FEDERAL UNIVERSITY OF AMAZONAS
INSTITUTE OF COMPUTING
GRADUATE PROGRAM IN COMPUTER SCIENCE

UNDERSTANDING PROGRAMMING BUGS IN ANSI-C SOFTWARE USING BOUNDED MODEL CHECKING COUNTER-EXAMPLES

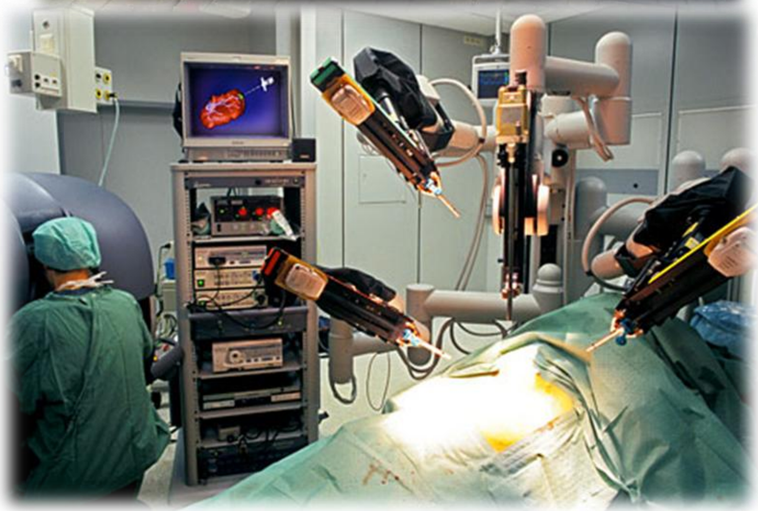
Herbert Oliveira Rocha, Raimundo Barreto,
Lucas Cordeiro and Arilo Dias Netto

iFM'2012

Agenda

1. Introduction
2. Background
3. Proposed Method
4. Experimental Results
5. Related Work
6. Conclusions and Future Work

Software Applications



Model Checking

- ✓ In the last few years, we can observe **a trend towards** the application of formal verification techniques to the **implementation level**;

- ✓ BMCs have gained **popularity** due to their ability to handle the **full semantics** of actual programming languages, and to support the verification of **a rich set of properties**.

And what are we proposing? **The EZProofC Method**

- ✓ To apply a **software bounded model checker**, in this case ESBMC (Efficient SMT-Based Context-Bounded Model Checker);
- ✓ To verify **critical parts of a software** written in the C programming language;
- ✓ To gather **data to show** the evidence that **failures** might happen.

The motivation of this work - **EZProofC**

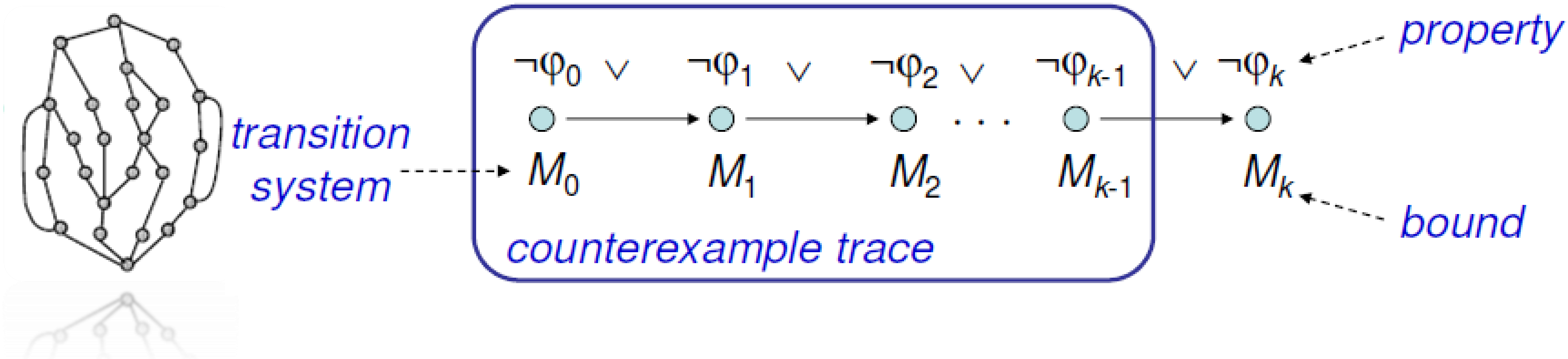
- ✓ Data collected by verification tools is usually **not trivial** to be understood:
  - **Amount of variables;**
  - **Values involved in the counter-example;**
  - **The lack of a standard output to represent the counter-example;**
- ✓ Our techniques can also be applied to other programming languages like C++ and Java

Agenda

1. Introduction
2. Background
3. Proposed Method
4. Experimental Results
5. Related Work
6. Conclusions and Future Work

Bounded Model Checking

✓ The basic idea of BMC is to check (**the negation of**) a given property at a given depth.



✓ Transition system M unrolled k times

- for programs: unroll loops, unfold arrays, …

✓ Translated into verification condition $\boldsymbol{\psi}$ such that

- $\boldsymbol{\psi}$ satisfiable iff $\boldsymbol{\varphi}$ has counterexample of max. depth $\boldsymbol{k}$.

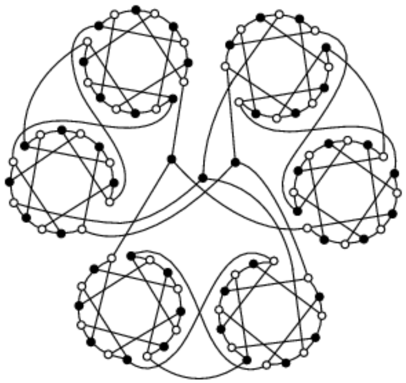
Context-Bounded Model Checking with ESBMC

ESBMC is a bounded model checker for embedded ANSI-C software based on SMT (Satisfiability Modulo Theories) solvers, which allows:

- ✓ Out-of-bounds array indexing;
- ✓ Division by zero;
- ✓ Pointers safety
- ✓ Dynamic memory allocation;
- ✓ Data races;
- ✓ Deadlocks;
- ✓ Underflow e Overflow;

Counter-Example

- ✓ A counter-example **<u>is a trace</u>** that shows that a given property does not hold in the model;



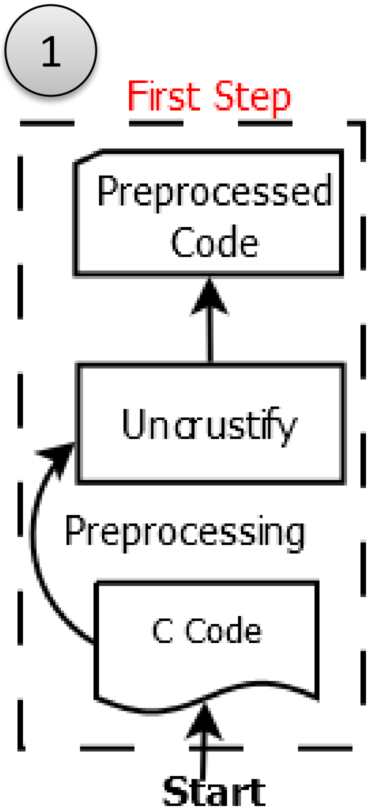
- ✓ Counter-examples allow the user:
  i. to analyze the failure;
  ii. to understand the root of the error;
  iii. to correct either the specification or the model, in this case, from the property and the program that has been analyzed respectively.

Agenda

1. Introduction
2. Background
3. Proposed Method
4. Experimental Results
5. Related Work
6. Conclusions and Future Work

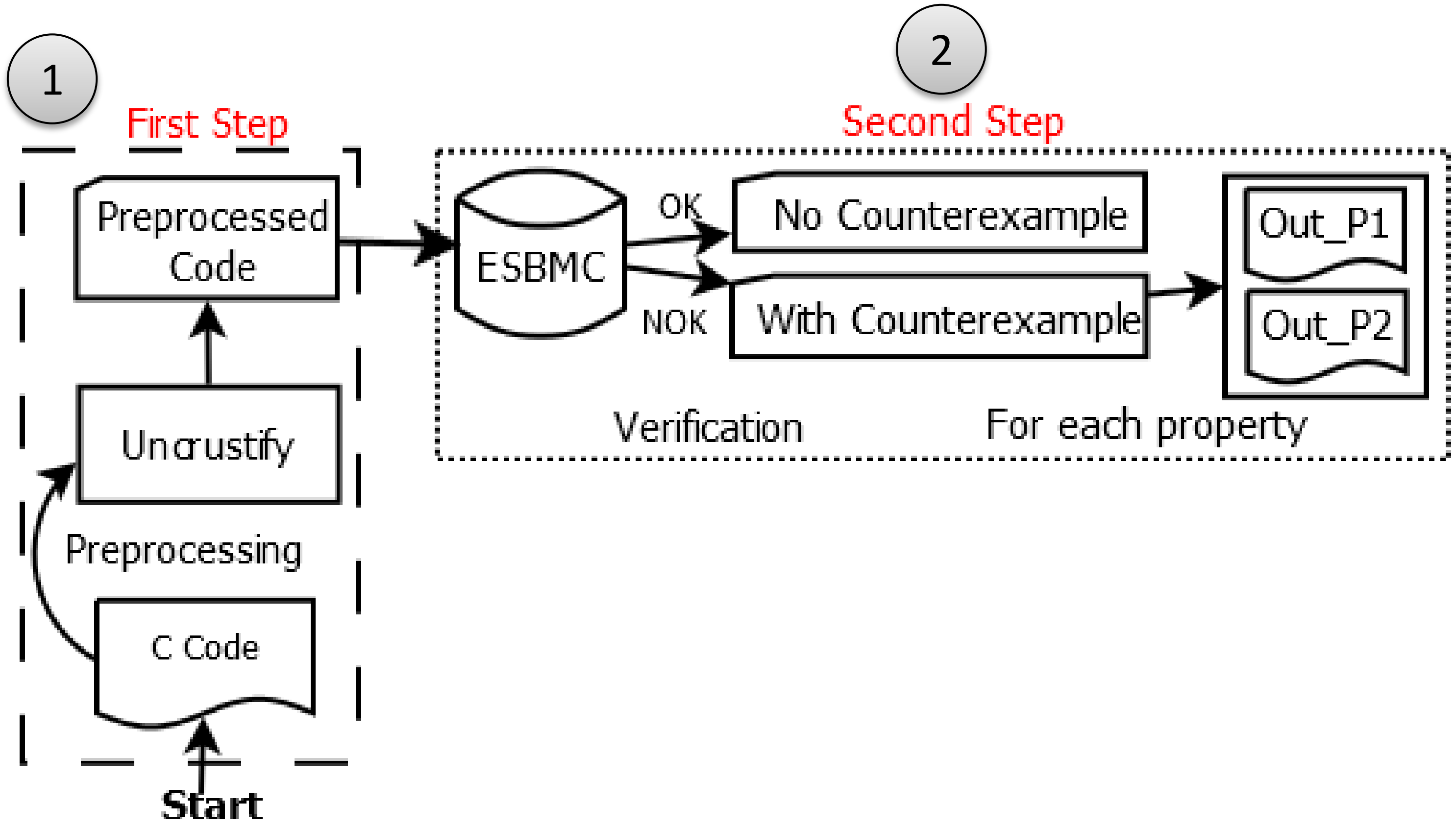
EZProofC

"An easy way to demonstrate and verify errors in C code"



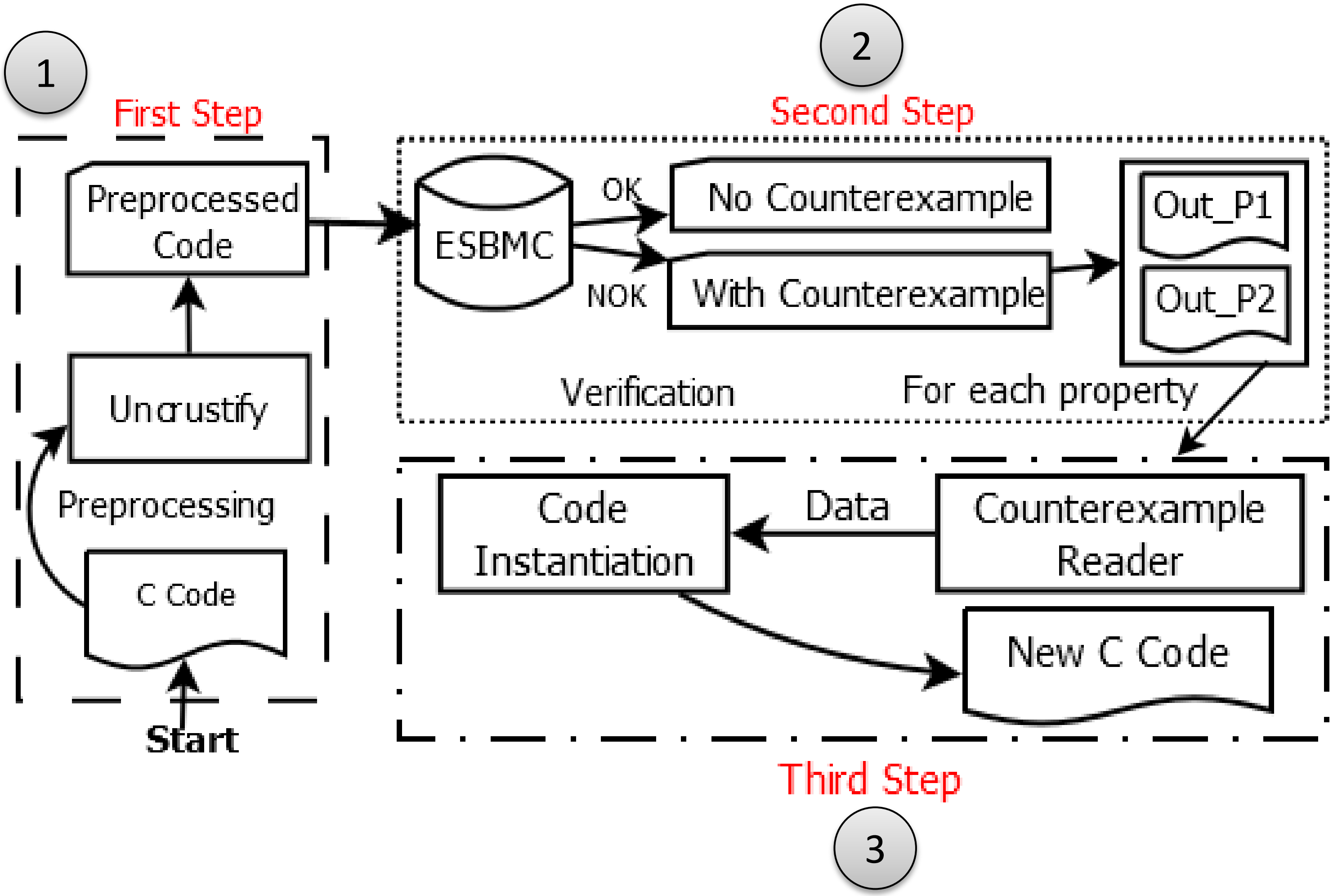
EZProofC

"An easy way to demonstrate and verify errors in C code"



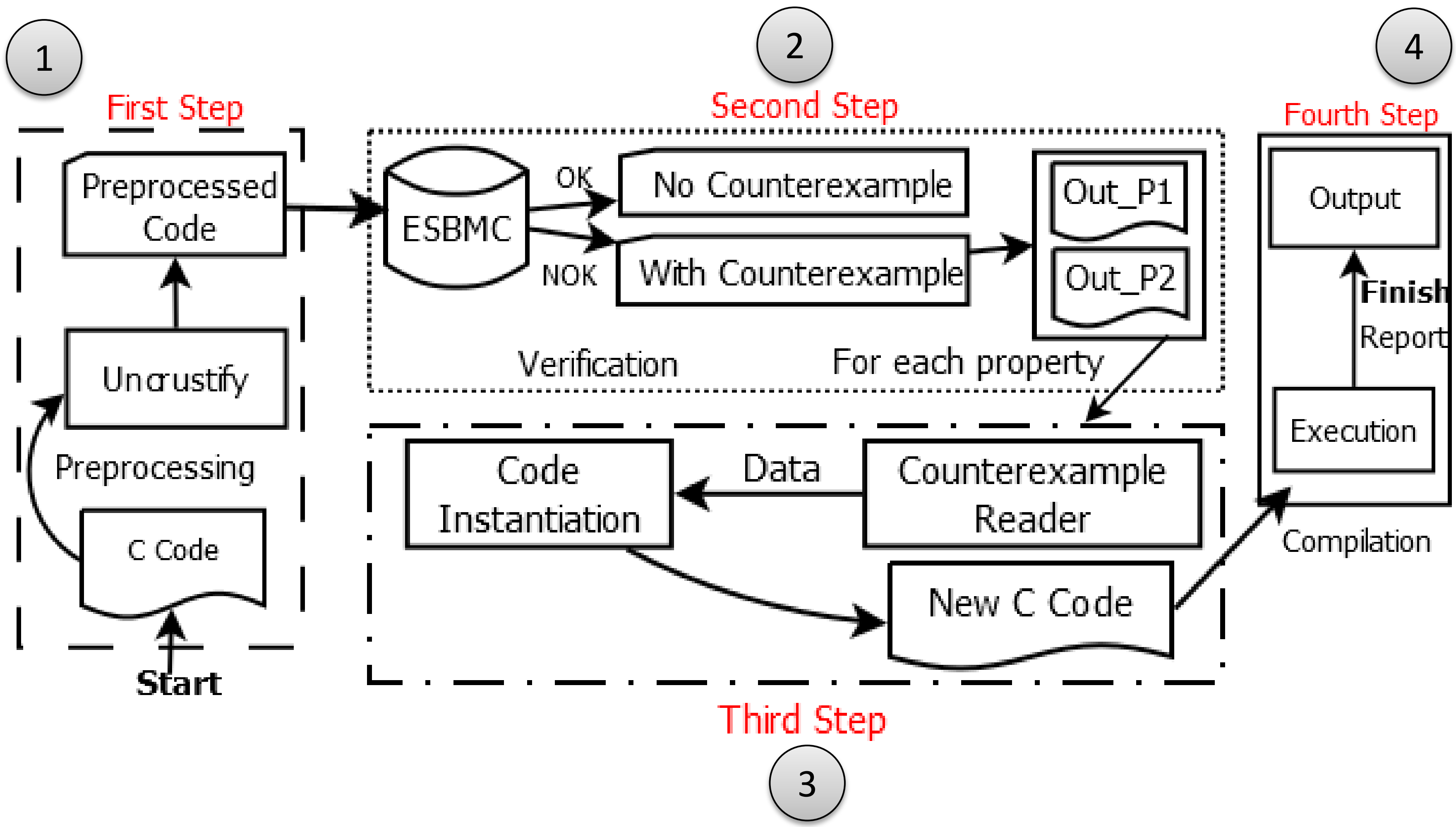
EZProofC

"An easy way to demonstrate and verify errors in C code"



EZProofC

"An easy way to demonstrate and verify errors in C code"



First Step: Code Preprocessing

UNCRUSTIFY

```
#define INSIZE 14
int main (void){
     unsigned char in[INSIZE+1];
     ...
    if (c == `-')
    {
        i=0;
        idx_in++;
        c = in[idx_in];
        while ((`0' <= c) && (c <= `9'))
       {
            j = c - `0';
            i = i * 10 + j;
            idx_in++;
            c = in[idx_in];
        }
    }
}
```

tTflag_arr_two_loops_bad.c
from Verisec benchmark

Second Step: Model Checking with ESBMC

Counterexample:

```
(......)
State 97 file ccode.pre/pre_tTflag_arr_two_loops_bad.c line 13 function main thread 0
----------------------------------------------------
  pre_tTflag_arr_two_loops_bad::main::1::i=33 (00000000000000000000000000100001)

State 98 file ccode.pre/pre_tTflag_arr_two_loops_bad.c line 14 function main thread 0
----------------------------------------------------
  pre_tTflag_arr_two_loops_bad::main::1::idx_in=15 (00000000000000000000000000001111)

State 93 file ccode.pre/pre_tTflag_arr_two_loops_bad.c line 15 function main thread 0
----------------------------------------------------
  pre_tTflag_arr_two_loops_bad::main::1::c=51 (00110011)

Violated property:
  file ccode.pre/pre_tTflag_arr_two_loops_bad.c line 15 function main
  array `in' upper bound
  idx_in < 15

VERIFICATION FAILED
```

Second Step: Model Checking with ESBMC

The line number

```
Counterexample:

(......)
State 97 file ccode.pre/pre_tTflag_arr_two_loops_bad.c line 13 function main thread 0
------------------------------------------------
  pre_tTflag_arr_two_loops_bad::main::1::i=33 (00000000000000000000000000100001)

State 98 file ccode.pre/pre_tTflag_arr_two_loops_bad.c line 14 function main thread 0
------------------------------------------------
  pre_tTflag_arr_two_loops_bad::main::1::idx_in=15 (00000000000000000000000000001111)

State 93 file ccode.pre/pre_tTflag_arr_two_loops_bad.c line 15 function main thread 0
------------------------------------------------
  pre_tTflag_arr_two_loops_bad::main::1::c=51 (00110011)

Violated property:
  file ccode.pre/pre_tTflag_arr_two_loops_bad.c line 15 function main
  array `in' upper bound
  idx_in < 15

VERIFICATION FAILED
```

Second Step: Model Checking with ESBMC

Counterexample:

The variables involved

The line number

le.pre/pre_tTflag_arr_two_loops_bad.c line **13** functio ain thread 0

\---

pre_tTflag_arr_two_lo _bad::main::1::i=33 (00000000000000 000000000000000100001)

State 98 file ccode.pre/pre_tTflag_arr_two_loops_bad.c line **14** function main thread 0

\---

pre_tTflag_arr_two_loops_bad::main::1::idx_in=15 (00000000000000000000000000001111)

State 93 file ccode.pre/pre_tTflag_arr_two_loops_bad.c line **15** function main thread 0

\---

pre_tTflag_arr_two_loops_bad::main::1::c=51 (00110011)

Violated property:

file ccode.pre/pre_tTflag_arr_two_loops_bad.c line **15** function main
array `in' upper bound
idx_in < 15

VERIFICATION FAILED

Second Step: Model Checking with ESBMC

Counterexample:

The variables involved

The line number

le.pre/pre_tTflag_arr_two_loops_bad.c line **13** functio main thread 0

---

pre_tTflag_arr_two_loo _bad::main::1::i=33 (0000000000000 0000000000000100001)

State 98 file ccode.pre/pre_tTflag arr_two_loops_bad.c line **14** function main thread 0

---

pre_tTflag_arr_two_loops_bad::main::1::idx_in=15 (00000000000000000000000000001111)

State 93 file ccode.pre/pre_tTflag_arr_two_loops_bad.c line

---

pre_tTflag_arr_two_loops_bad::main::1::c=51 (00110011)

The specific value for variable

Violated property:
file ccode.pre/pre_tTflag_arr_two_loops_bad.c line **15** function main
array `in' upper bound
idx_in < 15

VERIFICATION FAILED

Second Step: Model Checking with ESBMC

Counterexample:

The variables involved

The line number

(
S le.pre/pre_tTflag_arr_two_loops_bad.c line **13** functio nain thread 0
- ----------------------------
pre_tTflag_arr_two_loo _bad::main::1::i=33 (0000000000000 0000000000000100001)

State 98 file ccode.pre/pre_tTflag arr_two_loops_bad.c line **14** function main thread 0
----
pre_tTflag_arr_two_loops_bad::main::1::idx_in=15 (00000000000000000000000000001111)

State 93 file ccode.pre/pre_tTflag_arr_two_loops_bad.c line fu
----
pre_tTflag_arr_two_loops_bad::main::1::c=51 (00110011)

The specific value for variable

Violated property:
file ccode.pre/pre_tTflag_arr_two_loops_bad.c line **15** function main
array `in' upper bound
idx_in < 15

Violated Property

VERIFICATION FAILED

Second Step: Model Checking with ESBMC

```
#define INSIZE 14
int main (void){
    unsigned char in[INSIZE+1];
    ...
   if (c == `-')
   {
      i=0;
      idx_in++;
      c = in[idx_in];
      while ((`0' <= c) && (c <= `9'))
     {
        j = c - `0';
        i = i * 10 + j;
        idx_in++;
        c = in[idx_in];
     }
   }
}
```

Property "***idx_in < 15***" that has been violated

Second Step: Model Checking with ESBMC

```
#define INSIZE 14
int main (void){
    unsigned char in[INSIZE+1];
    ...
   if (c == `-')
   {
      i=0;
      idx_in++;
      c = in[idx_in];
      while ((`0' <= c) && (c <= `9'))
     {
         j = c - `0';
         i = i * 10 + j;
         idx_in++;
         c = in[idx_in];
      }
   }
}
```

Define the BOUND
[0 .. 14]

Property "***idx_in < 15***" that has been violated

Second Step: Model Checking with ESBMC

```
#define INSIZE 14
int main (void){
    unsigned char in[INSIZE+1];
    ...
   if (c == `-')
   {
      i=0;
      idx_in++;
      c = in[idx_in];
      while ((`0' <= c) && (c <= `9'))
     {
         j = c - `0';
         i = i * 10 + j;
         idx_in++;
         c = in[idx_in];
      }
   }
}
```

Define the BOUND [0 .. 14]

Loop doesn't control the value of the variable

Property "***idx_in < 15***" that has been violated

Third Step: Code Instantiation

```
Input: Code, CE_Out
Output: New_instanced_code.c

BEGIN
  Analysis The counter-example (CE_Out) and C program to collect several
      pieces of information
  FOREACH line from the C program
          IF the line number identified (counter-example) is equal to the line
             number of the C program
             IF the violated property is in a set of specific cases
                    Apply a Trigger for a specific case
                    Generate and write a new line using variable values from
                    counter-example
             ELSE Generate and write a new line using variable values from
             counter-example
          ELSE Write the line from the C program
END
```

The runtime complexity is $O(n + m)$

Third Step: Code Instantiation

FIRST PHASE: Collect several pieces of information

Input: Code, CE_Out
Output: New_instanced_code.c

BEGIN
Analysis The counter-example (CE_Out) and C program to collect several pieces of information
FOREACH line from the C program
    IF the line number identified (cou[illegible]xample) is equal to the line number of the C program
        IF the violated property is in a [illegible]
            Apply a Trigger for a specific case
            Generate and write a new line using variable values from counter-example
        ELSE Generate and write a new line using variable values from counter-example
    ELSE Write the line from the C program
END

line = 14, var =idx_in and value = 15

Third Step: Code Instantiation

SECOND PHASE: Generate a new instanced code

```
Input: Code, CE_Out
Output: New_instanced_code.c

BEGIN
  Analysis The counter-example (CE_Out) and C program to collect several
      pieces of information
  FOREACH line from the C program
        IF the line number identified (counter-example) is equal to the line
            number of the C program
            IF the violated property is in a set of specific cases
                  Apply a Trigger for a specific case
                  Generate and write a new line using variable values from
                  counter-example
             ELSE Generate and write a new line using variable values from
             counter-example
        ELSE Write the line from the C program
END
```

Third Step: Code Instantiation

It makes a copy and replaces variables assignments

Input: Code, CE_Out
Output: New_instanced_code.c

BEGIN
  Analysis The counter-example (CE_O...) and C program to collect several pieces of information
  FOREACH line from the C program
    IF the line number identified (counter-example) is equal to the line number of the C program
      IF the vio...
        App...
        Generate ... a new line using variable values from counter-example
      ELSE Generate and write a new line using variable values from counter-example
    ELSE Write the line from the C program
END

CE: line = 14, var =idx_in and value = 15
New Line: idx_in = 15;

Third Step: Code Instantiation

```
Input: Code, CE_Out
Output: New_instanced_code.c

BEGIN
    Analysis The counter-example (CE_Out) and C program to collect several
        pieces of information
    FOREACH line from the C program
            IF the line number identified (counter-example) is equal to the line
               number of the C program
               IF the violated property is in a set of specific cases
                    Apply a Trigger for a specific case
                    Generate and write a new line using variable values from
                    counter-example
                ELSE Generate and write a new line using variable values from
                counter-example
            ELSE Write the line from the C program
END
```

UPPER BOUND

Assert(idx_in < 15);

Third Step: Code Instantiation

```
#define INSIZE 14
int main (void){
    unsigned char in[INSIZE+1];
    ...
    if (c == `-')
    {
        i=0;
        idx_in= 9 ; //<- by EZProofC
        c =48 ; //<- by EZProofC
        while ((`0' <= c) && (c <= `9'))
        {
            j =3 ; //<- by EZProofC
            i =33 ; //<- by EZProofC
            idx_in= 15 ; //<- by EZProofC
            assert(idx_in<15); //<- by EZProofC
            c =51 ; //<- by EZProofC
        }
    }
}
```

Directly from the counter-example

Fourth Step: Code execution and confirmation of errors

Module	Result of the Execution
newz_tTflag_arr_two_loops_bad.c	Line:15:main:Assertion & ‘idx in<15’ failed. Aborted

```
Counterexample:

(......)
State 97 file ccode.pre/pre_tTflag_arr_two_loops_bad.c line 13 function main thread 0
----------------------------------------------------
  pre_tTflag_arr_two_loops_bad::main::1::i=33 (00000000000000000000000000100001)

State 98 file ccode.pre/pre_tTflag_arr_two_loops_bad.c line 14 function main thread 0
----------------------------------------------------
  pre_tTflag_arr_two_loops_bad::main::1::idx_in=15 (00000000000000000000000000001111)

State 93 file ccode.pre/pre_tTflag_arr_two_loops_bad.c line 15 function main thread 0
----------------------------------------------------
  pre_tTflag_arr_two_loops_bad::main::1::c=51 (00110011)

Violated property:
  file ccode.pre/pre_tTflag_arr_two_loops_bad.c line 15 function main
  array `in' upper bound
  idx_in < 15

VERIFICATION FAILED
```

Agenda

1. Introduction
2. Background
3. Proposed Method
4. **Experimental Results**
5. Related Work
6. Conclusions and Future Work

Planning and Design the Experiments

In order to evaluate the proposed method:

- ✓ We considered **211** ANSI-C programs
- ✓ Six different ANSI-C programs benchmarks:
  EUREKA, SNU, WCET, NEC, Siemens (SIR) and CBMC (C bounded model checker) tutorial.

During this empirical evaluation:

(1) Application of the EZProofC method;

(2) Application of the tool Frama-C with value analysis plug-in

frama-c -val <file.c>

(3) Application of the tool Frama-C with the plug-in Jessie

frama-c -jessie -jessie-atp=z3 <file.c>

Experiment's Execution and Results Analysis

ID	Module	#L	Frama-C		EZProofC/ESBMC					SC and SW
			#W	TW	#P	TC	TV	#V	CE	
1	bf5_20.c	49	6	<1s	33	<1s	<60s	0	-	0
2	bubble_sort1_13.c	51	2	<1s	25	<1s	<15s	0	-	0
3	fibonacci1_13.c	25	1	<1s	1	<1s	<1s	0	-	0
4	init_sel_sort1_13.c	54	2	<1s	25	<1s	<15s	0	-	0
5	minmax1_13.c	19	6	<1s	9	<1s	<3s	0	-	0
6	n_k_gray_codes1_13.c	45	36	<1s	22	<1s	<120s	0	-	11
7	prim4_8.c	79	12	<1s	30	<1s	<60s	0	-	3
8	selection_sort1_13.c	54	2	<1s	25	<1s	<15s	0	-	0
9	crc_det.c	125	1	<1s	15	<1s	≈840s	0	-	1
10	cnt_nondet.c	139	0	<1s	16	<1s	<60s	0	-	0
11	minmax_unsafe1_13.c	19	6	<1s	9	<1s	<4s	1	16	0
12	no_init_bubble_sort_safe1_13.c	25	2	<1s	14	<1s	<7s	1	32	1
13	no_init_sel_sort1_13.c	41	5	<1s	25	<1s	<15s	12	144	3
14	no_init_sel_sort_safe1_13.c	28	5	<1s	14	<1s	<7s	1	32	3
15	no_init_sel_sort_unsafe1_13.c	28	5	<1s	14	<1s	<7s	1	32	3
16	strcmp1_13.c	15	4	<1s	6	<1s	≈14400s	3	80	0
17	sum1_13.c	21	1	<1s	1	<1s	<1s	1	48	0
18	sum_array1_13.c	11	1	<1s	7	<1s	<3s	1	8	0
19	D_CBMC_assert_unsafy.c	15	4	<1s	1	<1s	<1s	1	24	0
20	D_CBMC_bound_array.c	16	2	<1s	10	<1s	<10s	1	30	1
21	D_CBMC_division_by_zero.c	32	3	<1s	1	<1s	<1s	1	24	1
22	ex26.c	29	4	<1s	8	<1s	≈420s	2	1236	1
23	select_det.c	122	3	<1s	39	<1s	≈14400s	3	40	1
24	Siemens_print_tokens2.c	508	90	<1s	51	<1s	≈18000s	1	3344	34

Experiment's Execution and Results Analysis

ID	Module	#L	Frama-C		EZProofC/ESBMC					SC and SW
			#W	TW	#P	TC	TV	#V	CE	

C Program name

LOC

Experiment's Execution and Results Analysis

ID	Module	#L	Frama-C		EZProofC/ESBMC					SC and SW
			#W	TW	#P	TC	TV	#V	CE	

Number of Warnings

Execution time of Frama-C with Value Analysis plug-in

Experiment's Execution and Results Analysis

ID	Module	#L	Frama-C		EZProofC/ESBMC					SC and SW
			#W	TW	#P	TC	TV	#V	CE	

#P: Number of Properties

TC: Time - Properties Identification

Experiment's Execution and Results Analysis

ID	Module	#L	Frama-C		EZProofC/ESBMC					SC and SW
			#W	TW	#P	TC	TV	#V	CE	

Time spent by ESBMC

Counterexample LOC

Violated Properties

Experiment's Execution and Results Analysis

ID	Module	#L	Frama-C		EZProofC/ESBMC					SC and SW
			#W	TW	#P	TC	TV	#V	CE	

Claims in common between EZProofC and Frama-C

EZProofC tool are available at :
https://sites.google.com/site/ezproofc/

Experiment's Execution and Results Analysis

ID	Module	#L	Frama-C		EZProofC/ESBMC					SC and SW
			#W	TW	#P	TC	TV	#V	CE	
1	bf5_20.c	49	6	<1s	33	<1s	<60s	0	-	0
2	bubble_sort1_13.c	51	2	<1s	25	<1s	<15s	0	-	0
3	fibonacci1_13.c	25	1	<1s	1	<1s	<1s	0	-	0
4	init_sel_sort1_13.c	54	2	<1s	25	<1s	<15s	0	-	0
5	minmax1_13.c	19	6	<1s	9	<1s	<3s	0	-	0
6	n_k_gray_codes1_13.c	45	36	<1s	22	<1s	<120s	0	-	11
7	prim4_8.c	79	12	<1s	30	<1s	<60s	0	-	3
8	selection_sort1_13.c	54	2	<1s	25	<1s	<15s	0	-	0
9	crc_det.c	125	1	<1s	15	<1s	≈840s	0	-	1
10	cnt_nondet.c	139	0	<1s	16	<1s	<60s	0	-	0

EZProofC did not find any violated property

Experiment's Execution and Results Analysis

ID	Module	#L	Frama-C		EZProofC/ESBMC					SC and SW
			#W	TW	#P	TC	TV	#V	CE	
11	minmax_unsafe1_13.c	19	6	<1s	9	<1s	<4s	1	16	0
12	no_init_bubble_sort_safe1_13.c	25	2	<1s	14	<1s	<7s	1	32	1
13	no_init_sel_sort1_13.c	41	5	<1s	25	<1s	<15s	12	144	3
14	no_init_sel_sort_safe1_13.c	28	5	<1s	14	<1s	<7s	1	32	3
15	no_init_sel_sort_unsafe1_13.c	28	5	<1s	14	<1s	<7s	1	32	3
16	strcmp1_13.c	15	4	<1s	6	<1s	≈14400s	3	80	0
17	sum1_13.c	21	1	<1s	1	<1s	<1s	1	48	0
18	sum_array1_13.c	11	1	<1s	7	<1s	<3s	1	8	0
19	D_CBMC_assert_unsafy.c	15	4	<1s	1	<1s	<1s	1	24	0
20	D_CBMC_bound_array.c	16	2	<1s	10	<1s	<10s	1	30	1
21	D_CBMC_division_by_zero.c	32	3	<1s	1	<1s	<1s	1	24	1
22	ex26.c	29	4	<1s	8	<1s	≈420s	2	1236	1
23	select_det.c	122	3	<1s	39	<1s	≈14400s	3	40	1
24	Siemens_print_tokens2.c	508	90	<1s	51	<1s	≈18000s	1	3344	34

All possible scenarios in terms of LOC???

Experiment's Execution and Results Analysis

ID	Module	#L	Frama-C		EZProofC/ESBMC					SC and SW
			#W	TW	#P	TC	TV	#V	CE	
11	minmax_unsafe1_13.c	19	6	<1s	9	<1s	<4s	1	16	0
12	no_init_bubble_sort_safe1_13.c	25	2	<1s	14	<1s	<7s	1	32	1
13	no_init_sel_sort1_13.c	41	5	<1s	25	<1s	<15s	12	144	3
14	no_init_sel_sort_safe1_13.c	28	5	<1s	14	<1s	<7s	1	32	3
15	no_init_sel_sort_unsafe1_13.c	28	5	<1s	14	<1s	<7s	1	32	3
16	strcmp1_13.c	15	4	<1s	6	<1s	≈14400s	3	80	0
17	sum1_13.c	21	1	<1s	1	<1s	<1s	1	48	0
18	sum_array1_13.c	11	1	<1s	7	<1s	<3s	1	8	0
19	D_CBMC_assert_unsafy.c	15	4	<1s	1	<1s	<1s	1	24	0
20	D_CBMC_bound_array.c	16	2	<1s	10	<1s	<10s	1	30	1
21	D_CBMC_division_by_zero.c	32	3	<1s	1	<1s	<1s	1	24	1
22	ex26.c	29	4	<1s	8	<1s	≈420s	2	1236	1
23	select_det.c	122	3	<1s	39	<1s	≈14400s	3	40	1
24	Siemens_print_tokens2.c	508	90	<1s	51	<1s	≈18000s	1	3344	34

Frama-C X EZProofC

Experiment's Execution and Results Analysis

ID	Module	#L	Frama-C		EZProofC/ESBMC					SC and SW
			#W	TW	#P	TC	TV	#V	CE	
11	minmax_unsafe1_13.c	19	6	<1s	9	<1s	<4s	1	16	0
12	no_init_bubble_sort_safe1_13.c	25	2	<1s	14	<1s	<7s	1	32	1
13	no_init_sel_sort1_13.c	41	5	<1s	25	<1s	<15s	12	144	3
14	no_init_sel_sort_safe1_13.c	28	5	<1s	14	<1s	<7s	1	32	3
15	no_init_sel_sort_unsafe1_13.c	28	5	<1s	14	<1s	<7s	1	32	3
16	strcmp1_13.c	15	4	<1s	6	<1s	≈14400s	3	80	0
17	sum1_13.c	21	1	<1s	1	<1s	<1s	1	48	0
18	sum_array1_13.c	11	1	<1s	7	<1s	<3s	1	8	0
19	D_CBMC_assert_unsafy.c	15	4	<1s	1	<1s	<1s	1	24	0
20	D_CBMC_bound_array.c	16	2	<1s	10	<1s	<10s	1	30	1
21	D_CBMC_division_by_zero.c	32	3	<1s	1	<1s	<1s	1	24	1
22	ex26.c	29	4	<1s	8	<1s	≈420s	2	1236	1
23	select_det.c	122	3	<1s	39	<1s	≈14400s	3	40	1
24	Siemens_print_tokens2.c	508	90	<1s	51	<1s	≈18000s	1	3344	34

Frama-C X EZProofC

Why??
Values Analysis plug-in

Experiment's Execution and Results Analysis

ID	Module	#L	Frama-C		EZProofC/ESBMC					SC and SW
			#W	TW	#P	TC	TV	#V	CE	
11	minmax_unsafe1_13.c	19	6	<1s	9	<1s	<4s	1	16	0
12	no_init_bubble_sort_safe1_13.c	25	2	<1s	14	<1s	<7s	1	32	1
13	no_init_sel_sort1_13.c	41	5	<1s	25	<1s	<15s	12	144	3
14	no_init_sel_sort_safe1_13.c	28	5	<1s	14	<1s	<7s	1	32	3
15	no_init_sel_sort_unsafe1_13.c	28	5	<1s	14	<1s	<7s	1	32	3
16	strcmp1_13.c	15	4	<1s	6	<1s	≈14400s	3	80	0
17	sum1_13.c	21	1	<1s	1	<1s	<1s	1	48	0
18	sum_array1_13.c	11	1	<1s	7	<1s	<3s	1	8	0
19	D_CBMC_assert_unsafy.c	15	4	<1s	1	<1s	<1s	1	24	0
20	D_CBMC_bound_array.c	16	2	<1s	10	<1s	<10s	1	30	1
21	D_CBMC_division_by_zero.c	32	3	<1s	1	<1s	<1s	1	24	1
22	ex26.c	29	4	<1s	8	<1s	≈420s	2	1236	1
23	select_det.c	122	3	<1s	39	<1s	≈14400s	3	40	1
24	Siemens_print_tokens2.c	508	90	<1s	51	<1s	≈18000s	1	3344	34

Jessie plug-in??

Frama-C X EZProofC

Why??
Values Analysis plug-in

Agenda

1. Introduction
2. Background
3. Proposed Method
4. Experimental Results
5. Related Work
6. Conclusions and Future Work

Related Work

- ✓ Ji et al.: **Design and Implementation of Retargetable Software Debugger Based on GDB**. In: Intl. Conf. on Convergence and Hybrid Information Technology (CHIT). 2008.
  - Fixed entry values X Tests exhautively

- ✓ Taghdiri, M.: **Inferring Specifications to Detect Errors in Code**. In: Intl. Conf. on Automated Software Engineering (ASE). 2004.
  - SAT solver X SMT solver
  - Drawback: Solving only structural properties (constrain configuration)

Related Work

- ✓ Cousot et al.: **The ASTRÉE analyzer**. In: Programming Languages and Systems (PLS). 2005.
  - Analyzes structured C programs, **BUT without** dynamic memory allocation and recursion
  - EZProofC provides support for structures **not supported** by Astrée

Agenda

1. Introduction
2. Background
3. Proposed Method
4. Experimental Results
5. Related Work
6. **Conclusions and Future Work**

Conclusions and Future Work

Proposed Method

- ✓ To help developers **not familiar** with formal verification techniques (find failures);
- ✓ EZProofC is a completely automatic method that **does not need to write specifications**;
- ✓ The experimental results have shown to be very effective over publicly available benchmarks;

Conclusions and Future Work

Future Work

- ✓ Verification with simplifications in the model (e.g. **function-by-function verification**);
- ✓ We intend to extend our experiments to **evaluate the usability** of the proposed method;
- ✓ We also plan to adapt the proposed method to use **other model checkers** (Blast and Java PathFinder) that rely on other abstraction techniques.

Questions ??

Thank you for your attention!

References

Baier, C., Katoen, J.P.: Principles of Model Checking. MIT Press (2008)

Beyer, D., Henzinger, T.A., Jhala, R., Majumdar, R.: The software model checker Blast: Applications to software engineering. In: Int. J. Softw. Tools Technol. Transf. (STTT). vol. 9, pp. 505–525 (2007)

Cordeiro, L., Fischer, B.: Verifying Multi-threaded Software using SMT-based Context-Bounded Model Checking. In: Intl. Conf. on Software Engineering (ICSE). pp. 331–340 (2011)

Cordeiro, L., Fischer, B., Marques-Silva, J.: SMT-Based Bounded Model Checking for Embedded ANSI-C Software. In: IEEE Transactions on Software Engineering (TSE). vol. 99 (2011), http://eprints.ecs.soton.ac.uk/22291/

Havelund, K.: Java PathFinder, A Translator from Java to Promela. In: Intl. SPIN Workshops on Theoretical and Practical Aspects of SPIN Model Checking. p. 152 (1999)

References

ESBMC. Efficient SMT-Based Context Bounded Model Checker. http://esbmc.org.

EUREKA. www.ai-lab.it/eureka/bmc.html.

SNU. http://archi.snu.ac.kr/realtime/benchmark;

WCET. www.mrtc.mdh.se/projects/wcet/benchmarks.html

NEC. http://www.nec-labs.com/research/system;

SIR-SIEMENS. http://sir.unl.edu/portal/index.htm;

CBMC. http://www.cprover.org/cbmc/doc/manual.pdf

UNCRUSTIFY. http://uncrustify.sourceforge.net

VERISEC. http://se.cs.toronto.edu/index.php/Verisec_Suite