# Collage of Static Analysis

Kwangkeun Yi

Seoul National University, Korea
http://ropas.snu.ac.kr/~kwang

2/26/2012 – 3/2/2012
17th Estonian Winter School in Computer Science, Palmse, Estonia

# About Me/Us

ROSAEC center

Research On Software Analysis for Error-free Computing

소프트웨어 무결점 연구센터 KOSEF ERC

We research to reduce/eliminate errors in software.

- statically: before execution, before sell/embed
- automatically: against explosive sw size
- to find bugs or verify their absence

# Our Activities

- Research areas: *static analysis, abstract interpretation, programming languge theory, type system, theorem proving, model checking, & whatever relevant*
- We published our works in:
  - POPL('11, '06), PLDI('12, '12), ESOP('12,'06), ICSE('11), TACAS('11), VMCAI('12, '11, '10), SAS, ISMM, OOPSLA, FSE, etc.
  - TOPLAS, TCS, JFP, SP&E, Acta Informatica, etc.
- Industrialization:

Sparrow
The Early Bird

Collage of Static Analysis

- 0.5hr: Static Analysis Overview
- 1.5hr: Static Analysis Design Framework
- 1.0hr: Static Analysis Engineering Framework
- 1.0hr: Static Analysis of Multi-Staged Programs

# Static Analysis Overview

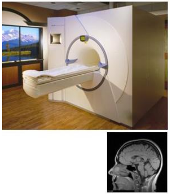
# One Challenge in SW

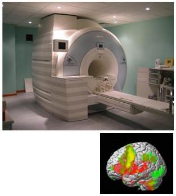
- How can we predict what our sw's do?
- Can it be done automatically?

One approach to respond: static analysis
Impossible! Approximate, usefully.

# Goals in Static Analysis

“SW MRI” “SW fMRI” “SW PET”





Or, more like “DNA diagnosis”.

# Static Analysis in Reality (1/3)

target: full C, memory leak + buffer overrun errors

detection rate: 6/KLOC, speed: 100Loc/sec, industrialized (as of 2008)

Sparrow The Early Bird

Bug-finder class: unsound, non-global

Memory leak detection

| Programs | Size KLOC | Time (sec) | True Alarms | False Alarms |
|---|---|---|---|---|
| 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 |

# Static Analysis in Reality (2/3)

target: full C, memory leak + buffer overrun errors
detection rate: 6/KLOC, speed: 100Loc/sec, industrialized (as of 2008)

Sparrow The Early Bird

Bug-finder class: unsound, non-global
Buffer overrun detection

| Programs | Size KLOC | Time (sec) | True Alarms | False Alarms |
|---|---|---|---|---|
| gzip-1.2.4 | 9.1 | 8.55 | 0 | 17 |
| gnuchess-5.07 | 17.8 | 179.58 | 1 | 8 |
| tcl8.4.14/unix | 17.9 | 585.99 | 1 | 14 |
| hanterm-3.1.6 | 25.6 | 52.25 | 34 | 1 |
| sed-4.0.8 | 26.8 | 49.34 | 2 | 11 |
| tar-1.13 | 28.3 | 57.98 | 1 | 10 |
| grep-2.5.1a | 31.5 | 47.26 | 0 | 1 |
| bison-2.3 | 48.4 | 281.84 | 0 | 18 |
| openssh-4.3p2 | 77.3 | 97.69 | 0 | 9 |
| fftw-3.1.2 | 184.0 | 102.17 | 9 | 4 |
| httpd-2.2.2 | 316.4 | 265.43 | 10 | 33 |
| net-snmp-5.4 | 358.0 | 899.73 | 3 | 36 |

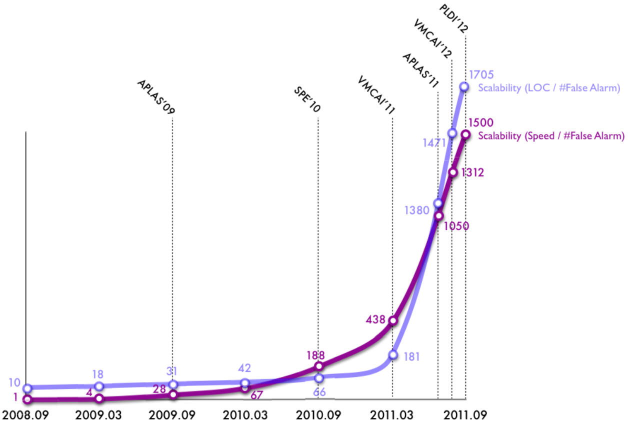
Sound, "semantically-deep", global analyzer for 1MLoC C program: a scalablility barrier knocked down (as of 2011)

Sound-&-global analyzer class: Sparrow The Early Bird

| Program | LOC | Baseline | | Localize | | Spd↑ | Mem↓ |
|---|---|---|---|---|---|---|---|
| | | Time | Mem | Time | Mem | | |
| gzip-1.2.4a | 7 K | 772 | 240 | 3 | 63 | 257 x | 74 % |
| bc-1.06 | 13 K | 1,270 | 276 | 7 | 75 | 181 x | 73 % |
| less-382 | 23 K | 9,561 | 1,113 | 33 | 127 | 289 x | 86 % |
| make-3.76.1 | 27 K | 24,240 | 1,391 | 21 | 114 | 1,154 x | 92 % |
| wget-1.9 | 35 K | 44,092 | 2,546 | 11 | 85 | 4,008 x | 97 % |
| a2ps-4.14 | 64 K | ∞ | N/A | 40 | 353 | N/A | N/A |
| sendmail-8.13.6 | 130 K | ∞ | N/A | 744 | 678 | N/A | N/A |
| nethack-3.3.0 | 211 K | ∞ | N/A | 16,373 | 5,298 | N/A | N/A |
| emacs-22.1 | 399 K | ∞ | N/A | 37,830 | 7,795 | N/A | N/A |
| python-2.5.1 | 435 K | ∞ | N/A | 11,039 | 5,535 | N/A | N/A |
| linux-3.0 | 710 K | ∞ | N/A | 33,618 | 20,529 | N/A | N/A |
| gimp-2.6 | 959 K | ∞ | N/A | 3,874 | 3,602 | N/A | N/A |
| ghostscript-9.00 | 1,363 K | ∞ | N/A | 14,814 | 6,384 | N/A | N/A |

# Static Analysis Scalability Improvement

Sound-&-global analyzer class



# 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`

```
  new_nexts = re_realloc (dfa->nexts, int, dfa->nodes_alloc);
  new_indices = re_realloc (dfa->org_indices, int, dfa->nodes_alloc);
  new_edests = re_realloc (dfa->edests, re_node_set, dfa->nodes_alloc);
  new_eclosures = re_realloc (dfa->eclosures, re_node_set,
      dfa->nodes_alloc);
  new_inveclosures = re_realloc (dfa->inveclosures, re_node_set,
 dfa->nodes_alloc);
  if (BE (new_nexts == NULL || new_indices == NULL
  || new_edests == NULL || new_eclosures == NULL
  || new_inveclosures == NULL, 0))
    return -1;
```

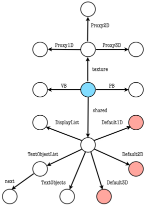
- in proprietary code

```
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,
491: ss->Default3D = gl_alloc_texture_object(ss, 0, 3);
----------------------
1257: void gl_destroy_context( GLcontext *ctx )
1258: {
      ...
1274: free_shared_state( ctx, ctx->Shared );
```



# Static Analysis

A general method for
automatic and sound approximations of
sw run-time behaviors
before the execution.

- applications: sw bug-finding, sw verification, sw optimization, sw management, etc $\infty$
- under many names:

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

# Static Analysis

A general method for
automatic and sound approximation of
sw run-time behaviors
before the execution.

- "before": statically, without running sw
- "automatic": sw analyzes sw
- "sound": all possibilities into account
- "approximation": cannot be exact
  - undecidable without approximation
- "general": for any source language and property
  - C, C++, C#, F#, Java, ML, Scala, Python, JVM, Dalvik, x86, etc.
  - "buffer overrun?", "memory leak?", "x=y at line 2?", "memory use $\leq$ 2K?", etc.
- One-on-one: 1 analyzer per language and property

# Static Analysis Analogy

$$128 \times 22 + (1920 \times -10) + 4$$

What value will it compute?

- static analysis: "an integer"
- static analysis: "an even number"
- static analysis: "a number in $[-100000, 10000]$"

```
x := 1; repeat x := x + 2 until readBool;
```

What value will `x` have?

- static analysis: "an integer"
- static analysis: "a positive integer"
- static analysis: "an odd number"

# Static Analysis Development Cycle

- Determine the target source:
  - Java, JavaScript, C, C#, ML, Haskell, F#, Scala, Python, Dalvik, binary?
- Determine the property to analyze:
  - buffer overrun, null dereference, constant, class, deadlock, race, uncaught exception, unhandled case, etc.
- $(\text{Design; Implementation; Test})^{+}$

# Every Static Analysis Has 3 Steps

- Set up equations about execution dynamics
  - in abstract semantic domains
  - about abstract execution dynamics
- Solve the equations
- Extract information from the solution
  - software errors, redundancies, parallelism, resource consumption, etc.

# Static Analysis Example

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

What values does the x variable have during the execution?

# Static Analysis Example: Equations

```
        x = readInt;
1:
        while (x ≤ 99)
2:
            x++;
3:
        end
4:
```

Capture the dynamics(semantics) by equations:

$$
\begin{array}{rcl}
x_1 & = & [-\infty, +\infty] \ or \ x_3 \\
x_2 & = & x_1 \ and \ [-\infty, 99] \\
x_3 & = & x_2 \dot{+} 1 \\
x_4 & = & x_1 \ and \ [100, +\infty]
\end{array}
$$

# Analogous to Other Disciplines

| Software | | Mechanical Engineering |
|---|---|---|
| program | $\Longleftrightarrow$ | machine design |
| run by computer | $\Longleftrightarrow$ | run by nature |
| equations of executions | $\Longleftrightarrow$ | equations of executions |
| solving the equations | $\Longleftrightarrow$ | solving the equations |
| "will run as we expected" | $\Longleftrightarrow$ | "will move as we expected" |
| "embed in devices" | $\Longleftrightarrow$ | "build the machine" |
| PL & Logic | $\Longleftrightarrow$ | Physics, Chemistry, & XX-equations |

# Need of Static Analysis Design Theory

- How can we derive correct equations from program text?
  - Does the equations capture all the execution dynamics?
  - Non-obvious: pointers, heap structures, exceptions, high-order functions, typeless low-level hacks, etc.

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

how? $\Longrightarrow$

$$
\begin{aligned}
x_1 &= [-\infty, +\infty] \ or \ x_3 \\
x_2 &= x_1 \ and \ [-\infty, 99] \\
x_3 &= x_2 \dot{+} 1 \\
x_4 &= x_1 \ and \ [100, +\infty]
\end{aligned}
$$

- Is there always a solution to the derived equations?
  - How do we compute the solution in a finite time?

$$
\begin{aligned}
x_1 &= [-\infty, +\infty] \ or \ x_3 \\
x_2 &= x_1 \ and \ [-\infty, 99] \\
x_3 &= x_2 \dot{+} 1 \\
x_4 &= x_1 \ and \ [100, \infty]
\end{aligned}
$$

how? $\Longrightarrow$

$$
\begin{aligned}
x_1 &= [-\infty, +\infty] \\
x_2 &= [-\infty, 99] \\
x_3 &= [-\infty, 100] \\
x_4 &= [100, +\infty]
\end{aligned}
$$