용하여 유사도를 측정한다. 즉, 노드 스트링 P1, P2는 다음과 같은 유사도를 갖는다.

$$sim(P1, P2) = \frac{2*MatchLength}{Length(P1) + Length(P2)}$$

$$= \frac{2*(3+8)}{16+18} = 0.647 \text{ 이다.}$$

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 3]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 3]의 반복문은 1번만 수행된다. 따라서 최상의 경우 시간 복잡도는 $O(n^2)$이 된다.

3.3 프로그램 유형 분석

본 알고리즘을 이용하면, 다음과 같은 프로그램 복제 유형의 유사도를 측정할 수 있다. 다음 예에서 P1, P2는 각각 원본 프로그램의 노드스트링과 복제된 노드 스트링을 나타낸다.

1) 변수나 함수 이름을 단순히 변환한 경우의 유사도

두 프로그램의 노드 스트링이 각각 $P1_{nodestring}$: { N_1, N_2, N_3, ..., N_i, N_{i+1}, ..., N_n}, $P2_{nodestring}$: { N_1, N_2, N_3, ..., N_i, N_{i-1}, ..., N_n}일 경우 두 프로그램의 유사도는 다음과 같다.

$$Sim(P1, P2) \fallingdotseq \frac{2 * Length(N1, N2, \ldots, Nn)}{Length(P1) + Length(P2)} \fallingdotseq 1$$

즉, 이 유사도의 의미는 AST를 이용하면, 프로그램의 구조적 검사를 할 수 있다는 것을 보여준다. 유사도 1 값은 두 프로그램이 구조적으로 완전히 일치하므로 완전히 복제되었다는 것을 의미한다.

2) 문장을 삽입하였을 경우의 유사도

두 프로그램의 노드 스트링이 각각 $P1_{nodestring}$: { N_1, N_2, N_3, ..., N_i, N_{i+1}, ..., N_n}, $P2_{nodestring}$: { N_1, N_2, N_3, ..., N_j, N_{j+1}, ..., N_{j+m}, N_i, N_{i+1}, ..., N_n }일 경우 두 프로그램의 유사도는 다음과 같다.

$$Sim(P1, P2) \fallingdotseq \frac{2 * Length(N1, N2, \ldots, Ni-1) + Length(N_i, Ni+1, \ldots Nn)}{Length(P1) + Length(P2)}$$

이 유사도는 원시 프로그램에 문장을 추가로 삽입하였을 경우에 해당되는 유사도 값을 나타낸다.

3) for, while, if 문 등을 변환하였을 경우의 유사도

두 프로그램의 노드 스트링이 각각 $P1_{nodestring}$: { N_1, N_2, N_3, ..., N_i, N_{i+1}, ..., N_{i+m}, N_{i+m+1}, ..., N_n }, $P2_{nodestring}$: { N_1, N_2, N_3, ..., N_j, N_{j+1}, ..., N_{i+m}, N_{i+m+1}, ..., N_n }일 경우 두 프로그램 사이의 유사도는 다음과 같다.

$$Sim(P1, P2) \fallingdotseq \frac{2 * \{Length(N1, N2, \ldots, Ni-1) + Length(Ni+m, Ni+m+1, \ldots Nn)\}}{Length(P1) + Length(P2)}$$

4) 문장들의 순서, 혹은 함수의 위치를 바꾸었을 경우의 유사도

두 프로그램의 노드 스트링이 각각 $P1_{nodestring}$: { N_1, N_2, N_3, ..., N_i, N_{i+1}, ..., N_{i+j}, N_{i+j+1}, N_{i+j+2}, ..., N_{i+j+m}, $N_{i+j+m+1}$, ..., N_n }, $P2_{nodestring}$: { N_1, N_2, N_3, ..., N_{i+j+1}, N_{i+j+2}, ..., N_{i+j+m}, N_i, N_{i+1}, ..., N_{i+j}, N_{i+j+m}, ..., N_n}일 경우의 프로그램 유사도는 다음과 같다.

$$Sim(P1, P2) \fallingdotseq 2 * \left\{ \frac{\begin{array}{c} Length(N1, N2, \ldots, Ni-1) + \\ Length(Ni, Ni+1, \ldots, Ni+j) + \\ Length(Ni+j+1, Ni+j+2, \ldots, Ni+j+m) + \\ Length(Ni+j+m+1, \ldots, Nn) \end{array}}{Length(P1) + Length(P2)} \right\} \fallingdotseq 1$$

즉, 이 수식의 의미는 두 프로그램이 구조적으로 똑같고, 함수 위치, 문장 순서가 뒤바뀌었을 경우에도 두 프로그램은 똑같음을 판명할 수 있다.

예를 들어 다음과 같은 두 프로그램을 살펴보자. 표 2의 프로그램은 입력으로 문자열을 읽어 들인 후, 문자, 단어, 라인 수를 출력하는 프로그램(P1)이다. 표 3의 프로그램(P2)은 원래의 프로그램(P1)에 "d = 0;"과 같은 의미 없는 문장을 추가하고, 변수 이름을 바꾸었으며, 문장의 순서를 바꾼 프로그램이다.

그림 4는 프로그램 P1, P2의 전체적인 AST 형태이다. 그림 4에서 첫 번째(왼쪽) AST가 프로그램 P1에

표 2 원본 프로그램 P1

```
void main() {
        int c, nl, nw, nc, state;
        state = 0; nl = nw = nc = 0;
        while((c = getchar()) != -1) {
                ++nc;
                if(c == '\n') ++nl;
                if(c == ' ' || c == '\n' || c == '\t')  state = 0;
                else    if(state == 0) { state = 1;  ++nw; }
        }
        printf("%d %d %d\n", nl, nw, nc);
}
```

표 3 표절된 프로그램 P2

```
void main() { int ch, newline, nw, nc, state, d; state = 0;
        newline = nw = nc = 0;
        while((ch = getchar()) != -1) {
          ++nc;
          if(ch == ' ' || ch == '\n' || ch == '\t')          state = 0;
          else      if(state == 0) { state = 1;  ++nw; }
          if(ch == '\n') ++newline;  /* 문장 순서 바꿈 */ }
        d = 0; printf("%d %d %d\n", newline, nw, nc); /*스타일 변경*/
}
```

관한 트리이며, 두 번째(오른쪽) AST에 프로그램 P2에 해당한다. 그림과 같이 왼쪽 AST에 비해 오른쪽 AST의 중간 부분에 표시되어 있는 부분이 추가되었음을 볼 수 있다. 이것은 "d=0;" 문장에 관한 노드가 추가된 것이다. 그 외에 변수 이름이나 문장의 순서를 바꾸었을 경우에도 나타난다.

그림 5는 프로그램 P1, P2의 부분적인 AST 형태이다. 즉, 그림 4에서 변경된 부분을 확대한 그림이다. 그림의 오른쪽 영역은 새롭게 추가된 문장("d=0;")이 잘 나타나 있다. 또한 그림 4와 같이 그림 5에서는 노드 스트링의 비교 결과와 유사도(91.6%) 값을 백분율로 보여주고 있다. 이 유사도 값은 이전에 언급한 프로그램 복

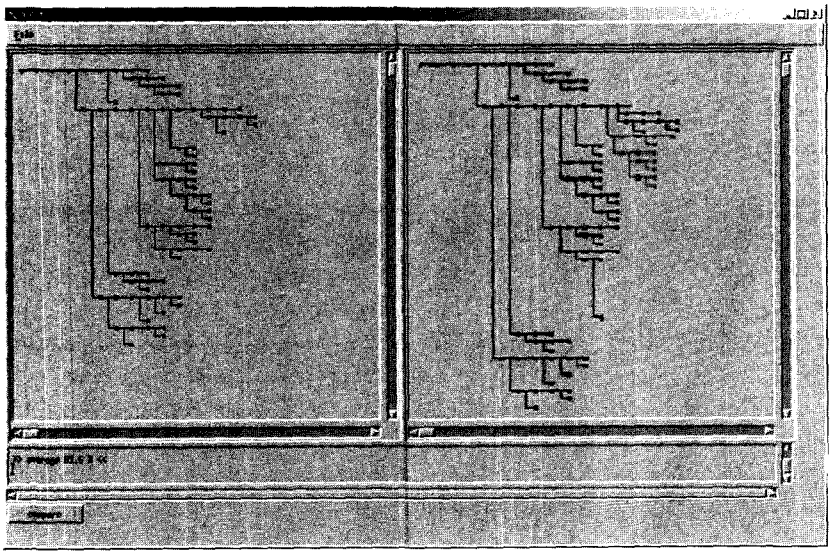

그림 4 프로그램 P1, P2의 AST(전체)

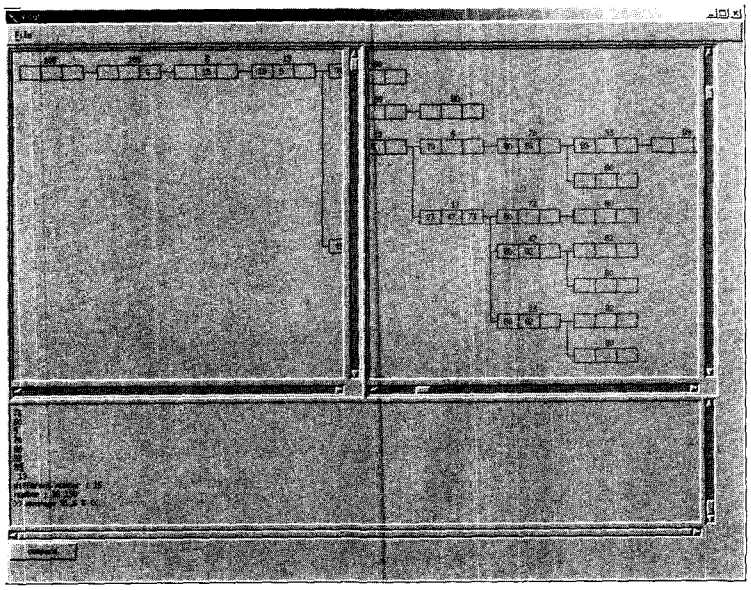

그림 5 프로그램 P1, P2의 AST(부분)

제 알고리즘을 적용한 결과로 나타났다.

4. 실 험

본 시스템은 Sun enterprise의 Solaris 7.0 환경에서 Motif로 구현되었으며, 부록 A와 같은 AST를 생성하기 위하여 lex & yacc[4]을 이용하였다. 본 시스템은 학생들이 제출한 과제물을 이용하여 유사도 검사를 수행하였으며, 관련된 웹 주소는 http://203.253.23.89/courseware/index.html이다. 본 논문의 실행 코드는 Unix의 Motif로 작성되었으며, http://203.253.23.89/courseware/Checker.zip으로 다운받을 수 있다. 본 프로그램을 실행하기 위해서는 UNIX Motif 환경에서 수행시켜야 한다.

본 시스템을 이용한 유사도 검사는 크게 두 부분으로 나눠서 측정되었다. 하나는 학생들 간의 프로그램 복제 정도를 검사하기 위한 목적으로 평가되었으며, 다른 하나는 강사가 프로그램 골격을 제공하여 최적의 프로그램을 찾기 위한 유사도 검사를 수행하였다. 즉, 전자는 실제로 제출된 프로그램들 중 복제 정도를 측정하여 표절된 프로그램을 찾는데 목적을 두고 있으나, 후자는 강사가 프로그램 골격을 직접 제공함으로써, 학생들은 강사가 제시한 프로그램과 얼마나 유사한지를 검사하게 된다. 따라서 후자는 유사도가 크면 클수록 강사가 원하는 프로그램이라 단정할 수가 있다. 학생들의 과제물을 대상으로 평가한 결과는 다음 그림 6과 같다. 그림 6의 가로축은 학생들의 번호(총 45명)이며, 세로축은 유사도를 백분율로 나타낸 것이다.

그림 6에서 보여주는 것처럼 프로그램 골격을 제공한 경우가 전체적으로 유사성이 높다. 이것은 강사가 프로그램 전체 골격을 제공하고, 학생들은 이 골격을 이용하여 프로그램을 하였기 때문이다. 따라서 골격을 제공한

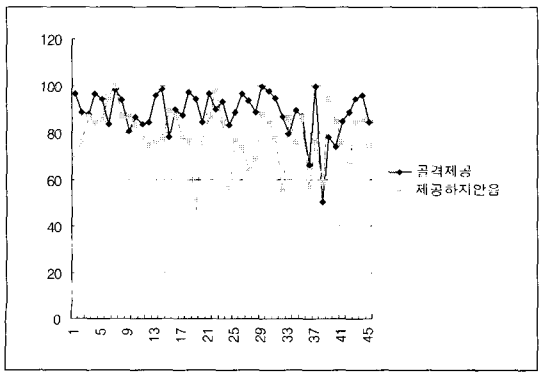

그림 6 유사도 측정 분포도

경우는 유사성 값이 1일 때, 가장 잘 설계된 프로그램임을 알 수 있다. 또한 골격을 제공하지 않았을 경우에는 n 명의 학생 과제물에 대해서 $\frac{n*(n-1)}{2}$ 번을 비교하여 가장 높은 유사도를 채택한 자료이다. 본 시스템의 신뢰성을 알아보기 위하여 강한 유사도(유사도 80% 100%)를 갖는 학생들을 대상으로 오프라인으로 검사해 보았다. 실제 유사도 100%에 해당하는 학생들의 두 프로그램을 수작업으로 비교해본 결과 똑 같은 프로그램으로 발견되었다. 더군구나 어떤 두 학생의 프로그램은 변수나 함수 이름 스타일(들여쓰기 및 줄바꿈 등) 부분도 똑같아 완전한 복제임이 밝혀졌으며, 다른 2명은 변수나 들여쓰기와 같은 스타일만을 변경하였다. 또한 유사도 80% - 90% 범위를 갖는 프로그램들은 Whale[2]에서 제시된 은닉 기술들을 많이 이용하였다. 또한 본 시스템을 이용하여 많은 프로그램의 유사도를 검사한 결과 printf 추가, do while 추가, for 추가, while 추가, switch 추가, if 추가, if else 추가 등도 모두 위와 같은 과정을 거치게 되며 85% 이상의 높은 유사도를 보였다. 여기서 statement에 대한 가중치를 적게 하면 90% 이상의 유사도를 나타내게 된다. 물론 이러한 결과는 완전히 정확할 수는 없다. 프로그램을 평가하는 일은 강사가 직접 수작업으로 수행한다고 해도 주관적일 수 있다. 따라서 본 시스템에서 평가한 결과는 어느 정도의 객관성을 근거로 구현되었으며, 보다 정확히 판단하기 위해서는 향후에 수정 보완해야할 것이다.

본 연구는 [16, 17, 18]의 연구와 같이 구문트리를 이용하여 유사도 검사를 수행한다. 특히 Clone-Checker[18]는 프로그램 유사도 검사를 수행하기 전에 구문트리에 해당되는 단말 노드를 하나의 문법적 범주를 가진 트리로 바꾸어 수행된다. 또한 부분 트리를 해시 값이나, 그룹짓기를 수행함으로써 비교할 대상을 많이 줄일 수 있다는 장점을 가지고 있다. 그러나 [18]과는 달리 본 연구에서는 구문트리의 구조에 따른 복제 검사 알고리즘을 제시하였으며, 프로그램 복제에 따른 유형을 분석하였다. 또한 [16, 17]에서 수행된 연구를 바탕으로 구체적인 검사방법과 유사도를 제시하였으며, 다음 표 4와 같은 특징을 가진다. 표 4는 기존의 다른 복제 검사 시스템과 간략하게 비교한 결과이다. 본 시스템은 AST를 이용하여 수행됨으로써, 기존의 다른 복제 검사 시스템과 달리 파싱 과정에서 생성된 AST를 이용함으로써 프로그램의 구조적 분석과 구문분석 과정도 거친다는 장점이 있다.

표 4 프로그램 복제 검사 시스템의 비교

시스템	이용알고리즘	다중화일지원	지원언어	특징
YAP3[11]	structure metric	O	3	함수 호출 순서 재정렬 유사도
MOSS[12]	structure metric	O	8	최다 언어 지원 라인 단위 유사도
Jplag[13]	token pattern matching	O	4	자기 유효성 검사 수행 다양한 형태의 플랫폼 지원
본 시스템	AST	O(AST_pool)	2	구문분석

5. 결 론

본 논문에서는 AST를 이용하여 서로 다른 두 프로그램의 유사도를 검사하는 프로그램 유형 복제 검사 시스템을 제시하였다. AST를 이용한 프로그램 유형 복제 검사는 프로그램의 구조적인 방법으로 유사도를 검사할 수 있다. 따라서 프로그램 스타일(들여쓰기, 여백, 설명문 등)에 많은 취약성을 보여준 기존의 프로그램 복제 검사 방법에 비해 많은 장점을 가지고 있다는 것을 보여주었다. 또한 본 시스템은 AST를 비교, 평가하기 위하여 역파싱 기법과 AST_pool 및 유사도 검사 알고리즘을 제시하였으며, 실험에서는 프로그램 복제 유형에 대해서 다루었다.

본 연구에서 제시한 프로그램 유사도 복제 알고리즘은 최악의 경우 $O(n^3)$와 최상의 경우 $O(n^2)$의 시간 복잡도를 갖는다. 그러나 이 시간 복잡도 역시 [18]에서 제시한 해시 기법과 Karp-Rabin 알고리즘[20]을 사용하면 시간을 줄일 수 있을 것으로 기대된다.

복제 검사 시스템은 실제로 많은 연구가 있었다. 그 중 현재 영국의 nottingham 대학교에서는 실제로 학생들에게 "ceilidh" 시스템[21]을 개발하여 복제 검사에 활용하고 있다. 이처럼 복제 검사 시스템은 후에 인터넷을 통해 C/C++ 언어 프로그래밍 과정의 원격강의를 지원하는 코스웨어와 학생들의 과제물을 효율적으로 평가할 수 있는 평가도구로 활용될 수 있다. 본 시스템을 활용한다면 가상대학(Cyber University), 주문형 강의(Lecture on Demand), 전자도서관 등에서 다양한 형태로 적용될 수 있으며, 컴퓨터 교육 분야에 많은 영향을 줄 것으로 기대 된다.

본 연구의 향후 과제로는 AST에서 검사된 중복된 부분을 활용하면, 소프트웨어 공학측면에서 연구되고 있는 중복코드 검사에 유용하게 이용될 수 있다.

참 고 문 헌

[1] E. L. Jones, "Metrics Based Plagiarism Monitoring," *In 6th Annual CCSC Nortesastern Conference*, Middlebury, Vermont, April 20-21, 2001.

[2] G. Whale, "Identification of Program Similarity in Large Populations," *The Computer Journal* 33(2), pp. 140-146, 1990.

[3] A.V. Aho, R. Sethi and J. D. Ullman. Compilers : *Principles, Techniques, and Tools*. Addison Wesley. March 1988.

[4] J. R. Levine, T. Mason & D. Brown, *lex & yacc*, O'Reilly & Associates, Inc. 1995.

[5] B. Baker. "On Finding duplication and near-duplication in large software systems," *In Proc. IEEE Working Conf. on Reverse Eng.*, pp. 86-95, July 1995.

[6] K. Kontogiannis, R. Demori, E. Merlo, M. Bernstein. "Pattern matching for clone and concept detection," *Automated Software Eng.*, 3(1-2): 77-108, 1996.

[7] Halstead, M. Howard, *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):30-41, 1976.

[9] J. L. Donaldson, A. M. Lancaster, and Paul H. "Sposato. A plagiarism detection system," *ASM SIGSCE Bulletin(Proc. of 12th SIGSCE Technical Symp.)*, 13(1):21-25, February 1981.

[10] H.L. Berghel and D. L. Sallach, "Measurements of program similarity in identical task environments," *ACM SIGPLAN Notices*, 19(8):65-76, August 1984.

[11] 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):268-271, March 1992.

[12] A. Aiken, "MOSS(Measure Of Software Similarity) Plagiarism detection system," http://www.

cs.cerkeley.edu/~moss/and personal communication, 1998. University of Berkeley, CA. April 2000.
[13] L. Prechelt, G. Malpohl, M. Philppsen. "JPlag :Finding plagiarism among a set of programs," http://wwwipd.ira.uka.de/EIR/D 76128 Karlsruhe, Germany. Technical Report 2000 1, march 28, 2000.
[14] D. Gitchell and N. Tran. "Sim: A Utility For Detecting Similarity in Computer Programs," In Proceeding of 30th SCGCSE Technical Sympo sium, New Orleans, USA. pp. 266 270, 1998.
[15] X. Chen, M. Li, B. Mckinnon and A. Seker. "A Theory of Uncheatable Program Plagiarism Dete ction and Its Practical Implementation," University of California, SantaBarbara, unpublished document. May 5, 2002.
[16] 정재은, 김영철, 김상헌, 유재우. "AST를 이용한 프로그램 유사도 평가 시스템 설계", *정보과학회 가을 학술대회*, 26(2):430 432, October, 1999.
[17] 정재은, 김영철, 유재우. "프로그램 복제 검사 시스템", *정보과학회 봄 학술대회*, 27(1):59 61, April, 2000.
[18] 장성순, 서선애, 이광근. "프로그램 유사성 검사기", *정보과학회 가을 학술대회*, 28(2):334 336, October, 2001.
[19] 황미녕, 강은미, 조환규. "유전체 서열의 정렬 기법을 이용한 소스 코드 표절 검사", 제21회 정보과학논문경진대회 입상작, 2002.
[20] M. J. Wise. "String similarity via greedy string tiling and running Karp Rabin matching," Dept. of CS, University of Sydney, Australia. ftp://ftp.cs.su.oz.au/michaelw/doc/RKR_GST.ps, December 1993.
[21] http://www.cs.nott.ac.uk/Department/Staff/ef/ceilidh/papers/Oracle.html, E. Foxley, "The Oracle program," LTR Report, Computer Science Dept, Nottingham University 1992.

부록 A. C언어의 AST 정의

(아래에서 *aId, *aId_list, *aType 등은 심볼테이블의 각 identifier에 대한 포인터를 지칭한다.)

```
program
        NProgram            (translation_unit)
translation_unit
        NExternalDeclList   (translation_unit,translation_unit)
        NFunction           (*aId, *aId_list, statement)
        NDeclaration        (*aId_list)
initializer
        NInitializerList    (initializer,initializer)
        NInitializer        (expression)
statement_list
        NStmtListNil        ()
        NStmtList           (statement_list,statement)
statement
        NLabeledStmt        (ID,statement)
        NCaseLabeledStmt    (expression,statement)
        NDefaultLabeledStmt (statement)
        NEmptyStmt          ()
        NExpStmt            (expression)
        NCompoundStmt       (*aId_list,statement_list)
        NIfThenStmt         (expression,statement)
        NIfThenElseStmt     (expression,statement,statement)
        NSwitchStmt         (expression,statement)
        NWhileStmt          (expression,statement)
        NDoWhileStmt        (statement,expression)
        NForStmt            (for_expression,statement)
        NGotoStmt           (ID)
        NContinueStmt       ()
        NBreakStmt          ()
        NReturnStmt         (expr_opt)
for_expression
        NForExp             (expr_opt,expr_opt,expr_opt)
expr_opt
        NExpOptNil          ()
        NExpOpt             (expression)
expression
        NCommaExp           (expression, expression)
        NAssignExp          (expression, expression)
        NStarAssignExp      (expression, expression)
        NPlusAssignExp      (expression, expression)
        NSlashAssignExp     (expression, expression)
        NMinusAssignExp     (expression, expression)
        NPrecentAssignExp(expression, expression)
        NShiftLeftAssignExp(expression, expression)
        NShiftRightAssignExp(expression, expression)
        NBitwiseAndAssignExp(expression, expression)
        NBitwiseXorAssignExp(expression, expression)
        NBitwiseOrAssignExp(expression, expression)
        NConditionalExp     (expression,expression,expression)
        NOrExp              (expression,expression)
        NAndExp             (expression,expression)
        NBitwiseOrExp       (expression,expression)
        NBitwiseXorExp      (expression,expression)
        NBitwiseAndExp      (expression,expression)
        NEqualTestExp       (expression,expression)
        NNotEqualTestExp    (expression,expression)
        NLessThanTestExp(expression,expression)
        NGreaterThanTestExp(expression,expression)
        NLessThanOrEqualTestExp(expression,expression)
        NGreaterThanOrEqualTestExp(expression,expression)
        NShiftLeftExp       (expression,expression)
        NShiftRightExp      (expression,expression)
```

```
        NAddExp          (expression,expression)
        NSubtractExp     (expression,expression)
        NMultiplyExp     (expression,expression)
        NDivideExp       (expression,expression)
        NModExp          (expression,expression)
        NTypeConversionExp  (*aType,expression)
        NPreIncrementExp (expression)
        NPreDecrementExp (expression)
        NAddressExp      (expression)
        NIndirectExp     (expression)
        NPlusExp         (expression)
        NMinusExp        (expression)
        NComplementExp   (expression)
        NNotExp          (expression)
        NSizeofExp       (expression)
        NSizeofTypeExp   (*aType)
        NIdentExp        (*aId)
        NIntegerConstantExp  (int_value)
        NCharacterConstantExp  (string)
        NFloatingConstantExp  (string)
        NEnumerationConstantExp  (int_value)
        NStringConstantExp  (string)
        NArrayExp        (expression,expression)
        NFunctionCallExp  (expression,argument_list_opt)
        NStructFieldExp  (expression,ID)
        NPointerFieldExp  (expression,ID)
        NPostIncrementExp  (expression)
        NPostDecrementExp  (expression)
argument_list_opt
        NArgumentExpListOptNil()
        NArgumentExpListOpt  (argument_exp_list)
argument_exp_list
        NArgumentExpList (argument_exp_list,argument_exp_list)
        NArgumentExp     (expression)
```

김 영 철

1990년 한남대학교 전자계산학과 졸업 1996~현재 숭실대학교 컴퓨터학과 박사과정. 2002년 3월~현재 명지전문대학 겸임교수. 관심분야는 컴파일러, 망관리, 프로그래밍 언어

김 성 근

1993년 숭실대학교 전자계산학과 졸업 1995년~현재 숭실대학교 컴퓨터학과 박사과정. 1998년~현재 가톨릭상지대학 컴퓨터정보계열 조교수. 관심분야는 프로그래밍 환경, 에이전트 프로그래밍 언어

염 세 훈

1992년 2월 국립 서울 산업대학교 전자계산학과 졸업(공학사). 1992년~1994년 숭실대학교 컴퓨터학과(공학 석사). 1994년~1999년 숭실대학교 컴퓨터학과 박사과정 수료. 2002년 3월~현재 동서울 대학 전자계산과 전임강사. 관심분야는 컴파일러, 프로그래밍언어, 인간과 컴퓨터 상호작용, Voice (Speech) Based Markup Language, XML

최 종 명

정보과학회논문지 : 소프트웨어 및 응용 제 30 권 제 3 호 참조

유 재 우

정보과학회논문지 : 소프트웨어 및 응용 제 30 권 제 3 호 참조