STIC Doctoral School of the university of Nice Sophia Antipolis
I3S laboratory

On going work on error localization with IIS

Field: *Computer Science*
March 2013

Presented by:
BEKKOUCHE MOHAMMED
Cooperation with:
MICHEL RUEHER
HELEN COLLAVIZA
YAHIA LEBBAH
OLIVER PONSINI

Abstract

- Error Localization $\sqsubset$ Software Debugging $\sqsubset$ Software Engineering
- A counterexample -> Faulty trace of the counterexample

  Our search space is the set of instructions in the trace of the counterexample
- The constraint programming formalism

  Why ?
  - To model the problem,
  - And to solve it.

Work objective

- Locate faults in imperative programs
- In which we have a counterexample

Introduction, the problematic and hypothesis

- A program may contain errors
- This errors can harm in proper operation of the program
- The process of debugging software is inevitable
  - The errors detection, **the faults localization**, the correction of fautes
- Program with errors :
  - A tool for model-checking (e.g. CPBPV, CBMC) to obtain a counterexample
  - Counterexample -> counterexample trace
- The problem :
  - The execution trace of the counterexample is often long and difficult to understand
  - The reason for which the localization problem is difficult

Our idea :

- Counterexample, counterexample trace and the postcondition -> set of infeasible constraints -> A **minimal conflict set of constraints** (**IIS**)

Introduction, the problematic and hypothesis

- We consider a set of assumptions :
  - A program with a single fault assignment statement
  - A counterexample provided by a model-checking tool
- In this context, we study the case where :
  - The path is right,
  - The path is bad.

```
class program{

 /*@ ensures
   @ (c >= d+e);
   @*/
void foo(int a,int b){
  int c;
  int d;
  int e;
  int f;
  if (a>=0){
    ...
  }
  else{
    c=b; /* error */
    d=1;
    e=-a;
    if (a>b){
      f=b+e+a;
      d=d+4;
    }
    else{
      ...
    }
  }
  c=c+d+e;
}
}
```

Program foo

Example of motivation

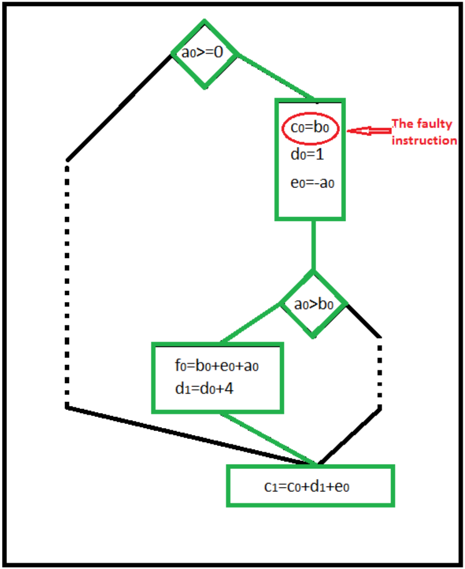

FIGURE: The control flow graph of the SSA form of the foo program

Example of motivation

Description of the example :

- The erroneous program above is written in java
- It is annotated with a JML specification
- The error in the program is an assignment instruction ("c=d")
- The erroneous instructionis in a dependency data-flow with postcondition variables
- Our goal :
  - Finding the minimum set of **suspect instructions** in the program
  - That covers the real faulty instruction

Example of motivation

Our approche to locate faults :

- Use a BMC tool to obtain a counterexample :
  $CE_{PROG}\ (a_0 = -1, b_0 = -2)$
- Generating the set of constraints which corresponds to the trace of the counterexample :
  $C_{TCE} = \{c_0 = b_0, d_0 = 1, e_0 = -a_0, a_0 > b_0, f_0 = b_0 + e_0 + a_0, d_1 = d_1 + 4, c_1 = c_0 + d_1 + e_0\}$
- Generating of the constraints set that corresponds to the postcondition :
  $C_{POST} = \{c_1 >= d_1 + e_0\}$
- Generating of the constraints set of the counterexample :
  $C_{CE_{PROG}} = \{a_0 = -1, b_0 = -2\}$

Example of motivation

Our approche to locate faults :

- Identification of the faulty contraints :
  - $C_{CE_{PROG}} \cup C_{TCE} \cup C_{POST}$ is infeasible
    It has at least an infeasible sub-system irreducible of constraints (**IIS**)
  - $C_{CE_{PROG}} \cup C_{LOC} \cup C_{POST}$ must be infeasible and C_{LOC} is minimum
    $C_{LOC} = \{c_0 = b_0, c_1 = c_0 + d_1 + e_0\}$
    - $\{a_0 = -1, b_0 = -2\} \cup \{c_0 = b_0, c_1 = c_0 + d_1 + e_0\} \cup \{c_1 >= d_1 + e_0\}$ is infeasible
    - $\{a_0 = -1, b_0 = -2\} \cup \{c_0 = b_0\} \cup \{c_1 >= d_1 + e_0\}$ is feasible
      $\{a_0 = -1, b_0 = -2\} \cup \{c_1 = c_0 + d_1 + e_0\} \cup \{c_1 >= d_1 + e_0\}$ is feasible
  - $C' = C_{CE_{PROG}} \cup C_{TCE} \backslash c_i \cup C_{POST}$ is feasible($c_i \in C_{LOC}$) Because the input infeasible system has a single **IIS**
- $LOC = \{\boldsymbol{ligne\ 15}, ligne\ 26\}$

Notations and definitions

- CSP
  $\mathcal{P} = < X, D, C >$
- Sol function
  $\Delta = D_{x_1} \times D_{x_2} \times ... \times D_{x_n}$
  $Sol : C \times D \longrightarrow \Delta$
- IS
  - $IS \subseteq C$.
  - $Sol(IS, D) = \emptyset$.
- MIN-UNCSP
  - $Sol(C \backslash MUC, D) \neq \emptyset$.
  - $\nexists\ MUC' \subset MUC$ such that $Sol(C \backslash MUC', D) \neq \emptyset$.
- IIS
  - S is an IS.
  - $\forall\ S' \subset S.Sol(S', D) \neq \emptyset$.
- MIN-IIS
  - MS is an IIS.
  - $\forall\ S \in \Sigma_{IIS}.|MS| \leq |S|$
    (Σ_{IIS} represents all the IISs in C).

Notations and definitions

- IIS-COVER
    - $\forall S \in \Sigma_{IIS}, \exists c \in SC$ such that $c \in S$ (Σ_{IIS} is the set of all the IISs in C).
- MIN-IIS-COVER
    - MSC is an IIS-COVER.
    - $\forall SC \in \Sigma_{SC}.|MSC| \leq |SC|$ (Σ_{SC} is the set of all the IISs in C).
- MIN-UNCSP $\equiv$ MIN-IIS-COVER

Notations and definitions

Example Let $\mathcal{P} = < X, D, C >$ with $C = \{C_1, C_2, ..., C_{16}\}$.

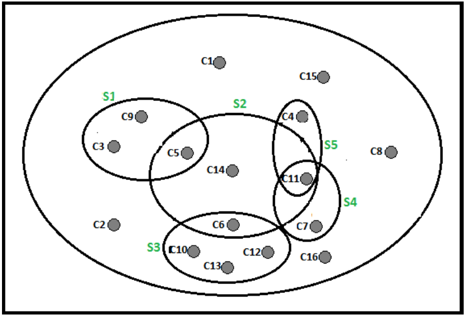

FIGURE: A constraint system with five IISs

$\Sigma_{IIS} = \{S_1, S_2, S_3, S_4, S_5\}$.
From the set Σ_{IIS}, we can compute :

- MIN-IIS : $\Sigma_{MS} = \{\{C_7, C_{11}\}, \{C_4, C_{11}\}\}, |MS| = 2$.
- The set that contains all the IIS-COVERs :
  $\Sigma_{SC} = \{C, \{S_1 \cup S_2 \cup S_3 \cup S_4 \cup S_5\}, ..., \{C_3, C_{11}, C_{13}\}, \{C_5, C_6, C_{11}\}, ...\}$.
- Le MIN-IIS-COVER (MIN-UNCSP) : There are exactly twelve ($|S_1| \times |S_3|$) MIN-IIS-COVERs for which the cardinality is three
  Exemple $MSC = \{C_3, C_{11}, C_{13}\}$

Notations and definitions

- Two classes of constraints
  - C_{HARD}
  - C_{SOFT}
- Conflict Set
  - $CS \subseteq C_{SOFT}$
  - $CS \cup C_{HARD}$ is an IS ($Sol(CS \cup C_{HARD}, D) = \emptyset$)
- Minimal Conflict Set
  - CS is a Conflict Set
  - $\forall\ CS' \subset CS$, CS' is not a Conflict Set

The definition of the fault localization problem

- An erroneous program *PROG*
- A postcondition violated *POST*
- A counterexample CE_{PROG}
- We can find the counterexample trace *TCE*

The localization problem in *TCE*

What is the minimal set of instructions to remove (or change) from TCE to reach the satisfiability of $CE_{PROG} \wedge POST$ *?*

The localization problem in *TCE*

What is the minimal set of instructions (one or many) in contradiction with $CE_{PROG} \wedge POST$ *?*

The definition of the fault localization problem

- The localization problem in *TCE* $\longrightarrow$ Isolating infeasibity problem in *P*
  - $\mathcal{P} = < X, D, C_{CE_{PROG}} \cup C_{TCE} \cup C_{POST} >$

Isolating infeasibity problem in *P*

What is the Minimal set of constraints to remove from C_{TCE} to reach the satisfiability of $C_{CE_{PROG}} \cup C_{POST}$?

Isolating infeasibity problem in *P*

What is the Minimal Conflict Set (one or many) in C_{TCE} towards to $C_{CE_{PROG}} \cup C_{POST}$?

Our approach

Algorithm 1 Fault localization algorithm

Input : *PROG* :A program ; *PRED* :A precondition ; *POST* :A postcondition
Output : *LOC* : The set of suspicious instructions in *PROG*

```
$CE_{PROG} \leftarrow BMC(PROG, PRED, POST)$
if $CE_{PROG}$ is $Nulle$ then
    $LOC \leftarrow Nulle$
    WRITE("The program is conform to the specification")
else
    $TCE \leftarrow$ GENERATE_TCE($CE_{PROG}, PROG, POST$)
    $< X, D, C_{CE} \cup C_{TCE} \cup C_{POST} > \leftarrow$ GENERATE_CSP($CE_{PROG}, TCE, POST$)
    $C_{LOC} \leftarrow$ ISOLATING-INFEASIBILITY($< X, D, C_{CE} \cup C_{TCE} \cup C_{POST} >$)
    $LOC \leftarrow Consts\_To\_inst(C_{LOC})$
end if
```

Our approach

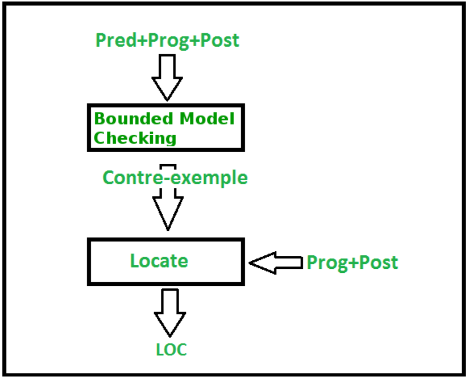

FIGURE: Our approach of localization

Modeling of the problem

- The starting point is cunterexample

  Obtained by the use of a model checking tool
- Generation of the counterexample trace
- $CSP\ \mathcal{P} = < X, D, C_{CE_{PROG}} \cup C_{TCE} \cup C_{POST} >$
  - $C_{CE_{PROG}}$ which corresponds to CE_{PROG}.
  - C_{TCE} which corresponds to TCE.
  - C_{POST} which corresponds to $POST$.

Solving the problem

Isolating infeasibility algorithm based on the Deletion Filter Method

Algorithm 2

Input : $\mathcal{P} = < X, D, C_{CE_{PROG}} \cup C_{TCE} \cup C_{POST} >$:An infeasible system of constraints.
Output : A minimal conflict set in C_{TCE}.

```
for each constraint $c_i$ in $C_{TCE}$ do
    Temporarily drop the constraint $c_i$ from $C_{TCE}$.
    Test the feasibility of $C_{CE_{PROG}} \cup (C_{TCE} \setminus c_i) \cup C_{POST}$ :
    if feasible then
        return dropped constraint to the set.
    else
        drop the constraint permanently.
    end if
    We take the set of constraints that remains in $C_{TCE}$
end for
```

Solving the problem

Isolating infeasibility algorithm based on the Additive Method

Algorithm 3

Input : $\mathcal{P} = < X, D, C_{CE_{PROG}} \cup C_{TCE} \cup C_{POST} >$:An infeasible system of constraints.
Output : I is a minimal conflict set in C_{TCE}.

```
1: $T \leftarrow \emptyset$, $I \leftarrow \emptyset$.
2: $T \leftarrow C_{CE_{PROG}} \cup C_{POST} \cup I$.
3: for each constraint $c_i$ in $C_{TCE}$ do
4:     $T \leftarrow T \cup \{c_i\}$.
5:     if $C_{CE_{PROG}} \cup C_{POST} \cup T$ infeasible then
6:         $I \leftarrow I \cup \{c_i\}$.
7:         Go to 10.
8:     end if
9: end for
10: if $C_{CE_{PROG}} \cup C_{POST} \cup I$ feasible then
11:     Go to 2.
12: end if
```

Solving the problem

Isolating infeasibility algorithm based on The Additive/Deletion method

Algorithm 4

Input : $\mathcal{P} = < X, D, C_{CE_{PROG}} \cup C_{TCE} \cup C_{POST} >$:An infeasible system of constraints.
Output : A minimal conflict set in C_{TCE}.

```
1: Set $T \leftarrow \emptyset$.
2: for each constraint $c_i$ in $C$ do
3:     Set $T \leftarrow T \cup c_i$.
4:     if $C_{CE_{PROG}} \cup C_{POST} \cup T$ infeasible then
5:         Go to 8.
6:     end if
7: end for
8: for each constraint $t_i$ in $t_{|T|-1}$ in $T$ : do
9:     Temporarily drop the constraint $t_i$.
10:    Test the feasibility of $C_{CE_{PROG}} \cup C_{POST} \cup T \setminus t_i$ :
11:    if feasible then
12:        return dropped constraint to $T$.
13:    else
14:        $T \leftarrow T \setminus t_i$.
15:    end if
16: end for
```

Solving the problem

Comparison

- All these methods are based on the principal of testing the feasibility of a sub-system of constraints
- The difference between them lies in the number of feasibility tests
  - The cardinality of the set of constraints of the counterexample trace is n
  - The cardinality of the set returned is k
    - The number of feasibility tests :
    - By using Deletion filter
      In all cases n
    - By using Additive method
      In worst case : $k/2 * (2n - k)$
      In the best case : $k/2 * (k + 1)$
    - By using Additive/Deletion method
      In worst case : $n + (n - 1)$
      In the best case : $k + (k - 1)$
    - By using QUICKXPLAIN
      In worst case : $2k * log(n/k) + 2k$
      In the best case : $log(n/k) + 2k$

Implementation

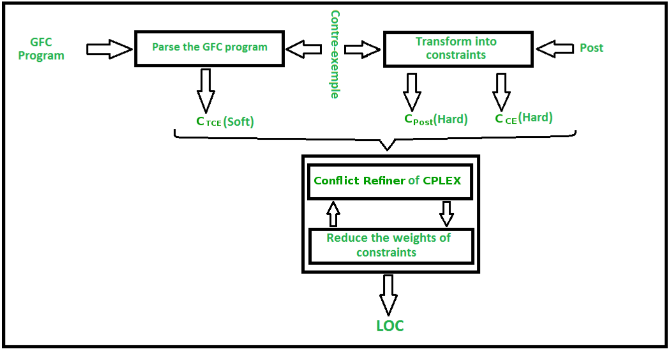

FIGURE: The localization process

Thank you for your attention