Towards a Globally Scalable Semantics-based Static Analysis

Kwangkeun Yi

School of Computer Science & Engineering
Seoul National University

11/15/2010 @ CSE, HKUST
(co-work with my students and postdocs)

ROSAEC center
Research On Software Analysis for Error-free Computing
소프트웨어 무결점 연구센터 KOSEF ERC

A Fundamental Challenge in SW

How can we reduce/eliminate errors in software?

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

Our approach:

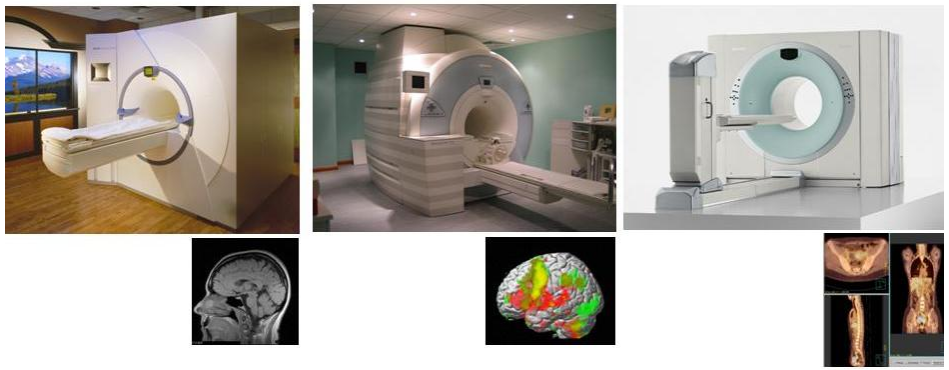
- "semantics-based static analysis"
- *abstract interpretation*, *type system*, *program logics*, *programming language theories*
- and hodge-podge of engineering

ROSAEC center
Research On Software Analysis for Error-free Computing
소프트웨어 무결점 연구센터 KOSEF ERC

Our Goal

R&D of static software analysis tools:

"SW MRI" "SW fMRI" "SW PET"



ROSAEC center
Research On Software Analysis for Error-free Computing
소프트웨어 무결점 연구센터 KOSEF ERC

Our Activities

We publicize our works in:

- POPL('06, '11), CAV('08), VMCAI('10, '11), ICSE('11), SAS, OOPSLA, FSE, etc.
- ACM TOPLAS, TCS, JFP, SP&E, Acta Informatica, etc.

A commercialization:

Sparrow
The Early Bird

Research areas: *static analysis, abstract interpreation, programming languge theory, type system, & whatever relevant*

ROSAEC center
Research On Software Analysis for Error-free Computing
소프트웨어 무결점 연구센터 KOSEF ERC

Talk Outline

1. Short overview of SPARROW: scalable yet non-global bug finder for C
2. New localization technology for scalable & global SPARROW
3. Concluding remark

ROSAEC center
Research On Software Analysis for Error-free Computing
소프트웨어 무결점 연구센터 KOSEF ERC

Sparrow System

Motivation

- prove by ourselves that static analysis is "useful in real world"

Status

- *shallow property, full automation, scalable*
  - C's buffer overrun, memory leak, null deref., / by zero, $\cdots$
- for *non domain-specific* C code
- detection ratio: $\sim$6/KLOC, speed: $\sim$100Loc/sec

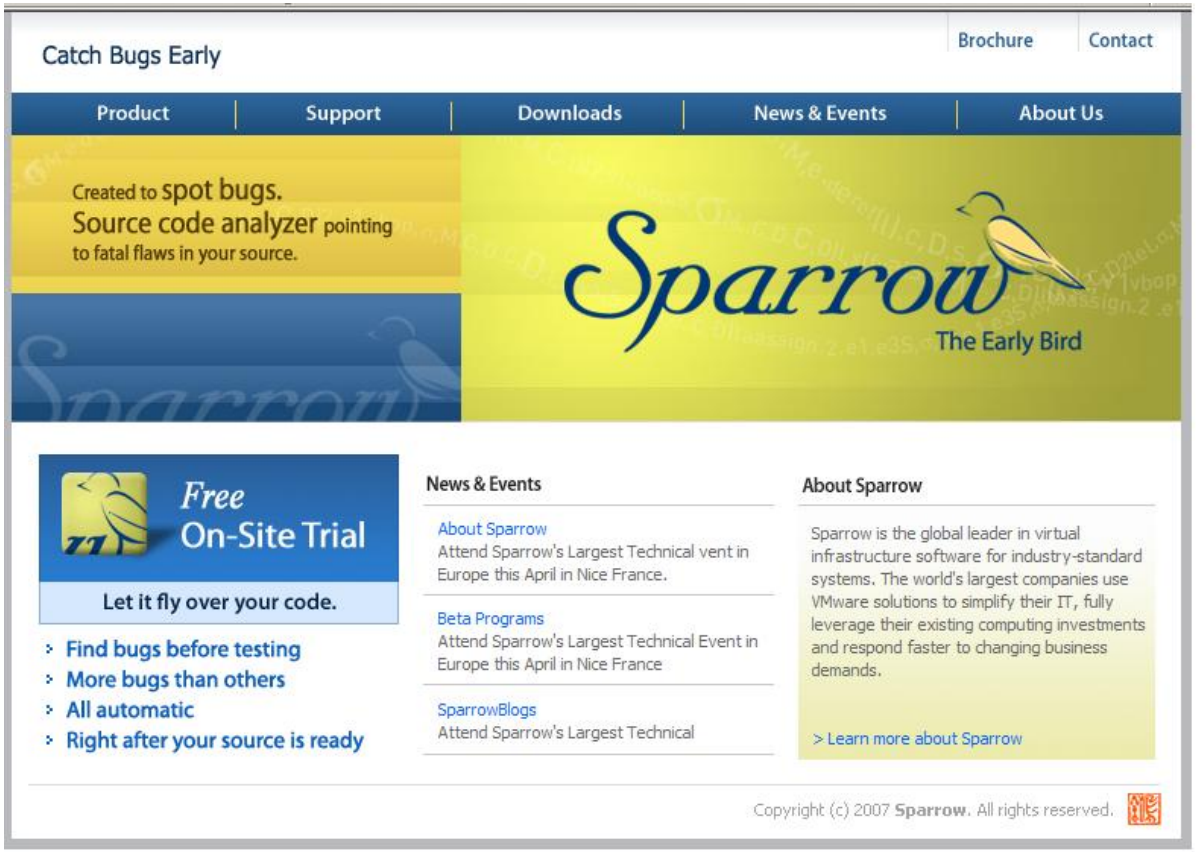
From the elusive three (deep property, scalability, automation)

– deep property

\+ domain-independence

ROSAEC center
Research On Software Analysis for Error-free Computing
소프트웨어 무결점 연구센터 KOSEF ERC

spa-arrow.com



ROSAEC center
Research On Software Analysis for Error-free Computing
소프트웨어 무결점 연구센터 KOSEF ERC

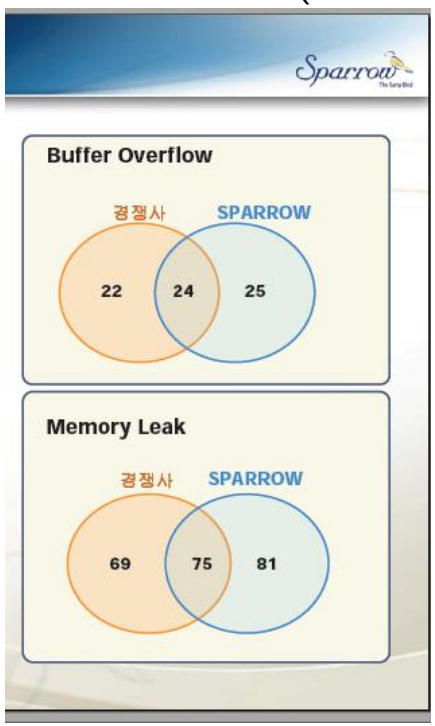
Customers

Domestic market at the moment

- electronics companies: Samsung, LG, $\cdots$
- network switching system sw developers
- other embedded sw developers
- bank system sw developers
- etc.

Complementing others (Coverity, GrammaTech, Klockworks, Polyspace).

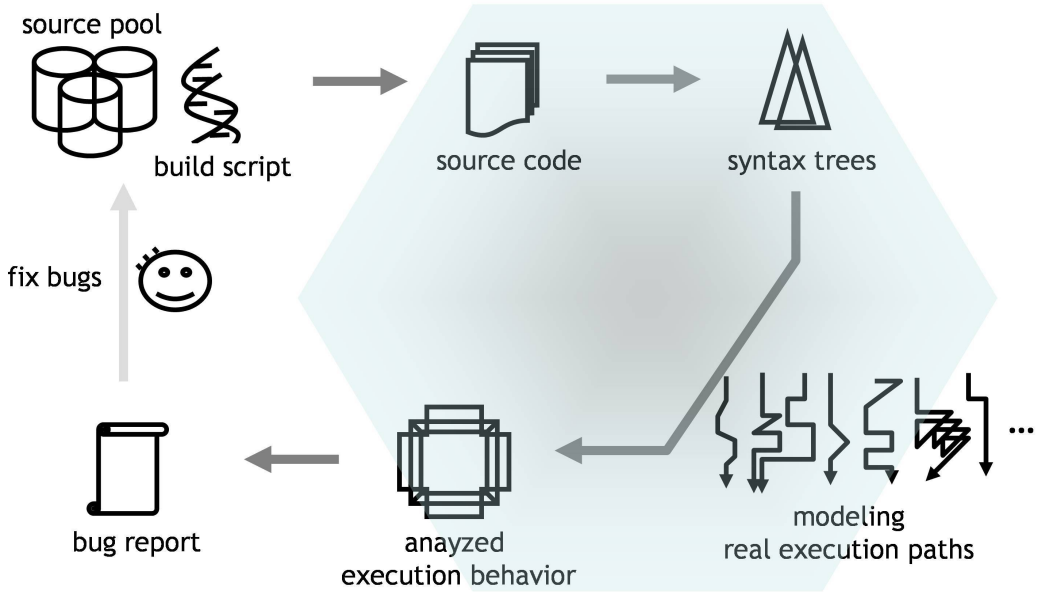
- BMT at a site (a network device OS, $\sim$ 700KLOC):



ROSAEC center
Research On Software Analysis for Error-free Computing
소프트웨어 무결점 연구센터 KOSEF ERC

Steps of Sparrow

Sparrow is a one-button solution with four steps:



Performance Numbers (1/2)

Memory leak detection (SPEC2000 and open sources)

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

ROSAEC center
Research On Software Analysis for Error-free Computing
소프트웨어 무결점 연구센터 KOSEF ERC

Performance Numbers (2/2)

Buffer overrun detection (SPEC2000 and open sources)

Programs	Size KLOC	Time (sec)	True Alarms	False Alarms
art	1.2	0.45	0	0
equake	1.5	2.89	0	1
mcf	1.9	0.33	0	0
bzip2	4.6	10.90	23	29
gzip	7.7	3.38	18	24
parser	10.9	260.94	4	13
twolf	19.7	8.59	0	0
ammp	13.2	10.20	6	0
vpr	16.9	11.15	0	3
crafty	19.4	139.80	1	5
mesa	50.2	47.88	2	10
vortex	52.6	40.12	2	0
gap	59.4	28.48	0	2
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

ROSAEC center
Research On Software Analysis for Error-free Computing
소프트웨어 무결점 연구센터 KOSEF ERC

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

```
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;
```

ROSAEC center
Research On Software Analysis for Error-free Computing
소프트웨어 무결점 연구센터 KOSEF ERC

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

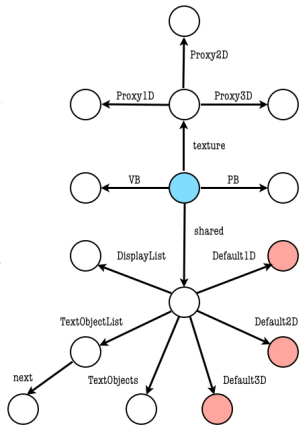
```
line = read_config_read_data(ASN_INTEGER, line,
                             &StorageTmp->traceRouteProbeHistoryHAddrType,
                             &tmpint);
...
line = read_config_read_data(ASN_OCTET_STR, line,
                             &StorageTmp->traceRouteProbeHistoryHAddr,
         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

ROSAEC center
Research On Software Analysis for Error-free Computing
소프트웨어 무결점 연구센터 KOSEF ERC

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 );
```



ROSAEC center
Research On Software Analysis for Error-free Computing
소프트웨어 무결점 연구센터 KOSEF ERC

Sparrow Approach: Pure Soup + Impure Catalysts

Pure soup

- firm formalization of basis: a simple & "sound" abstract interpretation

Impure catalysts

- unorthodox & unsound techniques to refine the bug-finding performance

As a pragmatic approach to find prevalent true errors

- miss rare true cases, reduce many false alarms

Towards Globally Scalable Sparrow

A New Localization Technology of Abstract Memories

ROSAEC center
Research On Software Analysis for Error-free Computing
소프트웨어 무결점 연구센터 KOSEF ERC

Motivation

Big cost of global analysis by SPARROW

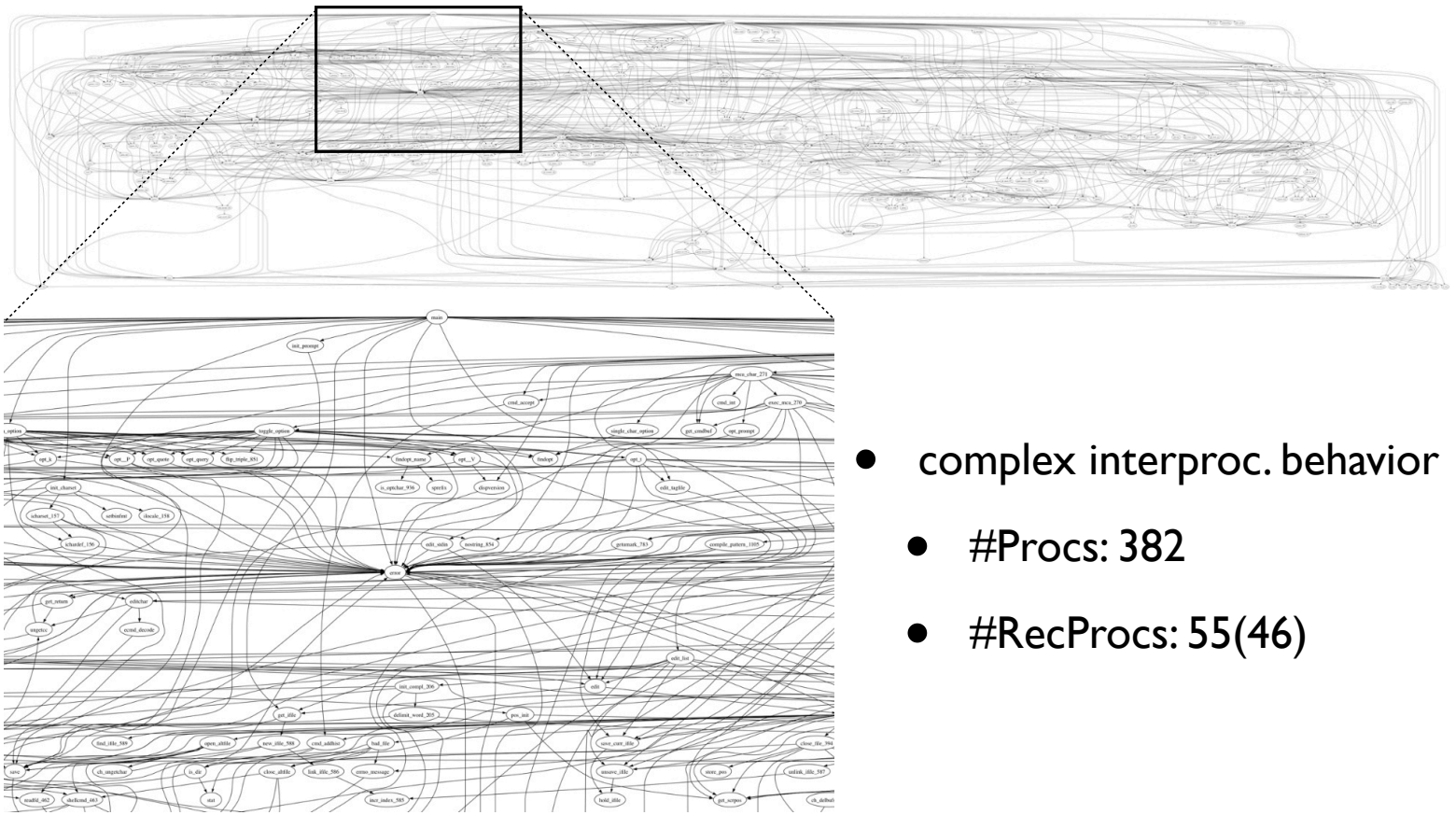
Program	LOC	Time	Mem
bc-1.06	13,093	13,879s	335MB
twolf	19,700	27,230s	1,199MB
less-382	23,822	137,827s	1,480MB

1.6 days

ROSAEC center
Research On Software Analysis for Error-free Computing
소프트웨어 무결점 연구센터 KOSEF ERC

Real-World Programs

less-382, 23,822 LOC



- complex interproc. behavior
  - #Procs: 382
  - #RecProcs: 55(46)

Bottleneck: Re-analysis

- In global analysis, procedures are wastefully re-analyzed by changes to input states.

```
int f() {...}
int m() {
  g = 1; f();
  g = 2; f();}
```

Suppose f does not access g

Change of g has no impact on the analysis of f

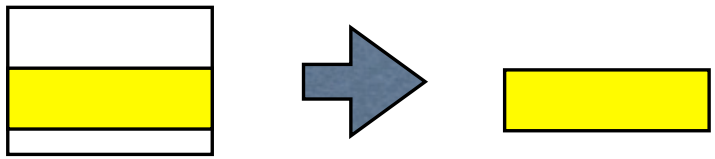
- One major reason for big global analysis cost.

program	LOC	Repeated analyses of procedures
gzip	7,327	211
parser	10,900	601
bc	13,093	1,151
less	23,822	755

ROSAEC center
Research On Software Analysis for Error-free Computing
소프트웨어 무결점 연구센터 KOSEF ERC

Solution

- When analyzing a procedure, pass only the memory parts that is necessary to analyze the procedure.



- Localization reduces re-analysis.

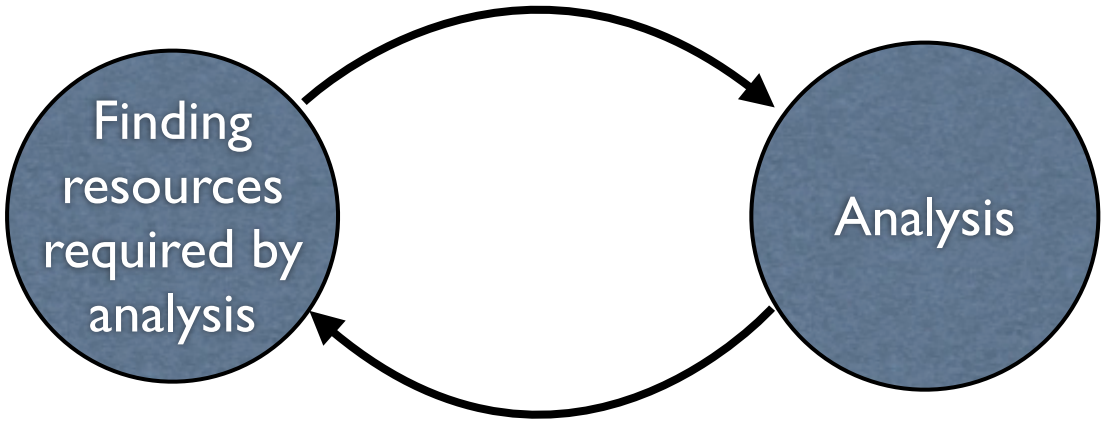
```
int f() {...}
int m() {
  g = 1; f();
  g = 2; f();}
```

Assume f does not access g

Need not to analyze f again

A Catch 22 Situation

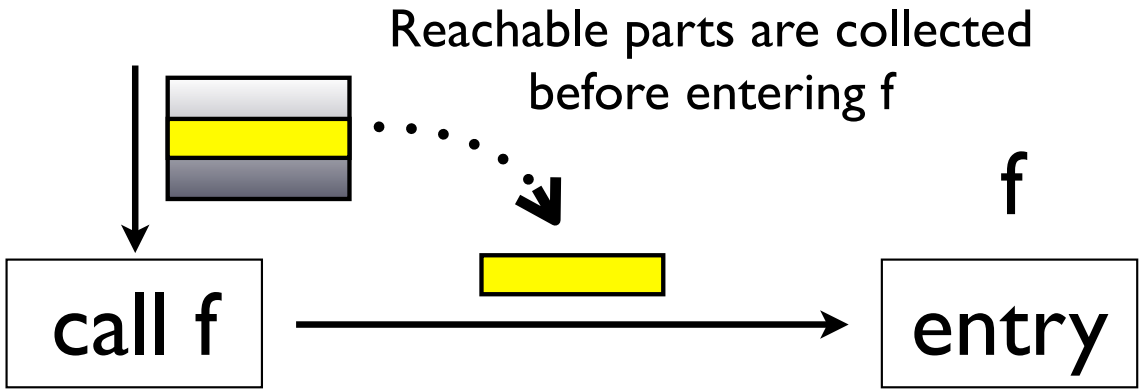
- How to find the memory parts that will be used during the *a*nalysis before the analysis



ROSAEC center
Research On Software Analysis for Error-free Computing
소프트웨어 무결점 연구센터 KOSEF ERC

Status Quo: Reachability-based Approach (Abstract Garbage Collection)

- Approximate what will be used by removing the unreachable from parameters and globals
- Conventional in control flow analysis, shape analysis, ...



Safe, but Crude

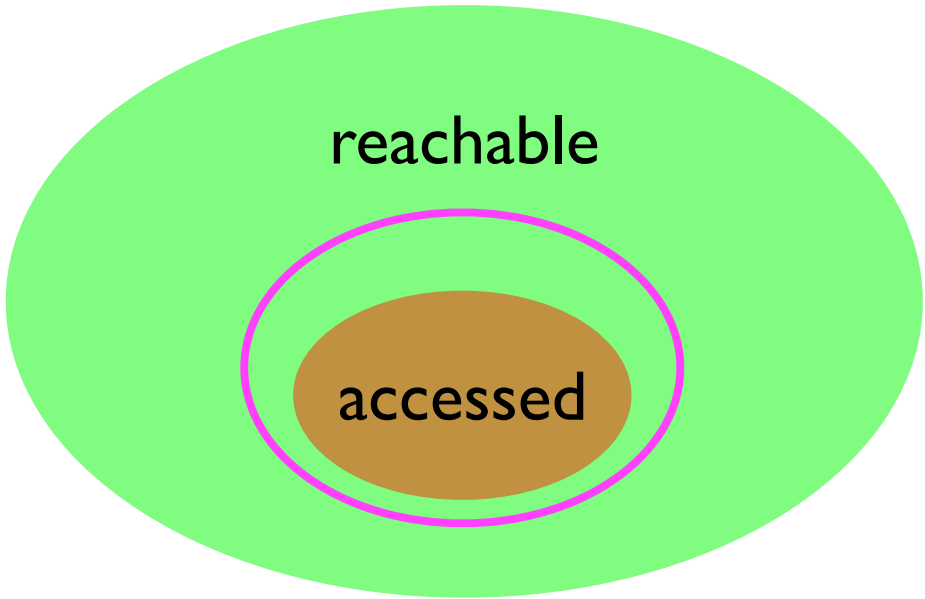
- Accessed parts are a tiny subset of the reachable
  - ex) only a small number of globals are actually accessed in a procedure

Program	LOC	accessed memory / reachable memory
spell-1.0	2,213	5 / 453 (1.1%)
barcode-0.96	4,460	19 / 1175 (1.6%)
httptunnel-3.3	6,174	10 / 673 (1.5%)
gzip-1.2.4a	7,327	22 / 1002 (2.2%)
jwhois-3.0.1	9,344	28 / 830 (3.4%)
parser	10,900	75 / 1787 (4.2%)
bc-1.06	13,093	24 / 824 (2.9%)
less-290	18,449	86 / 1546 (5.6%)

ROSAEC center
Research On Software Analysis for Error-free Computing
소프트웨어 무결점 연구센터 KOSEF ERC

Our Approach

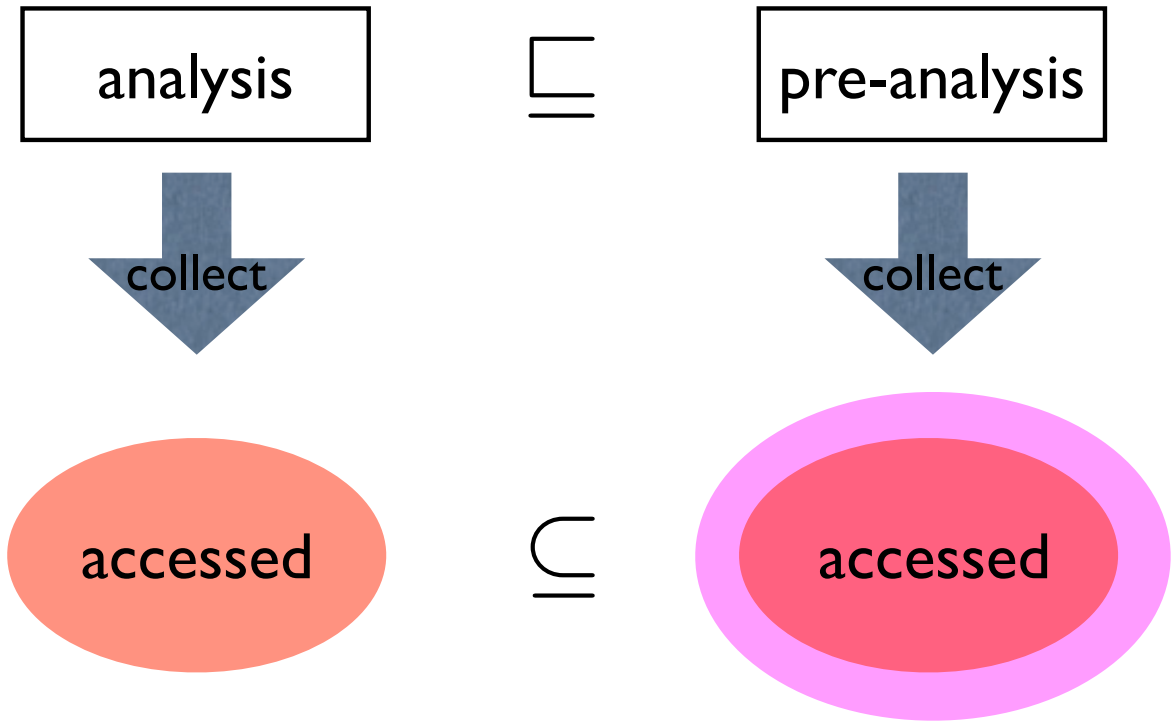
- More aggressive than reachability
  - Exclude some reachable that will not be accessed



ROSAEC center
Research On Software Analysis for Error-free Computing
소프트웨어 무결점 연구센터 KOSEF ERC

How?

- Basically, perform the analysis once more.
  - using a further abstraction of the analysis



ROSAEC center
Research On Software Analysis for Error-free Computing
소프트웨어 무결점 연구센터 KOSEF ERC

How to Get the Further Abstraction?

- Easy in our case, because the underlying analysis is designed as an abstract interpretation.
- Just apply conservative abstractions to the abstract semantics of the original analysis.

Original analysis

$$\hat{\mathcal{D}} = Node \to \hat{Mem}$$

$$\hat{\mathcal{F}} : \hat{\mathcal{D}} \to \hat{\mathcal{D}}$$
$$\hat{\mathcal{F}} = \lambda T.\lambda n.\hat{f}_n\ (\bigsqcup_{p \in \text{predof}(n)} T(p))$$

Pre-analysis

$$\hat{\mathcal{D}}_{pre} = \hat{Mem}$$

$$\hat{\mathcal{F}}_{pre} : \hat{\mathcal{D}}_{pre} \to \hat{\mathcal{D}}_{pre}$$
$$\hat{\mathcal{F}}_{pre} = \lambda m.(\bigsqcup_{n \in Node} \hat{f}_n(m))$$

ignore statements orders

KSAEC center
Research On Software Analysis for Error-free Computing
소프트웨어 무결점 연구센터 KOSEF ERC

Collecting Abstract Locations

- Access functions collects accessed abstract locations.
  - naturally derived from the definitions of the abstract semantics

abstract semantics

$$
\begin{aligned}
\hat{\mathcal{V}}(n)(\hat{m}) &= \langle [n,n], \bot, \bot, \bot \rangle \\
\hat{\mathcal{V}}(e_1 \texttt{ + } e_2)(\hat{m}) &= \hat{\mathcal{V}}(e_1)(\hat{m}) \hat{+} \hat{\mathcal{V}}(e_2)(\hat{m}) \\
\hat{\mathcal{V}}(lv)(\hat{m}) &= \bigsqcup \{ \hat{m}(A\hat{d}dr) \mid \\
&\qquad A\hat{d}dr \in \hat{\mathcal{L}}(lv)(\hat{m}) \} \\
\hat{\mathcal{V}}(\&lv)(\hat{m}) &= \langle \bot, \hat{\mathcal{L}}(lv)(\hat{m}), \bot, \bot \rangle
\end{aligned}
$$

$$
\begin{aligned}
\hat{\mathcal{L}}(x)(\hat{m}) &= \{x\} \\
\hat{\mathcal{L}}(*e)(\hat{m}) &= \hat{\mathcal{V}}(e)(\hat{m}).2 \cup \{ l \mid \langle l, o, s \rangle \in \hat{\mathcal{V}}(e)(\hat{m}).3 \} \\
&\qquad \cup \{ \langle l, x \rangle \mid \langle l, \{x\} \rangle \in \hat{\mathcal{V}}(e)(\hat{m}).4 \} \\
\hat{\mathcal{L}}(e_1 \texttt{[}e_2\texttt{]})(\hat{m}) &= \{ l \mid \langle l, o, s \rangle \in \hat{\mathcal{V}}(e_1)(\hat{m}).3 \} \\
\hat{\mathcal{L}}(e\texttt{.}x)(\hat{m}) &= \{ \langle l, x \rangle \mid \langle l, \{x\} \rangle \in \hat{\mathcal{V}}(e)(\hat{m}).4 \}
\end{aligned}
$$

access functions

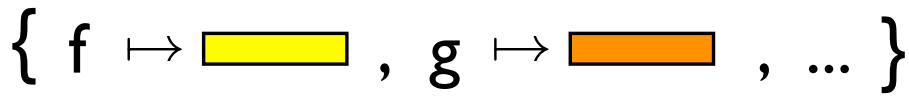
$$
\begin{aligned}
\hat{\mathcal{AV}}(n)(\hat{m}) &= \emptyset \\
\hat{\mathcal{AV}}(e_1 \texttt{ + } e_2)(\hat{m}) &= \hat{\mathcal{AV}}(e_1)(\hat{m}) \cup \hat{\mathcal{AV}}(e_2)(\hat{m}) \\
\hat{\mathcal{AV}}(lv)(\hat{m}) &= \hat{\mathcal{AL}}(lv)(\hat{m}) \cup \hat{\mathcal{L}}(lv)(\hat{m}) \\
\hat{\mathcal{AV}}(\&lv)(\hat{m}) &= \hat{\mathcal{AL}}(lv)(\hat{m})
\end{aligned}
$$

$$
\begin{aligned}
\hat{\mathcal{AL}}(x)(\hat{m}) &= \emptyset \\
\hat{\mathcal{AL}}(*e)(\hat{m}) &= \hat{\mathcal{AV}}(e)(\hat{m}) \\
\hat{\mathcal{AL}}(e_1 \texttt{[}e_2\texttt{]})(\hat{m}) &= \hat{\mathcal{AV}}(e_1)(\hat{m}) \\
\hat{\mathcal{AL}}(e\texttt{.}x)(\hat{m}) &= \hat{\mathcal{AV}}(e)(\hat{m})
\end{aligned}
$$

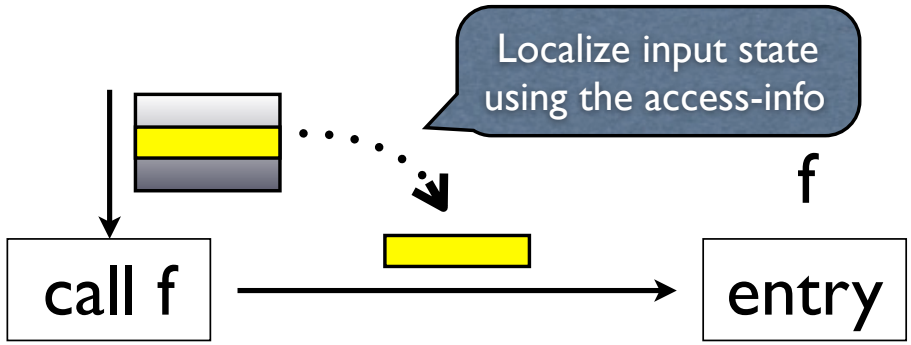
ROSAEC center
Research On Software Analysis for Error-free Computing
소프트웨어 무결점 연구센터 KOSEF ERC

Access-based Localization at Procedure Calls

1. Access-analysis



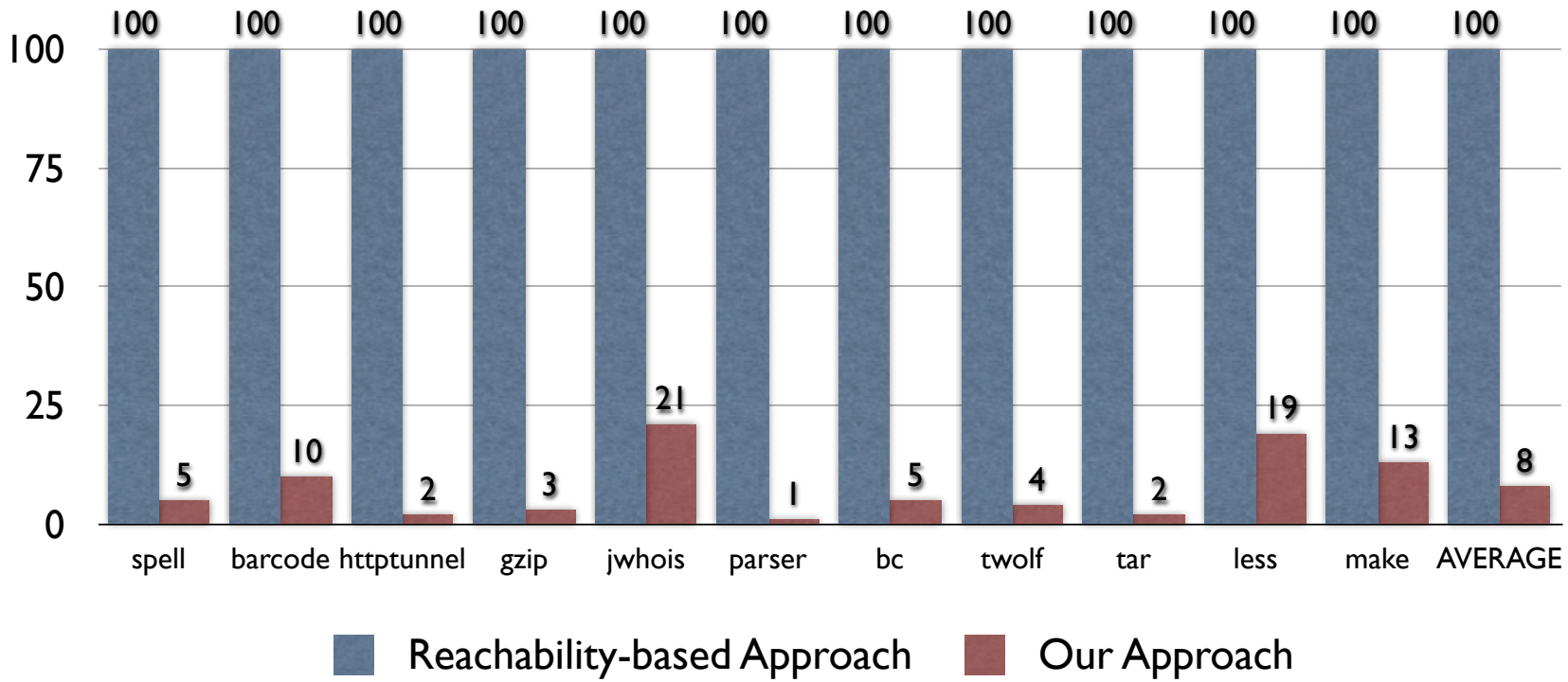
2. Actual analysis



ROSAEC center
Research On Software Analysis for Error-free Computing
소프트웨어 무결점 연구센터 KOSEF ERC

Performance

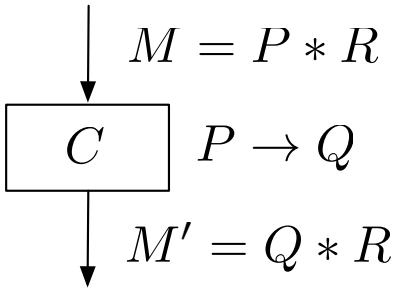
On average 92% reduction in time



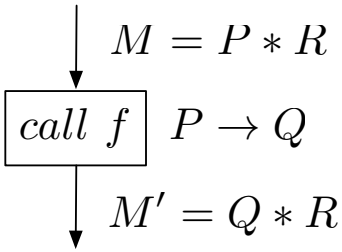
ROSAEC center
Research On Software Analysis for Error-free Computing
소프트웨어 무결점 연구센터 KOSEF ERC

Access-based Localization at Everywhere

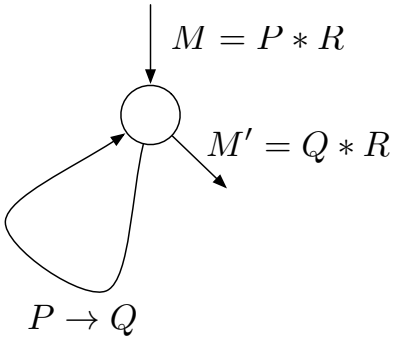
- We can apply access-based localization to any code block C.



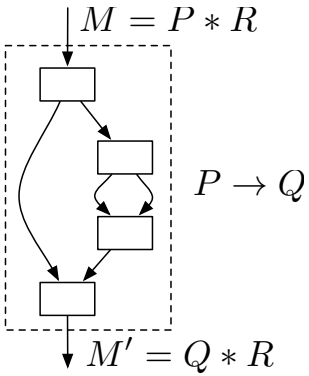
procedures



loops



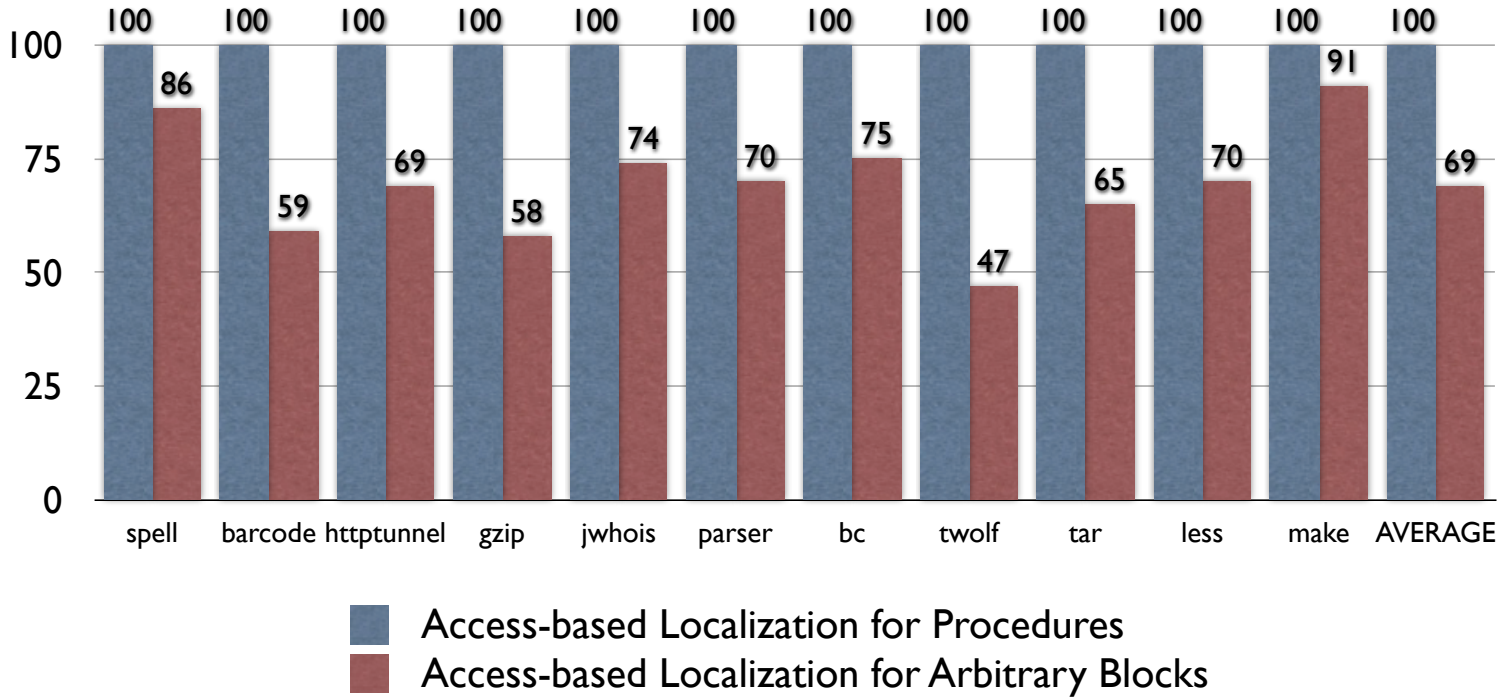
branches



RSAEC center
Research On Software Analysis for Error-free Computing
소프트웨어 무결점 연구센터 KOSEF ERC

Performance

On average 31% further reduction in time



ROSAEC center
Research On Software Analysis for Error-free Computing
소프트웨어 무결점 연구센터 KOSEF ERC

Concluding Remark

- Semantics-based static analysis technology: mature to solve realistic problems for sequential C programs
- Our humble proof: SPARROW
- Practical, semantically-dense global analysis for 100 MLoc will soon be possible
- But lots of problems still remain
  - reducing false alarms: cost vs soundness vs reality vs specialization
  - alarm explanation: analysis internals too complex to understand
  - daunting properties to check/verify

ROSAEC center
Research On Software Analysis for Error-free Computing
소프트웨어 무결점 연구센터 KOSEF ERC

Concluding Remark

- Semantics-based static analysis technology: mature to solve realistic problems for sequential C programs
- Our humble proof: SPARROW
- Practical, semantically-dense global analysis for 100 MLoc will soon be possible
- But lots of problems still remain
  - reducing false alarms: cost vs soundness vs reality vs specialization
  - alarm explanation: analysis internals too complex to understand
  - daunting properties to check/verify

Thank you.

ROSAEC center
Research On Software Analysis for Error-free Computing
소프트웨어 무결점 연구센터 KOSEF ERC