# Accurate detection of code clones

Hyo-Sub Lee

Jan. 07. 2010





## 개념

- 코드 클론(중복 코드)
  - 소스 프로그램에서 구문적 생김새가 동일한 코드 조각
- 코드 클론의 응용분야
  - 리팩토링
  - 소프트웨어 형상관리
  - 복사 후 붙이기 오류
  - 복제 감정평가 및 프로그램 과제의 복제 판별

# 기존 연구방법의 문제점

- 대부분 기존 코드 클론 도구들은 토큰들의 나열인 토큰열 비교.
- 토큰열은 프로그램의 1차원 비교로, 프로그램의 구문구조를 전혀 고려하지 않는다.
  - 틀린 코드 클론을 찾아내기 쉽다. (false positive)
- 의존 그래프기반 방식, 계량치 기반 방식
  - 모양이 다른 코드의 그래프나 계량치가 동일할 수 있다. (false positive)
- 문자열기반 방식
  - 단순한 프로그램 포맷변경에 민감하다. (false negative)

정확한 코드 클론을 찾기 위해서는 코드의 구문구조 정보가 포함되어 있는 AST를 비교하는 방식이 적절하다.





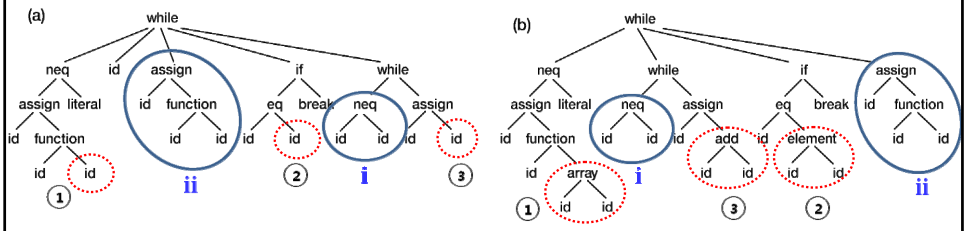
# 기존 트리기반 방식

- AST를 샅샅이 뒤져서 비교하는 방식은 계산적으로 비싸다. 따라서 기존의 방식은 AST를 적절히 요약하여 계산 비용을 줄이는 것이 일반적이다.
- 기존 트리기반 방식의 문제점
  - 요약된 AST의 잃어버린 구조정보로 인한 탐지결과의 불완전성과 불안전성
  - 불완전 : False Negatives
    - 골격이 비슷하여 상당부분 코드클론임에도 불구하고, 일부 하위트리의 상이함이 모든 상위트리의 유사성을 판단하는데 영향을 미쳐 클론을 놓친다.
  - 불안전 : False Positives
    - 트리의 구조적인 정보를 계량화하는 경우, 계량화 과정에서 구조정보의 일부를 잃어버리기 때문에 틀린 클론을 찾아내는 경우가 많다.

# False Negative

```
while ((result_code=dbresults(dbproc)) != 0.0)
{
    int result_code;
    sql = mkdb(query);
    if (result_code == FAIL) break;
    while(result_code != dbstore)
      result += result_code;
}
```

```
while ((resultcode=dbresult(pDb_Proc[dc])) != 0)
{
   while(resultcode != db_store)
      result = result + resultcode;
   if (resultcode == preload.arguments) break;
   sql = mkdb(query);
}
```





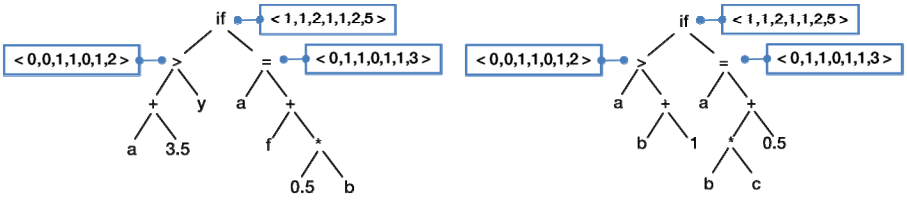



# False Positive

Characteristic vector
< if, =, +, >, *, literal, id >

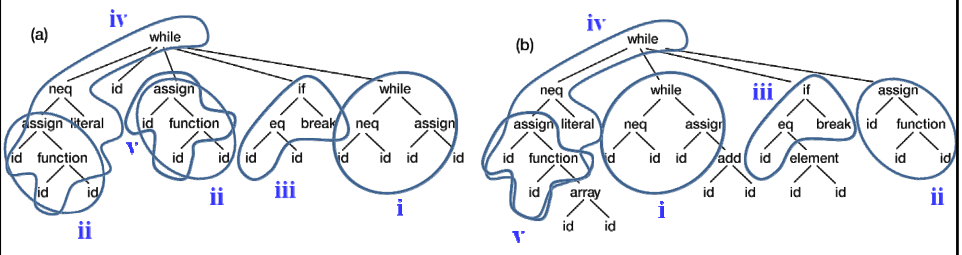
If( a+3.5 > y)  a = f+0.5 * b

If( a > b+1)  a = b *c + 0.5




두 AST의 characteristic vector는 동일하다.
그러나 두 코드는 실제로 다르다!

# 우리의 방식으로 찾아낸 코드 클론



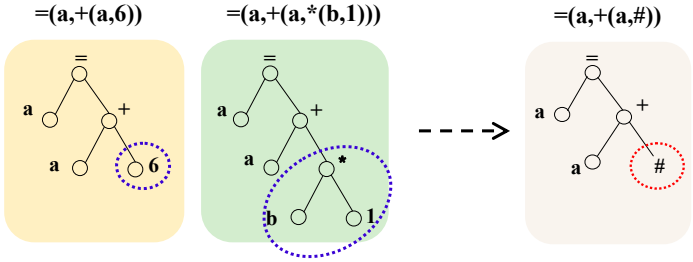





# 우리의 방식

1. 코드의 구문구조 정보가 포함되어 있는 AST를 비교
2. 동일한 코드 조각(tree pattern)을 모아 클러스터링

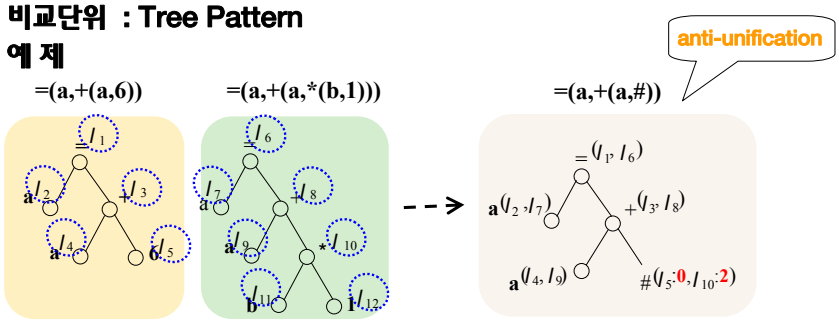
- **비교단위 : Tree Pattern**
- **예 제**



# 우리의 방식

1. 코드의 구문구조 정보가 포함되어 있는 AST를 비교
2. 동일한 코드 조각(tree pattern)을 모아 클러스터링

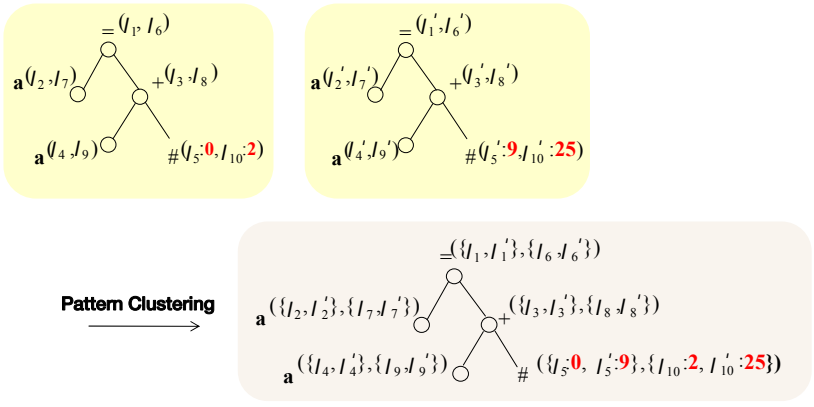
- **비교단위 : Tree Pattern**
- **예 제**








# 트리 패턴 클러스터링



# 트리패턴으로 정확한 코드클론 찾기 알고리즘

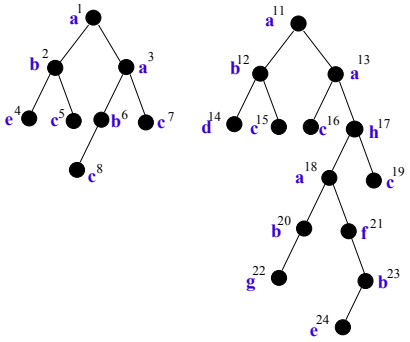
- 입력 : 두 개의 AST
- 결과 : Pattern clusters
- 알고리즘의 단계
  - 단계 1 : Top-down방식으로 AST의 트리패턴을 찾아나가면서 코드클론을 모은다.
    - Anti-unification 알고리즘을 이용해 AST에서 트리패턴을 찾는다.
    - 중복 비교를 피하기 위해 Work-Table에 이미 방문한 트리의 노드들을 등록하여 항상 최대 트리 패턴을 구한다.
  - 단계 2 : 클러스터링
    - 검출된 트리 패턴들은 동일한 모양을 가진 패턴끼리 모아 하나의 패턴 클러스터로 병합된다.





# 트리패턴을 이용한 코드 클론 탐지

- 예제

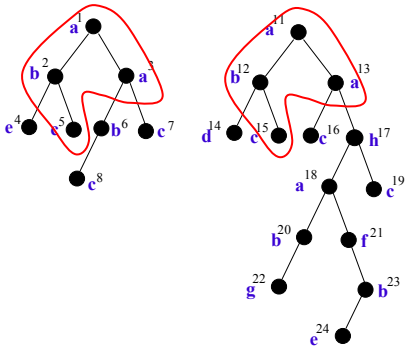



pattern-clusters

| Index | The set of pattern clusters |
|---|---|
| a | a({1},{11}) b({2},{12}) a({3},{13}) #({4:0},{14:0}) #({6:0},{16:0}) #({7:0},{17:7}) c({5},{15}) a({1,3},{11,18}) b({2,6},{12,20}) #({3:3,7:0},{13:9,21:2}) #({4:0,6:1},{14:0,22:0}) a({1},{13}) #({2:2},{16:0}) #({3:3},{17:7}) |
| b | b({2},{23}) e({4},{24}) b({6},{23}) #({8:0},{24:0}) |

# 트리패턴을 이용한 코드 클론 탐지

## - 예제



pattern-clusters

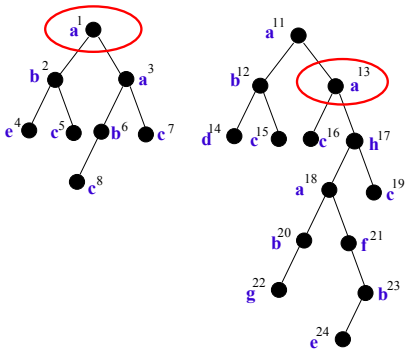
| Index | The set of pattern clusters |
|---|---|
| a | a({1},{11}) b({2},{12}) a({3},{13}) #({4:0},{14:0}) #({6:0},{16:0}) #({7:0},{17:7}) c({5},{15}) a({1,3},{11,18}) b({2,6},{12,20}) #({3:3,7:0},{13:9,21:2}) #({4:0,6:1},{14:0,22:0}) a({1},{13}) #({2:2},{16:0}) #({3:3},{17:7}) |
| b | b({2},{23}) e({4},{24}) b({6},{23}) #({8:0},{24:0}) |





# 트리패턴을 이용한 코드 클론 탐지

## - 예제

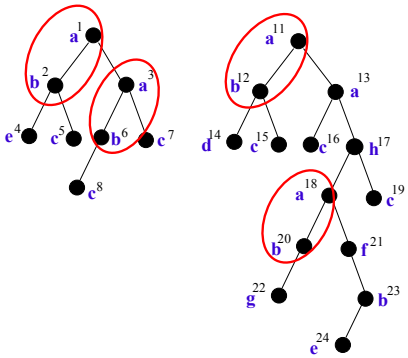


pattern-clusters

| Index | The set of pattern clusters |
|---|---|
| a | a({1},{11}) b({2},{12}) a({3},{13}) #({4:0},{14:0}) #({6:0},{16:0}) #({7:0},{17:7}) c({5},{15}) a({1,3},{11,18}) b({2,6},{12,20}) #({3:3,7:0},{13:9,21:2}) #({4:0,6:1},{14:0,22:0}) a({1},{13}) #({2:2},{16:0}) #({3:3},{17:7}) |
| b | b({2},{23}) e({4},{24}) b({6},{23}) #({8:0},{24:0}) |

# 트리패턴을 이용한 코드 클론 탐지

- 예제



pattern-clusters

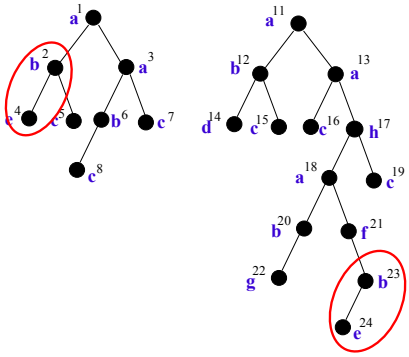
| Index | The set of pattern clusters |
|---|---|
| a | a({1},{11}) b({2},{12}) a({3},{13}) #({4:0},{14:0}) #({6:0},{16:0}) #({7:0},{17:7}) c({5},{15}) a({1,3},{11,18}) b({2,6},{12,20}) #({3:3,7:0},{13:9, 21:2}) #({4:0,6:1},{14:0,22:0}) a({1},{13}) #({2:2},{16:0}) #({3:3},{17:7}) |
| b | b({2},{23}) e({4},{24}) b({6},{23}) #({8:0},{24:0}) |





# 트리패턴을 이용한 코드 클론 탐지

- 예제

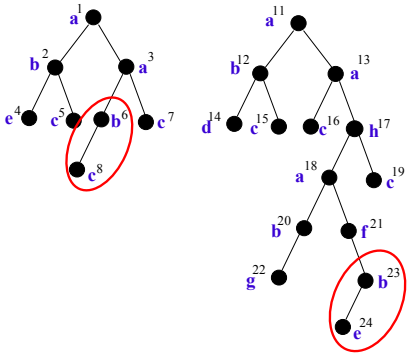


pattern-clusters

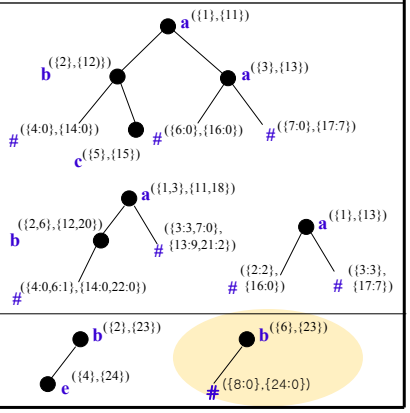
| Index | The set of pattern clusters |
|---|---|
| a | a({1},{11}) b({2},{12}) a({3},{13}) #({4:0},{14:0}) #({6:0},{16:0}) #({7:0},{17:7}) c({5},{15}) a({1,3},{11,18}) b({2,6},{12,20}) #({3:3,7:0},{13:9,21:2}) #({4:0,6:1},{14:0,22:0}) a({1},{13}) #({2:2},{16:0}) #({3:3},{17:7}) |
| b | b({2},{23}) e({4},{24}) b({6},{23}) #({8:0},{24:0}) |

# 트리패턴을 이용한 코드 클론 탐지

- 예제



pattern-clusters

| Index | The set of pattern clusters |
|---|---|
| a |  |
| b |  |





# 구현

- 대상 : C 프로그램
- 구현 환경
  - 구현 언어 : Objective Caml 3.09
  - 파서 : CIL 1.3.6 (C Intermediate Language)
  - 개발 환경
    Mac Pro at Xeon 2x2.8 GHz Quad Core processor
    8GB RAM running Mac OX X Server 10.5.8
  - 대상 응용 프로그램 : ***Tar, Apache Web Server*** and ***MySQL***

- 응용 프로그램 실행 시간

| source | files | lines | execution time |
|---|---|---|---|
| Tar 1.19 | 251 | 149,154 | 0h 03m 06s |
| Apache 2.2.8 | 543 | 278,172 | 1h 57m 17s |
| MySQL 6.0.3 | 802 | 716,215 | 7h 57m 49s |

# 지금까지 한 일 과 추후 해야 할 일

- **지금까지 한 일**
  - 정확한 코드 클론을 찾는 알고리즘의 개발/ 구현/ 시험
  - 실제 응용 프로그램에 적용 : 리팩토링
- **현재 하고 있는 일과 추후 해야 할 일**
  - 다른 도구들과의 성능 비교
  - 도구 개발 : Web 기반
  - 응용분야에 적용
    - 리팩토링, 소프트웨어 형상관리, 복사 후 붙이기 오류 찾기, 복제 감정평가