논문 22-2-05

프로그램 유사도 산정을 위한 육안보정 반자동화

Supporting Manual Review in Estimating Program Similarity

안효천, 이욱세, Yan Bin Tang
한양대학교 컴퓨터공학과
{hcahn;oukseh;tangyanbin}@pllab.hanyang.ac.kr

요 약

프로그램 코드의 유사도 측정에 대한 알고리즘은 여러 가지 존재하고 있으며 유사도 측정 프로그램도 많은 부분 향상되어 왔다. 이런 프로그램 유사도 측정 도구는 코드의 문법만을 검사의 대상으로 사용하게 되고 그런 점 때문에 실제 소프트웨어 코드 분쟁에 있어서 사용하기에는 한계가 존재한다. 복제 시 변형된 코드, 소스코드 미공개로 인한 역컴파일된 코드, 자동 생성 코드가 포함된 코드와 같은 경우에는 그 유사도의 신뢰성이 저하되게 된다. 본 논문에서는 이런 문제점을 보안하기 위해서 기존의 비교엔진에 더해서 사람의 경험과 판단이 개입되어 반자동으로 신뢰도 있는 유사도를 편리하고 빠르게 보정하는 방식을 제안한다. 그리고 제안하는 방법을 사용하여 기존의 방식보다 향상된 결과를 도출할 수 있었다.

1 서 론

많은 프로그램이 개발되는 요즘에는 프로그램 코드 복제에 대한 정확한 유사도 감정은 매우 중요한 일이다. 프로그램은 프로그래머의 허락없이 도용되어서는 안되는 독창적 창작 활동이다. 하지만 개발 시간을 단축하거나 성능을 향상시키기 위한 목적으로 다른 사람의 프로그램 코드를 무단으로 도용하는 일이 발생한다. 그래서 기업간 혹은 특정 단체간에 코드 복제에 대한 분쟁은 끊이지 않고 있다. 그런 분쟁은 결국 양측에서 개발한 코드간의 유사도에 따라 결정나게 된다. 이렇듯 프로그램 유사도는 기업,단체 혹은 개인간의 분쟁을 결정 짓는 중요한 요소이다.

유사도를 감정하는 기존의 연구와 도구는 코드에서 변경 없이 복제된 부분을 검출해 낼 수는 있지만, 변형 된 코드나 불필요한 코드에 대해서는 잘못된 유사도를 감정하는 한계가 있다. 본 논문은 아래 세 가지와 같은 경우의 잘못된 유사도 측정에 대해서 효율적인 방법을 제시하고 있다. 첫째로 코드 도용 시 그 코드에 의도적으로 변형을 하게 되면 복제한 코드이지만 검출도구에서는 다른 코드라고 판단 할 수 있다. 둘째로는 프로그램 유사도 감정에 있어서 반드시 소스 코드로만 비교하는 것은 아니다. 예를 들어 프로그램 복제 분쟁에 있어 피고발자는 자신들의 기밀유출을 우려해 소스코드가 아닌 실행 파일로 비교 하기를 원할 수도 있다. 이런 경우는 역컴파일을 사용해 코드를 비교하게 되는데 역컴파일은 프로그램 코드의 변경을 가져올 수 있다. 그래서 복제 코드 일지라도 컴파일-역컴파일의 과정을 거치면서 변경되어지고 유사도 측정도구는 그 변경을 모르고 비유사 코드로 결론 내리게 된다. 셋째로는 불필요하거나 유사도 측정에 개입되어서는 안되는 코드들도 유사도 측정에 개입 될 수가 있다. 예를 들어, 프로그램 개발툴 등에 의해서 코드에 자동적으로 삽입된 코드의 경우 그 부분은 창작된 부분도 아니고 복제된 부분도 아니기 때문에 유사도측정

에 개입 되어서는 안 된다. 하지만 실제로는 그 부분도 유사도 측정에 개입되어 신뢰도를 떨어뜨리는 결과를 가져온다.

원본 프로그램	컴파일 후 역컴파일로 복원한 프로그램

`int some_code(int n) {`
`  int i, j=1;`
`  for(i=2; i<n+1; i++)`
`  j *= i;`
`  if(j>127 || j%2)`
`  j -= (j+16)/2;`
`  return j;`
`}`
	`some_code(int param1) {`
	`  int local1, local2, local4, local5;`
	`  local1 = 1;`
	`  local2 = 2;`
	`  while (param1 + 1 > local2) {`
	`  local4 = local2 * local1;`
	`  local1 = local4;`
	`  local2++;`
	`  }`
	`  if (local1 > 127) {`
	`L2:`
	`  local4 = (local1 + 16) / 0x80000000 + local1 + 16;`
	`  local1 = local1 - (local4 >> 1);`
	`  } else {`
	`  if ((unsigned char) (local1 & 0x1) != 0)`
	`  goto L2;`
	`  }`
	`  return local1;`
	`}`

[그림 1] C# 프로그램 복원 예제

이런 형태로 떨어진 유사도의 신뢰도를 회복하기 위해선 기존의 방법으로는 많은 노력과 시간을 필요로 하게 된다. 문법만을 비교하는 비교엔진에는 한계가 존재하고 유사도에 대한 신뢰성이 떨어질 가능성이 많이 존재한다. 기존에는 그런 떨어진 유사도를 회복하기 위해선 코드에 대한 지식과 경험이 있는 전문가가 직접 두 코드를 비교하면서 잘못된 부분을 검출하는 방법을 사용해 왔다. 하지만 이런 방식은 전문가의 많은 노력과 시간을 소비하게 되고 전문가의 실수도 포함 될 가능성이 있어서 비효율적이고 위험하다.

본 논문에서는 이런 유사도의 신뢰성을 회복하는 효율적인 방법을 소개하고 실제로 테스트한 결과를 바탕으로 효율성을 비교해 보았다.

2 관련 연구

코드 복제 검출에 대한 여러 연구가 있었으며, Roy와 Cordy[18]는 코드 검출 방식을 크게 6가지로 분류하고 있다. 텍스트/토큰 비교[1, 3, 5, 6, 7, 13, 16]는 소스코드의 텍스트나 토큰을 나열하여 비교하는 방법이다. 매트릭스 비교[9, 11, 15, 17]는 코드 부분의 서로 다른 매트릭스를 생성하여 이 매트릭스의 유사도를 측정하는 방법이다. 문법 트리 비교[2, 4, 19, 20]는 소스 코드의 요약된 문법 트리를 생성하여 그 트리의 유사도를 측정하는 방법이다. 의존도 그래프 비교[8, 10, 14]는 프로그램 코드의 의존도 그래프를 그려 그 그래프를 비교하는 방법이다. 그 외로 비교 방식을 융합하여 사용하는 방법[12]이나 데이터마이닝에서의 그것과 유사한 방식으로 비교하는 방법[13, 19]이 있다.

본 논문에서는 특정 비교엔진을 제안하는 것이 아니라 기존의 비교엔진을 사용하여 유연성을 높이는 방법을 제안하고 있다. 본 논문에서는 토큰 비교 도구를 사용한다. 그 이유는 토큰 비교는 구문분석기(parser)를 필요로 하지 않아서 유사도 감정에 가장 현실적으로 적합하기 때문이다. 본 논문의 비교엔진으로는 토큰 비교 기반의 Sim[5]을 사용하고 있다.

3 반자동 유사도 감정 방법

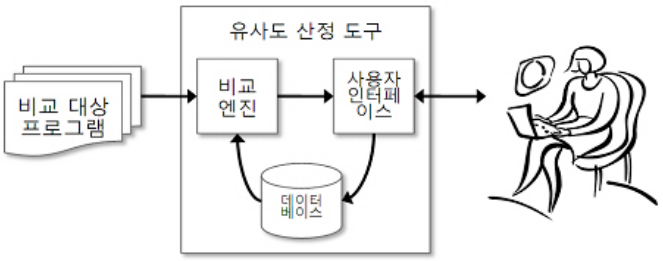
유사도의 신뢰도를 높이기 위해 고려해야 할 사항은 유사한 패턴 검사로 볼 수 있다. 원래 소스코드는 유사한데 컴파일을 거치면서 겉보기에는 유사하지 않은 두 형태의 코드 부분이 존재할 수 있다. 즉, 한 쪽에서 다른 한 쪽으로 컴파일 도중 변환되었으나 역컴파일시 복구되지 않은 경우가 다수 발생할 수 있다. 이를 감지하기 위해서는 원본 형태와 컴파일·역컴파일을 거친 후의 형태를 모두 알고 있어야 한다.

육안 검사 작업으로 컴파일러 지식이 있는 전문가에 한해서 이러한 패턴 검사를 통해 형태를 감지할 수 있다. 컴파일러에서의 변환작업은 언어에 따라 크게 종류에 차이가 없기 때문에, 전문가는 어떤 언어라 할지라도 두 패턴이 컴파일-역컴파일로 인해 모양만 다를 뿐 원래는 같은 코드임을 감지할 수 있는 것이다.

문제는 이런 육안 검사 작업이 오랜 시간과 노동력을 소비한다는 것이다. 단순히 패턴을 보는 것이 아니라, 두 패턴이 어떤 과정을 통해 같은 코드라고 볼 수 있는지 논리적 추론을 해야 한다. 이는 단순 노동이상의 고차원의 노동이다. 또한 유사한 변환 과정이 코드의 여러 부분에 반복적으로 적용되기 때문에, 반복적으로 감지해 주어야 한다. 이는 단순 노동에 속한다. 이렇게 고차원의 노동과 단순 노동이 교차되면서 아무리 전문가라 할지라도 실수를 할 수 있고 결과적으로 감정의 신뢰도가 훼손되는 일이 발생할 수 있다.

3.1 반자동 시나리오

이러한 불안 요소를 줄이기 위해 본 논문에서는 전문가가 육안으로 확인한 동일한 패턴의 경우 그 패턴의 형태를 데이터베이스에 저장하고 그 패턴의 형태를 유사도 측정 프로그램에 개입시키는 방법을 이용한다.



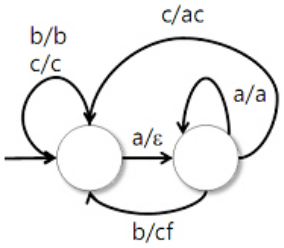
[그림 2] 사용자 경험에 기반한 유사패턴 쌍 데이터베이스 구축

예를 들어 전문가에 의해서 패턴 ab와 패턴 cd가 동일하다고 검증되었을 경우 (ab, cd)쌍이 생성되어지며 그 데이터가 저장되어지고 유사도 산출 방식에 영향을 주게 되는 것이다. 이전의 유사도 산출에서는 $cabd$와 $ccdd$는 다른 코드로 인식이 되며 검출이 되지 않았을 것이다. 하지만 저장된 패턴 (ab, cd)쌍에 의해 $cabd$와 $ccdd$는 같은 패턴이란 것이 밝혀지게 되는 것이다. 같은 방식으로 전문가에 의해서 ef는 프로그래밍 도구에 의해서 자동적으로 생성되거나 불필요한 코드라고 판명이 나면 그 패턴은 데이터베이스에 저장 된다. 그리고 그 부분이 비교엔진에 개입되어 그 전에 검출된 (ef, ef) 부분은 유사하지 않은 코드로 변경되는 것이다. 이런 형태로 전문가에 의해서 저장된 보정 패턴들은 다른 코드에도 영향을 주게 되고 그 다음 유사도 측정에는 더 신뢰도 있는 유사도가 산출되게 된다. 이를 반복할수록 유사도의 신뢰도는 높아지게 된다.

위에서 설명한 유사도 선정 도구의 유사패턴 쌍이나 불필요한 코드를 비교엔진에 개입시켜 유사도를 검출하는 방식을 구현하기 위해서는 한 가지 문제가 존재한다. 기존의 코드에서 유사패턴 쌍을 어떻게 찾아내고 찾아낸 유사패턴 쌍을 어떤 방법으로 유사도 산출 비교엔진에 개입시켜서 유사도를 검출할 것인가이다. 이 문제는 토큰열 변환 오토마타에서 오토마타를 통한 재작성 기법을 이용하여 해결하였다.

3.2 토큰열 변환 오토마타

토큰열 변환 오토마타는 일정 패턴을 정해진 다른 패턴으로 교체시키는 방법을 말한다. 사람의 육안 검사로 나온 결과는 목적 토큰 나열과 대체 토큰 나열의 쌍으로 구현 할 수 있다. 예를 들어 원본 소스의 cf라는 토큰 나열과 복제 코드에서 ab라는 토큰 나열이 복제된 코드라고 판명 된 경우 (cf, ab)라는 쌍이 데이터베이스에 저장된다. 이것은 원본코드의 cf토큰 나열은 복제코드의 ab라는 토큰 나열로 변경 혹은 수정 되었지만, 그 두 토큰 나열은 복제된 것이다. 라는 의미로 해석할 수 있다. 이 의미를 유사도 비교엔진에 개입 시키기 위한 방법으로 오토마타를 이용, 복제본의 전체 코드에서 ab토큰 나열을 찾아내고 그 ab부분을 cf로 재작성하는 방법을 이용하였다.

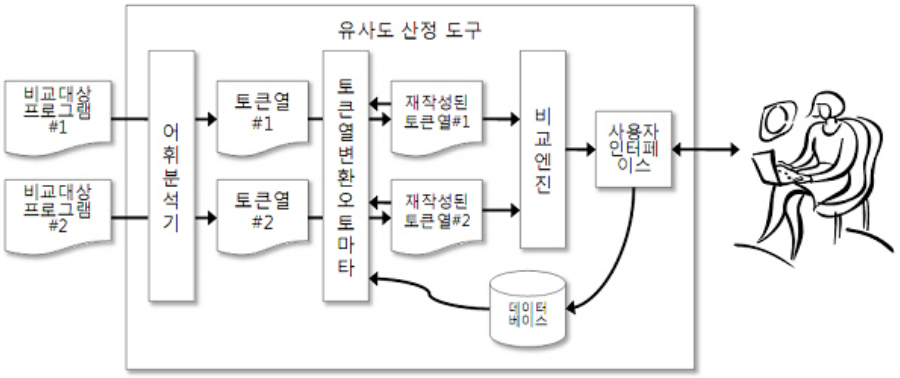


[그림 3] 토큰열 변환 오토마타

[그림 3]은 ab라는 입력을 받아서 cf라는 출력을 내어주는 오토마타를 도식화 한 것이다. 각각 입력/출력 에 따른 상태 이동을 보여주고 있다. 위 그림에서 입력 a에 따라서 다음 상태로 이동이 가능해지며 다음 상태에서는 b 라는 입력에 의해서만 출력 cf가 나오게 된다. 위의 오토마타 도식으로 예를 들면, 복제 코드의 전체 토큰 나열이 $acdabf$이고 전문가에 의해서 $(cf\ ,\ ab)$토큰쌍이 판명 되었다고 하자. 판명 된 토큰쌍은 [그림3]과 같은 토큰열 변환 오타마타로 생성되고 데이터베이스에 저장 된다. 그리고 전체 토큰 나열 $acdadf$는 생성된 오토마타를 통과하게 된다. 그러면 전체 토큰 나열 $abcdabf$는 $abccff$로 재작성 된다. 따라서 복제코드의 토큰 나열은 $abccff$가 되며, 이 토큰 나열은 기존 소소코드의 토큰나열과 비교 엔진을 통하여 소스 토큰 나열 cf부분과 복제 토큰 나열 cf가 복제 된 코드로 검출되게 된다. 같은 방식으로 복제 코드 전체 나열에서 불필요한 코드라고 판명 된 토큰 나열 bc가 있다고 하자. 이 나열은 (ε, bc) 쌍으로 저장되고 복제 코드 전체 나열에서 오토마타로 bc를 찾아내고 그 위치에 ε이 재작성되게 된다. 즉 그 부분은 전체 토큰 나열에서 삭제되어지고 위에서의 전체 복제 토큰 나열은 $acff$가 되고 비교엔진은 bc 토큰 나열을 제외하고 유사도를 측정하게 된다. [그림 4]에서 볼 수 있듯, 이런 과정을 반복함으로써 유사도의 신뢰성을 높아지고 유사도를 점점 보정해 나갈 수 있게 된다.

3.3 장점 및 단점

본 논문에서 제안한 방식은 몇 가지 장점을 가지고 있다. 첫째는 기존의 모든 부분을 사용자가 검사하고 삭제했던 것과는 달리 단 한 번의 수정, 삭제 패턴을 찾아내면 모든 코드에



[그림 4] 통합된 경험기반 유사도 반자동 산정 도구

서 그 패턴을 찾아 수정, 삭제 할 수 있다. 둘째는 그림에서 알 수 있듯이 비교엔진에 주워지는 토큰열만 재작성 하므로 기존의 비교엔진을 그대로 유지하고 사용할 수 있다. 셋째는 어휘분석기 부분을 모듈화 하여 그것만 변경 해 주면 다른 여러 가지 언어에서도 사용 가능하다. 넷째는 연속적인 반복 수행이 가능하여 사용자가 요구하는 만큼의 정확도를 구할 수 있다. 그리고 오토마타의 합성이 가능하다는 특성을 이용해서 데이터베이스에 저장된 패턴형태들의 합성을 진행해 데이터베이스의 저장 용량을 자체적으로 줄이는 기대도 가능하다.

위에서 언급한 장점도 있지만 토큰열 재작성 비교법으로 인해 생기는 단점도 존재한다. 첫째로 비교엔진 사용 횟수가 늘어나서 많은 시간이 소요된다는 점이다. 토큰열 변환 오토마타의 특성상 최소 한번 이상의 비교엔진 사용을 하게 되고 코드의 크기가 크다면 그 시간은 부담스러울 수 있다. 둘째로 데이터베이스가 커지면 커질수록 재작성 도구의 수행 시간은 늘어난다는 점이다.

번호	파일	내용	토큰수
[1]	A'	정적라이브러리 포함	3
[2]	A'	소멸자	9
[3]	A'	선언	4
[4]	A'	메시지 처리 맵	39
[5]	B'	컬러 도구 설정	13
[6]	B'	페인팅 도구 사용 함수	34
[7]	C'	생성자	8
[8]	C'	응용프로그램 간 호출 함수 등록	96
[9]	D'	응용프로그램 간 메소드 호출	22
[10]	E'	대화상자	54
[11]	F'	클래스 간 고유번호 지정	31
합계	6개 파일		282

[표 I]육안 검사로 알아낸 보정 패턴쌍

파일 원본 - 복제	Sim 유사도	보정 토큰쌍 번호	재작성 횟수	재작성 된 토큰수	보정 된 유사도	실제 유사도
A - A'	90.90%	[1]	3	9	80.51%	60.20%
		[2]	1	9		
		[3]	5	20		
		[4]	1	39		
B - B'	41.30%	[1]	4	12	42.79%	44.30%
		[5]	5	65		
		[6]	3	102		
C - C'	2.14%	[1]	8	24	0%	0%
		[7]	2	18		
		[8]	1	96		
D - D'	82.13%	[1]	2	6	79.74%	31.0%
		[9]	4	88		
E - E'	22.64%	[1]	3	9	0%	0%
		[10]	1	54		
F - F'	0%	[1]	11	33	0%	0.4%
		[7]	1	8		
		[11]	3	93		
G - G'	0%	[1]	4	12	0%	5.6%
		[2]	1	9		
		[3]	12	48		
		[4]	1	39		
H - H'	5.18%	[1]	2	6	5.54%	8.6%
		[3]	10	40		
		[4]	1	39		
I - I'	9.35%	[1]	24	72	20.42%	14.9%
		[3]	5	20		
		[8]	1	96		
		[11]	1	31		
J - J'	0%	[1]	7	21	0%	9.1%
		[2]	1	9		
		[8]	3	288		
		[11]	1	31		
합계			122	1369		

[표 II]선정된 패턴쌍에 대한 보정 결과

4 실험 결과

제안하는 방법을 사용 한 테스트는 기존의 방식보다 향상된 결과를 도출할 수 있었다.테스트는 [표 I]의 C++ 코드를 대상으로 했으며, 특정 솔루션에서 10 개의 소스코드를 선정하였다. 비교엔진으로는 Sim[5]을 사용했다. Sim을 통해 유사도를 검증 후, 신뢰성을 떨어뜨리는 몇가지 경우를 아래 표와 같이 선정하였다.

선정된 보정 부분으로 재작성 오토마타를 사용하여 [표 II]와 같은 효율적인 결과를 도출할 수 있었다. Sim만으로 유사도를 구했을때보다 어느정도 보정이 된 결과이며 실제 육안검사를 통해 구한 완벽한 유사도에 근접함을 알 수 있다. 보정 부분에서의 토큰쌍 번호는 위에서 미리 선정 된 토큰쌍이고 재작성 횟수만큼 그 코드에서 재작성 되었음을 의미한다. 그리고 총 재작성 된 토큰수를 알 수 있다. 육안으로 선정된 부분은 총 11 쌍의 282 토큰이였고, 재작성 오토마타로 재작성 된 토큰은 총 122 쌍 1369 토큰임을 알 수 있다. 이말은 즉 282 토큰 부분을 육안으로 찾아내고 실제로 코드에서는 1369 토큰을 보정했다는 말이다. 보정에 의해서 유사도가 증가한경우는 불필요한 코드가 지워지거나 유사하지 않다고

검증된 부분이 유사하다고 재작성 된 경우이고, 유사도가 감소하는 경우는 유사하다고 검증 되었으나 재작성에 의해 삭제 된 경우이다. 이 결과를 통해 알 수 있는 점은, 본 논문에서 제안하는 도구로 완벽한 유사도를 구할 수는 없지만 보다 빠르고 효율적으로 완벽한 유사도에 근접해 갈 수 있다.

5 결론

본 논문에서는 기존의 비교엔진에 재작성 오토마타를 포함하여 반자동으로 유사도를 보정하는 방법에 대하여 설명하였다. 그리고 아래와 같은 잘못된 유사도 측정에 대하여 사용자의 경험과 판단이 유사도 비교엔진과의 상호작용을 통해 그 노력과 시간을 줄이고 유사도의 신뢰도를 높일 수 있음을 보여준다.

– 실제로는 상당히 유사하나 검출 알고리즘에 의해 검출되지 않는 경우.
– 불필요 하거나 유사도에 영향을 주어서는 안 되는 코드를 알고리즘이 검출한 경우.
– 불필요 한 부분이 유사도 선정에 계산되어 잘못된 유사도가 출력된 경우.

유사도 측정에 대한 많은 알고리즘들이 연구 되었고 발전되고 있다. 본 논문의 제안은 그런 유사도 측정에 대한 알고리즘의 신뢰도를 높이고 실제 사용에 있어서 편리를 더하기 위함이라 할 수 있다. 아직 많은 문제점을 내제하고 있지만, 실제 그 유용성은 높을 것이며 유사도 선정에 대한 신뢰도가 높아질 것으로 예상한다.

토큰열 변환 오토마타의 문제점을 해결하고 효율성을 높히기 위해 몇가지 보완해야 할 부분이 있다. 첫째는 복제 부분으로 측정 되었으나 실제로는 복제가 아닌 것에 대한 보완이 필요하다. 둘째는 단점에서 언급 된 반복되는 검증에 소요되는 시간을 단축 시킬 필요가 있다. 셋째로 오토마타 합성을 이용한 데이터베이스의 축소화이다. 마지막으로 편리한 인터페이스를 추가하여 사용자가 사용하기 편리하게 해야한다.

참고문헌

[1] Brenda Baker. On finding duplication and near-duplication in large software systems. In *Proceedings of the Second Working Conference on Reverse Engineering*, pages 86–95, Toronto, Ontario, Canada, July 1995.

[2] Ira Baxter, Andrew Yahin, Leonardo Moura, and Marcelo Sant Anna. Clone detection using abstract syntax trees. In *Proceedings of the 14th International Conference on Software Maintenance*, pages 368–377, Bethesda, Maryland, November 1998.

[3] Stéphane Ducasse, Oscar Nierstrasz, and Matthias Rieger. On the effectiveness of clone detection by string matching. *International Journal on Software Maintenance and Evolution: Research and Practice*, 18(1):37–58, January 2006.

[4] Williams Evans and Christopher Fraser. Clone detection via structural abstraction. In *Proceedings of the 14th Conference on Reverse Engineering*, Vancouver, BC, Canada, October 2007.

[5] David Gitchell and Nicholas Tran. Sim: a utility for detecting similarity in computer programs. *ACM SIGCSE Bulletin*, 31(1):266–270, March 1999.

[6] John Johnson. Substring matching for clone detection and change tracking. In *Proceedings of the 10th International Conference on Software Maintenance*, pages 120–126, Victoria, British Columbia, Canada, September 1999.

[7] Toshihiro Kamiya, Shinji Kusumoto, and Katsuro Inoue. CCFinder: A multilinguistic token-based code clone detection system for large scale source code. *IEEE Transactions on Software Engineering*, 28(7):654–670, July 2002.

[8] Raghavan Komondoor and Susan Horwitz. Using slicing to identify duplication in source code. In *Proceedings of the 8th International Symposium on Static Analysis*, volume 2126 of *Lecture Notes in Computer Science*, pages 40–56, Paris, France, July 2001.

[9] K. Kontogiannis, R. DeMori, E. Merlo, M. Galler, and M.Bernstein. Pattern matching for clone and concept detection. *Automated Software Engineering*, 3(1–2):77–108, June 1996.

[10] Jens Krinke. Identifying similar code with program dependence graphs. In *Proceedings of the 8th Working Conference on Reverse Engineering*, pages 301–309, Stuttgart, Germany, October 2001.

[11] Filippo Lanubile and Teresa Mallardo. Finding function clones in web applications. In *Proceedings of the 7th European Conference on Software Maintenance and Reengineering*, pages 379–386, Benevento, Italy, March 2003.

[12] A. M. Leitão. Detection of redundant code using R^2D^2. *Software Quality Control*, 12(4):361–382, 2004.

[13] Zhenmin Li, Shan Lu, Suvda Myagmar, and Yuanyuan Zhou. CP-Miner: Finding copy-paste and related bugs in large-scale software code. *IEEE Transactions on Software Engineering*, 32(3):176–192, March 2006.

[14] Chao Liu, Chen Chen, Jiawei Han, and Philip S. Yu. GPLAG: Detection of software plagiarism by program dependence graph analysis. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 872–881, Philadelphia, USA, August 2006.

[15] G. A. Di Lucca, M. Di Penta, and A.R. Fasolino. An approach to identify duplicated web pages. In *Proceedings of the 26th International Computer Software and Applications Conference*, pages 481–486, Oxford, England, August 2002.

[16] Andrian Marcus and Jonathan I. Maletic. Identification of high-level concept clones in source code. In *Proceedings of the 16th IEEE International Conference on Automated Software Engineering*, pages 107–114, San Diego, CA, USA, November 2001.

[17] Jean Mayrand, Claude Leblanc, and Ettore Merlo. Experiment on the automatic detection of function clones in a software system using metrics. In *Proceedings of the 12th International Conference on Software Maintenance*, pages 244–253, Monterey, CA, USA, November 1996.

[18] Chanchal Kumar Roy and James R. Cordy. A survey on software clone detection research. Technical Report 2007-541, School of Computing, Queen's University, Canada, September 2007.

[19] V. Wahler, D. Seipel, Jurgen Wolff von Gudenberg, and G. Fischer. Clone detection in source code by frequent itemset techniques. In *Proceedings of the 4th IEEE International Workshop Source Code Analysis and Manipulation*, pages 128–135, Chicago, IL, USA, September 2004.

[20] Wuu Yang. Identifying syntactic differences between two programs. *Software Practice and Experience*, 21(7):739–755, July 1991.

안 효 천

– 2007- 한양대학교 컴퓨터공학과 (석사과정)
– 2000-2007 충주대학교 컴퓨터공학과 졸업 (학사)

<관심분야> 프로그램 분석,프로그램 복제 코드 검증

이 욱 세

– 2004- 한양대학교 컴퓨터공학과 교수
– 1997-2003 KAIST 전산학과 (박사)
– 1995-1997 KAIST 전산학과 (석사)
– 1991-1995 KAIST 전산학과 (학사)

<관심분야> 프로그램 분석, 포인터 분석, 프로그램 검증, 타입 시스템

Yan Bin Tang

– 2007- 한양대학교 컴퓨터공학과 (석사과정)
– 2001-2005 Tianjin University of Technology (학사)

<관심분야> 프로그래밍 분석