프로그램분석.
SW오류검증.
산과골.
미개한.

이 광근

08/29/2011 @ 학부교수 점심발표

문제

작성한 SW의 오류를 자동으로 미리 모두 찾아주거나,
없으면 없다고 확인해주는 기술들은 있는가?

그래서, SW의 오류때문에 발생하는
개인/기업/국가/사회적 비용을
절감시켜주는 기술들은 있는가?

목표

"소프트웨어 MRI" "소프트웨어 fMRI" "소프트웨어 PET"

기반 기술

정적 프로그램 분석(static program analysis)

프로그램의 실행 성질을

실행전에 자동으로

안전하게 어림잡는

일반적인 방법

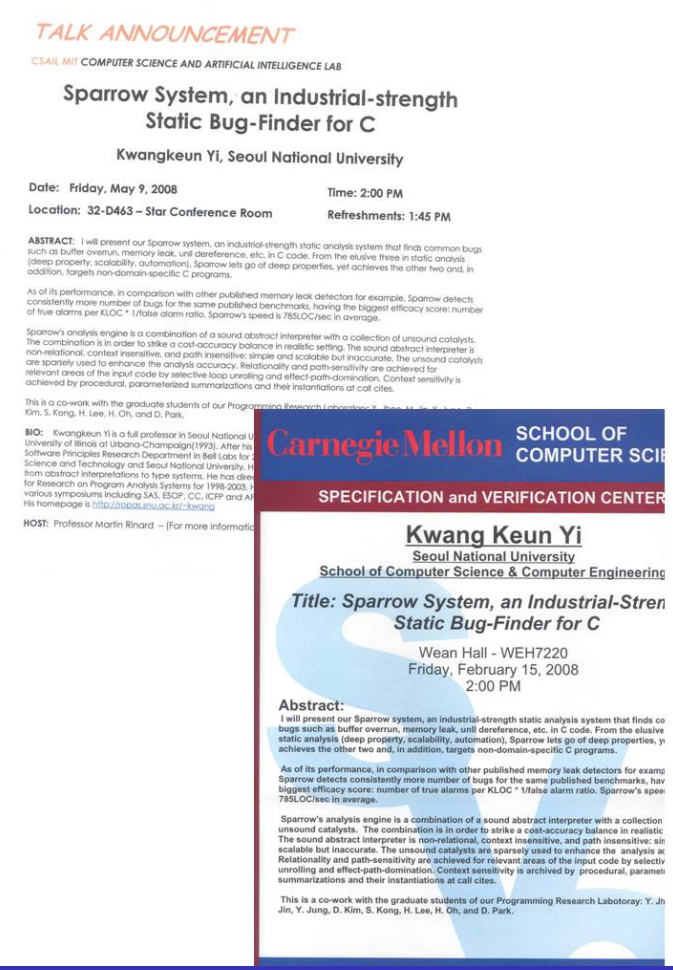
- 엄밀히 예측
- 테스팅의 단점을 보완
- 실용성 확산 (국내외 성과)
  - 우리 예) 산업화한 SPARROW

기반기술 실용성 예시: SPARROW

대상: C, memory leak/buffer overrun/etc, 오류 검출률 6/KLOC, 속도 100Loc/sec

Memory leak detection (SPEC2000 and open sources) (as of 01/04/2008)

Programs	Size KLOC	Time (sec)	True Alarms	False Alarms
art	1.2	0.68	1	0
equake	1.5	1.03	0	0
mcf	1.9	2.77	0	0
bzip2	4.6	1.52	1	0
gzip	7.7	1.56	1	4
parser	10.9	15.93	0	0
ammp	13.2	9.68	20	0
vpr	16.9	7.85	0	9
crafty	19.4	84.32	0	0
twolf	19.7	68.80	5	0
mesa	50.2	43.15	9	0
vortex	52.6	34.79	0	1
gap	59.4	31.03	0	0
gcc	205.8	1330.33	44	1
gnuchess-5.07	17.8	9.44	4	0
tcl8.4.14	17.9	266.09	4	4
hanterm-3.1.6	25.6	13.66	0	0
sed-4.0.8	26.8	13.68	29	31
tar-1.13	28.3	13.88	5	3
grep-2.5.1a	31.5	22.19	2	3
openssh-3.5p1	36.7	10.75	18	4
bison-2.3	48.4	48.60	4	1
openssh-4.3p2	77.3	177.31	1	7
fftw-3.1.2	184.0	15.20	0	0
httpd-2.2.2	316.4	102.72	6	1
net-snmp-5.4	358.0	201.49	40	20
binutils-2.13.1	909.4	712.09	228	25


TALK ANNOUNCEMENT

CSAIL MIT COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE LAB

Sparrow System, an Industrial-strength Static Bug-Finder for C

Kwangkeun Yi, Seoul National University

Date: Friday, May 9, 2008 Time: 2:00 PM

Location: 32-D463 – Star Conference Room Refreshments: 1:45 PM

ABSTRACT: I will present our Sparrow system, an industrial-strength static analysis system that finds common bugs such as buffer overrun, memory leak, unll dereference, etc. in C code. From the elusive three in static analysis (deep property, scalability, automation), Sparrow lets go of deep properties, yet achieves the other two and, in addition, targets non-domain-specific C programs.

As of its performance, in comparison with other published memory leak detectors for example, Sparrow detects consistently more number of bugs for the same published benchmarks, having the biggest efficacy score: number of true alarms per KLOC * 1/false alarm ratio. Sparrow's speed is 785LOC/sec in average.

Sparrow's analysis engine is a combination of a sound abstract interpreter with a collection of unsound catalysts. The combination is in order to strike a cost-accuracy balance in realistic setting. The sound abstract interpreter is non-relational, context insensitive, and path insensitive; simple and scalable but inaccurate. The unsound catalysts are sparsely used to enhance the analysis accuracy. Relationality and path-sensitivity are achieved for relevant areas of the input code by selective loop unrolling and effect-path-domination. Context sensitivity is achieved by procedural, parameterized summarizations and their instantiations at call cites.

This is a co-work with the graduate students of our Programming Research Laboratory: Y. Jh Kim, S. Kong, H. Lee, H. Oh, and D. Park.

BIO: Kwangkeun Yi is a full professor in Seoul National U University of Illinois at Urbana-Champaign(1993). After his Software Principles Research Department in Bell Labs for Science and Technology and Seoul National University. H from abstract interpretations to type systems. He has direc for Research on Program Analysis Systems for 1998-2003. H various symposiums including SAS, ESOP, CC, ICFP and AP His homepage is http://ropas.snu.ac.kr/~kwang

HOST: Professor Martin Rinard – (For more informatio

Carnegie Mellon SCHOOL OF COMPUTER SCIE

SPECIFICATION and VERIFICATION CENTER

Kwang Keun Yi

Seoul National University

School of Computer Science & Computer Engineering

Title: Sparrow System, an Industrial-Stren Static Bug-Finder for C

Wean Hall - WEH7220

Friday, February 15, 2008

2:00 PM

Abstract:

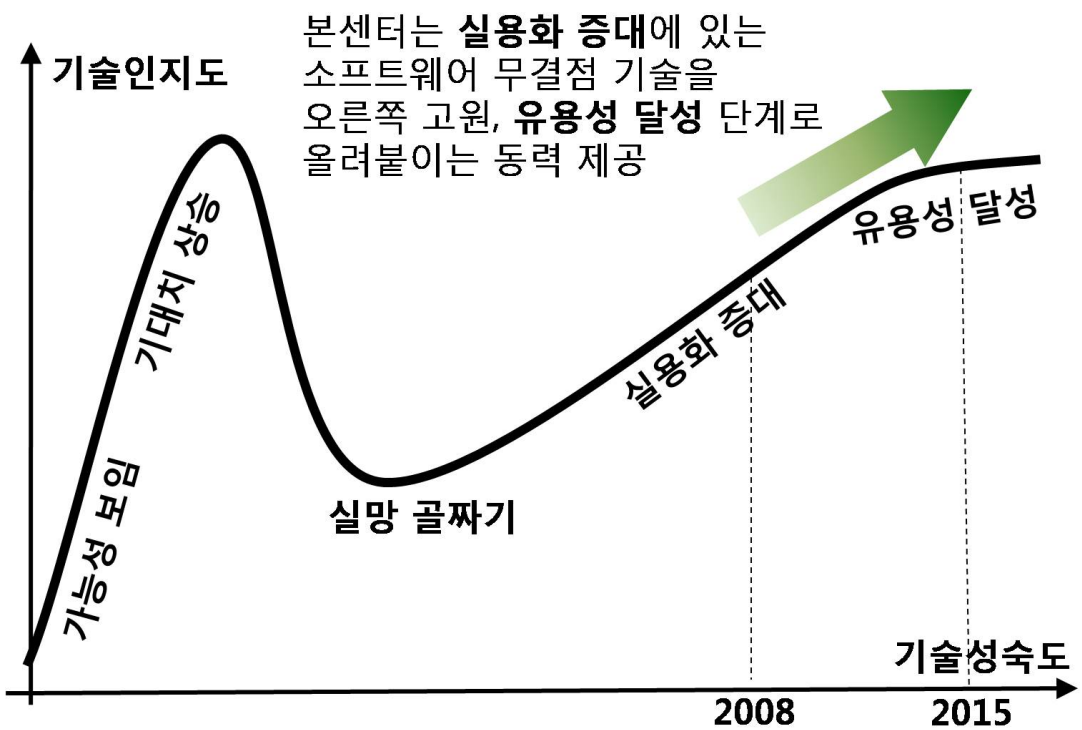
I will present our Sparrow system, an industrial-strength static analysis system that finds co bugs such as buffer overrun, memory leak, unll dereference, etc. in C code. From the elusive static analysis (deep property, scalability, automation), Sparrow lets go of deep properties, y achieves the other two and, in addition, targets non-domain-specific C programs.

As of its performance, in comparison with other published memory leak detectors for examp Sparrow detects consistently more number of bugs for the same published benchmarks, hav biggest efficacy score: number of true alarms per KLOC * 1/false alarm ratio. Sparrow's spee 785LOC/sec in average.

Sparrow's analysis engine is a combination of a sound abstract interpreter with a collection unsound catalysts. The combination is in order to strike a cost-accuracy balance in realistic The sound abstract interpreter is non-relational, context insensitive, and path insensitive: sin scalable but inaccurate. The unsound catalysts are sparsely used to enhance the analysis ac Relationality and path-sensitivity are achieved for relevant areas of the input code by selectiv unrolling and effect-path-domination. Context sensitivity is archived by procedural, paramet summarizations and their instantiations at call cites.

This is a co-work with the graduate students of our Programming Research Labotoray: Y. Jh Jin, Y. Jung, D. Kim, S. Kong, H. Lee, H. Oh, and D. Park.

기술 위치


본센터는 실용화 증대에 있는
소프트웨어 무결점 기술을
오른쪽 고원, 유용성 달성 단계로
올려붙이는 동력 제공
기술인지도
가능성 보임
기대치 상승
실망 골짜기
실용화 증대
유용성 달성
기술성숙도
2008
2015

오늘 내용

- 구체적인 예: SPARROW
- 기술
- 한계
- 진행연구

Sparrow (as of 2008)

- static analyzer
- *fully automatic* and *scalable*
- detects *common* bugs
  - buffer overrun, memory leak, null dereference, uninitialized access, divide by zero, etc.
- for *non domain-specific* C code

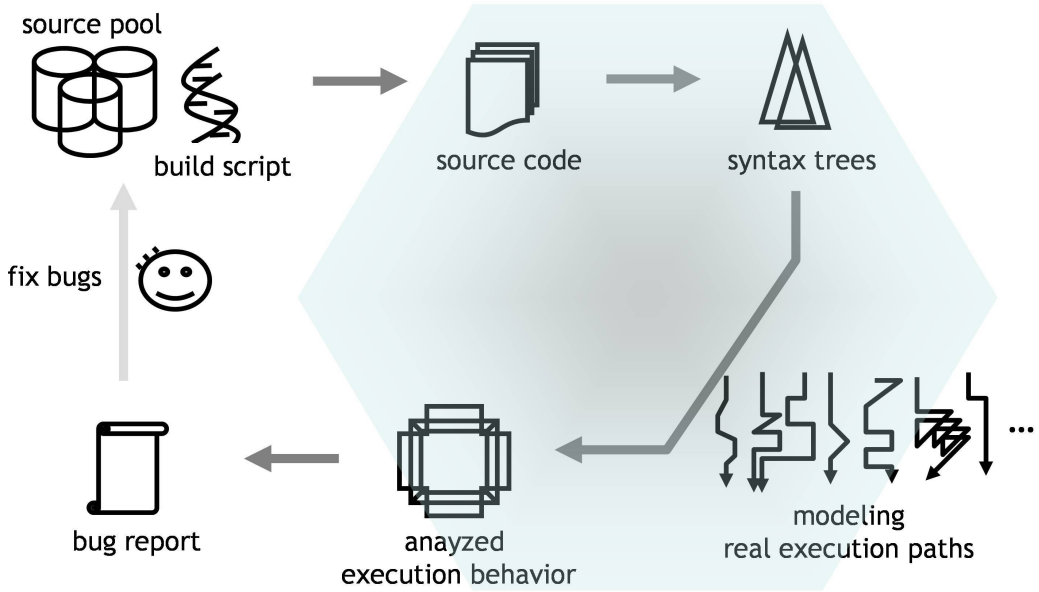
From the elusive three (deep property, scalability, automation)

– deep property

\+ domain-independence

Steps of SPARROW

One-button solution with four steps:



Our motivation (as of 2004)

- to prove that static analysis is "useful in real world"
- curious about "extra miles" from academia to industry

Of course, the reality

- has been challenging us a lot, and
- we've been struggling to respond to.

SPARROW-detected Overrun Errors (1/3)

- in Linux Kernel 2.6.4

```
625       for (minor = 0; minor < 32 && acm_table[minor]; minor++);
...       ...
713       acm_table[minor] = acm;
```

- in a proprietary code

```
if (length >= NET_MAX_LEN)
   return API_SET_ERR_NET_INVALID_LENGTH;
...
buff[length] |= (num << 4);
```

- in a proprietary code

```
   index = memmgr_get_bucket_index(block_size);
   ...
   mem_stats.pool_ptr[index] = prt
```

- in a proprietary code

```
imi_send_to_daemon(PM_EAP, CONFIG_MODE, set_str, sizeof(set_str));
...
imi_send_to_daemon(int module, int mode, char *cmd, int len)
{
...
        strncpy(cmd, reply.str, len);
        cmd[len] = 0;
```

SPARROW-detected Leak Errors (2/3)

- in sed-4.0.8/regexp_internal.c

```
948:   new_nexts = re_realloc (dfa->nexts, int, dfa->nodes_alloc);
949:   new_indices = re_realloc (dfa->org_indices, int, dfa->nodes_alloc);
950:   new_edests = re_realloc (dfa->edests, re_node_set, dfa->nodes_alloc);
951:   new_eclosures = re_realloc (dfa->eclosures, re_node_set,
952:       dfa->nodes_alloc);
953:   new_inveclosures = re_realloc (dfa->inveclosures, re_node_set,
954:  dfa->nodes_alloc);
955:   if (BE (new_nexts == NULL || new_indices == NULL
956:   || new_edests == NULL || new_eclosures == NULL
957:   || new_inveclosures == NULL, 0))
958:     return -1;
```

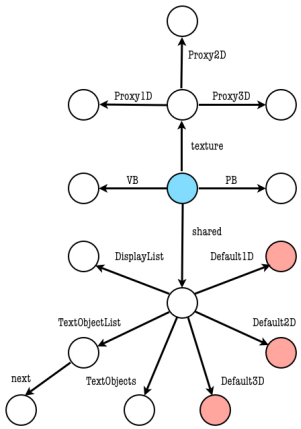
- in proprietary code

```
line = read_config_read_data(ASN_INTEGER, line,
                             &StorageTmp->traceRouteProbeHistoryHAddrType,
                             &tmpint);
...
line = read_config_read_data(ASN_OCTET_STR, line,
                             &StorageTmp->traceRouteProbeHistoryHAddr,
                             &StorageTmp->traceRouteProbeHistoryHAddrLen);
...
if (StorageTmp->traceRouteProbeHistoryHAddr == NULL) {
     config_perror
         (``invalid specification for traceRouteProbeHistoryHAddr'');
     return SNMPERR_GENERR;
}
```

SPARROW-detected Leak Errors (3/3)

- in mesa/osmesa.c(in SPEC 2000)

```
276:   osmesa->gl_ctx = gl_create_context( osmesa->gl_visual );
...
287:     gl_destroy_context( osmesa->gl_ctx );
------------------
1164: GLcontext *gl_create_context( GLvisual *visual,
                                    GLcontext *share_list,
                                    void *driver_ctx ) {
...
1183:  ctx = (GLcontext *) calloc( 1, sizeof(GLc
...
1211:     ctx->Shared = alloc_shared_state();
---------------------
476: static struct gl_shared_state *alloc_shared
477: {
...
489: ss->Default1D = gl_alloc_texture_object(ss,
490: ss->Default2D = gl_alloc_texture_object(ss, 0, 2);
491: ss->Default3D = gl_alloc_texture_object(ss, 0, 3);
----------------------
1257: void gl_destroy_context( GLcontext *ctx )
1258: {
      ...
```



기술 소개

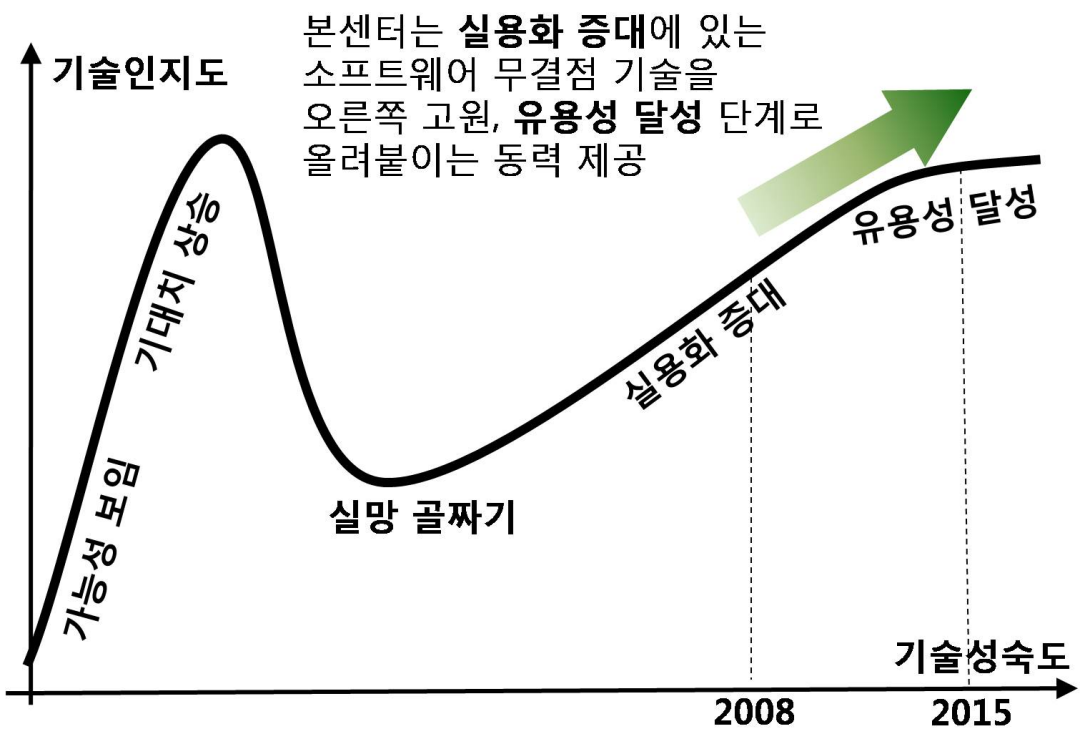
정적 프로그램 분석(static program analysis)

프로그램의 실행 성질을
실행전에 자동으로
안전하게 어림잡는
일반적인 방법

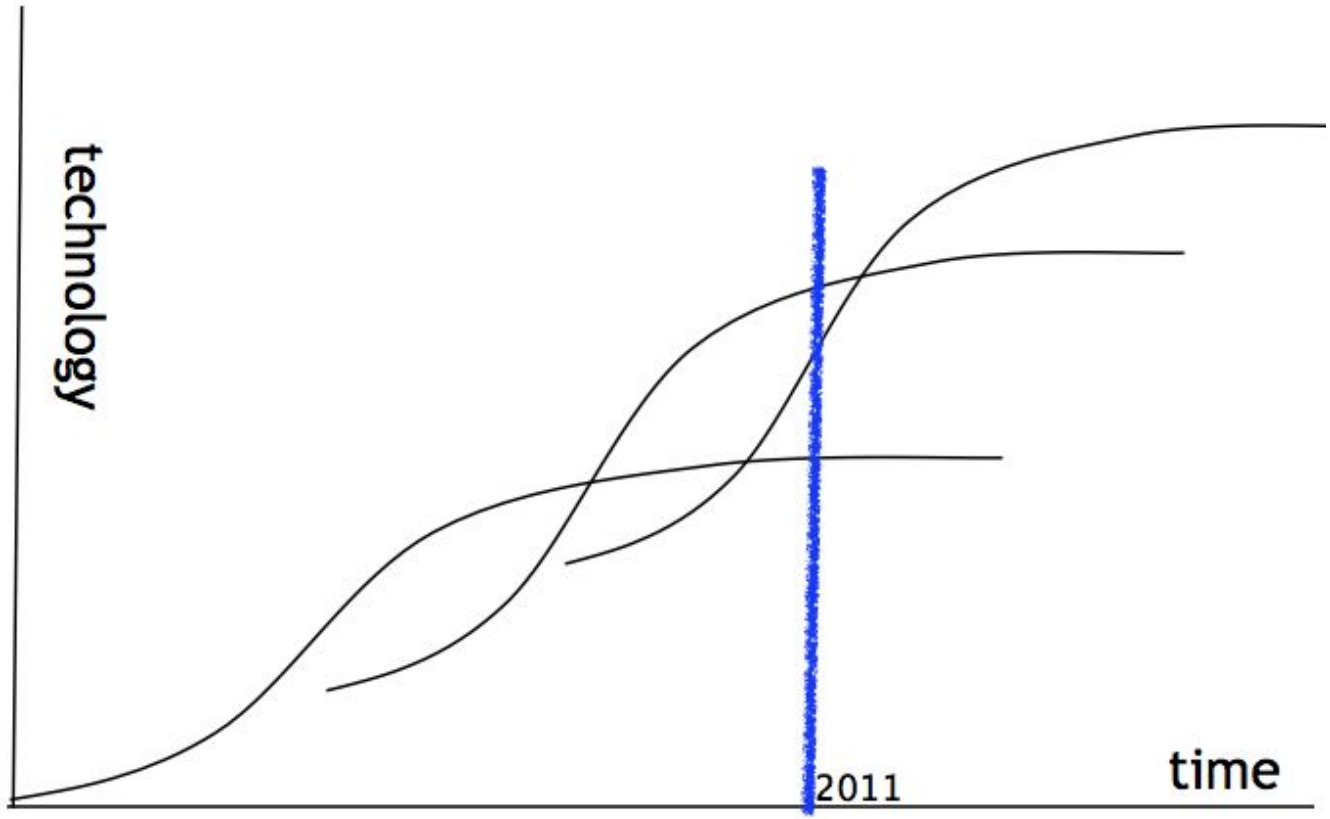
- 응용: sw 오류검증, sw 관리, sw 테스트, sw 최적화, 등등
- 다양한 레벨/목적/이름으로 존재:

theory	"요약해석 abstract interpretation"
pl, se, veri.	"type system", "model checking", "theorem proving"
cmplr	"data-flow analysis", etc.

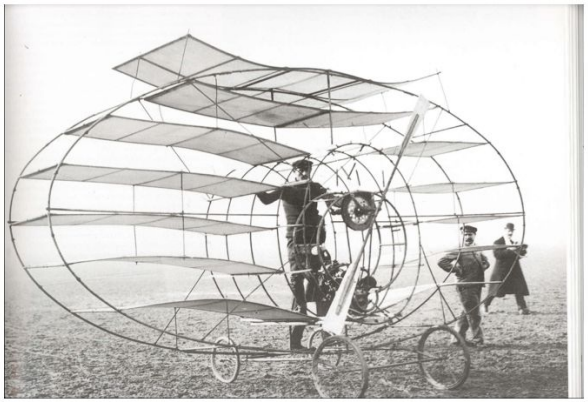
기술 위치 (우물안)

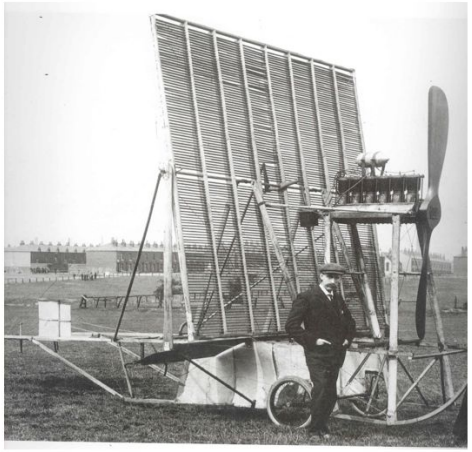

본센터는 실용화 증대에 있는
소프트웨어 무결점 기술을
오른쪽 고원, 유용성 달성 단계로
올려붙이는 동력 제공
기술인지도
가능성 보임
기대치 상승
실망 골짜기
실용화 증대
유용성 달성
기술성숙도
2008
2015

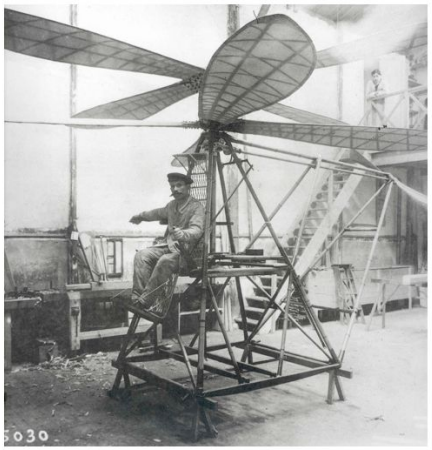
기술 단계 (우물밖)



현 수준







아직은 아닌

(여담) 정적분석은 가까이에 늘

- 대학입학시험
- 궁합
- 점, 손금, 관상
- 나이트클럽 기도

무이론 + 대충. 틀리는 것이 문제.

다른 공학분야의 정적분석

- 기계설계 분석검증 후 제작
- 전기설계 분석검증 후 제작
- 공정설계 분석검증 후 설비
- 건축설계 분석검증 후 건설

정적 프로그램 분석

- "실행전": 프로그램을 실행시키지 않고
- "자동으로": 프로그램이 프로그램을 분석
- "안전하게": 모든 가능성을 포섭
- "어림잡는": 실제 이외의 것들이 포함됨
  - 어림잡지 않으면 불가능
- "일반적": 소스언어와 성질에 무제한
  - C, Java, ML, Bluespec, Dalvik, x86, binary, etc.
  - "buffer overrun?", "memory leak?", "unhandled exn?" "x=y at line 2?", "memory use $\leq$ 2K?", "terminate?", "race?" etc.
- "One-on-One": 분석기1개당, 1언어 1성질.

정적 프로그램 분석 비유

$128 \times 22 + (1920 \times -10) + 4$는 어떤 값을 계산합니까?

- 정적 분석: "정수입니다."
- 정적 분석: "짝수입니다."
- 정적 분석: "$-10,0000$과 $1,0000$ 사이의 수입니다."
- 정적 분석: "음수입니다."

정적 프로그램 분석 예

```
x = readInt;
while (x ≤ 99)
    x++;
```

실행중/후에 변수 x가 가질 값은?

정적 분석기 개발 싸이클

1. 분석할 소스 언어 결정:
   - C, Java, ML, Bluespec, Dalvik, JVM, x86, binary
2. 분석할 실행 성질 결정:
   - buffer overrun? memory leak? race? uncaught exn? time/power/memory/wire consumption? equivalent? terminate? etc.
3. (디자인; 구현; 테스트)$^+$

모든 정적 분석은 3스텝

1. "연립방정식"을 세운다
   - 요약된 세계에서 (abstract semantic domains)
   - 프로그램의 실행 다이나믹스에 관한 (abstract execution flows)
2. 그 방정식을 푼다
3. 그 해를 가지고 결론을 내린다
   - 있는가 없는가? 같은가 다른가?

프로그램 분석 이론

PL 이론과 어휘를 사용

- 올바른 연립방정식을 어떻게 유도하는가?
  - 방정식이 실제 실행의 모든 것을 포착하는가?
- 유도한 연립방정식의 해는 항상 있는가?
  - 그 해를 어떻게 계산하는가?
  - 유한시간내에 해를 구할 수 있는가?

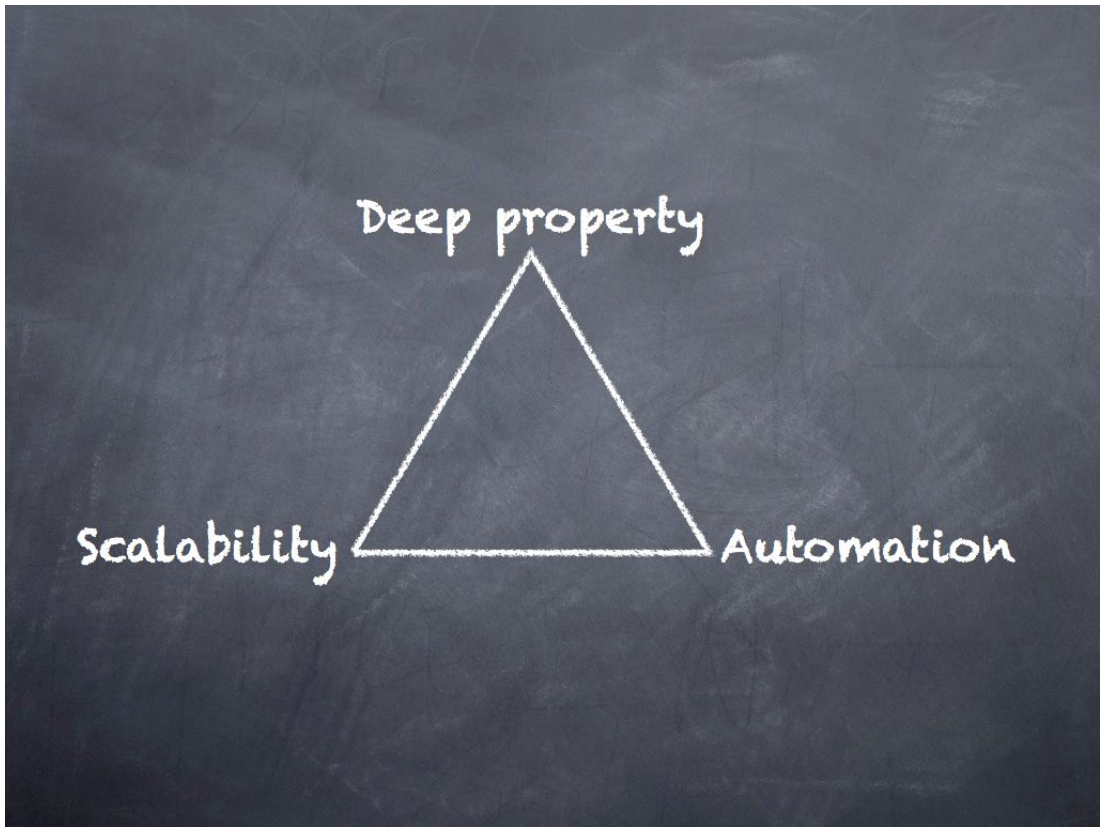
다른 분야와 다르지 않은

프로그램	$\Longleftrightarrow$	기계디자인
실행 = 컴퓨터 (언어정의)	$\Longleftrightarrow$	작동 = 자연 (자연법칙)
실행에대한 방정식	$\Longleftrightarrow$	작동에대한 방정식
방정식 풀기	$\Longleftrightarrow$	방정식 풀기
“생각대로 돌것이다”	$\Longleftrightarrow$	“생각대로 작동할것이다”
“무인비행기에 심자”	$\Longleftrightarrow$	“만들어 팔자”
PL & Logic 이론	$\Longleftrightarrow$	물리-화학법칙, XX방정식

정적 프로그램 분석 기술의 한계 I (SW 오류 분석에서)

마의 삼각형

단 둘 만 가능



정적 프로그램 분석 기술의 한계 II (SW 오류 분석에서)

허위 경보(false alarm)

- 오류가 아닌 것을 오류라고 판별
- 이론적으로 불가능: 허위경보가 항상 0

우리가보는 틈새 (SW 오류 분석에서)

오류 검출과 무결점 검증

- 오류 검출(bug-finding)
  - 허위경보가 항상 적음 ($\leq 20\%$, non domain-specific)
  - 오류를 모두 찾지 못함
- 무결점 검증(verification)
  - 허위경보가 거의 0
  - 오류를 모두 찾는 것이 보장
  - 특정 SW에 대해서만 (domain-specific)

오류 검출기(bug-finder)

있는 오류 대부분을 검출

- not exhaustive
- a few false-alarm
- domain-independent

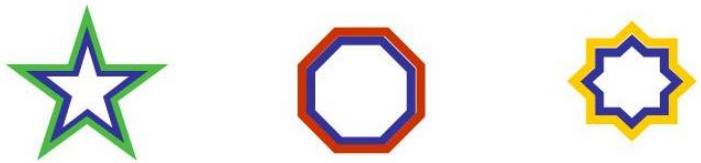
허위 경보율 $\leq 20\%$

무결점 검증기(verifier)

오류가 없으면 없다고 확인

- exhaustive
- zero false-alarm
- domain-specific

허위 경보율 $\leq 1\%$



정적 분석 기술 제품상황

전세계적으로

- 무결점 검증기(verification-level analyzer) 개발 업체는 아직.
- 오류 검출기(bug-finder)들이 시장에 있슴
  - 경쟁 잣대: 오류검출력 vs 허위경보율 vs "센스있는"오류들 vs UI+UX

진행 연구들

- 이론
  - "multi-staged programming language" (e.g. web pgm): 분석 어떻게?
  - "implicit programming" (e.g. Haskell, Scala, C++ Concept): 분석 어떻게?
- 실제
  - Sparrow 개선: more scalable, alarm clustering, etc.
  - Android app 분석기: privacy leak detector
  - semantic clone detector
  - binary malware detector
  - Bluespec analyzers
  - Hypervisor Xen core verification by Coq
- 논문들: POPL'06, POPL'11, ICSE'11, VMCAI'11, TACAS'11, TOPLAS, SPE, TCS, Acta Info. etc.

SW Clinic Service

- 대상: C 프로그램 실습 학생, C 프로그램 개발자 등
- `http://rosaec.snu.ac.kr/clinic`