Corpus-based Abduction

Kihong Heo Suwon Jang

Programming Research Laboratory
Seoul National University

ROSAEC Workshop
July 11, 2009

Problem

- Importance of heap memory analysis
  - dangling pointer
  - NULL dereference
  - memory leak
- Difficulty
  - ...
  - guessing pre-condition
  - ...

Previous Approach

Bi-abduction

- $\frac{A \rightarrow B \quad B}{A}$
- fuction body $\longrightarrow$ preconditions
- (preconditions, function body) $\longrightarrow$ postconditions

Calcagno, Cristiano and Distefano, Dino and O'Hearn, Peter and Yang, Hongseok.
Compositional shape analysis by means of Bi-Abduction
In *POPL '09: Proceedings of the 36th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 289-300, 2009

Motivation

- Bi-abduction is NOT always safe.
- Only list can be analyzed.

Program	MLOC	Procs #	Proven Procs	Procs Coverage %	Time(1)	Time(8)
Linux kernel 2.6.25.4	2.473	101330	59215	58.4	6869.09	1739.28
Gimp 2.4.6	0.708	15114	6364	42.1	3601.16	1067.60
OpenSSL 0.9.8g	0.214	4818	2967	61.6	605.36	446.60
Sendmail 8.14.3	0.108	684	353	51.6	184.50	184.83
Apache 2.2.8	0.102	1870	881	47.1	294.67	104.48
OpenSSH 5.0	0.073	1135	519	45.7	142.56	30.24
Spin 5.1.6	0.019	357	197	55.2	772.82	253.96

- Preconditions found only for 42 $\sim$ 62 % of total procedures.

Our approach

Observations

- Most analyzer have NO knowledge which data structures are frequently used.
- Common data structures are not so complex. (e.g. linked-list, circular linked-list with linked-list, tree, graph, . . .)

Key idea

- Collect real-world data structures.
- Don't abduct, just enumerate and check.

Example

```
/* [list(l), list(l)],
   [dlist(l), ???]     */
list_one_more(l) {
  t := l;
  while (t.tl != NULL){
    t := t.tl;
  }

  t.tl := malloc;
  t := t.tl;
  t.tl = NULL;
}
```

```
main () {
  ...
  p := (S.L.L.) or (D.L.L.)
  ...
  list_one_more(p);

  dlist_prev(p);
}
```

```
/* [list(l), XXX], [dlist(l), dlist(l)] */
dlist_prev(l){
  l := l.hd
}
```

Future works

- Loop invariant
- Large survey on frequently used data structures.