AST를 이용한 프로그램 유사도 평가 시스템 설계

o
정재은, 김영철, 김상헌, 유재우
숭실대학교 컴퓨터학과

A Design of the Similarity Evaluation System using Abstract Syntax Tree

o
Jae-Une Jung, Young-Chul Kim, Sang-Heon Kim, Chae-Woo Yoo
Department of Computing, Soongsil University.

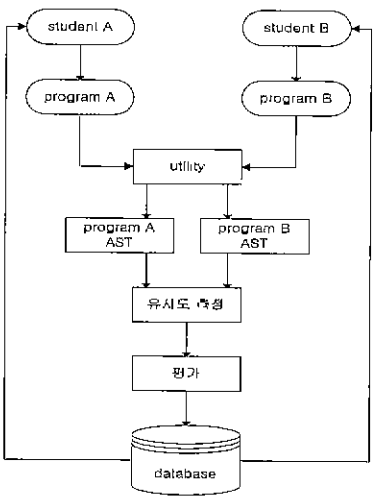
요 약

본 논문에서는 AST(Abstract Syntax Tree)를 이용하여 서로 다른 프로그램의 유사도를 측정하는 방법을 제시한다 지금까지 유사도 평가 방법은 거의 제시되지 않고 있으며 프로그램의 비교평가 연구는 찾아볼 수가 없다 본 시스템은 서로 다른 여러 프로그램을 입력받아 파싱함으로써 AST를 생성하여 생성된 AST를 유사도 측정에 이용한다 따라서 다른 비교 측정 시스템보다 비용과 속도 면에서 경제적이고 빠르게 유사도를 검출해 낼 수 있으며, 또한 신속히 평가 결과를 학생들에게 피드백 함으로써 큰 학습성과를 기대할 수 있다

1. 서론

본 논문은 학생들이 제출한 과제물에 대한 유사도를 측정하여 보다 빠른 평가와 복제 검사를 할 수 있도록 설계하였다. 현재 학생들이 제출한 프로그램 과제물에 대한 평가에서 교원은 객관적인 프로그램 평가 방법이 어렵다 특히 실기 측정은 교원이 직접 프로그램을 실행시켜봐야 하며, 더욱 어려운 것은 표절에 관한 비교 측정방법이다 만약 60명의 학생들에 대한 과제물을 교원이 표절검사하려면 모두 224번의 반복작업이 필요하다 더 나아가서 프로그램의 양과 학생 수가 많아지면 비교평가는 더욱 많은 시간과 노력이 필요하게 된다. 또한 실기 측정이 여러명의 조교나 학생들에 의해서 이루어 지면 실기 측정에 대한 신빙성을 상당히 의심받게 된다 검수자의 심한 편치를 줄이기 위해 나름대로의 작업을 서로 해주어야 하는 불편함을 감수 해야 함은 물론이다. 본 논문에서는 AST를 이용하여 유사도를 측정함으로써 교사 입장에서 학생들의 많은 과제물을 비교검토할 수 있고, 학생들이 제출한 레포트에 대한 평가 시간을 최소화하고 평가 결과를 알려주어 학생들의 학습능력을 향상시킨다 또한 표절이나 특정 패턴의 과제물을 직접히 검사할 수기 있다. 학생들은 자신의 과제물에 대한 평가와 더불어 잘못된 점에 대한 오류를 지적받을 수 있으므로 학습에 도움을 받을 수 있으며, 또한 복제를 근절할 수 있는 장치를 마련 함으로써 학생들의 학습능력을 향상 시킬 수 있다.

2. 시스템 구성도



[그림 1] 시스템 구성도

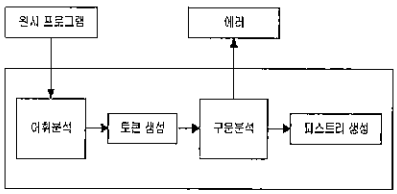
시스템 구성도에서 학생들은 교사에게 과제물을 제출하며 이 과제물은 utility에 의해서 각각에 대한 AST를 생성하게 된다 각각의 생성 AST는 AST 평가 시스템에 의해서 평가되어 복제 여부에 대한 등급이 정해지

게 된다 평가 시스템에서 나온 평가와 오류등은 곧바로 학생들에게 피드백 된다. 이때 평가에 대한 등급이 정해지며, 강한 유사도를 보이는 학생에 대한 평가는 정밀 검사에 들어간다

3. AST 를 이용한 유사도 검사 방법

3.1 AST

AST는 프로그램을 tree 형태로 구성한 것으로써 일종의 피스 트리이다



[그림 2] AST 구성도

원시 프로그램은 어휘분석, 즉, lex에 의해서 하나의 작은 토큰으로 잘라게 되며 이 잘린 토큰들은 구문 분석기인 yacc에 의해서 트리로 구성된다. 결국 AST 는 원시 프로그램을 트리 형태로 구성한 것이라고 볼 수 있다

AST는 최상위 root NODE를 시작으로 숫자나 변수를 담을 수 있는 마지막 NODE 까지 트리 형태로 구성된다

AST는 bottom-up 방법으로 만들어 지며, bottom up은 주어진 스트링으로부터 시작 심볼로 축약되어 가는 과정으로, 주어진 스트링이 시작 심볼로 축약 또는 reduce 될 수 있으면 올바른 문장이고 그렇지 않으면 틀린 문장으로 간주하는 방법이다 이 때 트리를 단말 노드로부터, 즉 아래에서 위로 거꾸로 만들 수 있는데 이로 인해 bottom-up 이란 이름이 붙여진 것이다 노드들은 노드의 이름과 왼쪽, 오른쪽 자노드를 저장할 수 있는 영역으로 구성된다

Lex는 프로그램에서 토큰을 분리하며 yacc은 분리된 토큰을 가지고 문법에 맞추어 노드를 만들어 나간다 이때 프로그램에 에러가 있다면 트리를 생성할 수 없다.

3.2 장점

① AST는 방대한 프로그램에서 작은 프로그램까지 유사도 검사에 적용될 수 있다

② 복제를 한 당사자가 약간의 프로그램을 수정했을 경우에도 유사도를 추정하는 것이 가능하다.

③ 컴퓨터가 정확히 판별하기 어려운 프로그램도 검사자에 의해서 유사도 측정이 가능하다 즉, 검사자가 눈으로 식별가능하도록 AST 트리를 생성하는 프로그램을 만든후 컴퓨터에 의해서 강한 유사도를 보이는 프로그램을 선별한 후 검사자는 AST 트리를 눈으로 직접 판독하면서 쉽게 유사도를 검사할 수 있다

④ AST는 프로그램에 에러가 있을 경우 트리를 생성할 수 없으므로 에러를 쉽게 판별해 낼 수 있다

⑤ AST는 함수 각각에 대해서도 유사도를 검사할 수 있다

⑥ 복제를 한 당사자가 프로그램의 변수나, 함수 이름, 변수에 할당된 값등을 바꾸 었을 경우 유사도 검사를 정확히 판별해 낼 수 있다.

⑦ While 이나 if, swich 문등의 안에 복제자가 간단한 변수나, 수치를 추가 했을 경우 AST에 대한 전체 적인 트리는 영향을 받지 않으므로 이 때는 프로그램이 강한 유사도를 나타내게 되고 검사자는 이 프로그램에 대해서 정밀 검사를 진행하게 된다

⑧ 함수나 순환문, if – then, switch 등의 순서를 바꾸 었을 경우는 AST가 크게 변화 하게 된다 하지만 함수, 순환문, if – then, switch 등의 개개의 트리를 따로 유사도 검사 한다면 이를 해결할 수 있다. 즉, 함수, 순환문, if – then, switch 등의 순서를 바꾸어도 개개의 트리를 따로 검사함으로써 강한 유사도를 유추해 낼 수 있다

3.3 단점

① AST는 악의를 가진 복제자에 대해서는 유사도를 판독하기 힘들다. 즉, 프로그램을 크게 변형 시켰을 경우는 유사도 검사에서 약한 유사도를 나타냄으로써 복제 시스템을 비켜가게 된다. 여기서 크게 변형 시켰다함은 함수나 순환문등의 순서를 바꾼 후 함수나 순환문의 내용에서 다시 변형을 가했을 경우이다.

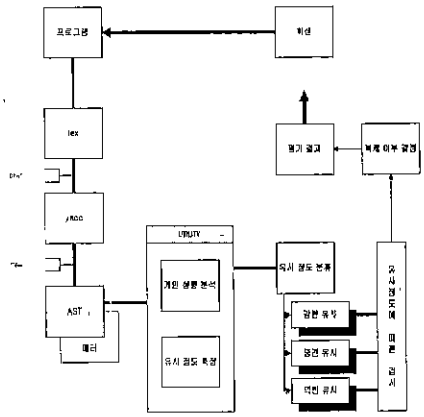
② 논리적인 에러를 판별할 수 없다. 실행에러는 AST 트리가 만들어 지지 않음으로써 쉽게 판독이 가능하지만 논리적인 에러는 유사도 검사 프로그램이 AST 트리를 생성함으로 다른 보조 프로그램에 의해서 에러를 판별해야 한다

③ 프로그램이 여러 개의 파일로 구성되어 있을 경우는 AST 구성에 어려움이 따른다.

④ 복제자가 프로그램안에 엉뚱한 문장을 삽입했을 경우는 유사도 검사 프로그램이 유사도를 판별하기가 어려워 진다 가령, 복제자가 정상적인 프로그램 안에 전혀 아무 영향을 미치지 않는 삽입문(순환문, if-then, switch 등)을 넣었을 경우 AST 트리는 큰 변화가 생겨 결국 유사도 검사 프로그램이 약한 유사도를 출력하게 된다

3.4 AST 와 utility 의 결합

위와 같은 단점으로 AST 트리와 더불어 개인의 개성에 의한 유사도 검사 방법을 병용한 Utility를 결합함으로써 좀더 효율적으로 유사도를 검사할 수 있다 그림 3과 같은 구성으로 AST와 Utility를 결합하여 효율성을 증대 시킨다. 유사도는 크게 강한 유사, 중간 유사, 약한 유사로 나누며 세 부류 모두 세밀한 점수로 다시 나눈다 강한 유사를 나타내는 것만 검사하더라도 검사자의 검사 시간을 상당히 절약할 수 있음을 알 수 있다



[그림 3] AST 와 UTILITY 를 결합한 유사도 검사 구성도

평가 결과는 학생들에게 다시 피드백 되므로 학생들의 학습능력을 향상시킨다 AST를 도출해 내는 과정에서 에러를 검출하여 실행되지 않는 파일이나 잘못된 파일에 대해서 곧바로 색출이 가능하며, 이러한 프로그램은 유사도 측정을 하지 않고 평가 결과를 거쳐 곧 바로 학생에게 다시 전달된다. 전달받은 학생은 자신의 프로그램에 대해 검토하고 다시 작성함으로써 학습능력을 향상 시킨다

utility 중에서 개인 성향 분석은 다음과 같은 특징을 가지고 있다

① "원시 소스는 어느 정우든 약간의 에러를 포함한다"고 가정한다 예를 들면, 변수는 정의하고 사용하지 않는 정우, 함수를 정의하고 사용하지 않는 경우, 무한 루프, 한번도 진행되지 않는 프로그램 영역, 잘못된 타입선언으로 생성되는 경고등등 표절학생이 원래의 소스를 분석해서 이러한 경고나 에러를 고칠경우는 일단 전무하다고 생각할 수 있다. 프로그램이 커지면 그럴 가능성은 더욱 희박해 진다 결국 복제자의 소스 에러는 원본 소스 에러와 같으므로 이를 분석하고 AST와 결합하여 복제 검사를 강화 한다.

② "사람마다 지문처럼 그들만의 프로그램 스타일이 존재한다"고 가정한다 "int reverse(char to[], char from[]),"라는 문장을 예로 들면 int 후에 tab키로 공백을 처리하는 경우와 스페이스 바로 처리하는 경우가 있으며 함수를 정의할 때 괄호 다음에 두는 경우와 두지 않는 경우, char *to 라고 정의 하는 경우와 char to[]라고 정의하는 경우, return 문에서 괄호를 꼭 써서 return 문을 쓰는 경우와 괄호를 전혀 쓰지 않는 경우, 대입문 a=0을 사용할 때 a문자 다음 공백을 두는 경우와 두지 않는 경우등 그사람만의 스타일이 존재한다 소스 코드를 분석해 본 결과 각자의 개성은 매우 뚜렷하게 나타 났으며 프로그램이 커질수록 개성을 정확히 파악할 수 있었다 이를 적절히 이용하는 방법이 "개인 성향 분석" 방법이다

4. 결론 및 향후 연구 과제

지금까지 AST를 이용한 평가 시스템에서의 장점과 단점 그리고 영향에 대해서 알아 보았다. 이 시스템은 과제물의 수나 양이 많을 때 큰 힘을 발휘한다 또한 일관성 있는 과제물의 평가가 가능하고 학생들의 학습능력을 높일 수 있는 시스템이다

향후 과제로는 AST를 보완 하는 유틸리티를 제작해야 할 것이다. AST는 악의를 가진 복제자에게는 약한 유사도를 보이는 취약점이 있으므로 이를 극복할 수 있는 유틸리티를 추가함으로써 보다 강력한 유사도 측정 시스템을 구축할 수 있도록 해야 할 것이다

참고 문헌

[1] John R.Levine, Tony Mason and Doug Brown, UNIX Programming Tools lex & yacc, O'Reilly pub. 1995.

[2] A.V. Aho and S C. Johnson "LR Parsing," ACM Computing Surveys, Vol. 6, 1974.

[3] D.E. Knuth, "Top-down Syntax Analysis, " Acta Informatica, 1971

[4] M E. Lesk, "Lex-A Lexical Analyzer Generate, " Comp Sci Tech Rep No 39, Bell Laboratories, Murray Hill, New Jersy, 1975

[5] *Amsterdam Compiler Kit Manual*, UniPress Software Inc,1987

[6] Robert W. Sebesta, *concepts of . Programming Languages*, Third Edition, Addison Wesley, 1996.

[7] Alfred V. Aho, Ravi Sethi & Jeffrey D. Ullman, *Compilers Principles, Techniques, and Tools*, Addison Wisley, 1986.

[8] J M. Spivey, *The Z Notation · A Reference Manual*, Prentice-Hall, 1989

[9] B. W. Kernighan & P J Plauger, *The Elements of Programming Style*, McGraw-Hill, 1974.