Sound Non-statistical Clustering of Static Analysis Alarms

Woosuk Lee, Wonchan Lee, and Kwangkeun Yi

Seoul National University

VMCAI '12 @ Philadelphia

Contents

Contents

- Problem & Our approach
- Overall result
- Clusterings
- Framework
- Conclusion

Motivation

- Manual alarm investigation is painful!
- Our commercial tool Sparrow
  - For 3.5 MLOC program
  - Over 1060 alarms are reported.

Our approach

- Cluster similar alarms of the same origin
- Clusters have its own representatives (= dominant alarms).
- Users may inspect only dominant alarms.

How It Works

gzip-1.2.4

```
void pqdownheap(int k)
{
  int j = 2 * k;
  while(j <= heap_len)
  {
   heap[k] = heap[j];
   k = j;
   j = 2 * j;
  }
  heap[k] = ...;
}
```

3 buffer-overflow alarms

How It Works

gzip-1.2.4

```
void pqdownheap(int k)
{
  int j = 2 * k;
  while(j <= heap_len)
  {
   heap[k] = heap[j];
   k = j;
   j = 2 * j;
  }
  heap[k] = ...;
}
```

A user identifies **heap[j]** to be false

How It Works

gzip-1.2.4

```
void pqdownheap(int k)
{
  int j = 2 * k;
  while(j <= heap_len)
  {
    heap[k] = heap[j];
    k = j;
    j = 2 * j;
  }
  heap[k] = ...;
}
```

The others are automatically deduced false.

($\because$ loop invariant : $j = 2k$)

How It Works

gzip-1.2.4

```
void pqdownheap(int k)
{
  int j = 2 * k;
  while(j <= heap_len)
  {
    heap[k] = heap[j];
    k = j;
    j = 2 * j;
  }
  heap[k] = ...;
}
```

Users may check only **heap[j]** instead of all.

Contents

- Problem & Our approach
- Overall result
- Clusterings
- Framework
- Conclusion

Results: Example (1/2)

```
char invmergerules[8];
char invmergerules_nn[8];

int lookup (char *rule) {
  for (i = 1; invmergerules[i]; i++)
    if (strcasecmp(rule, invmergerules_nn[i] == 0)
      return (i);
}

int rule (struct sketch *s, int rule, int rcount) {
  if (debug)
    printf("%s %d", invmergerules[rule], rcount);

}

int apply (char *rule, struct sketch *sketch) {
  if (code = lookup (rule))
       res = rule (sketch, code, rcount);
  ...
```

Appcontour 1.1.0

Results: Example (2/2)

```
char cboard[64];
char ephash[64];

void MakeMove(int side, int *move) {

  fpiece = cboard[f];
  tpiece = cboard[t];

  if (fpiece == pawn && abs(f-t) == 16) {
    sq = (f + t) / 2;

    HashKey ^= ephash[sq];
  }
}
```

gnuchess-5.05

Results: Overall Effectiveness

Program	LOC	# Alarms	# Alarms after Clustering	% Reduction
nlkain-1.3	831	124	93	25%
polymorph-0.4.0	1,357	25	13	48%
ncompress-4.2.4	2,195	66	30	55%
sbm-0.0.4	2,467	237	125	47%
stripcc-0.2.0	2,555	194	127	35%
barcode-0.96	4,460	435	302	31%
129.compress	5,585	57	29	49%
archimedes-0.7.0	7,569	711	132	81%
man-1.5h1	7,232	276	165	40%
gzip-1.2.4	11,213	385	263	32%
combine-0.3.3	11,472	733	294	60%
gnuchess-5.05	11,629	976	333	66%
bc-1.06	12,830	593	198	67%
coan-4.2.2	22,414	461	291	37%
grep-2.5.1	31,154	115	85	26%
lsh-2.0.4	110,898	616	264	57%
Total	245,861	6,004	2,744	**54%**

Contents

Clusterings

- On top of the interval-domain-based industrialized commercial tool Sparrow
- Three alarm clustering analyses
  1. Syntactic clustering
  2. Semantic clustering with non-relational analysis
  3. Semantic clustering with relational analysis

1. Syntactic Clustering

```
while (*optarg && *optarg >= '0' && *optarg <= '9')
   val = *optarg  - '0';
   optarg++;
```

- Expressions are the same.
- Variables have the same definition point.

2. Semantic Clustering
(w/ non-relational analysis)

- key idea (alarm dependence)

```
int buffer[10];
...
buffer[i] = 10;     // i = [0, ∞]
...
j = i / 3;          // j = [0, ∞]
foo = buffer[j];    // j = [0, ∞]
```

Two alarms occurred.

2. Semantic Clustering
(w/ non-relational analysis)

- key idea (alarm dependence)

```
int buffer[10];
...
buffer[i] = 10;     // i = [0, 9]
...
j = i / 3;          // j = [0, ∞]
foo = buffer[j];    // j = [0, ∞]
```

assume **buffer[i]** false

2. Semantic Clustering
(w/ non-relational analysis)

- key idea (alarm dependence)

```
int buffer[10];
...
buffer[i] = 10;     // i = [0, 9]
...
j = i / 3;          // j = [0, 3]
foo = buffer[j];    // j = [0, 3]
```

propagate the refinement

2. Semantic Clustering
(w/ non-relational analysis)

- key idea (alarm dependence)

```
int buffer[10];
...
buffer[i] = 10;     // i = [0, 9]
...
j = i / 3;          // j = [0, 3]
foo = buffer[j];    // j = [0, 3]
```

It kills the other.

2. Semantic Clustering
(w/ non-relational analysis)

- key idea (alarm dependence)

```
int buffer[10];
...
buffer[i] = 10;     // i = [0, 9]
...
j = i / 3;          // j = [0, 3]
foo = buffer[j];    // j = [0, 3]
```

If **buffer[i]** is false, so is the other.

We cluster two alarms.

3. Semantic Clustering
(w/ relational analysis)

- key idea (alarm dependence)

```
char * p, * str;

for (p = str; *p; p++)     // 0 ≤ p.offset
  *p = TOLOWER(*p);



if (*str == ‘*’) ...       // 0 ≤ str.offset
```

Two alarms occurred.

3. Semantic Clustering
(w/ relational analysis)

- key idea (alarm dependence)

```
char * p, * str;

for (p = str; *p; p++)     // 0 ≤ p.offset < p.size
  *p = TOLOWER(*p);



if (*str == '*') ...       // 0 ≤ str.offset
```

assume ***p** false

3. Semantic Clustering
(w/ relational analysis)

- key idea (alarm dependence)

```
char * p, * str;

for (p = str; *p; p++)      // 0 ≤ p.offset < p.size
  *p = TOLOWER(*p);

// Loop inv :
//   0 ≤ str.offset ≤ p.offset < p.size = str.size

if (*str == ‘*’) ...        // 0 ≤ str.offset < str.size
```

propagate the refinement

3. Semantic Clustering
(w/ relational analysis)

- key idea (alarm dependence)

```
char * p, * str;

for (p = str; *p; p++)     // 0 ≤ p.offset < p.size
  *p = TOLOWER(*p);

// Loop inv :
//   0 ≤ str.offset ≤ p.offset < p.size = str.size

if (*str == '*') ...       // 0 ≤ str.offset < str.size
```

It kills the other.

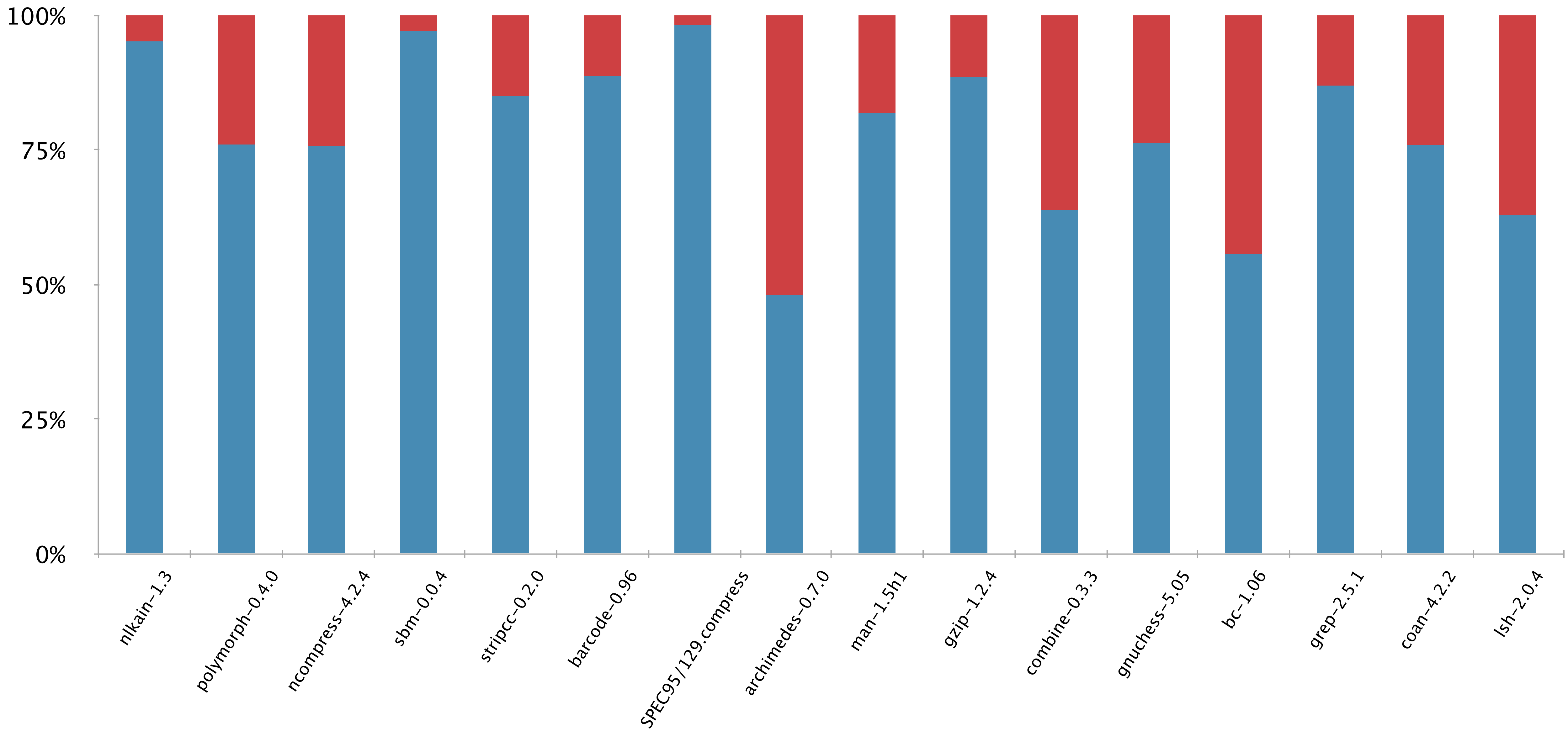
3. Semantic Clustering
(w/ relational analysis)

- key idea (alarm dependence)

```
char * p, * str;

for (p = str; *p; p++)     // 0 ≤ p.offset < p.size
  *p = TOLOWER(*p);

// Loop inv :
//   0 ≤ str.offset ≤ p.offset < p.size = str.size

if (*str == '*') ...       // 0 ≤ str.offset < str.size
```

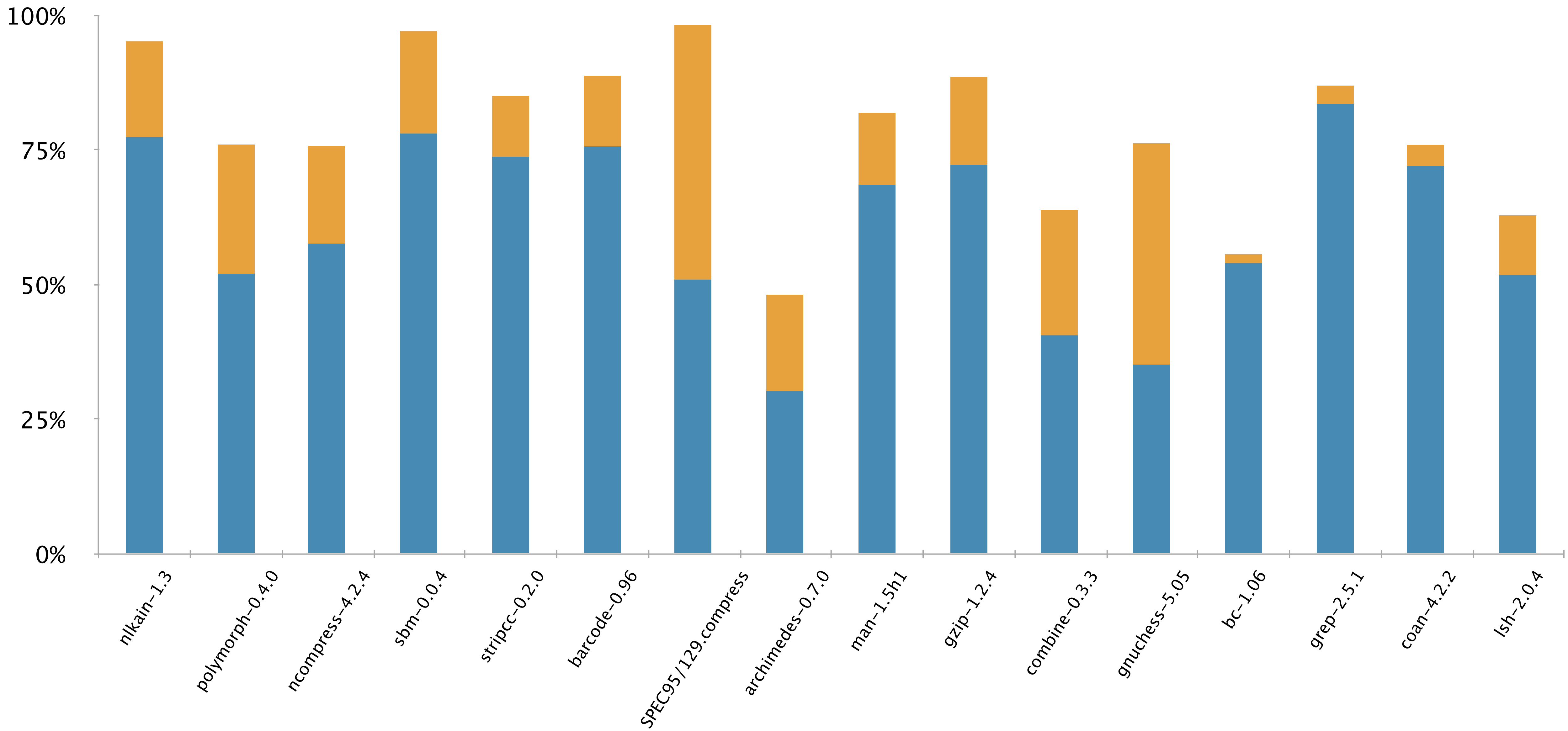
If ***p** is false, so is the other.
We cluster two alarms.

Result



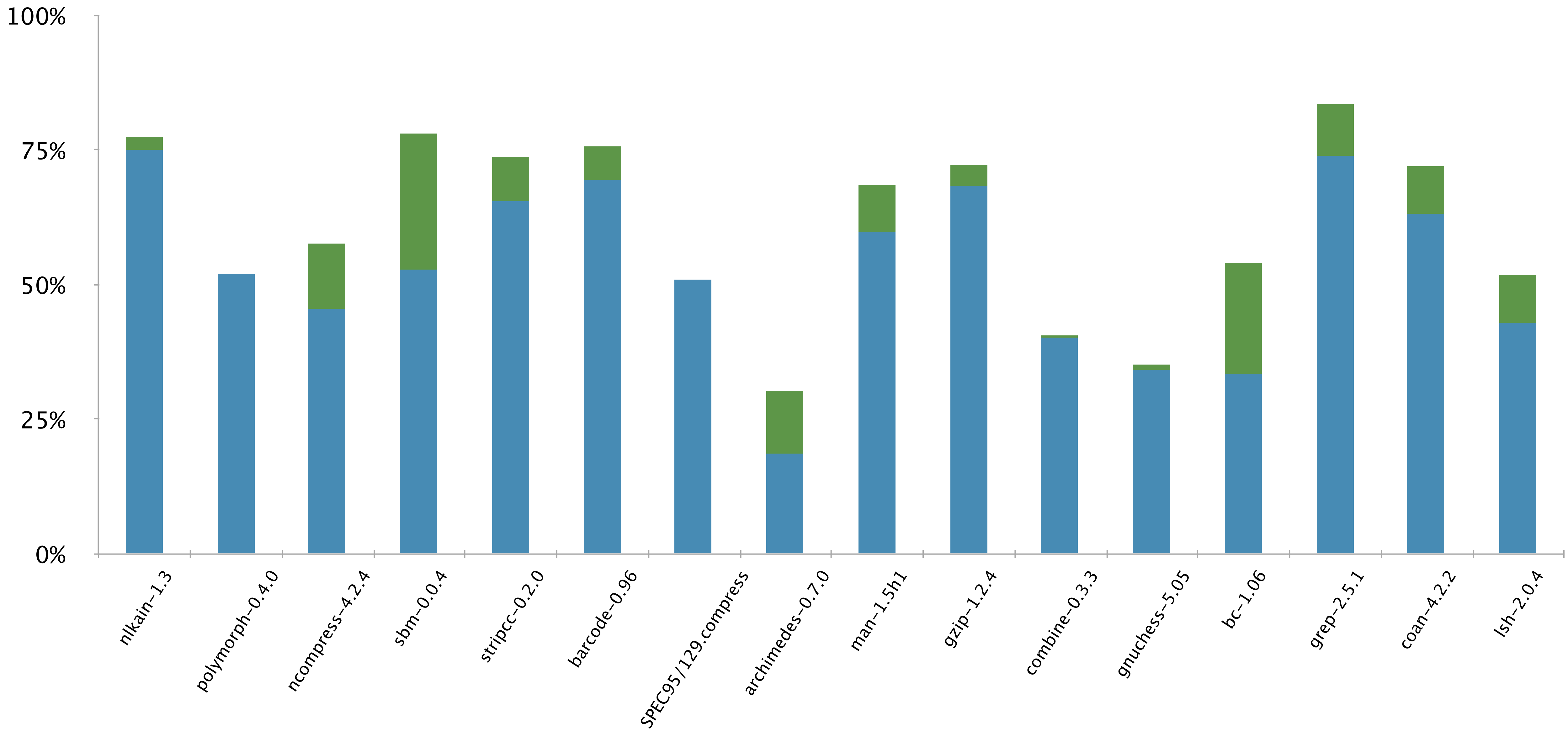
- Syntactic clustering

Result



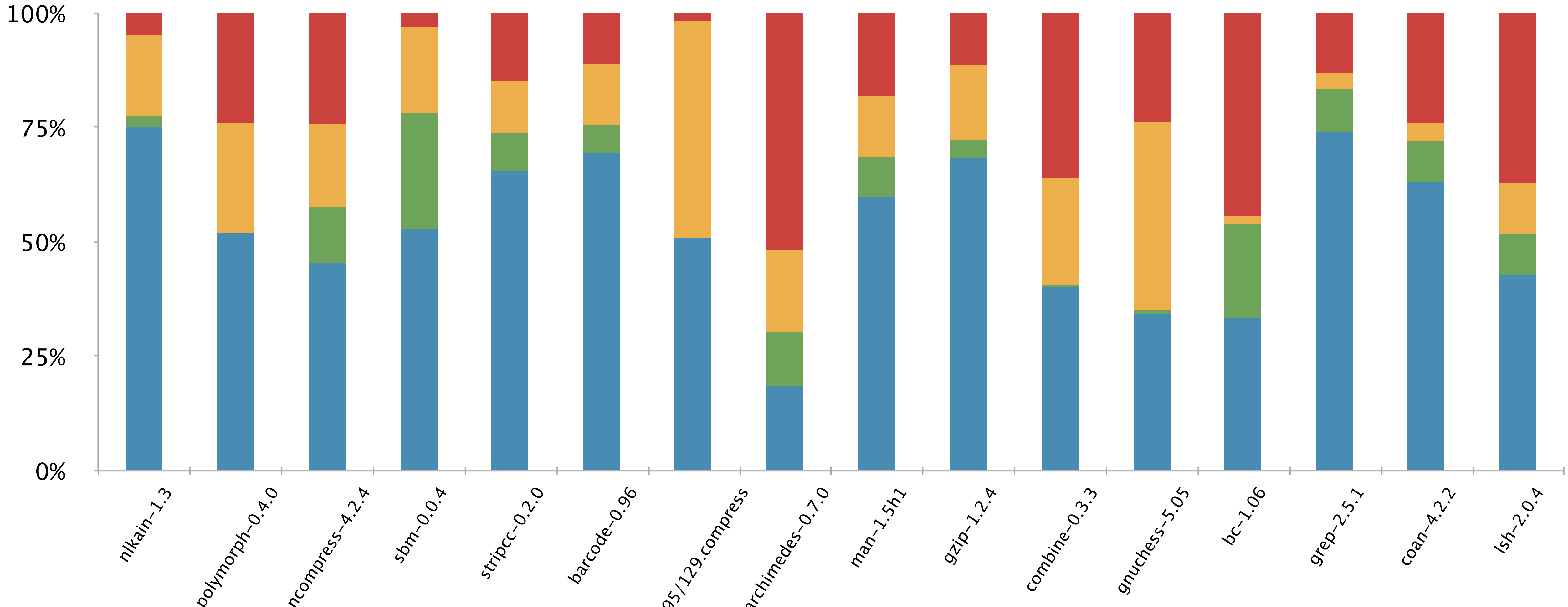
- Semantic clustering (non-relational)

Result



- Semantic clustering (relational)

Overall



	Syntactic	+Semantic (non-relational)	+Semantic (relational)
#Alarm↓	28%	18%	8%
Time↑	0%	4%	88%

Contents

- Problem & Our approach
- Overall result
- Clusterings
- Framework
- Conclusion

Formalization & Soundness

- Three methods have the same strategy.
  - (1) Assume some alarms are false
    (2) propagate the refinement
    (3) get alarm dependences
- We formalize a general alarm clustering method, and prove the correctness.

Alarm Clustering Framework

- Three clusterings are instances of the framework.
- Applicable to any semantics-based static analysis
- Guarantees the soundness of alarm clustering

Notations

- Set of program points Φ , and set of the states S
- Concrete semantics $[\![P]\!] : \Phi \to 2^S$
- Galois connection $2^S \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \hat{S}$
- Abstract semantics $\hat{T} : \Phi \to \hat{S}$

$$\forall \varphi \in \Phi . \alpha([\![P]\!](\varphi)) \sqsubseteq \hat{T}(\varphi)$$

$$\hat{T} = \mathsf{fix}\, \hat{F}$$

- Erroneous states $\Omega : \Phi \to 2^S$

Goal

- For any two alarms at $\varphi_1, \varphi_2 \in \Phi$, to find concrete dependence

$$[\![P]\!](\varphi_1) \cap \Omega(\varphi_1) = \varnothing \implies [\![P]\!](\varphi_2) \cap \Omega(\varphi_2) = \varnothing$$

- Using abstract dependence

Abstract Alarm Dependence $\varphi_1 \rightsquigarrow \varphi_2$

Definition 1 **($\varphi_1 \rightsquigarrow \varphi_2$)** *Given two alarms φ_1 and φ_2, alarm φ_2 has abstract dependence on alarm φ_1 if and only if,*

$$\gamma(\tilde{T}_{\varphi_1}(\varphi_2)) \cap \Omega(\varphi_2) = \varnothing$$

Abstract Alarm Dependence $\varphi_1 \rightsquigarrow \varphi_2$

Definition 1 **($\varphi_1 \rightsquigarrow \varphi_2$)** *Given two alarms φ_1 and φ_2, alarm φ_2 has abstract dependence on alarm φ_1 if and only if,*

$$\gamma(\tilde{T}_{\varphi_1}(\varphi_2)) \cap \Omega(\varphi_2) = \emptyset$$

where

$$\tilde{T}_{\varphi_1} = \mathsf{fix}\, \lambda X. \boxed{} \hat{F}(X) \qquad (\hat{T} = \mathsf{fix}\, \hat{F})$$

Abstract Alarm Dependence $\varphi_1 \rightsquigarrow \varphi_2$

Definition 1 **($\varphi_1 \rightsquigarrow \varphi_2$)** *Given two alarms φ_1 and φ_2, alarm φ_2 has abstract dependence on alarm φ_1 if and only if,*

$$\gamma(\tilde{T}_{\varphi_1}(\varphi_2)) \cap \Omega(\varphi_2) = \varnothing$$

where

$$\tilde{T}_{\varphi_1} = \mathsf{fix}\, \lambda X.\hat{T}_{\neg\varphi_1} \sqcap \hat{F}(X)$$

38

Abstract Alarm Dependence $\varphi_1 \leadsto \varphi_2$

Definition 1 **($\varphi_1 \leadsto \varphi_2$)** *Given two alarms φ_1 and φ_2, alarm φ_2 has abstract dependence on alarm φ_1 if and only if,*

$$\gamma(\tilde{T}_{\varphi_1}(\varphi_2)) \cap \Omega(\varphi_2) = \varnothing$$

where

$$\tilde{T}_{\varphi_1} = \mathsf{fix}\, \lambda X. \hat{T}_{\neg\varphi_1} \sqcap \hat{F}(X)$$

$$\hat{T}_{\neg\varphi_1} = \hat{T}\{\varphi_1 \mapsto \hat{T}(\varphi_1)\, \hat{\ominus}\, \alpha(\Omega(\varphi_1))\}$$

39

Abstract Alarm Dependence $\varphi_1 \rightsquigarrow \varphi_2$

Definition 1 ($\varphi_1 \rightsquigarrow \varphi_2$) *Given two alarms φ_1 and φ_2, alarm φ_2 has abstract dependence on alarm φ_1 if and only if,*

$$\gamma(\tilde{T}_{\varphi_1}(\varphi_2)) \cap \Omega(\varphi_2) = \emptyset$$

where

$$\tilde{T}_{\varphi_1} = \mathsf{fix}\,\lambda X.\hat{T}_{\neg\varphi_1} \sqcap \hat{F}(X)$$

$$\hat{T}_{\neg\varphi_1} = \hat{T}\{\varphi_1 \mapsto \underline{\hat{T}(\varphi_1)\,\hat{\ominus}\,\alpha(\Omega(\varphi_1))}\}$$

slice out error states at φ_1 in a way that it approximates

$$[\![P]\!](\varphi_1) \ominus \Omega(\varphi_1)$$

Abstract Alarm Dependence $\varphi_1 \rightsquigarrow \varphi_2$

Definition 1 ($\varphi_1 \rightsquigarrow \varphi_2$) *Given two alarms* φ_1 *and* φ_2*, alarm* φ_2 *has abstract dependence on alarm* φ_1 *if and only if,*

$$\gamma(\tilde{T}_{\varphi_1}(\varphi_2)) \cap \Omega(\varphi_2) = \emptyset$$

where

$$\tilde{T}_{\varphi_1} = \underline{\mathsf{fix}\, \lambda X. \hat{T}_{\neg\varphi_1} \sqcap \hat{F}(X)}$$

propagate the refinement until fixpoint

$$\hat{T}_{\neg\varphi_1} = \hat{T}\{\varphi_1 \mapsto \underline{\hat{T}(\varphi_1)\, \hat{\ominus}\, \alpha(\Omega(\varphi_1))}\}$$

slice out error states at φ_1 in a way that it approximates

$$[\![P]\!](\varphi_1) \ominus \Omega(\varphi_1)$$

Soundness of $\varphi_1 \rightsquigarrow \varphi_2$

Lemma 1 $\varphi_1 \rightsquigarrow \varphi_2 \implies (\text{alarm } \varphi_1 \text{ false} \implies \text{alarm } \varphi_2 \text{ false})$

φ_1 can be lifted to a set of alarms

Lemma 2 $\overrightarrow{\varphi} \rightsquigarrow \varphi \implies ((\forall \varphi_i \in \overrightarrow{\varphi}.\text{alarm } \varphi_i \text{ false}) \implies \text{alarm } \varphi \text{ false})$

42

Alarm Cluster $\mathcal{C}_{\vec{\varphi}}$

Definition 3 (Alarm Cluster) *Given set $\mathcal{A}$ of all alarms and dependence relation $\leadsto$, a false alarm cluster $\mathcal{C}_{\overrightarrow{\varphi}}$ is $\{\varphi \in \mathcal{A} \mid \overrightarrow{\varphi} \leadsto \varphi\}$.*

$\overrightarrow{\varphi} \subseteq \mathcal{A}$: dominant alarms of cluster $\mathcal{C}_{\overrightarrow{\varphi}}$

Clustering Algorithm

- Dependences determine the clustering.
- Brute-force search requires $2^{\#\mathrm{Alarms}}$ fixpoint computation.
- Our algorithm requires one fixpoint computation.
  - but misses some dependences.
- The algorithm derives sound dependences.[†]

† Not in the paper. Please refer to technical memo : http://ropas.snu.ac.kr/~wslee/vmcai12_techmemo.pdf

Conclusion

A sound, general, and effective way
to reduce alarm-investigation efforts

Thank you!

Backup slides

Example

		$\tilde{T}_{\mathcal{A}}$	R

	`int large[7];`
	`int medium[5];`
	`int small[3];`		

φ_1	**`large[i] = ...;`**	$[0,6]$	$\{\varphi_1\}$
φ_2	`... =` **`medium[i];`**	$[0,4]$	$\{\varphi_2\}$
φ_3	`... =` **`large[i];`**	$[0,4]$	$\{\varphi_2\}$
φ_4	`... =` **`medium[i-1];`**	$[1,4]$	$\{\varphi_2,\ \varphi_4\}$
φ_5	`... =` **`small[i-1];`**	$[1,4]$	$\{\varphi_2,\ \varphi_4\}$

- Clustering result

$$C_{\varphi_2} = \{\varphi_3\}$$

$$C_{\{\varphi_2,\varphi_4\}} = \{\varphi_5\}$$

- Naive algorithm

$$C_{\varphi_1} = \{\varphi_3\}$$

$$C_{\varphi_2} = \{\varphi_3\}$$

$$C_{\{\varphi_2,\varphi_4\}} = \{\varphi_5\}$$

Experimental result

Table 1. Alarm clustering results.
B : baseline analysis, **S**: syntactic alarm clustering, **I** : semantic alarm clustering with interval domain, **O** : semantic clustering with octagon domain.

Program	LOC	# Alarms				% Reduction				Time(s)		
		B	S	S+I	S+I+O	S	+I	+O	S+I+O	B	I	O
nlkain-1.3	831	124	118	96	93	5%	18%	2%	25%	0.17	0.03	0.1
polymorph-0.4.0	1,357	25	19	13	13	24%	24%	0%	48%	0.12	0	0.06
ncompress-4.2.4	2,195	66	50	38	30	24%	18%	12%	55%	0.54	0.03	0.69
sbm-0.0.4	2,467	237	230	185	125	3%	19%	25%	47%	2.28	0.3	1.15
stripcc-0.2.0	2,555	194	165	143	127	15%	11%	8%	35%	2.76	0.07	25.44
barcode-0.96	4,460	435	386	329	302	11%	13%	6%	31%	3.23	0.1	2.59
129.compress	5,585	57	56	29	29	2%	47%	0%	49%	2.46	0.02	0.19
archimedes-0.7.0	7,569	711	342	215	132	52%	18%	12%	81%	6.48	0.27	16.11
man-1.5h1	7,232	276	226	189	165	18%	13%	9%	40%	11.65	0.28	1.86
gzip-1.2.4	11,213	385	341	278	263	11%	16%	4%	32%	10.03	0.3	2.92
combine-0.3.3	11,472	733	468	297	294	36%	23%	0%	60%	19.74	0.81	26.93
gnuchess-5.05	11,629	976	744	343	333	24%	41%	1%	66%	42.49	4.78	8.66
bc-1.06	12,830	593	330	320	198	44%	2%	21%	67%	33.75	7.04	27.23
grep-2.5.1	31,154	115	100	96	85	13%	3%	10%	26%	4.19	0.01	11
coan-4.2.2	22,414	461	350	332	291	24%	4%	9%	37%	126.66	1.91	6.14
lsh-2.0.4	110,898	616	387	319	264	37%	11%	9%	57%	115.13	2.12	204.12
TOTAL	245,861	6,004	4,312	3,222	2,744	28%	18%	8%	54%	381.68	15.94	335.19