

SAD 2021
Static Analysis Days
Online
5.-6. Mai 2021

Verifysoft
TECHNOLOGY

Testen zur Laufzeit und statische Codeanalyse – zwei komplementäre Verfahren

Royd Lütke
Verifysoft Technology GmbH
luedtke@verifysoft.com
+49 781 127 8118-0

1

SAD 2021
Static Analysis Days
Online
5.-6. Mai 2021

Verifysoft
TECHNOLOGY

Agenda

- Fiktives Beispiel: Kaffeeautomat
- Einsatzschwerpunkt für die statische Analyse: Unbestimmtes Verhalten
- Einsatzschwerpunkt für die dynamische Analyse: Funktionale Fehler
- Paradedisziplin der statischen Analyse: Aufdecken von Nebenläufigkeitsfehlern
- Zusammenfassung

www.verifysoft.com

2

2



Fiktives Beispiel: Fehler in Kaffeeautomat


Foto: www.shutterstock.com

www.verifysoft.com

3

3

```
size_t coffeeCharge(size_t product, size_t cups, size_t hardness) {  
    if ((cups < 5) && ((product == 1) || (product == 2))) {  
        return hardness * 4;  
    }  
    else if ((cups < 5) && ((product == 3) || (product == 4))) {  
        return hardness * 6;  
    }  
    else if ((cups < 3) && ((product == 3) || (product == 4))) {  
        return hardness * 3;  
    }  
    else {  
        return hardness * 7;  
    }  
}
```



www.verifysoft.com

4

4



```
size_t getWaterHardness(size_t machine_version) {
    size_t hardness;

    if (machine_version < 7) { //Maschinen ohne Härtesensor
        hardness = 2;
    }
    else if (machine_version > 7) { //Maschinen mit Härtesensor
        hardness = readSensor();
    }
    return hardness;
}
```

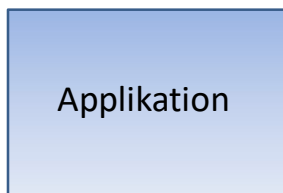
5

Dynamische Analyse

Test #	Funktion	Eingangsparameter	Erwartetes Ergebnis
1	init()	0	5
2	init()	200000	4
3	init()	a	0
4	init()	-1	3,14
5	getParam()	12, 16, 25	0
6			

Testfälle

Überprüfung zur Laufzeit



Messung der Testabdeckung

TER % - MC/DC	TER % - statement	File
Directory: C:\Projects\hcontrol		
79 % - (22/28)	83 % - (20/24)	regulators.c
100 % - (8/8)	100 % - (4/4)	sensors.c
71 % - (5/7)	93 % - (14/15)	service_functions.c
74 % - (14/19)	75 % - (15/20)	zhome.c
79 % - (49/62)	84 % - (53/63)	DIRECTORY OVERALL
79 % - (49/62)	84 % - (53/63)	OVERALL

Testergebnisse



Erwartete Ergebnisse



6

Testfall Nr.	machine_version	product	cups	hardness	Ergebnis	erw. Ergebnis
1	6	1	4	2	8	8
2	6	2	4	2	8	8
3	6	1	6	2	14	14
4	6	2	6	2	14	14
5	7	3	4	2	12	12
6	7	4	4	2	12	12
7	8	3	6	1	7	7
8	8	4	6	1	7	7

7

Testfall Nr.	machine_version	product	cups	hardness	Ergebnis	erw. Ergebnis
9	6	3	2	2	12	6
10	6	4	2	2	12	6
11	6	7	4	2	14	14
12	6	3	6	2	14	14

8

CTC++ Coverage Report - Execution Profile

[Directory Summary](#) | [Files Summary](#) | [Functions Summary](#) | [Untested Code](#) |
To files: [First](#) | [Previous](#) | [Next](#) | [Last](#) | [Index](#) | [No Index](#)

Source file: C:\Testwell\Samples\SAD21\SAD21.cpp
Instrumentation mode: multicondition
TER: 82 % (29/35) structural, 97 % (33/34) statement

Hits/True	False	Line	Source
12		30	size_t coffeeCharge(size_t product, size_t cups, size_t hardness) {
2	10	31	if ((cups < 5) && ((product == 1) (product == 2))) {
1		31	1: (T) && ((T) (C))
1		31	2: (T) && ((F) (T))
5		31	3: (T) && ((F) (F))
5		31	4: (F) && ((C) (C))
2		32	return hardness * 4;
		33	}
4	6	34	else if ((cups < 5) && ((product == 3) (product == 4))) {
2		34	1: (T) && ((T) (C))
2		34	2: (T) && ((F) (T))
1		34	3: (T) && ((F) (F))
5		34	4: (F) && ((C) (C))
4		35	return hardness * 6;
		36	}
0	6	37	else if ((cups < 3) && ((product == 3) (product == 4))) {
0		37	1: (T) && ((T) (C))
0		37	2: (T) && ((F) (T))
0		37	3: (T) && ((F) (F))
0	6	37	4: (F) && ((C) (C))
0		38	return hardness * 3;
		39	}
		40	else {
6		41	return hardness * 7;
		42	}
		43	}
		44	}
		45	}

9

Kontrollflussanalyse

Kontrollflussanomalien

- Nicht erreichbare Anweisungen (Dead Code)
- Sprünge aus Schleifen heraus
- Sprünge in Schleifen hinein
- Programmstücke mit mehreren Ausgängen

10

SAD 2021
Static Analysis Days
03-04

Fiktives Beispiel: Fehler in Kaffeeautomat

Verifysoft TECHNOLOGY

CODESONAR Search code in this analysis for Search | Advanced Search

Home > Kaffeeautomat > Kaffeeautomat analysis 1 > Warning 61977.163756

< Prev (Warning 3 of 3) Next >

Unreachable Computation at SAD21.cpp:32 No properties have been set. | edit properties
Jump to warning location | warning details...

Options

coffeeCharge() C:\Tests\Static_Analysis_Day\SAD21\SAD21\SAD21.cpp

```

24 size_t coffeeCharge(size_t product, size_t cups, size_t hardness) {
25     if ((cups < 5) && ((product == 1) || (product == 2))) {
26         return hardness + 4;
27     }
28     else if ((cups < 5) && ((product == 3) || (product == 4))) {
29         return hardness + 6;
30     }
31     else if ((cups < 3) && ((product == 3) || (product == 4))) {
32         return hardness + 3;
33     }
34     else {
35         return hardness + 7;
36     }
37 }

```

Unreachable Computation

The highlighted code will not execute under any circumstances. This may be because of:

- A function call that does not return.
- A test whose result is always the same: look for a preceding Redundant Condition warning.
- A crashing bug. Look for a preceding Null Pointer Dereference or Division By Zero warning.

www.verifysoft.com 11

11

SAD 2021
Static Analysis Days
03-04

Datenflussanalyse

Verifysoft TECHNOLOGY

Datenflussanomalien

- Referenzierende Verwendung nicht initialisierter Variablen
- Nichtverwenden eines Variablenwertes

- Auf einem Programmpfad wird ein undefinierter Wert einer Variablen gelesen
- Einer Variablen wird ein Wert zugewiesen. Dieser wird jedoch ungültig, ohne jemals verwendet zu werden
- Eine Variable erhält zum zweiten Mal einen Wert, ohne dass der erste verwendet wurde

www.verifysoft.com 12

12

SAD 2021
Static Analysis Days
03-04. Mar 2021

Fiktives Beispiel: Fehler in Kaffeeautomat **Verifysoft**
TECHNOLOGY

CODESONAR Search code in this analysis for

Home > Kaffeeautomat > Kaffeeautomat analysis 1 > Warning 61979.163759

< Prev (Warning 1 of 3) Next >

Uninitialized Variable at SAD21.cpp:20 *No properties have been set.* | edit properties
Jump to warning location ↓ *warning details...*

Show Events | Options

```

getWaterHardness() C:\Tests\Static_Analysis_Day\SAD21\SAD21\SAD21.cpp
12 size_t getWaterHardness(size_t machine_version) {
13     size_t hardness;
14     if (machine_version < 7) { //machines w/o hardness sensor
15         hardness = 2;
16     }
17     else if (machine_version > 7) { //machines with hardness sensor
18         hardness = readSensor();
19     }
20     return hardness;

```

Uninitialized Variable
hardness was not initialized.
• hardness was defined at SAD21.cpp:13.
The issue can occur if the highlighted code executes.
Show: All events | Only primary events

www.verifysoft.com 13

13

SAD 2021
Static Analysis Days
03-04. Mar 2021

Verifysoft
TECHNOLOGY

**Einsatzschwerpunkt für
die statische Analyse:**

Unbestimmtes Verhalten

www.verifysoft.com 14

14

Memory Leak

```
void simple_leak(void){
    char *p;
    p = (char*)malloc(12);
    if (!p)
        return;
    if (!some_function()){
        free(p);
        return;
    }
    if (!some_other_function())
        return;
    free(p);
    return;
}
```

15

Buffer Overrun

```
char initBuffer(void){
    char buffer[5];

    buffer[0] = 'A';
    buffer[1] = 'B';
    buffer[2] = 'C';
    buffer[3] = 'D';
    buffer[4] = 'E';
    buffer[5] = 'F'; /* Ueberlauf */

    return buffer[5];
}
```

16

Datei_1.c

```
void double_free(void){  
    int *p;  
  
    p = (int*)malloc(sizeof(int));  
  
    if (p){  
        afunction(p);  
        free(p);  
    }  
}
```

Datei_2.c

```
void afunction(int* p1){  
    if (p1)  
        free(p1);  
}
```

17



18

Fehlende Funktionalität

```
void featureSelect(int button) {  
    switch (button){  
        case 1:  
            feature_1();  
            break;  
        case 2:  
            feature_2();  
            break;  
        case 3:  
            feature_3();  
            break;  
        default:  
            return;  
    }  
}
```



Fehlende Funktionalität

19

Optimierungsfehler

Originalcode

```
volatile int pressure = 25;  
  
bool checkPressure() {  
    if (pressure > 56) {  
        openReliefValve();  
        return false;  
    }  
    return true;  
}
```

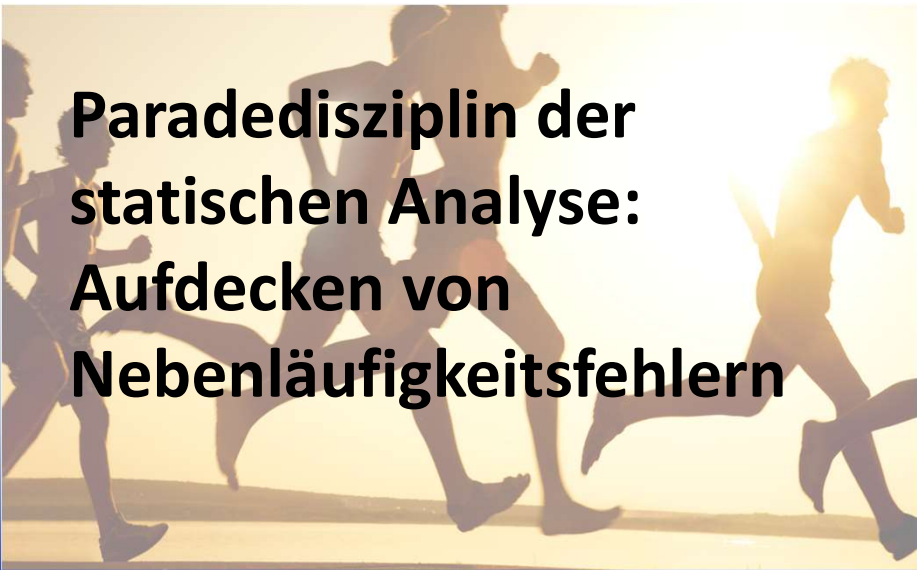
Fehlerhafte Optimierung

```
int pressure = 25;  
  
bool checkPressure() {  
    return true;  
}
```

20

```
int sum(int number) {  
    int i, sum = 0;  
    while (i <= number) {  
        if (i == 42)  
            continue;  
        sum = sum + i;  
        i++;  
    }  
    return sum;  
}
```

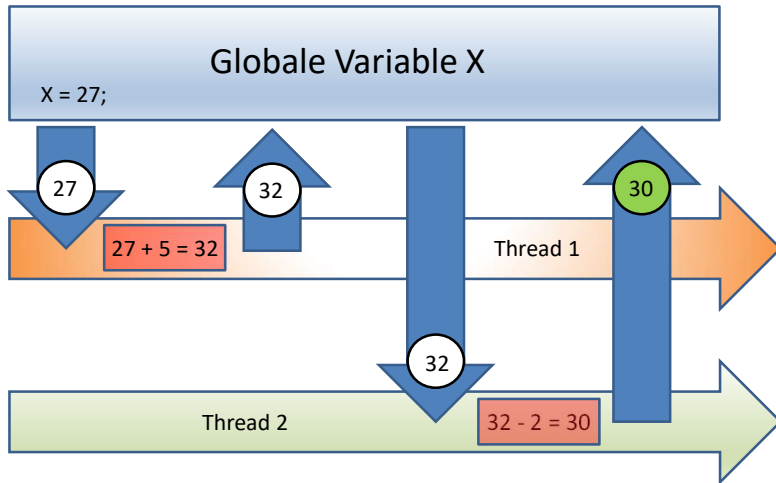
21



**Paradedisziplin der
statischen Analyse:
Aufdecken von
Nebenläufigkeitsfehlern**

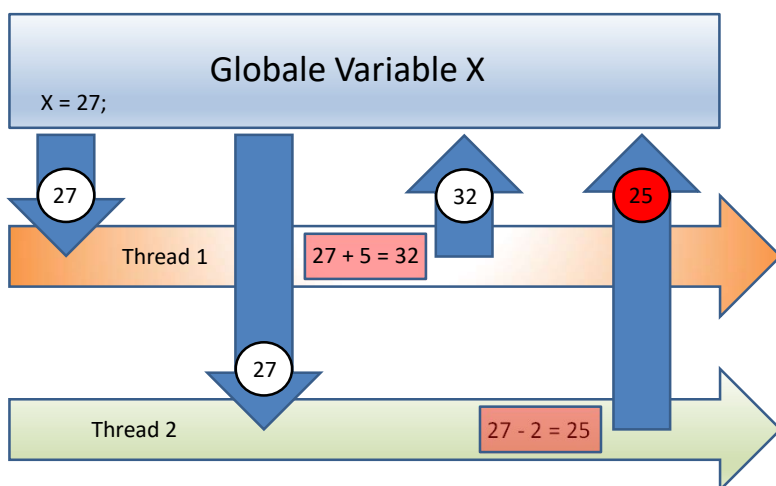
22

Nebenläufigkeitsprobleme Threads synchronisiert



23

Nebenläufigkeitsprobleme Threads nicht synchronisiert (Data Race)



24

Beispiel für Data Race

```
pthread_mutex_t mutex0 = PTHREAD_MUTEX_INITIALIZER;
double sum = 0.;

void *calculate(void *params){
    double d, w;
    unsigned long l;
    unsigned long* part = (unsigned long*) params;
    d = 1.0 / part[2];
    for (l = part[0]; l < part[1]; ++l){
        w = d * (1 + 0.5);
        /* pthread_mutex_lock(&mutex0); */
        sum += 4.0 / (1.0 + w * w);
        /* pthread_mutex_unlock(&mutex0); */
    }
}
```

25

Beispiel für Data Race

```
int main(void){
    pthread_t pth1, pth2;
    unsigned long segment_info0[3] = {0, 500000, 1000000};
    unsigned long segment_info1[3] = {500001, 1000000, 1000000};
    if (pthread_create(&pth1, NULL, calculate, (void*) &segment_info0) != 0){
        printf("Thread creation failed!\n");
        exit (EXIT_FAILURE);
    }
    if (pthread_create(&pth2, NULL, calculate, (void*) &segment_info1) != 0){
        printf("Thread creation failed!\n");
        exit (EXIT_FAILURE);
    }
    pthread_join(pth1, NULL);
    pthread_join(pth2, NULL);
    printf("PI = %25.201f\n", sum / segment_info0[2]);
    return 0;
}
```

26

SAD 2021
Static Analysis Days
02/04 - 06/04/2021

Beispiel für Data Race

Verifysoft
TECHNOLOGY

CODESONAR Search code in this analysis for Search Advanced Search

Home > concurrency > concurrency analysis 1 > Warning 662.163761

< Prev (Warning 2 of 3) Next >

Data Race at pi.c:21 No properties have been set. | edit properties
Jump to warning location | warning details...

Show Events | Change View | Options

thread 1

```

calculate() C:\Users\vroyd\uedtke\Documents\Static_Analysis_Day_2021\pi.c
12 void *calculate(void *params) {
13     double d, w;
14     unsigned long l;
15
16     unsigned long* part = (unsigned long*)params;
17     d = 1.0 / part[2];
18     for (l = part[0]; l < part[1]; ++l) {
19         w = d * (l + 0.5);
20         /* pthread_mutex_lock(&mutex0); */
21         sum += 4.0 / (1.0 + w * w);
    
```

Data Race

This code reads from global variable `sum`.

- The other thread writes to `sum`. See **other access**.
- No locks are currently held so a race with the other thread may occur.
- Compilers and processors reorder accesses to shared variables, so even source code that looks safe can be vulnerable to data races.

The issue can occur if the highlighted code executes.

Show: All events | Only primary events

www.verifysoft.com

27

27

SAD 2021
Static Analysis Days
02/04 - 06/04/2021

Deadlock

Verifysoft
TECHNOLOGY

The diagram shows two threads, Thread 1 and Thread 2, represented as horizontal arrows pointing right. Thread 1 (orange) has a yellow box labeled 'A' (lock A) and a black padlock icon. Below it is the text 'Freigabe A' (Release A). Thread 2 (green) has a blue box labeled 'B' (lock B) and a black padlock icon. Below it is the text 'Freigabe B' (Release B). Red arrows point from the padlock on Thread 1 to the lock 'B' on Thread 2, and from the padlock on Thread 2 to the lock 'A' on Thread 1, forming a cycle of dependencies.

www.verifysoft.com

28

28

Funktion durchlaufen von Thread 1

```
void *calculate1(void *params){
    double d, w;
    unsigned long l;
    unsigned long* part = (unsigned long*)params;
    d = 1.0 / part[2];
    for (l = part[0]; l < part[1]; ++l){
        w = d * (1 + 0.5);
        pthread_mutex_lock(&mutex0);
        pthread_mutex_lock(&mutex1);
        sum += 4.0 / (1.0 + w * w);
        pthread_mutex_unlock(&mutex1);
        pthread_mutex_unlock(&mutex0);
    }
}
```

Funktion durchlaufen von Thread 2

```
void *calculate2(void *params){
    double d, w;
    unsigned long l;
    unsigned long* part = (unsigned long*)params;
    d = 1.0 / part[2];
    for (l = part[0]; l < part[1]; ++l){
        w = d * (1 + 0.5);
        pthread_mutex_lock(&mutex1);
        pthread_mutex_lock(&mutex0);
        sum += 4.0 / (1.0 + w * w);
        pthread_mutex_unlock(&mutex0);
        pthread_mutex_unlock(&mutex1);
    }
}
```

CODESONAR Search code in this analysis for Search Advanced Search

Home > deadlock > deadlock analysis 1 > Warning 61900.163765

< Prev (Warning 8 of 9) Next >

Nested Locks at pl_lock.c:22 No properties have been set. | edit properties
Jump to warning location | warning details..

Show Events | Options

calculate1() C:\Users\vroyd\Idea\Documents\Static_Analysis_Day_2021\pl_lock.c

```

3 #include <pthread.h>
4
5 pthread_mutex_t mutex0 = PTHREAD_MUTEX_INITIALIZER;
6 pthread_mutex_t mutex1 = PTHREAD_MUTEX_INITIALIZER;
7
8 double sum = 0.;
9
10
11
12
13 void *calculate1(void *params) {
14     double d, w;
15     unsigned long l;
16
17     unsigned long* part = (unsigned long*)params;
18     d = 1.0 / part[2];
19     for (l = part[0]; l < part[1]; ++l) {
20         w = d * (1 + 0.5);
21         pthread_mutex_lock(&mutex0);
22         pthread_mutex_lock(&mutex1);
23     }
24 }
```

Event 2: pthread_mutex_lock() acquires mutex0. See related event 2. ▲ ▼ hide

Event 4: mutex1 is passed to pthread_mutex_lock(). ▲ ▼ hide

Nested Locks

The execution thread acquires lock mutex1 while already holding lock mutex0.

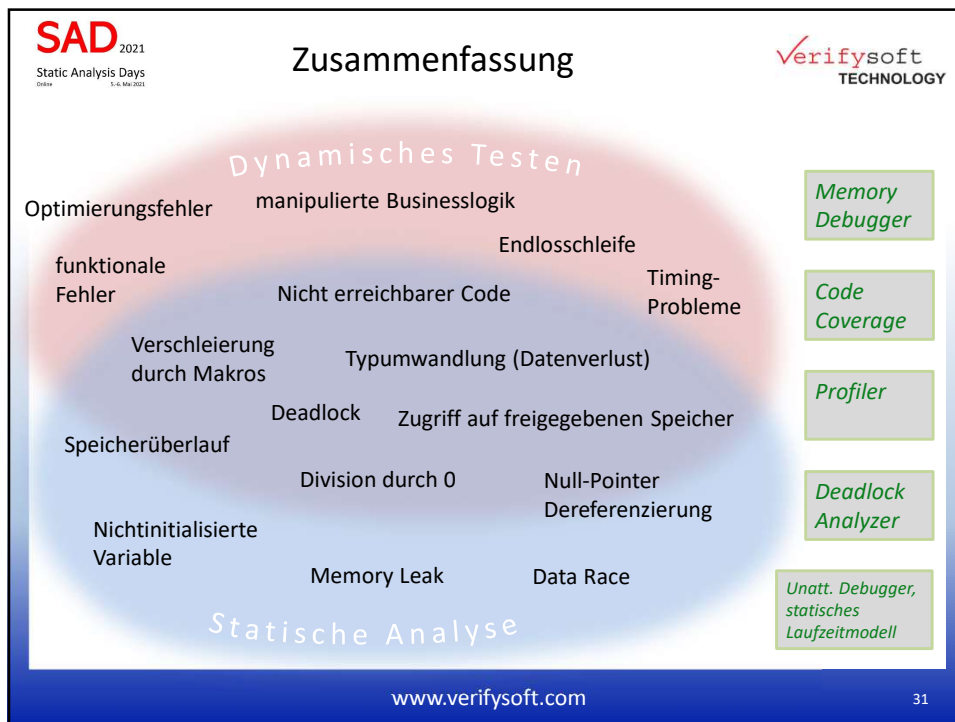
- pthread_mutex_lock() acquires lock mutex1. See related event 4
- Lock mutex0 was acquired at pl_lock.c:21 and has not been released. See related event 3
- Warnings of this class indicate cases where a thread holds multiple locks at the same time. If no thread can hold more than one lock at a time, the code is guaranteed to be deadlock-free.
- If any of the lock acquisition or release operations in this warning are misidentified, see the manual section on Resolving Lock Operation Identification Problems.

The issue can occur if the highlighted code executes.

Show: All events | Only primary events

```

23     sum += 4.0 / (1.0 + w * w);
24     pthread_mutex_unlock(&mutex1);
```



31

SAD 2021
Static Analysis Days
03.-06. Mai 2021

Verifysoft
TECHNOLOGY

Das Testen zur Laufzeit triggert Fehler, die sich funktional auswirken. Voraussetzungen für ein sicheres Aufdecken sind das Vorhandensein eines zum jeweiligen Fehler spezifischen Testfalles und ein deterministisches, reproduzierbares Fehlverhalten.

Die statische Quellcodeanalyse beweist ihre Stärke insbesondere beim Auffinden von Fehlern, die zu undefiniertem Verhalten führen.

Nur der gemeinsame Einsatz beider Verfahren ist Voraussetzung für eine hinreichende Softwarequalität.

www.verifysoft.com

32

32

VIELEN DANK !

Royd Lüdtké
Verifysoft Technology GmbH
luedtke@verifysoft.com
+49 781 127 8118-8