CodeHawk

Sound Static Analysis through Customization

Henny Sipma
Kestrel Technology LLC

HCSS
Linthicum Heights, MD
May 19, 2009

What is CodeHawk?

CodeHawk

sound static analysis tool (goal: no false negatives)

underlying technology: abstract interpretation

developed under SBIR contracts from AF and Army

current version specialized for buffer overflow

support for additional properties under development

Outline

1. Approach to property checking
2. Underlying Theory: Abstract interpretation
3. CodeHawk architecture
4. Errors, warnings and safe conditions
5. Three case studies
    1. Generic: SAMATE 115 - 1278
    2. Bugfinding: SAMATE 1291
    3. Customized analyzer for verification: Boeing
6. Conclusions

Approach to property checking: buffer overflow

1. All buffer accesses in the program are identified
2. For each buffer access a safety condition is constructed that guarantees there is no out-of-bounds access
3. The safety condition is evaluated against invariants generated

Approach to property checking: buffer overflow

Invariants

Over-approximation of the reachable state space of the program

constants

intervals
(variable ranges)

polyhedra
(variable relationships)

flow/context sensitive

$\models$

Safety conditions

constructed using

size of allocated memory blocks (stack or heap)

current offset into allocated memory block

semantics of library functions

generated using abstract interpretation

Abstract Interpretation

Mathematical Theory of Approximation

Developed in 1970's by Cousot and Cousot in France

Theory is well established - hundreds of research papers (1977 - present)

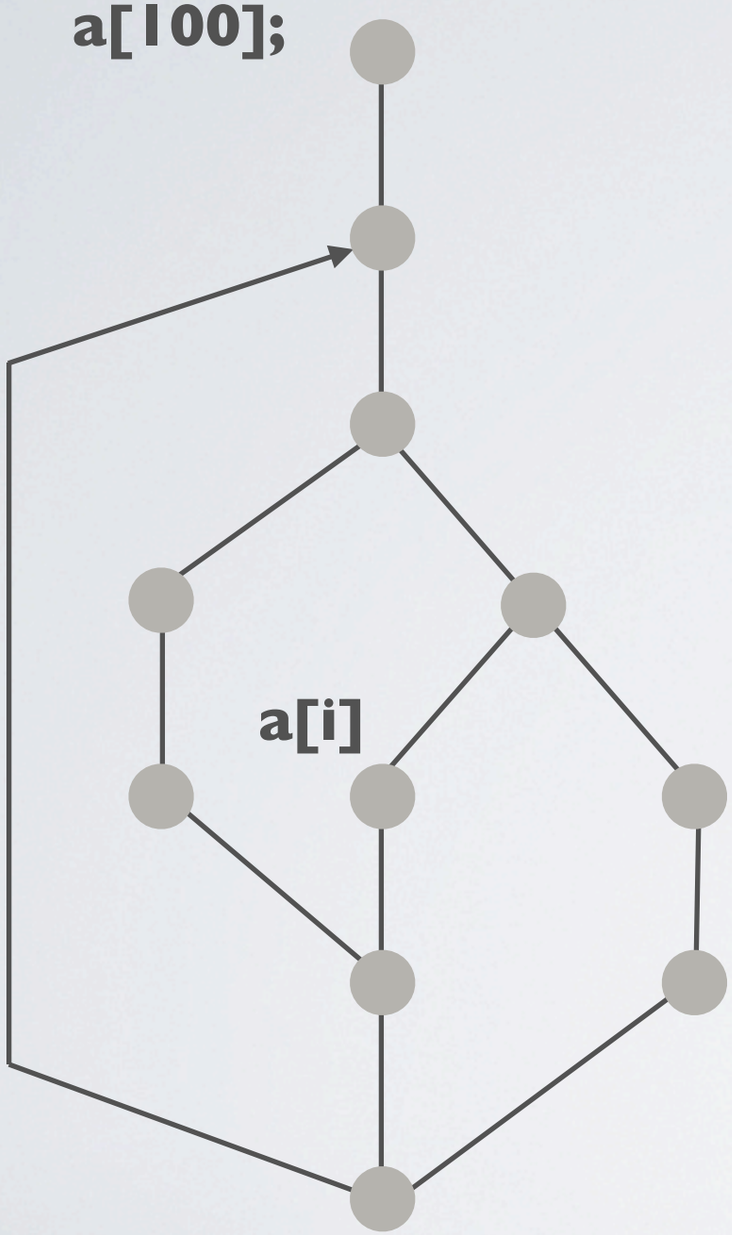
Challenge is the engineering: how to create a commercially usable tool

ASTREE : highly specialized analyzer for Airbus flight control software

Polyspace (acquired by MathWorks)

Abstract interpretation: theory

How do we know if a[i] is safe? $0 \leq i < 100$



Symbolic simulation for all possible input values

? Keep track of all possible values for all variables at all program points ?

No: sets of values may be infinite

? Describe in first-order logic?

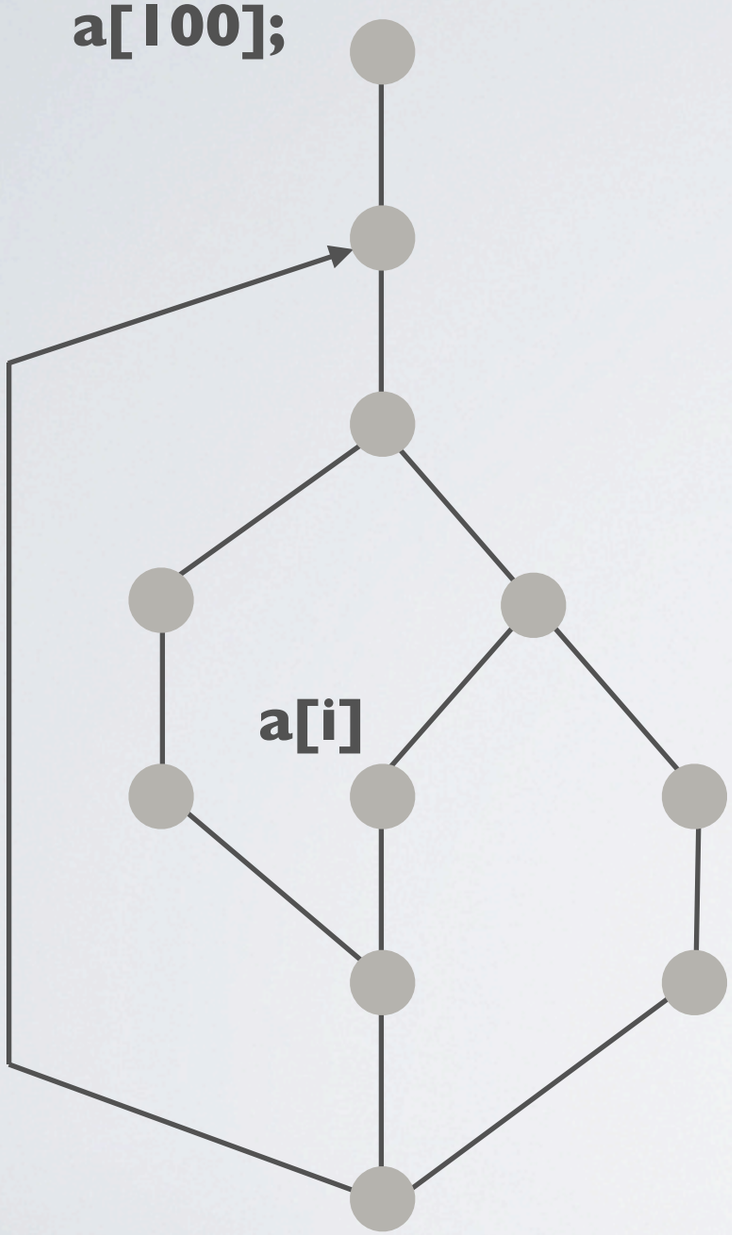
No: detecting convergence is not decidable

? Approximate in some decidable theory?

Ok, but be sure it is a **conservative** approximation

! ***Abstract interpretation*** can provide that guarantee

Abstract interpretation: theory



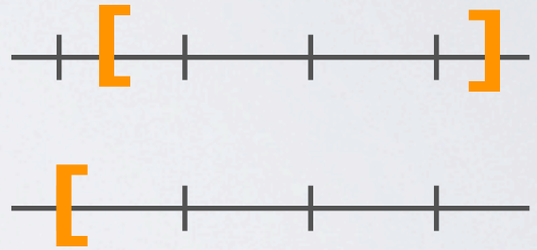
Some suitable numerical domains (decidable theories)

Constants

(fixed value or top)

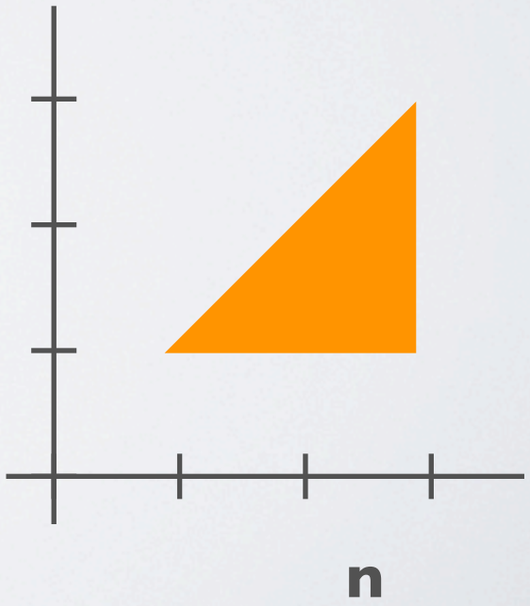
Intervals

(ranges on variables)

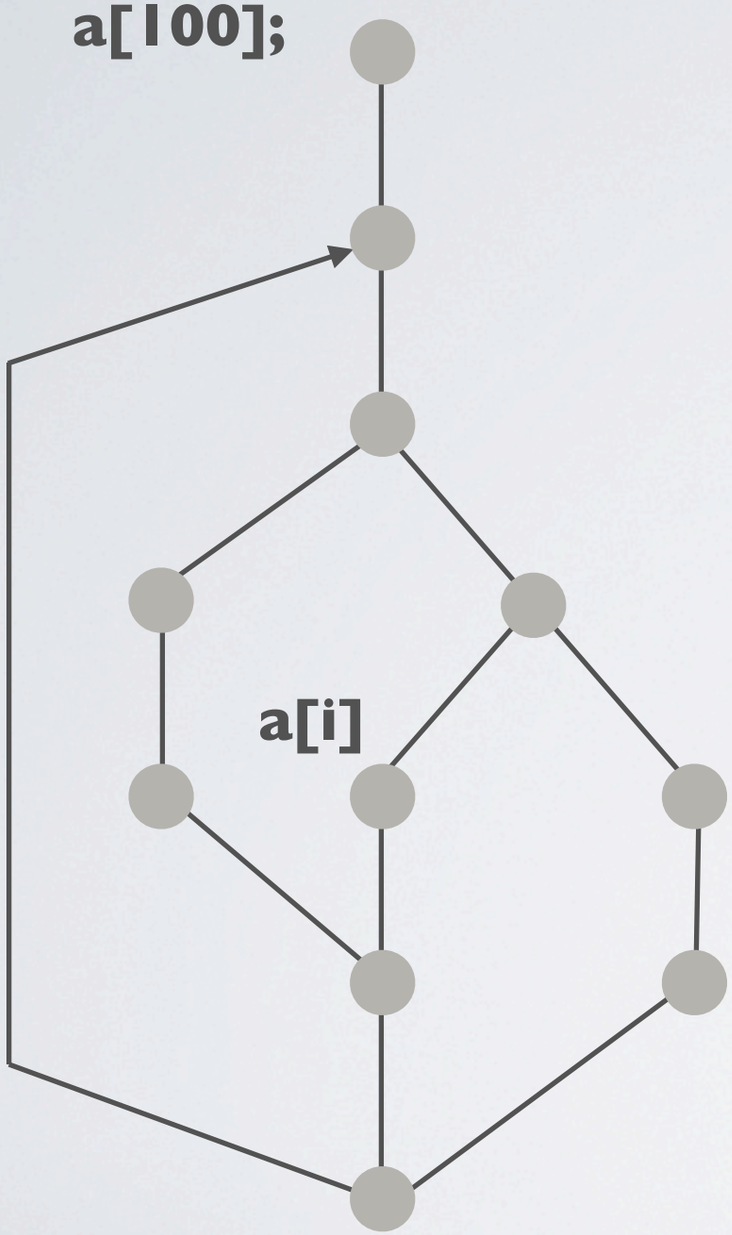


Polyhedra

(linear relationships
between multiple variables)



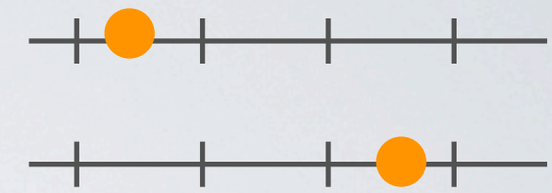
Abstract interpretation: theory



Some suitable numerical domains (decidable theories)

Constants

(fixed value or top)



Intervals

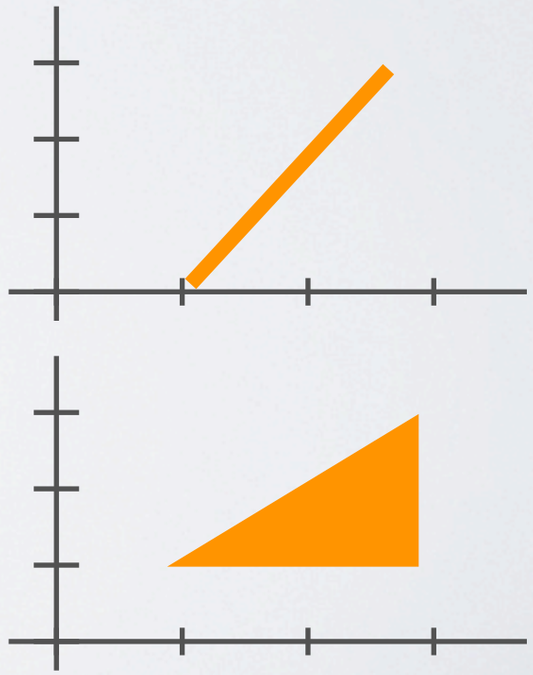
(ranges on variables)

several **weakly relational** domains

Linear equalities

Polyhedra

(linear relationships
between multiple variables)



Approach to property checking: buffer overflow

Invariants

Overapproximation of the reachable state space of the program

constants

intervals
(variable ranges)

polyhedra
(variable relationships)

flow/context sensitive

generated using abstract interpretation

$\models$

Safety conditions

constructed using

size of allocated memory blocks (stack or heap)

current offset into allocated memory block

semantics of library functions

Abstract interpretation: practice

In theory, practice and theory are the same

In practice, they are not

Abstract interpretation **theory**: well studied, many research papers

Abstract interpretation **in practice**: challenging **engineering** task

main challenges:

- managing computational complexity
- trade-off between scalability and precision

CodeHawk approach:

- analyzer generator
- customization for class of applications and properties

CodeHawk Architecture

Abstract interpretation engine

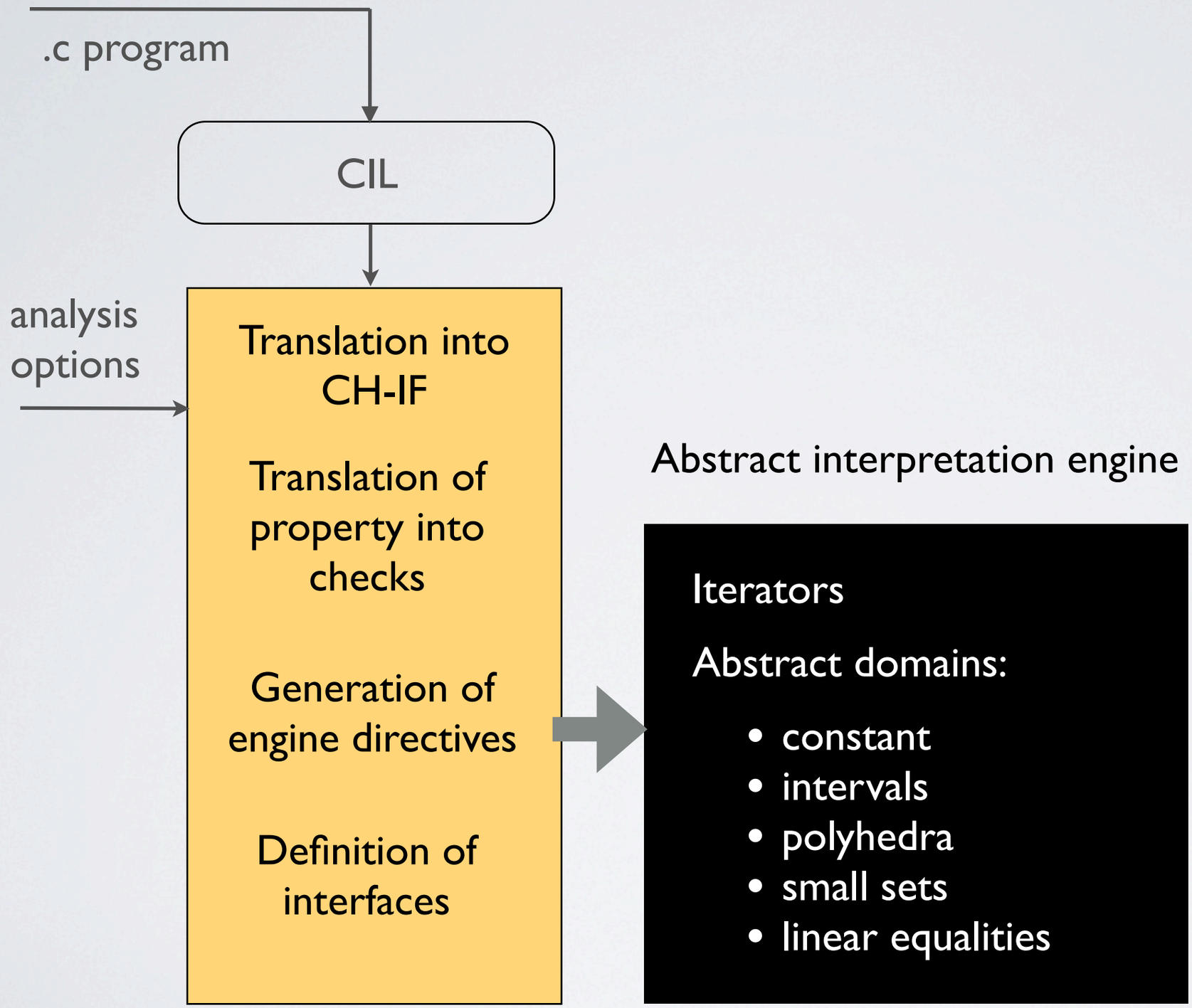
Iterators

Abstract domains:

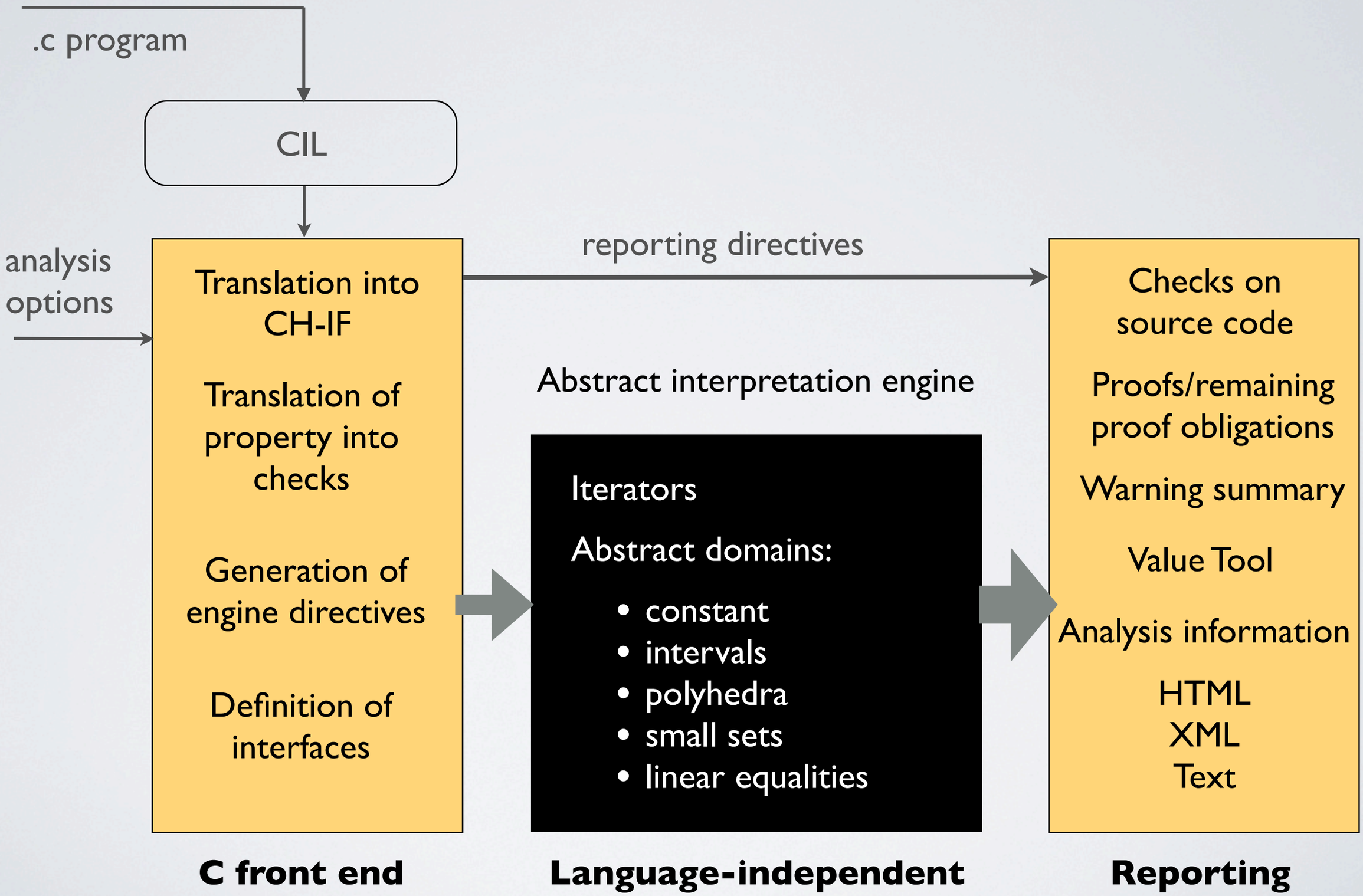
- constant
- intervals
- polyhedra
- small sets
- linear equalities

Language-independent

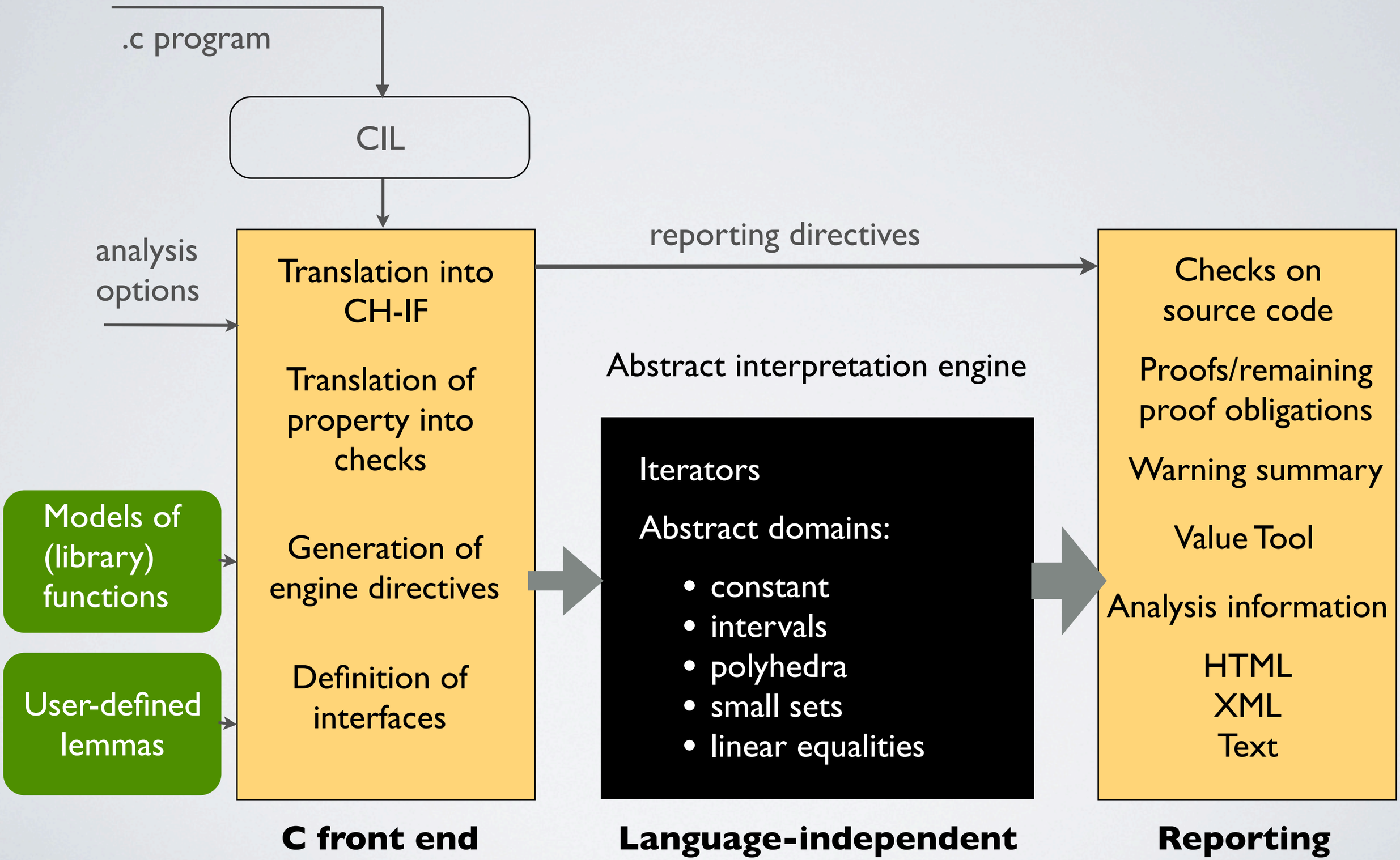
CodeHawk Architecture



CodeHawk Architecture



CodeHawk Architecture



Background on proving errors

safe range:

range computed:

(guaranteed to be an over approximation of the actual range)

an access is guaranteed safe if its computed range is fully within the safe range

an access is guaranteed an error if its computed range is fully outside the safe range

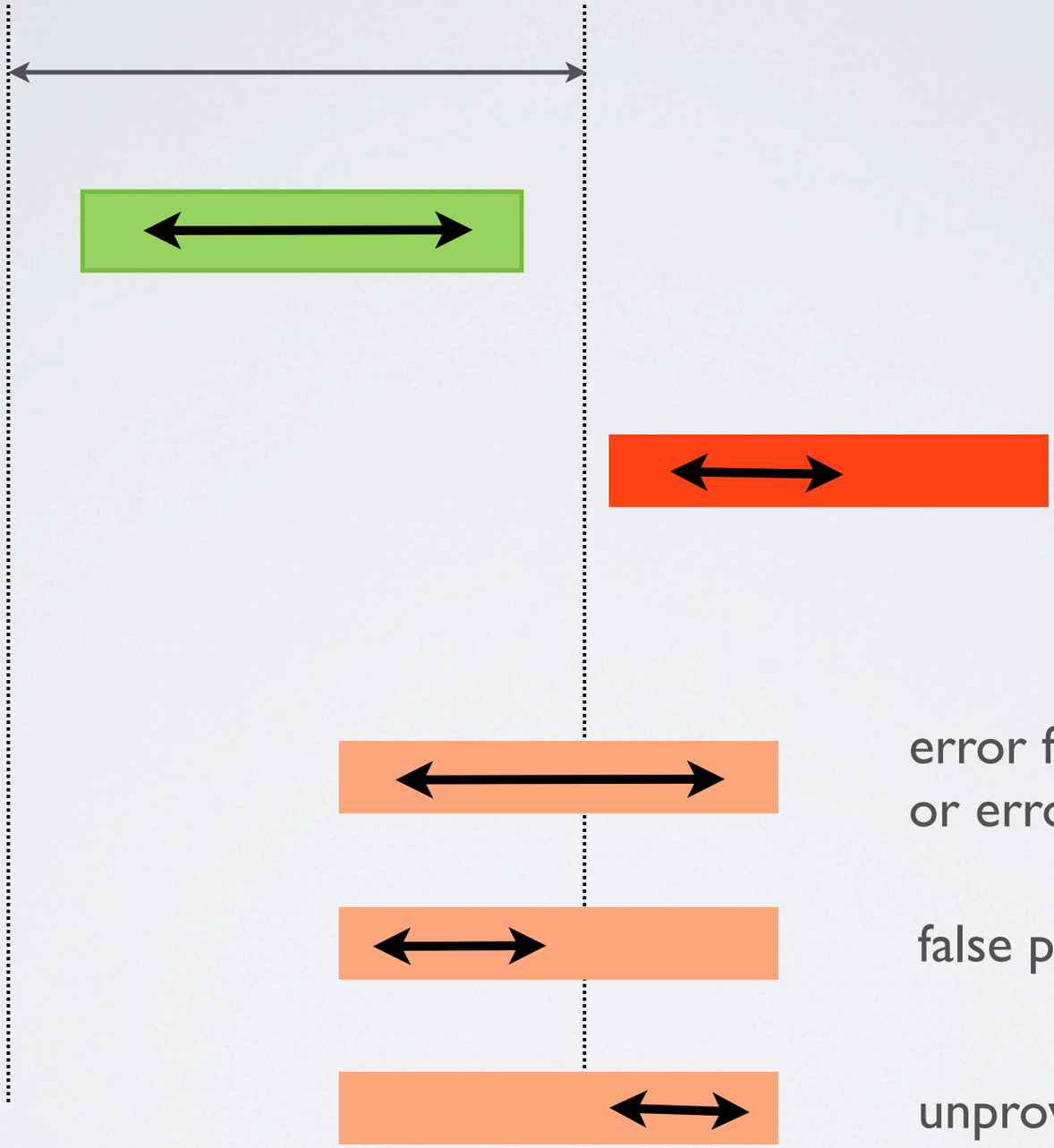
if overlap with the safe range, but not fully within the safe range, not sure whether safe or an error, due to over approximation

Background on proving errors

safe range:

range computed:

(guaranteed to be an over approximation of the actual range)



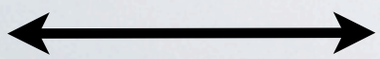
safe for all inputs

error for all inputs

error for some, but not all inputs
or error for some part of loops

false positive

unproven error



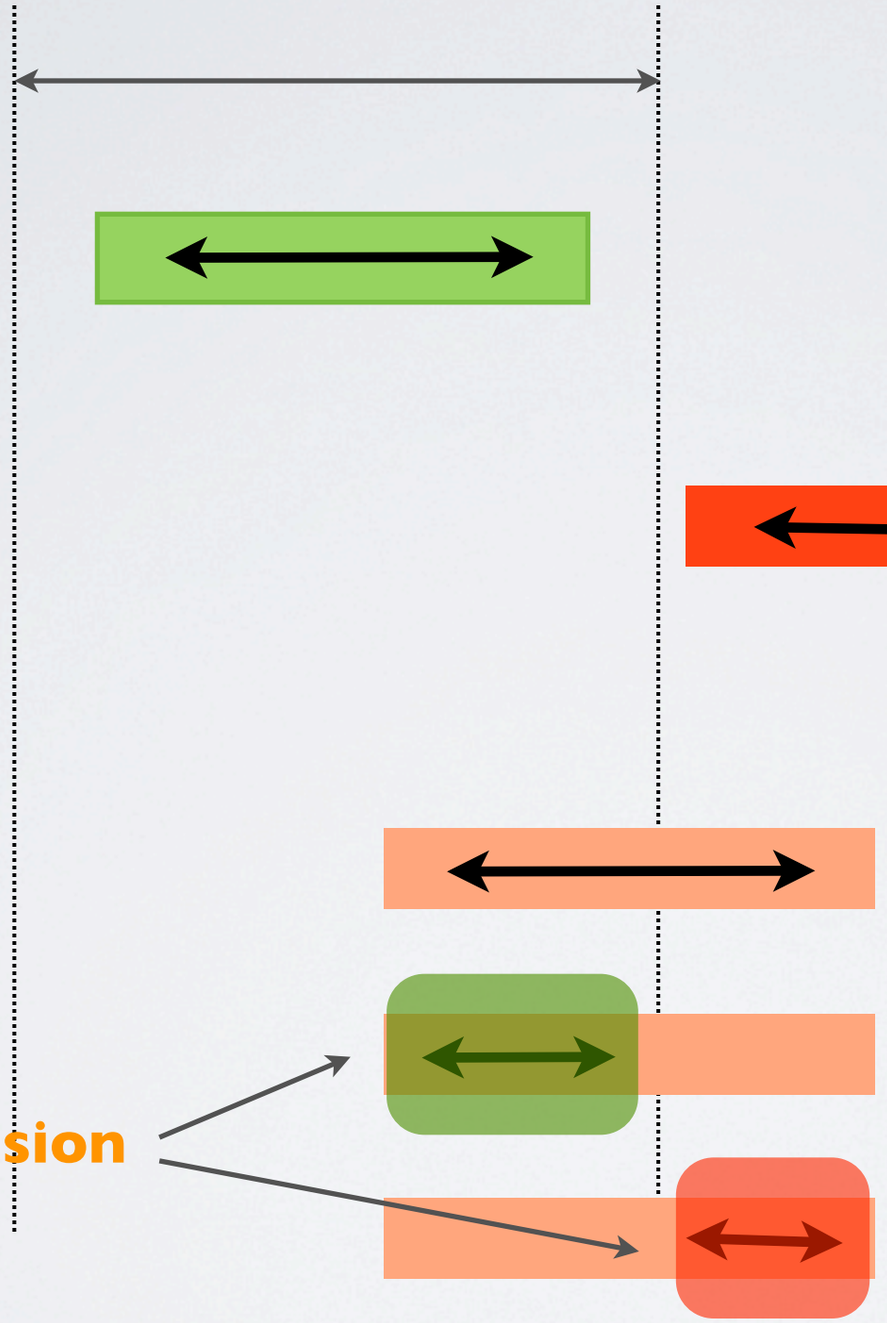
actually reachable

Background on proving errors

safe range:

range computed:

(guaranteed to be an over approximation of the actual range)



safe for all inputs

error for all inputs

error for some, but not all inputs
or error for some part of loops

increased precision

~~false positive~~ **safe**

~~unproven error~~ **error**

actually reachable

Case studies

1. NIST SAMATE benchmarks 115 - 1278

1164 small programs (5 - 20 l.o.c.) with wide variety of buffer overflows

constructed at MIT, 2004

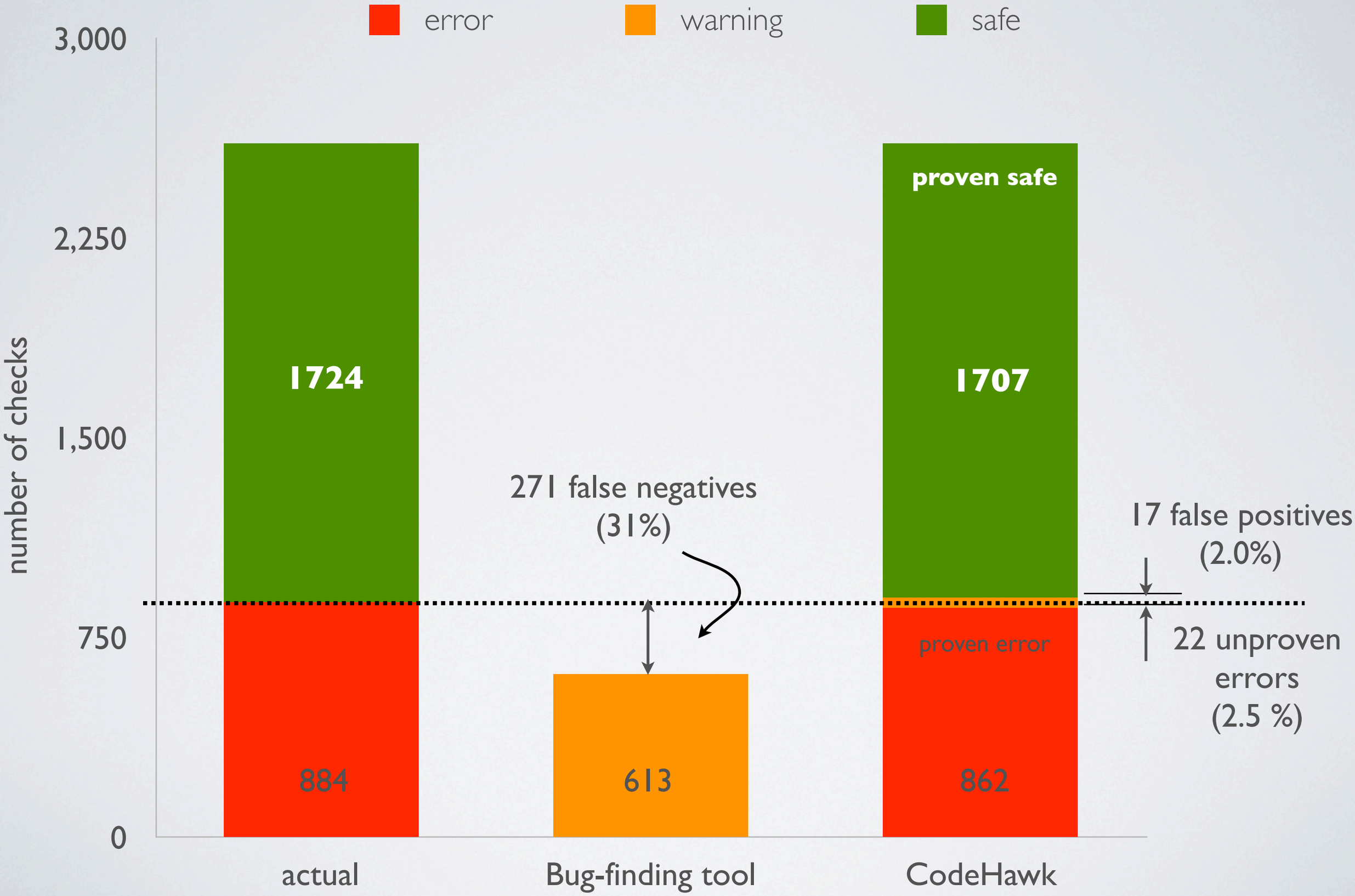
2. NIST SAMATE benchmark 1291

Fragment extracted from BIND with known vulnerability

3. BOEING CASE STUDY

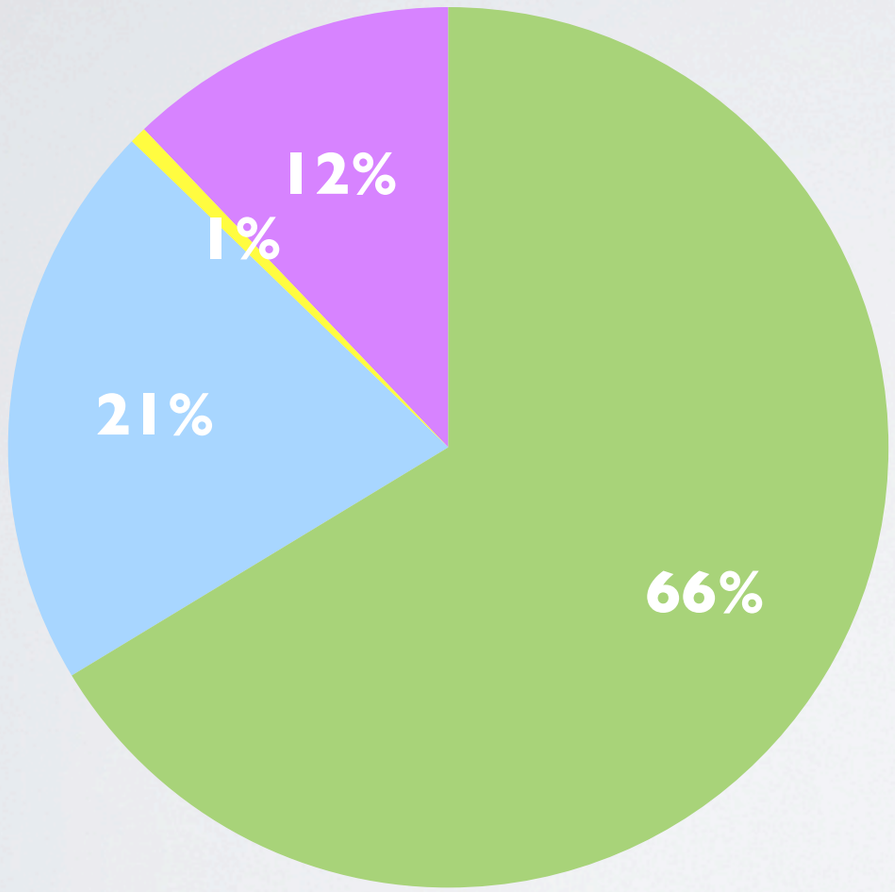
Example flight software developed on a prior research project

Results for SAMATE benchmarks 115-1278



Use of Domains

SAMATE small benchmarks
115-1278
(2569 proven checks)



CodeHawk case study: BIND - 2

source: NIST SAMATE Reference Dataset -- benchmark 1291

program fragment
- extracted from BIND by Zitser et al (MIT)
- includes a reported exploitable vulnerability

statistics: 750 lines of code ; 615 statements

Why do we care?

BIND (Berkeley Internet Name Domain) is the most commonly used DNS server on the Internet

Wikipedia

National Cyber-Alert System

Vulnerability Summary for CVE-1999-0835

Original release date: 11/10/1999

Last revised: 09/09/2008

Source: US-CERT/NIST

Static Link: http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-1999-0835

Overview

Denial of service in BIND named via malformed SIG records.

Impact

CVSS Severity (version 2.0):

CVSS v2 Base Score: 10.0 (HIGH) (AV:N/AC:L/Au:N/C:C/I:C/A:C) (legend)

Impact Subscore: 10.0

Exploitability Subscore: 10.0

CVSS Version 2 Metrics:

Access Vector: Network exploitable

Access Complexity: Low

Authentication: Not required to exploit

Impact Type: Provides administrator access, Allows complete confidentiality, integrity, and availability violation; Allows unauthorized disclosure of information; Allows disruption of service

Apply CodeHawk

out-of-the-box analysis results:

CodeHawk buffer overflow report

System: sig-bad_mac_mod

Buffer Overflow Analysis Results

Click on filename(s) to access the analysis results on the source code

File	L.O.C.	#stmts	Safe	Warning	ERROR	Not reachable	Buffer Checks	Concrete	Intervals	Polyhedra	Indirect	No Proof
sig-bad_mac_mod.c	750	624	130	67	0	2	199	7	121	4	0	67
	750	632	130	67	0	2	199	7	121	4	0	

Analysis time: 51.92750 sec

Summary of the results:

130 buffer access checks proven safe

2 buffer access checks proven unreachable

67 buffer access checks without proof

Analysis time: 52 sec

Analysis Results on the Source Code

```
307
308         cp += n;
309         len += n;
310         len += sizeof(HEADER);
311         BOUNDS_CHECK(cp, 2*INT16SZ + INT32SZ + INT16SZ);
312         GETSHORT(type, cp);
313         cp += 2;
314         len += 2;
315         printf("type = %d\n", type);
316         GETSHORT(class, cp);
317         cp += 2;
318         len += 2;
319
320         if (class > CLASS_MAX) {
321           printf("bad class in rrextract");
322           hp-> rcode = FORMERR;
323           return (-1);
324         }
325         GETLONG(ttl, cp);
326         printf("ttl = %d\n",ttl);
327         cp += 4;
328         len += 4;
329
330         if (ttl > MAXIMUM_TTL) {
331           printf("%s: converted TTL > %u to 0",
332                dname, MAXIMUM_TTL);
333
334               ttl = 0;
335         }
336         GETSHORT(dlen, cp);
337         cp += 2;
```

buffer overflow	O[V#876] + 1 <= S[V#876]
calling context	main@737

Simplification (cstCopyProp)	
to	O[cp] + 1 <= S[cp]
with	O[V#876] = O[cp] S[V#876] = S[cp]

Abstraction	
O[cp] + 1 <= S[cp]	(O[cp] + 1) <= S[cp]

Intervals	
to	(O[cp] + 1) <= S[cp]
with	O[cp] >= 12 S[cp] = 1000

Polyhedra	
to	(O[cp] + 1) <= S[cp]
with	n >= 0 S[V#874] == 1000 S[cp] == 1000 (O[cp] - n) == 12 S[tmp___0] == 1000 O[tmp___0] == 0 S[tmp] == 100 O[tmp] == 0 (O[V#874] - n) == 12 S[eom] == 1000

Polyhedra	
to	n <= 987
with	n >= 0 S[cp] == 1000 (O[cp] - n) == 12 S[eom] == 1000

Summary of Warnings

CodeHawk buffer overflow report

System: sig-bad_mac_mod

Buffer Overflow Analysis Results

Click on filename(s) to access the analysis results on the source code

File	L.O.C.	#stmts	Safe	Warning	ERROR	Not reachable	Buffer Checks	Concrete	Intervals	Polyhedra	Indirect	No Proof
sig-bad_mac_mod.c	750	624	130	67	0	2	199	7	121	4	0	67
	750	632	130	67	0	2	199	7	121	4	0	67

Analysi[illegible] 31.92750 sec

Summary of Warnings

Click on the function names to access the warnings

line nrs	function	number of warnings
283 - 606	rrextract	31
607 - 720	createSig	36

Summary of Warnings

linenr	context	description	remaining proof obligation
312	main(737)	buffer overflow	n <= 987
		buffer overflow	n <= 986
316	main(737)	buffer overflow	n <= 983
		buffer overflow	n <= 982
325	main(737)	buffer overflow	n <= 979
		buffer overflow	n <= 978
		buffer overflow	n <= 977
		buffer overflow	n <= 976
336	main(737)	buffer overflow	n <= 971
		buffer overflow	n <= 970
412	main(737)	buffer overflow	n <= 967
		buffer overflow	n <= 966
417	main(737)	buffer overflow	n <= 963
		buffer overflow	n <= 962
425	main(737)	buffer overflow	n <= 959
		buffer overflow	n <= 958
		buffer overflow	n <= 957
		buffer overflow	n <= 956
430	main(737)	buffer overflow	n <= 951
		buffer overflow	n <= 950
		buffer overflow	n <= 949
		buffer overflow	n <= 948
435	main(737)	buffer overflow	n <= 943
		buffer overflow	n <= 942
		buffer overflow	n <= 941
		buffer overflow	n <= 940
440	main(737)	buffer overflow	n <= 935
		buffer overflow	n <= 934
484	main(737)	memcpy source overflow	n <= 932
561	main(737)	memcpy source overflow	(O[cp] + n) <= 1000
		memcpy target overflow	(O[cp1] + n) <= 4140

29 warnings:

n <= some upper bound

Find root cause of 29 warnings

✓	299	if ((n = dn_expand(msg, eom, cp, (char *) dname, namelen)) < 0) {
	300	printf("dn_expand returned %d\n", n);
✓✓	301	hp-> rcode = FORMERR;
	302	
	303	return (-1);
	304	}
	305	
	306	printf("First dn_expand returned n = %d\n", n);
	307	
	308	cp += n;
	309	len += n;
	310	len += sizeof(HEADER);
✓✓	311	BOUNDS_CHECK(cp, 2*INT16SZ + INT32SZ + INT16SZ);
?? ✓✓	312	GETSHORT(type, cp);
	313	cp += 2;
	314	len += 2;
	315	printf("type = %d\n", type);
?? ✓✓	316	GETSHORT(class, cp);

dn_expand: Derive assumption from documentation

```
int dn_expand(unsigned char *msg,
              unsigned char *eomorig,
              unsigned char *comp_dn,
              unsigned char *exp_dn,
              int length);
```

The dn_expand() function expands the compressed domain name comp_dn to a full domain name, which is placed in the buffer exp_dn of size length.

Upon successful completion, the dn_expand subroutine returns the size of the expanded domain name.

Assumption

return-value ≤ length

Add assumption to the analyzer:

now: 5 lines of Ocaml code

new design: 1 line in external model repository

38 Warnings left

Buffer Overflow Analysis Results

Click on filename(s) to access the analysis results on the source code

File	L.O.C.	#stmts	Safe	Warning	ERROR	Not reachable	Buffer Checks	Concrete	Intervals	Polyhedra	Indirect	No Proof
sig-bad_mac_mod.c	750	624	159	38	0	2	199	7	150	4	0	38
	750	632	159	38	0	2	199	7	150	4	0	38

Analysis time: 51.97137 sec

Summary of Warnings

Click on the function names to access the warnings

line nrs	function	number of warnings
283 - 606	rrextract	2
607 - 720	createSig	36

Before:

130 buffer access checks proven safe

2 buffer access checks proven unreachable

67 buffer access checks without proof

After adding model for dn_expand:

159 buffer access checks proven safe

2 buffer access checks proven unreachable

38 buffer access checks without proof

The new results

	307			cp += n;
	308			len += n;
	309			len += sizeof(HEADER);
	310		307	cp += n;
✓✓	311		308	len += n;
?? ✓✓	312		309	len += sizeof(HEADER);
	313		310	
	314	✓✓	311	BOUNDS_CHECK(cp, 2*INT16SZ + INT32SZ + INT16SZ);
	315	✓✓✓✓	312	GETSHORT(type, cp);
?? ✓✓	316		313	cp += 2;
	317		314	len += 2;
	318		315	printf("type = %d\n", type);
	319	✓✓✓✓	316	GETSHORT(class, cp);
	320		317	cp += 2;
	321		318	len += 2;
✓✓	322		319	
	323		320	if (class > CLASS_MAX) {
	324		321	printf("bad class in rrextract");
???? ✓✓✓✓	325	✓✓	322	hp-> rcode = FORMERR;
	326		323	return (-1);
	327		324	}
	328	✓✓✓✓✓✓✓✓	325	GETLONG(ttl, cp);
	329		326	printf("ttl = %d\n",ttl);
	330		327	cp += 4;
	331		328	len += 4;
	332		329	
	333		330	if (ttl > MAXIMUM_TTL) {
	334		331	printf("%s: converted TTL > %u to 0",
	335		332	dname, MAXIMUM_TTL);
?? ✓✓	336		333	
	337		334	ttl = 0;
			335	}
		✓✓✓✓	336	GETSHORT(dlen, cp);
			337	cp += 2;

All assumptions are recorded

299	Assumption for res_9_dn_expand: dn_expand returns successfully: V#66 <= V#65
299	Assumption for res_9_dn_expand: dn_expand returns with failure: V#66 = -1
345	inlining: ns_ownercontext
345	inlining: ns_nameok
445	Unmodeled function: time
445	No assumptions: time
457	Unmodeled function: res_9_p_secstodate
457	No assumptions: res_9_p_secstodate
463	Unmodeled function: res_9_p_secstodate
463	No assumptions: res_9_p_secstodate
492	Assumption for res_9_dn_expand: dn_expand returns successfully: V#776 <= V#772
492	Assumption for res_9_dn_expand: dn_expand returns with failure: V#776 = -1

38 More Warnings

654	main(729)	buffer overflow	(O[comp_dn] + O[temp]) <= 998
657	main(729)	buffer overflow	O[p] <= 999
	main(729)	buffer overflow	(O[comp_dn] + O[temp]) <= 994
660	main(729)	buffer overflow	O[p] <= 999
	main(729)	buffer overflow	(O[comp_dn] + O[temp]) <= 990
	main(729)	buffer overflow	(O[comp_dn] + O[temp]) <= 989
	main(729)	buffer overflow	(O[comp_dn] + O[temp]) <= 988
663	main(729)	buffer overflow	O[p] <= 999
	main(729)	buffer overflow	(O[comp_dn] + O[temp]) <= 982
	main(729)	buffer overflow	O[p] <= 999

O[p], O[comp_dn]

```
644     comp_size = dn_comp((const char *) exp_dn, comp_dn, 200, dnptrs, lastdnptr);
645     printf("uncomp_size = %d\n", strlen(exp_dn));
646     printf("comp_size = %d\n", comp_size);
647     printf("exp_dn = %s, comp_dn = %s\n", exp_dn, (char *) comp_dn);
648
649     for(i=0; i< comp_size; i++)
650       *p++ = *comp_dn++;
```

682	main(729)	buffer overflow	(O[comp_dn] + O[temp]) <= 962
	main(729)	buffer overflow	(O[comp_dn] + O[temp]) <= 961
	main(729)	buffer overflow	(O[comp_dn] + O[temp]) <= 960
684	main(729)	buffer overflow	O[p] <= 999
	main(729)	buffer overflow	(O[comp_dn] + O[temp]) <= 954
	main(729)	buffer overflow	(O[comp_dn] + O[temp]) <= 953
	main(729)	buffer overflow	(O[comp_dn] + O[temp]) <= 952
687	main(729)	buffer overflow	O[p] <= 999
	main(729)	buffer overflow	(O[comp_dn] + O[temp]) <= 946

comp_size:
result of library function

lookup documentation

add the assumptions

Only **2** More Warnings

Buffer Overflow Analysis Results

Click on filename(s) to access the analysis results on the source code

File	L.O.C.	#stmts	Safe	Warning	ERROR	Not reachable	Buffer Checks	Concrete	Intervals	Polyhedra	Indirect	No Proof
sig-bad_mac_mod.c	750	624	195	2	0	2	199	7	150	40	0	2
	750	632	195	2	0	2	199	7	150	40	0	2

Analysis time: 52.28277 sec

Summary of Warnings

Click on the function names to access the warnings

line nrs	function	number of warnings
283 - 606	rrextract	2

Original Summary of the results:

130 buffer access checks proven safe

2 buffer access checks proven unreachable

67 buffer access checks without proof

New Summary of the results:

195 buffer access checks proven safe

7 buffer access checks proven unreachable

2 buffer access checks without proof

```
544
545                 case NS_ALG_DSA:
546                   if (n != NS_DSA_SIG_SIZE)
547                     hp-> rcode = FORMERR;
548
549                   break;
550
551                 default:
552                   printf("DEFAULT ALG!\n");
553                   break;
554               }
555             if (hp-> rcode == FORMERR)
556                 return (-1);
557
558             printf ("memcpying %u bytes \n", (unsigned int) n);
559
560             /* BAD */
561             memcpy(cp1, cp, n);
562
563             cp += n;
564             cp1 += n;
565
566             /* compute size of data */
567             n = cp1 - (u_char *)data;
568             cp1 = (u_char *)data;
569             break;
570           }
571
572       default:
573         printf("unknown type %d", type);
574         return ((cp - rrp) + dlen);
```

Polyhedra	
to	(O[cp] + n) <= 1000
with	(n + O[cp]) >= (dlen + 32) O[cp1] >= 19 dlen >= (n + 18) n >= (dlen - 4139) (dlen + ((-n) - O[cp])) >= -131 O[cp1] <= 4158 type == 24 S[eom] == 1000 S[cp] == 1000 S[cp1] == 4140

Polyhedra	
to	(O[cp1] + n) <= 4140
with	(n + O[cp]) >= (dlen + 32) O[cp1] >= 19 dlen >= (n + 18) n >= (dlen - 4139) (dlen + ((-n) - O[cp])) >= -131 O[cp1] <= 4158 type == 24 S[eom] == 1000 S[cp] == 1000 S[cp1] == 4140

Value Tool

line	function rrextract	id	n	len
283	static int			
284	rrextract(u_char *msg, int msglen, u_char *rrp, u_char *dname, int namelen)			
285	{			
406	BOUNDS_CHECK(cp, 18);	0	[0, 100]	[40, 140]
407	memcpy(cp1, cp, 18);	0	[0, 100]	[40, 140]
408				
409	cp1 += 18;	0	[0, 100]	[40, 140]
410				
411	n = dn_expand(msg, eom, cp+18, (char *)cp1, (sizeof data) - 18);	0	[0, 100]	[40, 140]
412				
413	/* finally, we copy over the variable-length signature.			
414	Its size is the total data length, minus what we copied. */			
415	n = dlen - (NS_SIG_SIGNER + n);	0	<..,4122]	[40, 140]
416				
417	/* BAD */			
418	memcpy(cp1, cp, n);	0		[40, 140]
419	cp += n;	0		[40, 140]
420	cp1 += n;	0		[40, 140]

Case study: Summary

```
/* create the signature file this model needs */

int create5ig (u_char *buf) {
 u_char *p;
 char *temp, *temp1;
 u_char *comp_dn, *comp_dn2;
 char exp_dn[200], exp_dn2[200];
 u_char **dnptrs, **lastdnptr, **dnptrs2;
 int i,len = 0, comp_size;
 u_long now;


 dnptrs = (unsigned char **) malloc(2 * sizeof(unsigned char *));
 dnptrs2 = (unsigned char **) malloc(2 * sizeof(unsigned char *));

 comp_dn = (unsigned char *) malloc(200*sizeof(unsigned char));
 comp_dn2 = (unsigned char *) malloc(200*sizeof(unsigned char));

 temp1 = (char *) malloc(400*sizeof(char));

 temp = temp1;

 p = buf;

 strcpy(temp, "HEADER JUNK:");

 len += strlen(temp);

 while (*temp != '\0')
   *p++ = *temp++;

 strcpy(exp_dn, "lcs.mit.edu");

 *dnptrs++ = (u_char *) exp_dn;
 *dnptrs-- = NULL;

 lastdnptr = NULL;

 printf("Calling dn_comp.\n");
 comp_size = dn_comp((const char *) exp_dn, comp_dn, 200, dnptrs, lastdnptr);
 printf("uncomp_size = %d\n", strlen(exp_dn));
 printf("comp_size = %d\n", comp_size);
 printf("exp_dn = %s, comp_dn = %s\n", exp_dn, (char *) comp_dn);

 for(i=0;i<comp_size;i++)
   *p++ = *comp_dn++;

 len += comp_size;

 PUTSHORT(24, p); /* type = T_SIG = 24 */
 p += 2;

 PUTSHORT(C_IN, p);  /* class = C_IN = 1*/
 p += 2;

 PUTLONG(255, p);  /* ttl */
 p += 4;

 PUTSHORT(30, p);  /* dlen = len of everything starting with the covered byte (the length
of the entire resource record... we lie about it
                                                                                */
 p += 2;

 len += 10;

 PUTSHORT(15, p);  /* covered type */
 p += 2;

 PUTSHORT(256*2, p);  /* algorithm and labels.. MAKE ALG = 2,i.e default ALG*/
 p += 2;

 PUTLONG(255, p);  /* orig ttl */
 p += 4;

 now = time(NULL);

 printf("Signing at = %d\n", now);
 PUTLONG(now+20000, p);   /* expiration time */
 p  += 4;
 PUTLONG(now, p);         /* time signed */
 p += 4;

 PUTSHORT(100, p);          /* random key footprint */
 p += 2;

 len += 18;

 strcpy(exp_dn2, "ils.lcs.mit.edu."); /* signer */

 *dnptrs2++ = (u_char *) exp_dn2;
 *dnptrs2-- = NULL;
 lastdnptr = NULL;

 printf("Calling dn_comp..\n");
 comp_size = dn_comp((const char *) exp_dn2, comp_dn2, 200, dnptrs2, lastdnptr);
 printf("uncomp_size = %d\n", strlen(exp_dn2));
 printf("comp_size = %d\n", comp_size);
 printf("exp_dn2 = %s, comp_dn2 = %s\n", exp_dn2, (char *) comp_dn2);

 len += comp_size;

 for(i=0;i<comp_size;i++)
   *p++ = *comp_dn2++;

 for(i=0;i<11;i++)
 {
   PUTLONG(123, p);          /* fake signature */
   p += 4;
   len += 4;
 }

 return (p-buf);

}
```

Apply CodeHawk out-of-the-box

Result: 199 buffer access checks
130 proven safe
67 warnings

65 warnings eliminated in two steps:

29 eliminated by dn_expand model

36 eliminated by dn_comp model

Value tool aids in final diagnosis

After fixing the bug .....

```
/* create the signature file this model needs */

int createSig (u_char *buf) {
 u_char *p;
 char *temp, *temp1;
 u_char *comp_dn, *comp_dn2;
 char exp_dn[200], exp_dn2[200];
 u_char **dnptrs, **lastdnptr, **dnptrs2;
 int i,len = 0, comp_size;
 u_long now;


 dnptrs = (unsigned char **) malloc(2 * sizeof(unsigned char *));
 dnptrs2 = (unsigned char **) malloc(2 * sizeof(unsigned char *));

 comp_dn = (unsigned char *) malloc(200*sizeof(unsigned char));
 comp_dn2 = (unsigned char *) malloc(200*sizeof(unsigned char));

 temp1 = (char *) malloc(400*sizeof(char));

 temp = temp1;

 p = buf;

 strcpy(temp, "HEADER JUNK.");

 len += strlen(temp);

 while (*temp != '\0')
   *p++ = *temp++;

 strcpy(exp_dn, "lcs.mit.edu");

 *dnptrs++ = (u_char *) exp_dn;
 *dnptrs-- = NULL;

 lastdnptr = NULL;

 printf("Calling dn_comp.\n");
 comp_size = dn_comp((const char *) exp_dn, comp_dn, 200, dnptrs, lastdnptr);
 printf("uncomp_size = %d\n", strlen(exp_dn));
 printf("comp_size = %d\n",comp_size);
 printf("exp_dn = %s, comp_dn = %s\n", exp_dn, (char *) comp_dn);

 for(i=0; i<comp_size; i++)
   *p++ = *comp_dn++;

 len += comp_size;

 PUTSHORT(24, p); /* type = T_SIG = 24 */
 p += 2;

 PUTSHORT(C_IN, p);  /* class = C_IN = 1*/
 p += 2;

 PUTLONG(255, p);  /* ttl */
 p += 4;

 PUTSHORT(30, p);  /* dlen = len of everything starting with the covered byte (the length

of the entire resource record... we lie about it
                                                                */
 p += 2;

 len += 10;

 PUTSHORT(15, p);  /* covered type */
 p += 2;

 PUTSHORT(256*2, p);  /* algorithm and labels.. MAKE ALG = 2,i.e default ALG*/
 p += 2;

 PUTLONG(255, p);  /* orig ttl */
 p += 4;

 now = time(NULL);

 printf("Signing at = %d\n", now);
 PUTLONG(now+20000, p);   /* expiration time */
 p += 4;
 PUTLONG(now, p);         /* time signed */
 p += 4;

 PUTSHORT(100, p);          /* random key footprint */
 p += 2;

 len += 18;

 strcpy(exp_dn2, "sls.lcs.mit.edu");  /* signer */

 *dnptrs2++ = (u_char *) exp_dn2;
 *dnptrs2-- = NULL;
 lastdnptr = NULL;

 printf("Calling dn_comp.\n");
 comp_size = dn_comp((const char *) exp_dn2, comp_dn2, 200, dnptrs2, lastdnptr);
 printf("uncomp_size = %d\n", strlen(exp_dn2));
 printf("comp_size = %d\n", comp_size);
 printf("exp_dn2 = %s, comp_dn2 = %s\n", exp_dn2, (char *) comp_dn2);

 len += comp_size;

 for(i=0; i<comp_size; i++)
   *p++ = *comp_dn2++;

 for(i=0; i<11; i++)
 {
  PUTLONG(123, p);          /* fake signature */
   p += 4;
   len += 4;
 }

 return (p-buf);

}
```

Independently checkable proofs for **all** buffer accesses

buffer overflow	O[V#202] + 1 <= S[V#202]
calling context	main@729

Simplification (cstCopyProp)	
to	O[tmp___12] + 1 <= S[tmp___12]
with	O[V#202] = O[tmp___12] S[V#202] = S[tmp___12]

Abstraction	
O[tmp___12] + 1 <= S[tmp___12]	(O[tmp___12] + 1) <= S[tmp___12]

Intervals	
to	(O[tmp___12] + 1) <= S[tmp___12]
with	O[tmp___12] >= 0 S[tmp___12] = 200

Polyhedra	
to	i <= 199
with	O[temp] <= 399 O[temp] >= 0 O[comp_dn] >= 1 comp_size <= 199 comp_size >= O[comp_dn] ((-i) - (-O[comp_dn])) == 1 S[p] == 1000 (O[p] + ((-O[temp]) - O[comp_dn])) == 0 S[temp] == 400 S[comp_dn2] == 200 O[comp_dn2] == 0 S[comp_dn] == 200

No mention of abstract interpretation !!

You don’t need to be an expert in formal methods
or abstract interpretation to use an
abstract-interpretation-based analyzer

Domains Used

Buffer Overflow Analysis Results

Click on filename(s) to access the analysis results on the source code

File	L.O.C.	#stmts	Safe	Warning	ERROR	Not reachable	Buffer Checks	Concrete	Intervals	Polyhedra	Indirect	No Proof
sig-bad_mac_mod.c	750	624	195	2	0	2	199	7	150	40	0	2
	750	632	195	2	0	2	199				0	2

Analysis time: 52.28277 sec

Summary of Warnings

Click on the function names to access the warnings

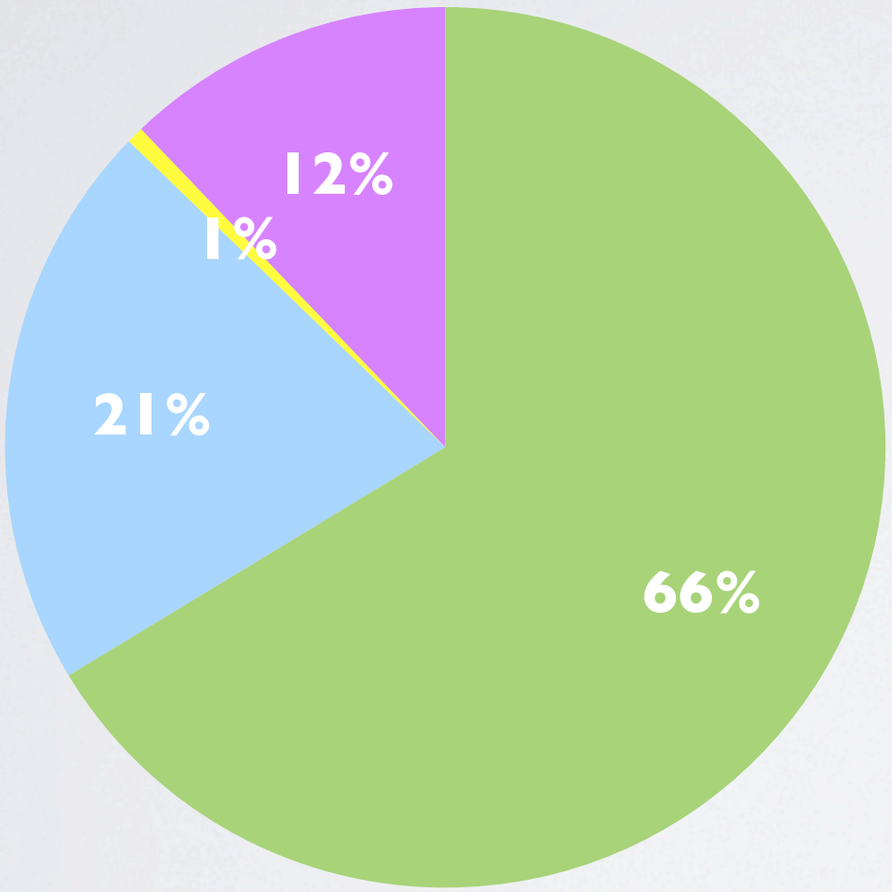
line nrs	function	number of warnings
283 - 606	rrextract	2

Domains required for proofs of safety:

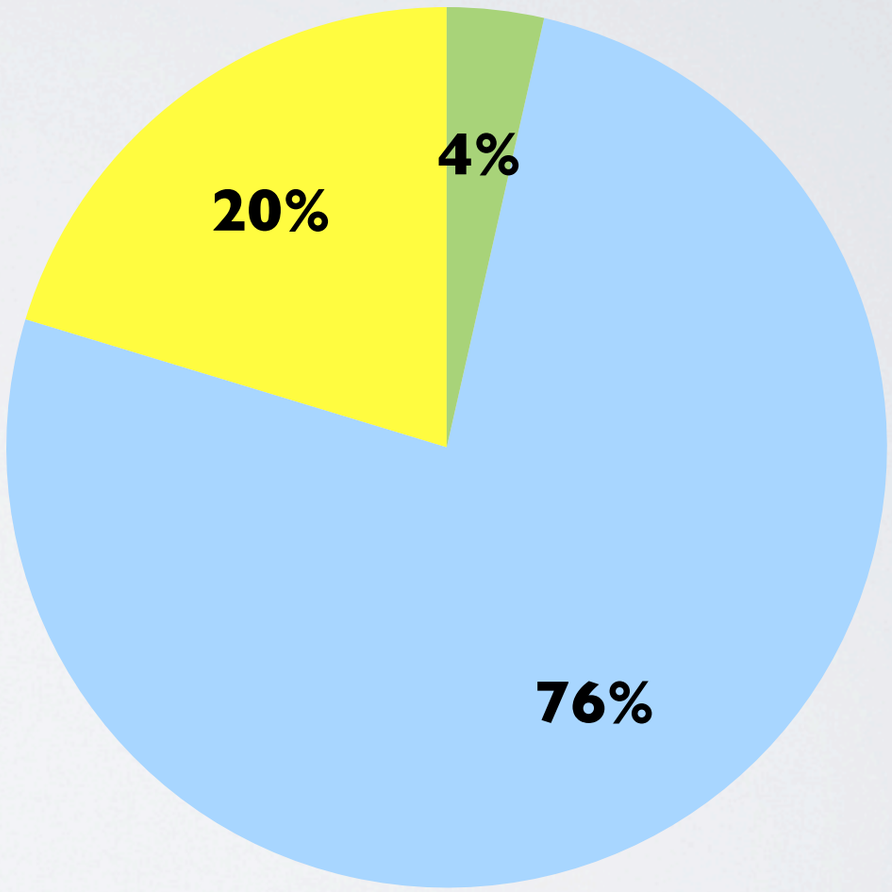
Concrete (constants):	7
Intervals:	150
Polyhedra:	40

Use of Domains

SAMATE small benchmarks
115-1278
(2569 proven checks)



SAMATE medium benchmark 1291



Concrete Intervals Polyhedra Indirect

Boeing Case Study

Example flight software from Boeing developed on a prior research project that included contributions from a group of academic researchers

Statistics: 2069 lines of code (4 files) ; 2034 statements

Apply CodeHawk

out-of-the-box analysis results:

Buffer Overflow Analysis Results

Click on filename(s) to access the analysis results on the source code

File	L.O.C.	#stmts	Safe	Warning	ERROR	Not reachable	Buffer Checks	Concrete	Intervals	Polyhedra	Indirect	No Proof
hifi_Data.c	965	689	332	84	0	0	416	84	248	0	0	84
interp.c	254	189	1078	2794	0	0	3872	0	1078	0	0	2794
main.c	19	2	0	0	0	0	0	0	0	0	0	0
plant.c	831	1174	1822	74	0	0	1896	1676	146	0	0	[illegible]
	2069	2054	3232	2952	0	0	6184	1760	1472	0	[illegible]	2952

Analysis time: 1183.29907 sec

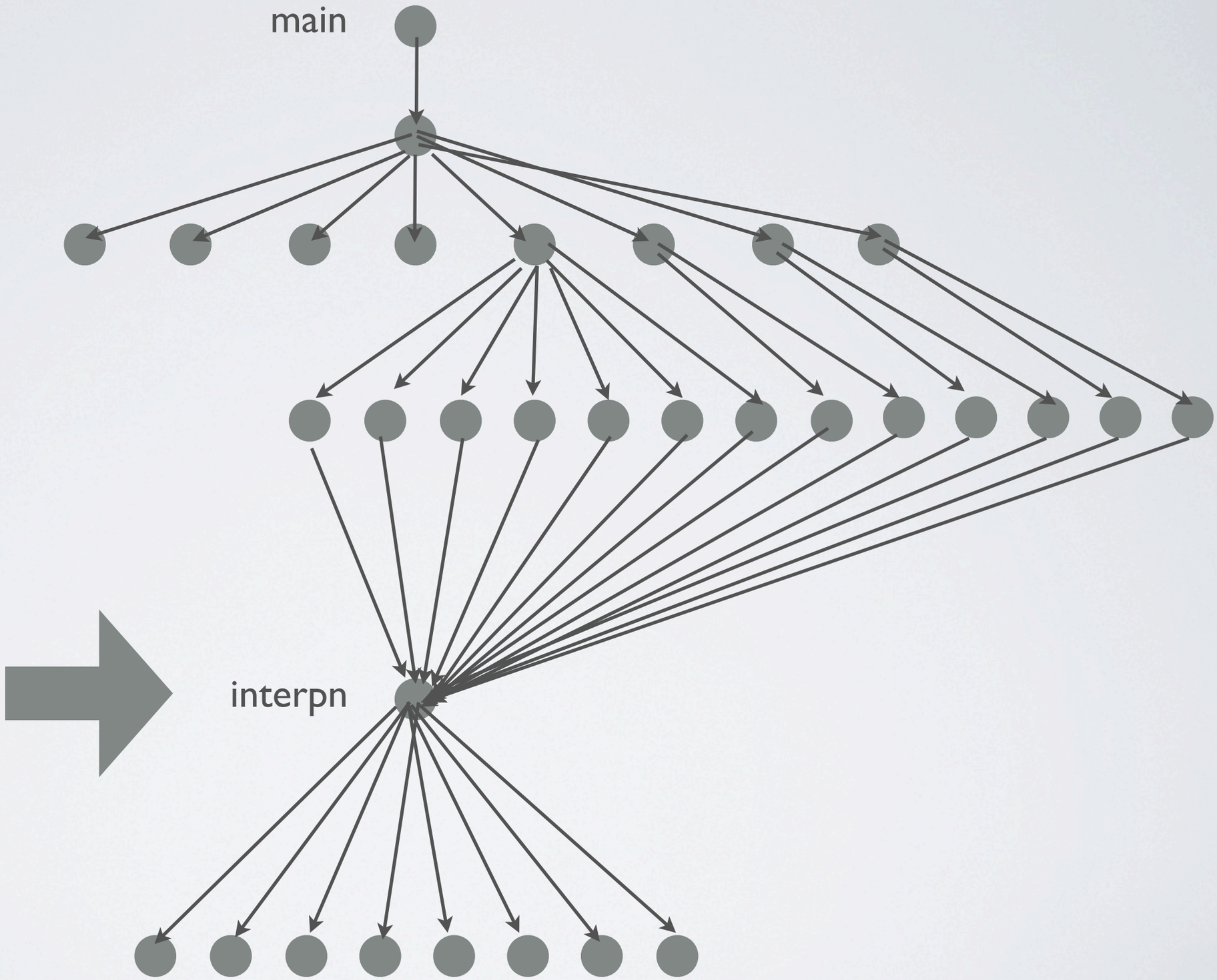
Summary of the results:

3232 buffer access checks proven safe

2952 buffer access checks without proof

Analysis time: > 20 min

System Structure



Divide and conquer

Buffer Overflow Analysis Results

Click on filename(s) to access the analysis results on the source code

File	L.O.C.	#stmts	Safe	Warning	ERROR	Not reachable	Buffer Checks	Concrete	Intervals	Polyhedra	Indirect	No Proof
hifi_Data.c	965	689	332	84	0	0	416	84	248	0	0	84
interp.c	254	189	35	141	0	0	176	0	35	0	0	141
main.c	19	2	0	0	0	0	0	0	0	0	0	0
plant.c	831	1174	1822	74	0	0	1896	1676	146	0	0	74
	2069	2054	2189	299	0	0	2488	1760	429	0	0	299

Analysis time: 197.89787 sec

Summary of the results:

2189 buffer access checks proven safe

299 buffer access checks without proof

Analysis time: ~ 3 min

hifi_Data.c: Create more variables

Warnings in hifi_Data.c:

	172	ndinfo.nDimension = nDimension;
	173	ndinfo.nPoints = intVector(nDimension);
??	174	ndinfo.nPoints[0] = 14; /* Alpha npoints */
??	175	ndinfo.nPoints[1] = 19; /* Beta npoints */

array access within struct

Expand structs: Create new variables for all struct fields

Result: eliminates 84 warnings

plant.c: Introduce lemmas

```
427
428     if (k < = -2) { /*bounds of table for extrapolation*/
429         k = -1;
430     }
431     else if (k > = 9){
432         k = 8;
433     }
434     da = s - k; /* amount from closest lower grid point*/
435
436     L = k + fix(1.1*(da/fabs(da)));
437     k = k + 3;
438     L = L + 3;
```

```
?? ✓✓✓✓✓✓ 439     coeff[0] = A[0][k-1] + fabs(da)*(A[0][L-1] - A[0][k-1]);
?? ✓✓✓✓✓✓ 440     coeff[1] = A[1][k-1] + fabs(da)*(A[1][L-1] - A[1][k-1]);
?? ✓✓✓✓✓✓ 441     coeff[2] = A[2][k-1] + fabs(da)*(A[2][L-1] - A[2][k-1]);
?? ✓✓✓✓✓✓ 442     coeff[3] = A[3][k-1] + fabs(da)*(A[3][L-1] - A[3][k-1]);
?? ✓✓✓✓✓✓ 443     coeff[4] = A[4][k-1] + fabs(da)*(A[4][L-1] - A[4][k-1]);
?? ✓✓✓✓✓✓ 444     coeff[5] = A[5][k-1] + fabs(da)*(A[5][L-1] - A[5][k-1]);
?? ✓✓✓✓✓✓ 445     coeff[6] = A[6][k-1] + fabs(da)*(A[6][L-1] - A[6][k-1]);
?? ✓✓✓✓✓✓ 446     coeff[7] = A[7][k-1] + fabs(da)*(A[7][L-1] - A[7][k-1]);
?? ✓✓✓✓✓✓ 447     coeff[8] = A[8][k-1] + fabs(da)*(A[8][L-1] - A[8][k-1]);
          448
```

L = k + some floating point operation (v) ; L = [-∞ , ∞]

Lemma: $\forall v . \; -1 \leq \ldots . (v) \leq 1$ L = [min(k) -1 , max(k) + 1]

Use of Lemmas

line	function cxcm	id	k	L	m	n
260	void cxcm(double alpha, double dele, double *coeff){					
292						
293	L = k + fix(1.1*(da/fabs(da)));	0	[-1, 8]			
294						
295	s = dele/12.0;	0	[-1, 8]	[-2, 9]		
296	m = fix(s);	0	[-1, 8]	[-2, 9]		

use of lemma

293	Custom assertion (by lemma): insertion of -1 <= tmp___0 <= 1 for tmp___0 = fix (1.1 * (da/fabs(da)))

301	m = 1;	0	[-1, 8]	[-2, 9]	[2,..>	
302	}					
303	de = s - m;	0	[-1, 8]	[-2, 9]	[-1, 1]	
304	n = m + fix(1.1*de/fabs(de));	0	[-1, 8]	[-2, 9]	[-1, 1]	
305						
306	k = k + 3;	0	[-1, 8]	[-2, 9]	[-1, 1]	[-2, 2]

use of lemma

304	Custom assertion (by lemma): insertion of -1 <= tmp___2 <= 1 for tmp___2 = fix (1.1 * de/fabs(de))

Buffer Overflow Analysis Results

Click on filename(s) to access the analysis results on the source code

File	L.O.C.	#stmts	Safe	Warning	ERROR	Not reachable	Buffer Checks	Concrete	Intervals	Polyhedra	Indirect	No Proof
hifi_Data.c	965	689	416	0	0	0	416	84	332	0	0	0
interp.c	254	189	0	0	0	0	0	0	0	0	0	0
main.c	19	2	0	0	0	0	0	0	0	0	0	0
plant.c	831	1190	1888	8	0	0	1896	1676	212	0	0	8
	2069	2070	2304	8	0	0	2312	1760	544	0	0	8

Analysis time: 205.44681 sec

Result:

hifi_Data.c	84 warnings eliminated by struct expansion	0 warnings left
plant.c	66 warnings eliminated by introduction of lemma	8 warnings left

unproven errors

Analyze interpn standalone

Buffer Overflow Analysis Results

Click on filename(s) to access the analysis results on the source code

File	L.O.C.	#stmts	Safe	Warning	ERROR	Not reachable	Buffer Checks	Concrete	Intervals	Polyhedra	Indirect	No Proof
interp.c	254	189	35	141	0	0	176	0	35	0	0	141
	254	189	35	141	0	0	176	0	35	0	0	141

Analysis time: 14.01881 sec

Buffer Overflow Analysis Results

Click on filename(s) to access the analysis results on the source code

File	L.O.C.	#stmts	Safe	Warning	ERROR	Not reachable	Buffer Checks	Concrete	Intervals	Polyhedra	Indirect	No Proof
interp.c	254	189	68	108	0	0	176	0	35	33	0	108
	254	189	68	108	0	0	176	0	35	33	0	108

Analysis time: 177.81718 sec

Analyze interpn standalone:

Intervals:	141 warnings	14 sec
Polyhedra:	108 warnings	3 min

We need context: Define the interface

```
typedef struct  {
    int dim;
    int *points;
    }  S;
```

interpn(double **X, double *Y, double *x, S s)

size(s.points) = s.dim

size(x) = s.dim

size(X) = s.dim

$\forall$ i: 0 .. s.dim-1 . size(X[i]) = s.points[i]

size(Y) = $\prod_{i = 0..s.dim-1}$ s.points[i]

4 + s.dim verification conditions for every call to **interpn**

(not included in results yet)

Provide context

Analyze interpn standalone:

Intervals:	141 warnings	14 sec
Polyhedra:	108 warnings	3 min

Provide context:

Polyhedra:	74 warnings	3 min
Custom size domain:	35 warnings	3 min

Boeing Case Study: Analysis Summary

We started with: 2952 checks without proof Analysis time: > 20 min

Decomposition: 299 checks without proof

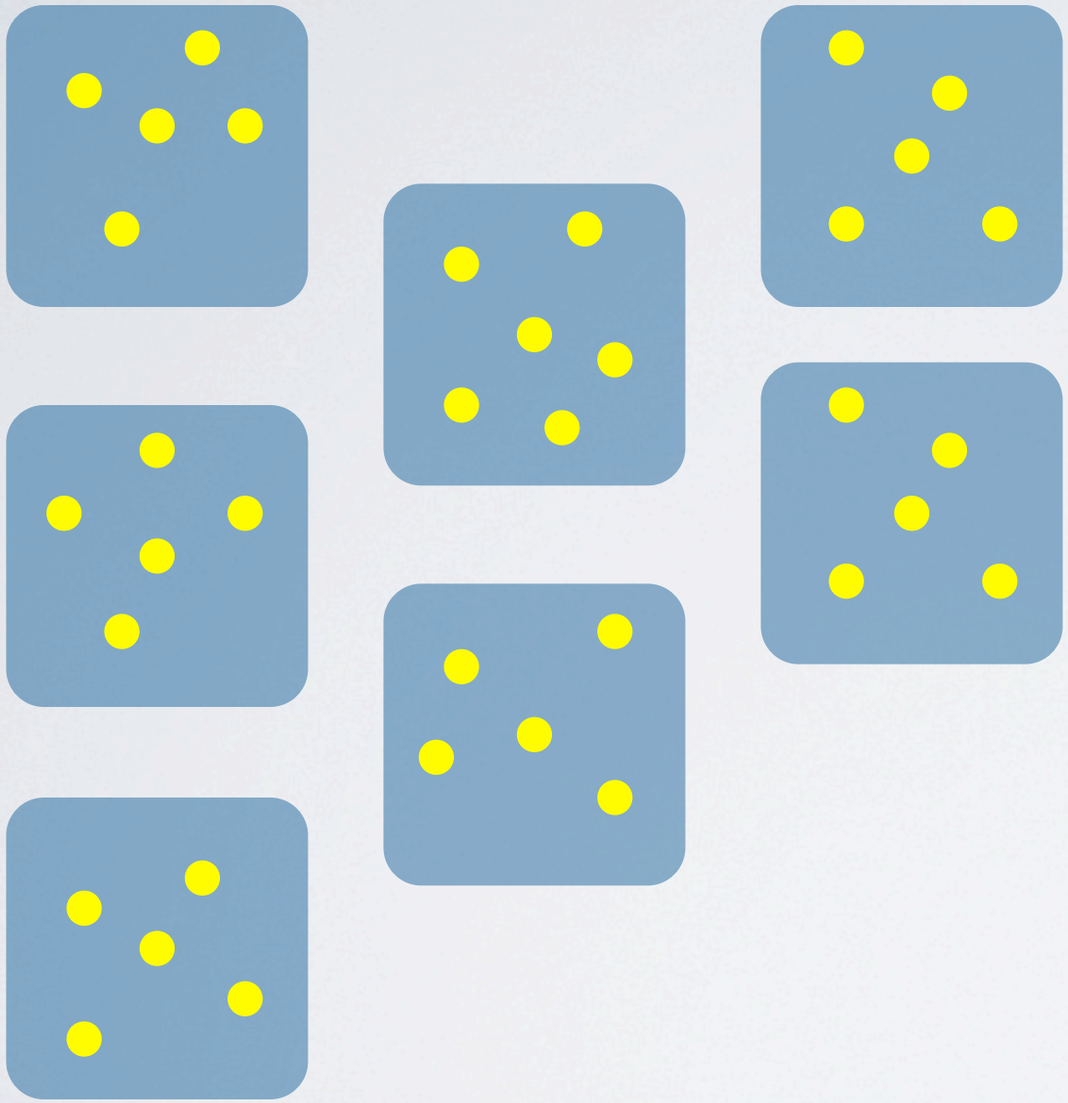
Elimination of false positives

Struct expansion:	84
Lemmas	66
Polyhedra	33
Context assumptions	34
Custom size domain	39

Checks without proof left: 43 (8 of which are confirmed bugs) Analysis time: 6 min

Custom analyzer that can be reused on software with the same architecture

Analysis of systems of systems



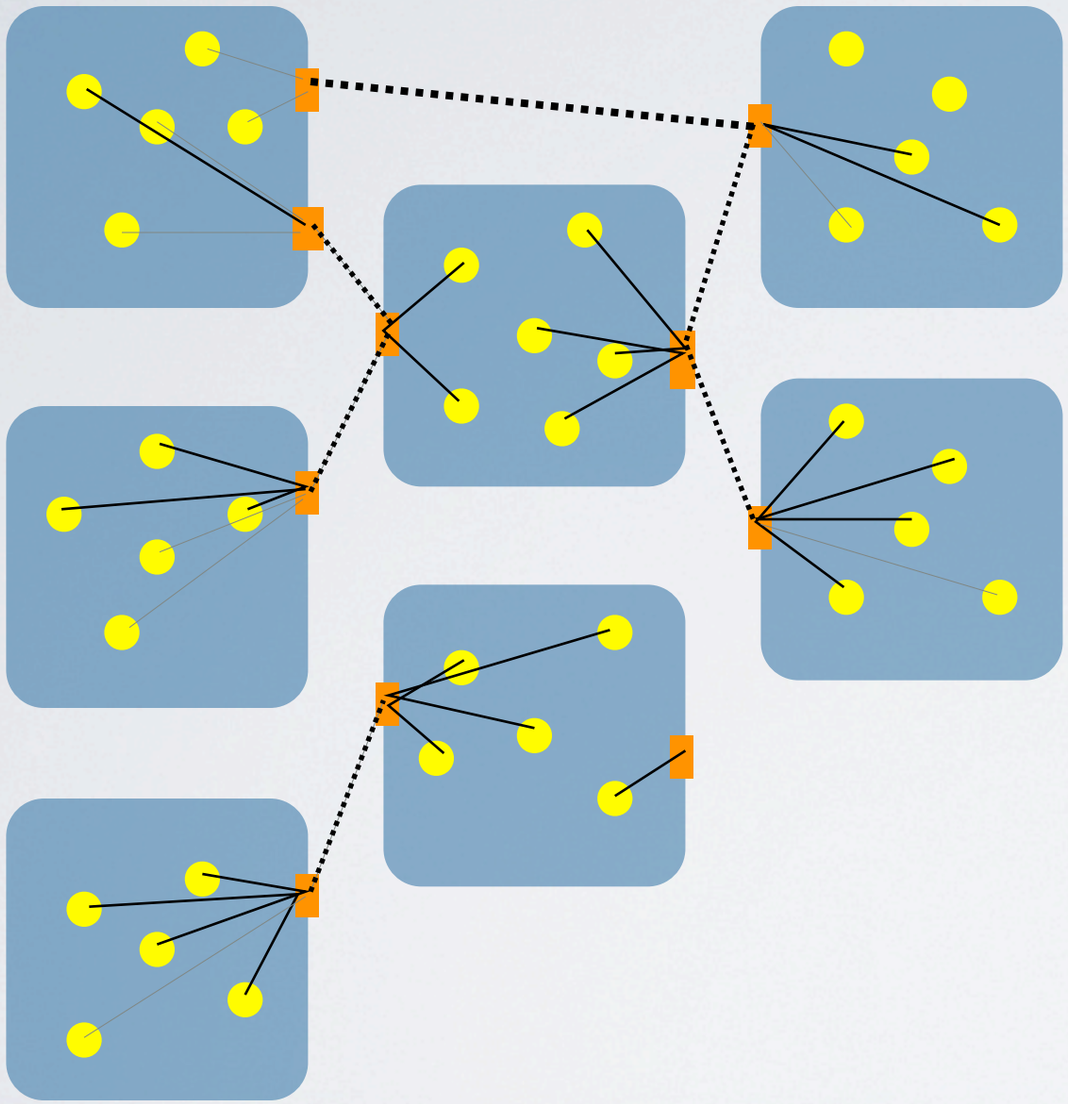
Application of generic
buffer overflow analyzer

Too many warnings

Too many false positives

User is overwhelmed

Analysis of systems of systems



Use backward analysis to

- relate warnings to user/ device input values
- relate warnings to function arguments
- collect warnings that originate from the same input condition
- identify input conditions that eliminate warnings
- assist the user in constructing an API

Position

	Bug finders	Theorem Provers	CodeHawk
Generality	++	– –	+
Scalability	++	– –	+
Application to source code	++	?	++
Usable by software engineers	++	– –	+
Assurance	– –	++	++
Possibility for independent checking of evidence	– –	++	++

Planned enhancements

SBIR driven

- SAGE: Desktop tool to visualize intermediate results, gain insight in the code
- IFEX: verification of systems of systems

Market-driven

- GCC front end: provide support for other programming language (Java, C++)
- More language-level properties
- Architectural customization to reduce cost of certification

Customer driven

- Boeing
- Lockheed Martin

Application-level properties

Conclusions

Promising and proven technology

- Key distinction for assurance: no false negatives
- Can be used for verification and to find defects
- Can be specialized for customer needs

Technology transfer

- Three-day class to learn first-level customization
- No formal methods or abstract-interpretation knowledge required for use

Current situation

- New abstract interpretation engine design is complete
- Ready for customer requests