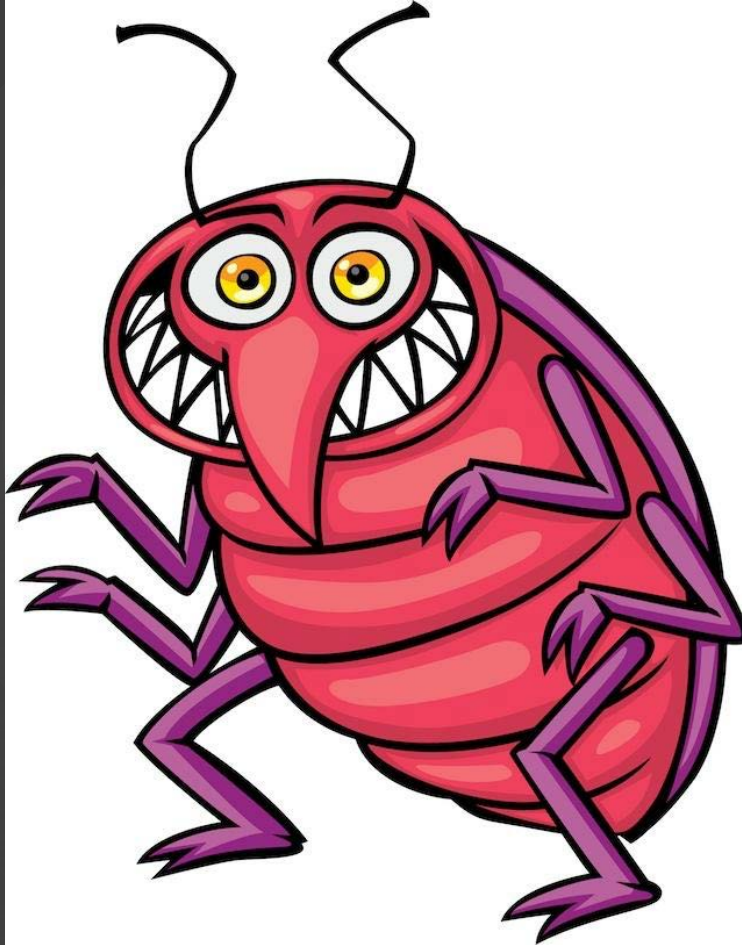
Using Likely Invariants for Automated Software Fault Localization

Research by Swarup Kumar Sahoo, John Criswell, Chase Geigle, and Vikram Adve
Department of Computer Science at UIUC

presenter name(s) removed for FERPA considerations

Introduction.



Contributions.

- A novel invariant based approach for fault localization.
- A novel heuristic for reducing false positives in the diagnosis results.
- Evaluate their approach against a set of applications much bigger and more realistic than most previous work.
- Results show that their approach is effective at reducing root causes in large programs. Also, each step of filtering is important in the reduction of the set.

Key Idea.

- Invariants that are "similar" to training inputs are more effective than other types of invariants generated.
- Sophisticated filtering techniques allow us to start with a large number of suspected bug locations and lets us narrow it down to a much smaller number of locations.

Approach

Program
Optional Input Specification
Failure-inducing (Bad) input
Generate Inputs
Good Inputs
Generate Invariants
Instrumented Program
Test with Bad Input
Failed Invariants
Backward Slicing
Dependence Filtering
Multi-faulty Inp Filter
False+ve Filters
Bad Inputs

Figure 1. Diagnosis Tool Architecture

Provided Example

```
1  long calc_daynr(uint year,uint month,uint day)
2  {
3    long delsum;
4    int temp;
5
6    if (year == 0 && month == 0 && day == 0)
7      return (0); /* Skip errors */
8    delsum= (long)(365L*year+31*(month-1)+day);
9    if (month <= 2)
10       year--;
11   else
12     delsum-= (long) (month*4+23)/10;
13   temp=(int) ((year/100+1)*3)/4;
14   return (delsum+(int) year/4-temp);
15 }
16
17 int calc_weekday(long daynr, bool first_week_day)
18 {
19   return ((int)((daynr+5L+(first_week_day ? 1L : 0L)) % 7));
20 }
21
22 bool make_date_time(TIME_FORMAT *format, M_TIME *t,
23                     timestamp_type type, String *str)
24 {
25    str->length(0);
26    if (l_time->neg)
27      str->append('-');
28 ...
29   weekday= calc_weekday(calc_daynr(t->year,t->month,
30                         t->day),0);
31   str->append(loc->d_names->type_names[weekday],
32       strlen(loc->d_names->type_names[weekday]),
33                         system_charset_info);
34 ...
35 }
```

Fails on inputs with (`year=0, month<=2`).

A buffer overflow occurs at Line 31 when `type_names` is indexed with `weekday`, a negative value on said failed input.

A brief explanation…

Provided Example

```
1   long calc_daynr(uint year,uint month,uint day)
2   {
3     long delsum;
4     int temp;
5
6     if (year == 0 && month == 0 && day == 0)
7       return (0); /* Skip errors */
8     delsum= (long)(365L*year+31*(month-1)+day);
9     if (month <= 2)
10        year--;
11    else
12      delsum-= (long) (month*4+23)/10;
13    temp=(int) ((year/100+1)*3)/4;
14    return (delsum+(int) year/4-temp);
15  }
16
17  int calc_weekday(long daynr, bool first_week_day)
18  {
19    return ((int)((daynr+5L+(first_week_day ? 1L : 0L)) % 7));
20  }
21
22  bool make_date_time(TIME_FORMAT *format, M_TIME *t,
23                timestamp_type type, String *str)
24  {
25     str->length(0);
26     if (l_time->neg)
27       str->append('-');
28  ...
29    weekday= calc_weekday(calc_daynr(t->year,t->month,
30                       t->day),0);
31    str->append(loc->d_names->type_names[weekday],
32        strlen(loc->d_names->type_names[weekday]),
33                        system_charset_info);
34  ...
35  }
```

Fails on inputs with (`year=0, month<=2`).

`year` is of type `uint`, and when decremented from value ‘0’, becomes the maximum unsigned value (2^b - 1) do to modular wraparound...

Provided Example

```
1   long calc_daynr(uint year,uint month,uint day)
2   {
3     long delsum;
4     int temp;
5
6     if (year == 0 && month == 0 && day == 0)
7       return (0); /* Skip errors */
8     delsum= (long)(365L*year+31*(month-1)+day);
9     if (month <= 2)
10      year--;
11    else
12      delsum-= (long) (month*4+23)/10;
13    temp=(int) ((year/100+1)*3)/4;
14    return (delsum+(int) year/4-temp);
15  }
16
17  int calc_weekday(long daynr, bool first_week_day)
18  {
19    return ((int)((daynr+5L+(first_week_day ? 1L : 0L)) % 7));
20  }
21
22  bool make_date_time(TIME_FORMAT *format, M_TIME *t,
23                      timestamp_type type, String *str)
24  {
25     str->length(0);
26     if (l_time->neg)
27       str->append('-');
28  ...
29    weekday= calc_weekday(calc_daynr(t->year,t->month,
30                          t->day),0);
31    str->append(loc->d_names->type_names[weekday],
32        strlen(loc->d_names->type_names[weekday]),
33                             system_charset_info);
34  ...
35  }
```

Fails on inputs with (`year=0, month<=2`).

...so `temp` becomes huge...

Provided Example

```
1   long calc_daynr(uint year,uint month,uint day)
2   {
3     long delsum;
4     int temp;
5
6     if (year == 0 && month == 0 && day == 0)
7       return (0); /* Skip errors */
8     delsum= (long)(365L*year+31*(month-1)+day);
9     if (month <= 2)
10       year--;
11    else
12      delsum-= (long) (month*4+23)/10;
13    temp=(int) ((year/100+1)*3)/4;
14    return (delsum+(int) year/4-temp);
15  }
16
17  int calc_weekday(long daynr, bool first_week_day)
18  {
19    return ((int)((daynr+5L+(first_week_day ? 1L : 0L)) % 7));
20  }
21
22  bool make_date_time(TIME_FORMAT *format, M_TIME *t,
23                      timestamp_type type, String *str)
24  {
25     str->length(0);
26     if (l_time->neg)
27       str->append('-');
28  ...
29    weekday= calc_weekday(calc_daynr(t->year,t->month,
30                          t->day),0);
31    str->append(loc->d_names->type_names[weekday],
32        strlen(loc->d_names->type_names[weekday]),
33                            system_charset_info);
34  ...
35  }
```

Fails on inputs with (`year=0, month<=2`).

...and `calc_daynr` returns a very negative value...

Provided Example

```
1   long calc_daynr(uint year,uint month,uint day)
2   {
3     long delsum;
4     int temp;
5
6     if (year == 0 && month == 0 && day == 0)
7       return (0); /* Skip errors */
8     delsum= (long)(365L*year+31*(month-1)+day);
9     if (month <= 2)
10      year--;
11    else
12      delsum-= (long) (month*4+23)/10;
13    temp=(int) ((year/100+1)*3)/4;
14    return (delsum+(int) year/4-temp);
15  }
16
17  int calc_weekday(long daynr, bool first_week_day)
18  {
19    return ((int)((daynr+5L+(first_week_day ? 1L : 0L)) % 7));
20  }
21
22  bool make_date_time(TIME_FORMAT *format, M_TIME *t,
23                 timestamp_type type, String *str)
24  {
25    str->length(0);
26    if (l_time->neg)
27      str->append('-');
28  ...
29    weekday= calc_weekday(calc_daynr(t->year,t->month,
30                        t->day),0);
31    str->append(loc->d_names->type_names[weekday],
32        strlen(loc->d_names->type_names[weekday]),
33                        system_charset_info);
34  ...
35  }
```

Fails on inputs with (`year=0, month<=2`).

...and this negative value carries through to the index of `type_names`, causing the buffer overflow.

Provided Example

```
1   long calc_daynr(uint year,uint month,uint day)
2   {
3     long delsum;
4     int temp;
5
6     if (year == 0 && month == 0 && day == 0)
7       return (0); /* Skip errors */
8     delsum= (long)(365L*year+31*(month-1)+day);
9     if (month <= 2)
10      year--;
11    else
12      delsum-= (long) (month*4+23)/10;
13    temp=(int) ((year/100+1)*3)/4;
14    return (delsum+(int) year/4-temp);
15  }
16
17  int calc_weekday(long daynr, bool first_week_day)
18  {
19    return ((int)((daynr+5L+(first_week_day ? 1L : 0L)) % 7));
20  }
21
22  bool make_date_time(TIME_FORMAT *format, M_TIME *t,
23                      timestamp_type type, String *str)
24  {
25     str->length(0);
26     if (l_time->neg)
27       str->append('-');
28  ...
29    weekday= calc_weekday(calc_daynr(t->year,t->month,
30                          t->day),0);
31    str->append(loc->d_names->type_names[weekday],
32        strlen(loc->d_names->type_names[weekday]),
33                          system_charset_info);
34  ...
35  }
```

ROOT CAUSE

CRASH

Fails on inputs with (`year=0, month<=2`).

The buffer overflow occurs at Line 31 but the root cause is at Line 10.

How does it work? User Input

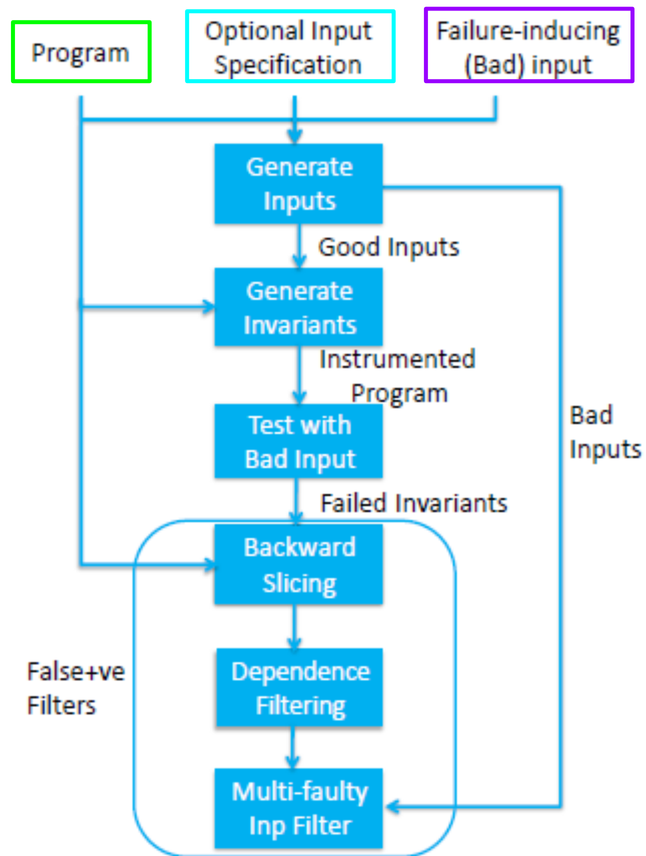


Figure 1. Diagnosis Tool Architecture

The system needs to be provided:

(1) A program

(2) A specification for valid inputs (tokenizer or lexical analyzer)

(3) A set of failing "bad" inputs that expose the bug in the program

How does it work? Phase 1 and Phase 2

Program
Optional Input Specification
Failure-inducing (Bad) input
Generate Inputs
Good Inputs
Generate Invariants
Instrumented Program
Test with Bad Input
Bad Inputs
Failed Invariants
Backward Slicing
False+ve Filters
Dependence Filtering
Multi-faulty Inp Filter

Figure 1. Diagnosis Tool Architecture

Phase 1 generates a large set of "candidate" root causes. The approach minimizes false-negatives but gets a lot of false-positives. Phase 2 uses sophisticated filtering to cut down on the many false-positives.

Why? It is important not to miss a root cause from the very beginning.

How does it work? Generating Inputs

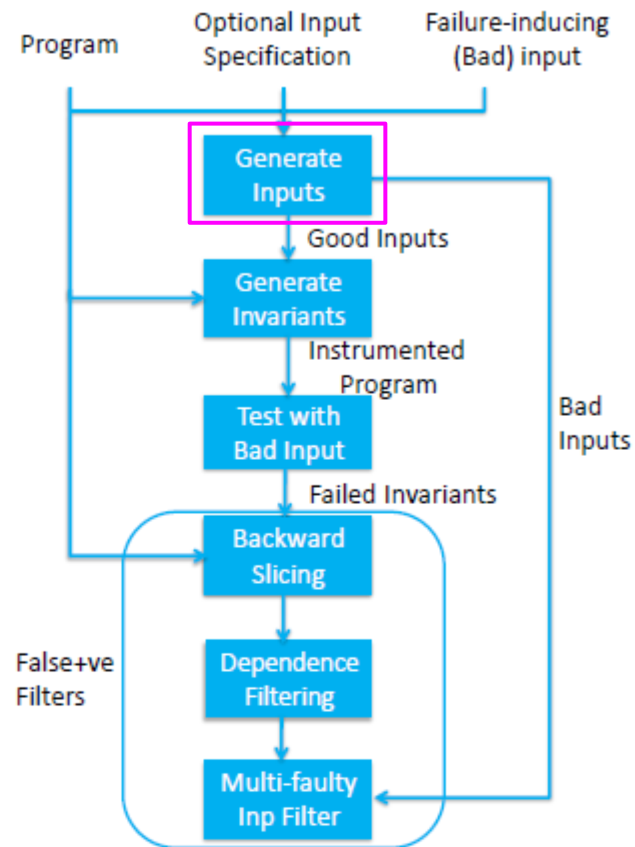


Figure 1. Diagnosis Tool Architecture

Based on what the user provided, a set of passing "good" inputs that do not expose the bug in the program is generated.

The "good" inputs are generated to be lexicographically close to the failing inputs.

"Bad" `(year=0, month=2)`
"Good" `(year=1, month=3)`

How does it work? Generating Likely Invariants

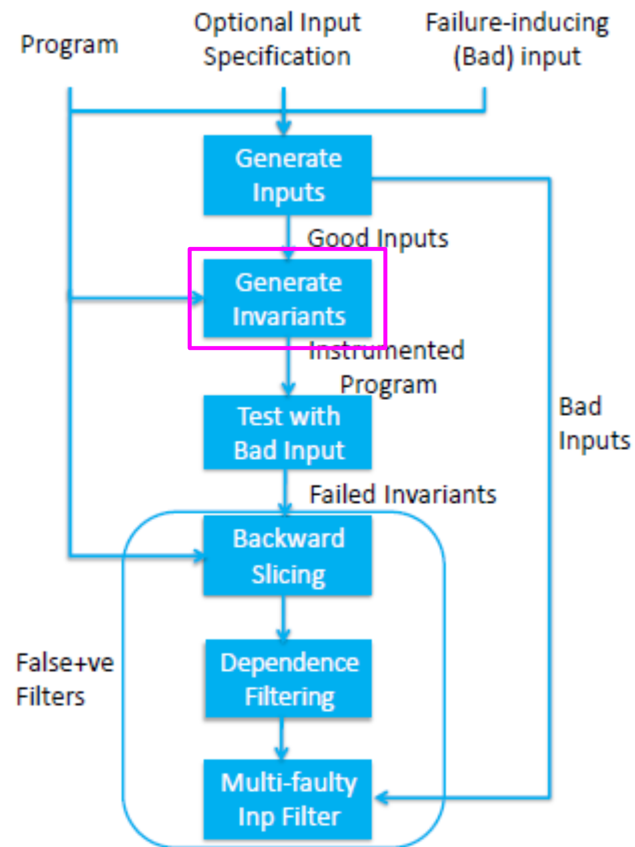


Figure 1. Diagnosis Tool Architecture

The provided program is executed with the "good" inputs that were just generated, and a set of "narrow" range invariants is derived.

- `year` is >= 0
- `month` is in the range [0,+12]
- `calc_daynr` returns >= 0
- `calc_weekday` returns >= 0

The invariant sets are limited to load, store, and function return values.

How does it work? Testing “Bad” Inputs

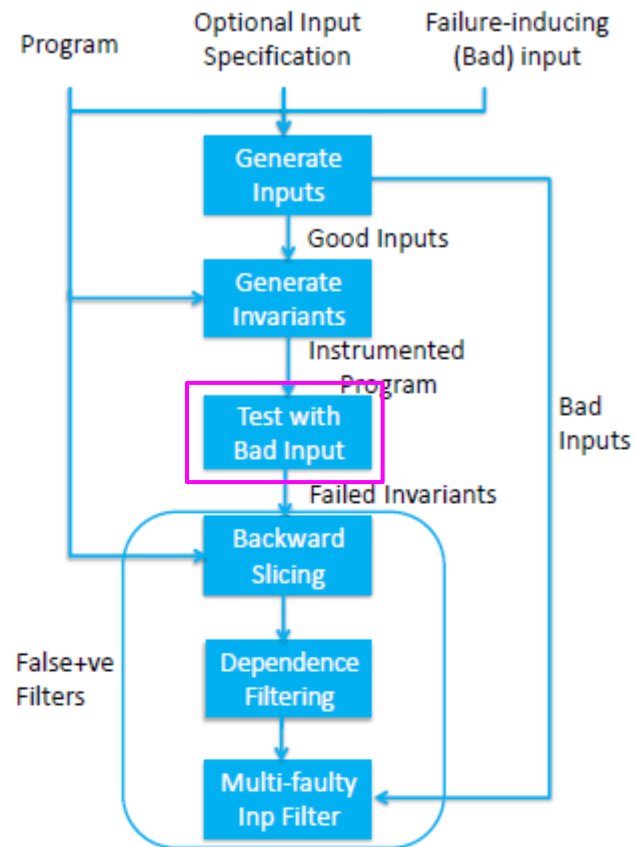


Figure 1. Diagnosis Tool Architecture

“Bad” traces are evaluated for violated range invariants.

In the example earlier, the range invariant (`year` is >= 0) is violated for the provided bad inputs.

However, there are also 94 other invariants that were violated. <u>These need to be filtered!</u>

How does it work? Dynamic Backwards Slicing

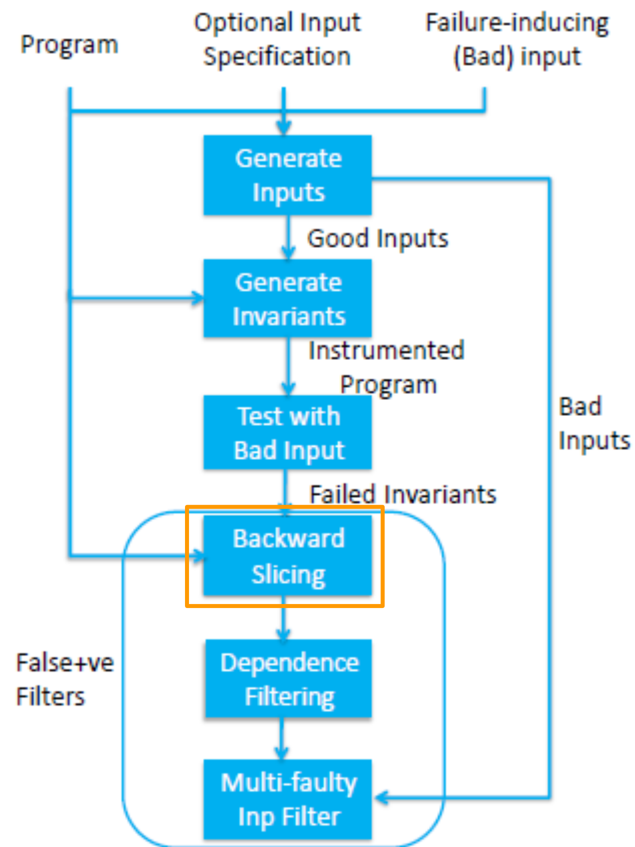


Figure 1. Diagnosis Tool Architecture

DBS takes as input the buggy statement.

i.e. Line 31 `type_names[weekday]`

DBS then uses data flow and control flow to find invariant statements in the execution that affected the buggy statement.

These form the Dynamic Backward Slice. All the rest are filtered out!

How does it work? Dependence Filtering

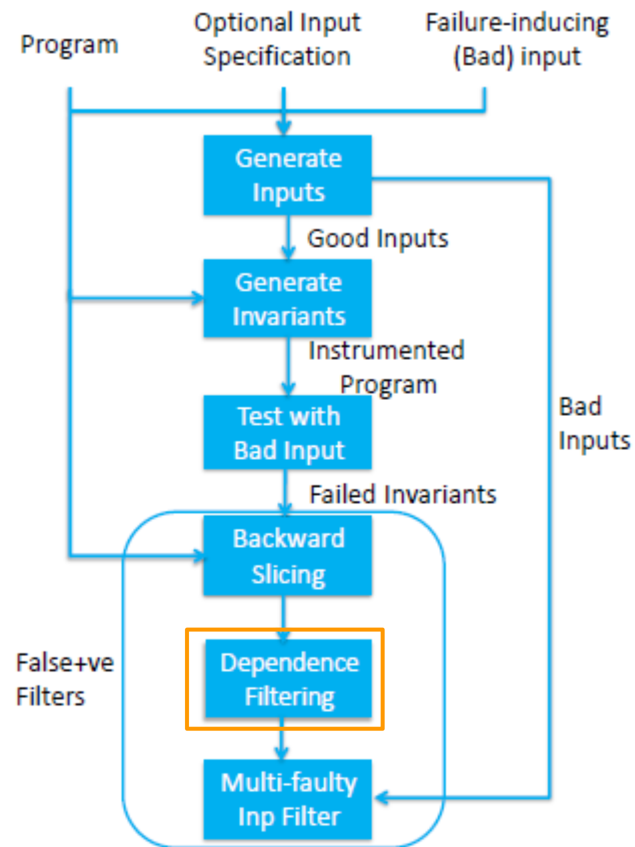


Figure 1. Diagnosis Tool Architecture

Evaluates the data flow and control dependence graphs to filter out any statements with violated invariants that depend on another statement with violated invariants.

The candidate `calc_weekday` is dependent on another candidate `calc_weekday`.

```
29      weekday= calc_weekday(calc_daynr(t->year,t->month,
30                                  t->day),0);
31      str->append(loc->d_names->type_names[weekday],
```

How does it work? Multiple Faulty Input Filtering

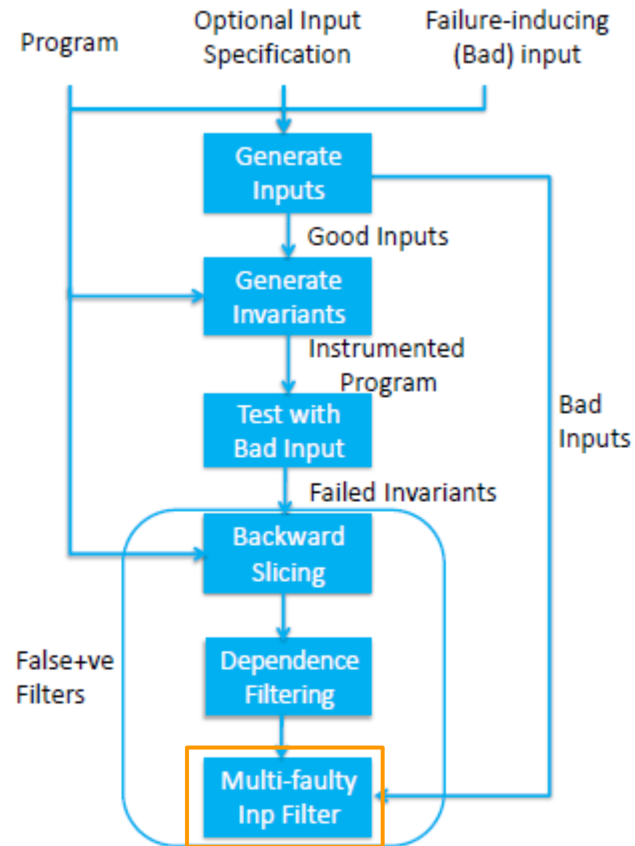


Figure 1. Diagnosis Tool Architecture

All of the filtering steps so far have been working on information gathered from single "bad" input run.

Taking the DBS and Dependence filtered candidate sets of ALL the "bad" input runs, any candidates not common between all sets is filtered out.

The resulting candidate set is reported to the user!

Experiment

Tested on Squid, MySQL, Apache

Three Programs -> Eight Bugs

Constraints

Bug#	Application	LOC	Symptom	Bug Description
Bug-1	Squid 2.3	70K	buffer overflow	Incorrect computation of buffer length leads to buffer overflow
Bug-2	MySQL 5.1.30	1019K	buffer overflow	Use of unsigned variable causes integer overflow which leads to buffer overflow
Bug-3	MySQL 5.1.30	1019K	Incorrect output	Wrong algorithm to convert microsecond field to integer results in incorrect value
Bug-4	MySQL 5.1.23a	1054K	buffer overflow	Use of a negative number along with an aggregate function results in seg fault
Bug-5	MySQL 5.0.18	949K	Incorrect output	Loss of precision in a sequence of computation results in wrong value
Bug-6	MySQL 5.0.18	949K	Incorrect output	Overflow during decimal multiplication results in garbage output
Bug-7	MySQL 5.0.15	937K	Incorrect output	Loss of data when inserting big values in a table
Bug-8	Apache 2.2	225K	buffer overflow	For particular value of output size, buffer overflow occurs

Difficulty In Diagnosis

#Bug	Static #LOC executed in failed run	Distance (Dyn #LLVM inst)	Distance (Static #LOC)	Distance (Static #Functions)
Bug-1	6927	12	6	2
Bug-2	9822	18	7	3
Bug-3	9982	86	27	10
Bug-4	11308	4	4	2
Bug-5	7874	124	41	17
Bug-6	7835	114	36	17
Bug-7	9835	429	16	5
Bug-8	6217	32780	6	2

Results of Filtering False Positives

Bug#	#Invs	Failed Invs	Slice	Dependence filter	Multiple faulty inputs	Src-expr-tree	Root Cause in final step?
Bug-1	3358	357	30	9	9	49	Yes
Bug-2	5917	95	36	16	12	48	No
Bug-3	5942	93	27	9	6	64	Yes
Bug-4	6847	156	44	14	8	28	Yes
Bug-5	4566	130	34	18	17	89	Yes
Bug-6	4652	83	13	7	5	26	Yes
Bug-7	5836	153	35	17	11	152	Yes
Bug-8	2295	120	12	6	6	171	Yes

Dynam ... es

Depend

Multiple Faulty Input Filter effectively pares none

Limitations

Limited to range invariants

System design may not be efficient if expanded to accommodate more invariants

Needs more robust input generation scheme

Conclusion

So what?

More robust algorithms

Less false positives

More bugs identified

Discussion

- The tool only considers loads, stores, and function returns for invariants (range invariants). It worked fine for the example bug, but how useful is this to developers in general?

Discussion

- Restriction to only range invariants was essential to the design of the system. Would expanding the invariants of interest be efficient?

Discussion

- The system is designed to primarily avoid missing possible candidates of root cause and secondarily reduce false positives. Would this still be a smart design choice if the invariant choices were expanded?

Discussion

- Is the experimental methodology sound?

Discussion

- The execution time for this tool was not reliable for the 8 test cases used in the authors' experiment, ranging from 8 minutes to 4 hours depending on the program size and trace size. Would execution time scale well if more invariants were considered?

References

Sahoo, S. K., Criswell, J., Geigle, C., & Adve, V. (2013). Using likely invariants for automated software fault localization. *Proceedings of the eighteenth international conference on Architectural support for programming languages and operating systems - ASPLOS '13*. doi:10.1145/2451116.2451131