Incremental Retrospective Assertion Checking

10 August 2023

19th KeY Symposium

—Bergen, Norway—

Lukas Grätz

Software Engineering Group
Departement of Computer Science
Technical University of Darmstadt, Germany

Introduction

The Problem

- Legacy software system (in use since years)
- Computes or decides
- White box, we have the sources!

The Problem

- Legacy software system (in use since years)
- Computes or decides
- White box, we have the sources!

- But: Is every result or decision *correct*?

The Problem

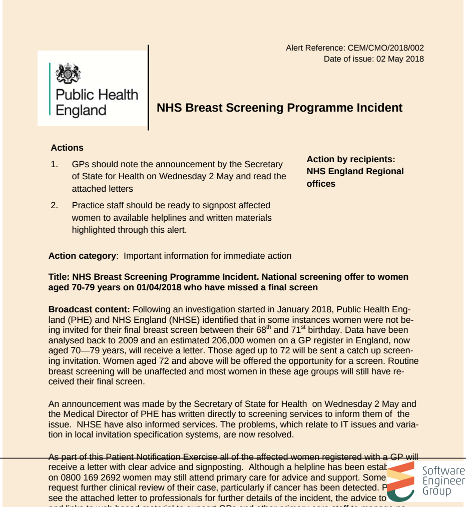
- Legacy software system (in use since years)
- Computes or decides
- White box, we have the sources!

- But: Is every result or decision *correct*?

  (*right* or *intended* or *expected* or *in accordance with the law* or ...)

Examples

- Sending invitations for breast cancer screenings
- Ticket booking for railway trips
- Your application
- ...



Alert Reference: CEM/CMO/2018/002
Date of issue: 02 May 2018

Public Health England

NHS Breast Screening Programme Incident

Actions

1. GPs should note the announcement by the Secretary of State for Health on Wednesday 2 May and read the attached letters
2. Practice staff should be ready to signpost affected women to available helplines and written materials highlighted through this alert.

Action by recipients:
NHS England Regional offices

Action category: Important information for immediate action

Title: NHS Breast Screening Programme Incident. National screening offer to women aged 70-79 years on 01/04/2018 who have missed a final screen

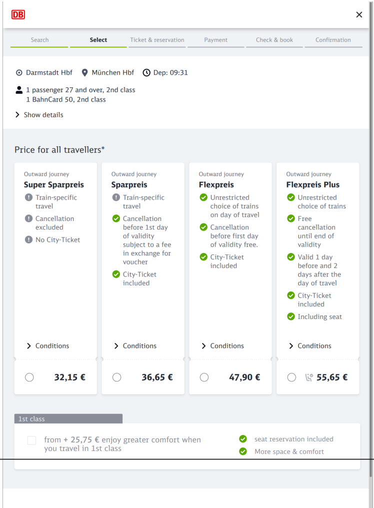
Broadcast content: Following an investigation started in January 2018, Public Health England (PHE) and NHS England (NHSE) identified that in some instances women were not being invited for their final breast screen between their 68th and 71st birthday. Data have been analysed back to 2009 and an estimated 206,000 women on a GP register in England, now aged 70—79 years, will receive a letter. Those aged up to 72 will be sent a catch up screening invitation. Women aged 72 and above will be offered the opportunity for a screen. Routine breast screening will be unaffected and most women in these age groups will still have received their final screen.

An announcement was made by the Secretary of State for Health on Wednesday 2 May and the Medical Director of PHE has written directly to screening services to inform them of the issue. NHSE have also informed services. The problems, which relate to IT issues and variation in local invitation specification systems, are now resolved.

As part of this Patient Notification Exercise all of the affected women registered with a GP will receive a letter with clear advice and signposting. Although a helpline has been estab
on 0800 169 2692 women may still attend primary care for advice and support. Some
request further clinical review of their case, particularly if cancer has been detected. P
see the attached letter to professionals for further details of the incident, the advice to
and links to web based material to support GPs and other primary care staff to manage ne

Examples

- Sending invitations for breast cancer screenings
- Ticket booking for railway trips
- Your application
- ...



DB ×

Search | **Select** | Ticket & reservation | Payment | Check & book | Confirmation

Darmstadt Hbf München Hbf Dep: 09:31

1 passenger 27 and over, 2nd class
1 BahnCard 50, 2nd class

› Show details

Price for all travellers*

Outward journey **Super Sparpreis**	Outward journey **Sparpreis**	Outward journey **Flexpreis**	Outward journey **Flexpreis Plus**
Train-specific travel Cancellation excluded No City-Ticket	Train-specific travel ✓ Cancellation before 1st day of validity subject to a fee in exchange for voucher ✓ City-Ticket included	✓ Unrestricted choice of trains on day of travel ✓ Cancellation before first day of validity free. ✓ City-Ticket included	✓ Unrestricted choice of trains ✓ Free cancellation until end of validity ✓ Valid 1 day before and 2 days after the day of travel ✓ City-Ticket included ✓ Including seat
› Conditions	› Conditions	› Conditions	› Conditions
○ 32,15 €	○ 36,65 €	○ 47,90 €	○ 55,65 €

1st class

□ from + 25,75 € enjoy greater comfort when you travel in 1st class

✓ seat reservation included
✓ More space & comfort

Examples

- Sending invitations for breast cancer screenings
- Ticket booking for railway trips
- Your application

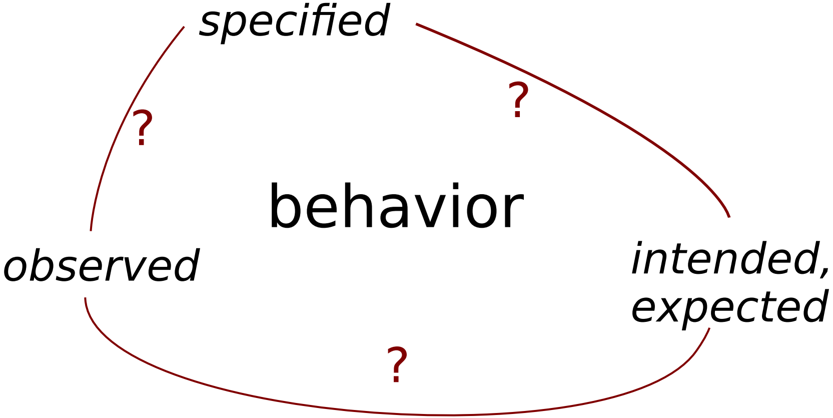
- ...

Examples

- Sending invitations for breast cancer screenings
- Ticket booking for railway trips
- Your application

- ...

Semantic Bugs (Logic Bugs)



specified

?

?

behavior

observed

intended,
expected

?

Semantic Bugs (Logic Bugs)

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Definition

A *semantic bug* violates:

- observed behavior = specified behavior
- observed behavior = expected behavior

Software
Engineering
Group

Semantic Bugs (Logic Bugs)

Definition

A *semantic bug* violates:

- observed behavior = specified behavior
- observed behavior = expected behavior

☞ We need *specification* or *domain expert* judgment!

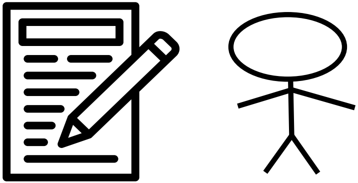
Semantic Bugs (Logic Bugs)

Definition

A *semantic bug* violates:

- observed behavior = specified behavior
- observed behavior = expected behavior

☞ We need *specification* or *domain expert* judgment!



Semantic Bugs (Logic Bugs)

Definition

A *semantic bug* violates:

- observed behavior = specified behavior
- observed behavior = expected behavior

☞ We need *specification* or *domain expert* judgment!

Crashes, deadlocks, memory errors etc. are *not* semantic bugs

- Other approaches & tools for bug finding
- No specification needed
- No domain expert needed

Existing bug finding approaches

TECHNISCHE UNIVERSITÄT DARMSTADT

- Design by contract
- Post-hoc static verification
- Regression test cases
- Code review
- (Trace) debugging

Software Engineering Group

Goals

- Design by contract less specification effort
- Post-hoc static verification less specification effort
- Regression test cases
- Code review
- (Trace) debugging

Goals

- Design by contract less specification effort
- Post-hoc static verification less specification effort
- Regression test cases better coverage
- Code review
- (Trace) debugging

Goals

- Design by contract less specification effort
- Post-hoc static verification less specification effort
- Regression test cases better coverage
- Code review more systematic
- (Trace) debugging

Goals

- Design by contract less specification effort
- Post-hoc static verification less specification effort
- Regression test cases better coverage
- Code review more systematic
- (Trace) debugging better visualization/navigation

Goals

- Design by contract less specification effort
- Post-hoc static verification less specification effort
- Regression test cases better coverage
- Code review more systematic
- (Trace) debugging better visualization/navigation

SO MANY GOALS!

The Setting

Software to validate

- With source code

The Setting

Software to validate

- With source code

Program runs R

- $R = \{r_1, r_2, \ldots, r_N\}$
- Large collection $N \gg 0$
- Real recorded data

The Setting

Software to validate

- With source code

Domain expert

- Understands source code
- Knows expected behavior in domain
- Can validate & justify

Program runs *R*

- $R = \{r_1, r_2, \ldots, r_N\}$
- Large collection $N \gg 0$
- Real recorded data

The Setting

Software to validate

- With source code

Domain expert

- Understands source code
- Knows expected behavior in domain
- Can validate & justify

Program runs R

- $R = \{r_1, r_2, \ldots, r_N\}$
- Large collection $N \gg 0$
- Real recorded data

Validation assistant

- IDE

Assists expert with:

- Debugging visualization/navigation
- Specification instrumentation

Incremental Specification

Lukas Grätz, Reiner Hähnle, Richard Bubel

Finding Semantic Bugs Fast

FASE 2022

https://doi.org/10.1007/978-3-030-99429-7_8

Cinema Example

```
private double calcDscdPrice(Movie movie, int age) {
  //@ assert dscdReg: getDiscount(age) == 0   assuming <regular>;
  return movie.getPrice() * (1 - getDiscount(age)/100.0);
}

public void nextTicket(Scanner input) {
  int age = input("Enter age: ");
  //@ assume regular: 16 <= age && age < 65;

  int movieNumber = input("Select movie (1/2): ");
  Movie movie = movies[movieNumber];

  double dscdPrice = calcDscdPrice(movie, age);
  printf("Your price: %.2f €\n", dscdPrice);
}
```

Buggy max Example

```
int max(int a[]) {

  int m = a[0];
  for (int k=0; k < a.length; k++) {
     if ( m < a[k]) {
        m = a[k++];
     }
  }

  return m;
}
```

Checking—Buggy max Example

100 random runs from input arrays

```
{61, 66, 86},
{53, 37},
{84, 95, 53},
{70},
{11, 26,  6},
{15, 41,  2, 65},
{83, 33},
{79, 89},
{85, 28},
{43, 60, 59, 61},
{59, 73,  7},
{35, 85,  2, 79},
{26},
{85, 59, 68},
{38},
{35, 38, 36, 55},
{ 4, 78},
{32, 71, 22},
{25, 23, 34},
{47, 92, 64},
{60, 10},
{25, 96, 54,  0},
{89, 30, 39, 27},
{ 5,  7, 17},
{94, 19},
{66, 89, 77},
{57,  0, 53, 68},
{87, 18, 93, 76},
{49},
{96, 60},
{21, 92,  1},
{34, 89, 46},
{10, 68},
{34, 41, 91},
{25, 68, 41, 27},
{35, 47, 75, 66},
{27, 54, 47, 69},
{91, 84},
{10, 50, 82, 22},
{81, 73, 26, 15},
{63, 16, 74, 23},
{34},
{30, 10, 46, 16},
{40, 73, 63},
{80, 58, 67},
{47, 53, 83, 71},
{80,  8},
{47, 36},
{30, 50,  1, 44},
{23},
{35},
{35, 77, 92},
{67, 16},
{85, 90, 80, 15},
{ 4, 76, 70},
{48,  4, 33},
{81, 28, 92},
{71,  1, 56, 27},
{76, 94, 27, 15},
{56, 56, 69, 91},
{80, 60},
{40, 49, 95, 8},
{32, 12, 34},
{11, 30},
{87, 53, 59},
{62, 76},
{0},
{93, 54, 46, 23},
{15, 75},
{35, 50, 32, 37},
{50, 61, 99, 43},
{99},
{91, 51, 77},
{22, 65, 63},
{95, 82,  0},
{63},
{ 5, 58},
{ 8,  3},
{66, 80},
{87, 36, 87, 45},
{24},
{85},
{15, 44, 17, 24},
{92, 37, 47},
{ 6},
{44, 31},
{96, 20, 51},
{56},
{ 4, 27, 88, 21},
{30},
{63, 69},
{57, 34, 29, 52},
{38, 76, 90},
{96},
{86},
{34, 30, 10, 27},
{21, 22},
{57, 27, 66, 60},
{85, 73, 39},
{47, 32, 12}
```

Checking—Buggy max Example

100 random runs from input arrays

```
{61, 66, 86},
{53, 37},
{84, 95, 53},
{70},
{11, 26,  6},
{15, 41,  2, 65},
{83, 33},
{79, 89},  Assistant
{85, 28},
{43, 60, 59, 61},
{59, 73,  7},
{35, 85,  2, 79},
{26},
{85, 59, 68},
{38},
{35, 38, 36, 55},
{ 4, 78},
{32, 71, 22},
{25, 23, 34},
{47, 92, 64},
{60, 10},
{25, 96, 54,  0},
{89, 30, 39, 27},
{ 5,  7, 17},
{94, 19},
{66, 89, 77},
{57,  0, 53, 68},
{87, 18, 93, 76},
{49},
{96, 60},
{21, 92,  1},
{34, 89, 46},
{10, 68},
{34, 41, 91},
{25, 68, 41, 27},
{35, 47, 75, 66},
{27, 54, 47, 69},
{91, 84},
{10, 50, 82, 22},
{81, 73, 26, 15},
{63, 16, 74, 23},
{34},
{30, 10, 46, 16},
{40, 73, 63},
{80, 58, 67},
{47, 53, 83, 71},
{80,  8},
{47, 36},
{30, 50,  1, 44},
{23},
{35},
{35, 77, 92},
{67, 16},
{85, 90, 80, 15},
{ 4, 76, 70},
{48,  4, 33},
{81, 28, 92},
{71,  1, 56, 27},
{76, 94, 27, 15},
{56, 56, 69, 91},
{80, 60},
{40, 49, 95, 8},
{32, 12, 34},
{11, 30},
{87, 53, 59},
{62, 76},
{0},
{93, 54, 46, 23},
{15, 75},
{35, 50, 32, 37},
{50, 61, 99, 43},
{99},
{91, 51, 77},
{22, 65, 63},
{95, 82,  0},
{63},
{ 5, 58},
{ 8,  3},
{66, 80},
{87, 36, 87, 45},
{24},
{85},
{15, 44, 17, 24},
{92, 37, 47},
{ 6},
{44, 31},
{96, 20, 51},
{56},
{ 4, 27, 88, 21},
{30},
{63, 69},
{57, 34, 29, 52},
{38, 76, 90},
{96},
{86},
{34, 30, 10, 27},
{21, 22},
{57, 27, 66, 60},
{85, 73, 39},
{47, 32, 12}
```

Buggy max Example

Validation Step (1)

Assistant

```
int max(int a[]) {                                   // a = {79, 89}

  int m = a[0];
  for (int k=0; k < a.length; k++) {
     if ( m < a[k]) {
        m = a[k++];
     }
  }

  return m;                                          // m = 89
}
```

ssistant

Buggy max Example

Validation Step (1)

```
    int max(int a[]) {                              Assistant  // a = {79, 89}

      int m = a[0];
      for (int k=0; k < a.length; k++) {
         if ( m < a[k]) {
            m = a[k++];
         }
      }

      return m;                                                // m = 89
ssistant }
```

Assistant: Is *this run* valid?

Buggy max Example

Validation Step (1)

Assistant

```
int max(int a[]) {                                   // a = {79, 89}

  int m = a[0];
  for (int k=0; k < a.length; k++) {
     if ( m < a[k]) {
        m = a[k++];
     }
  }
  //@ assert max1res: m==a[1] assuming;
  return m;                                          // m = 89
}
```

ssistant

Assistant: Is *this run* valid?

Expert: ...adds `max1res`...

Buggy max Example

Validation Step (1)

Assistant

```
int max(int a[]) {                                       // a = {79, 89}

  int m = a[0];
  for (int k=0; k < a.length; k++) {
     if ( m < a[k]) {
        m = a[k++];
     }
  }
  //@ assert max1res: m==a[1] assuming;
  return m;                                              // m = 89
}
```

ssistant

Assistant: Is *this run* valid?

Expert: ...adds `max1res`...

Expert: Because `a[1]` is the result (`max1res`).

Buggy max Example

Validation Step (2)

```
    int max(int a[]) {                            Assistant  // a = {79, 89}

      int m = a[0];
      for (int k=0; k < a.length; k++) {
         if ( m < a[k]) {
            m = a[k++];
         }
      }
Assistant //@ assert max1res: m==a[1] assuming;
      return m;                                              // m = 89
    }
```

Buggy max Example

Validation Step (2)

```
    int max(int a[]) {                            Assistant   // a = {79, 89}

      int m = a[0];
      for (int k=0; k < a.length; k++) {
         if ( m < a[k]) {
            m = a[k++];
         }
      }
Assistant //@ assert max1res: m==a[1] assuming;
      return m;                                               // m = 89
    }
```

Assistant: Why is it valid for *this run* to satisfy `max1res`?

Buggy max Example

Validation Step (2)

Assistant

```
int max(int a[]) {                                     // a = {79, 89}
  //@ max1of2: a.length==2 && a[0]<=a[1];
  int m = a[0];
  for (int k=0; k < a.length; k++) {
     if ( m < a[k]) {
        m = a[k++];
     }
  }
  //@ assert max1res: m==a[1] assuming <max1of2>;
  return m;                                            // m = 89
}
```

Assistant

Assistant: Why is it valid for *this run* to satisfy `max1res`?

Expert: Because `max1of2`.

Buggy max Example

100 random runs from input arrays

```
{61, 66, 86},
{53, 37},
{84, 95, 53},
{70},
{11, 26,  6},
{15, 41,  2, 65},
{83, 33},
{79, 89},  Assistant
{85, 28},
{43, 60, 59, 61},
{59, 73,  7},
{35, 85,  2, 79},
{26},
{85, 59, 68},
{38},
{35, 38, 36, 55},
{ 4, 78},
{32, 71, 22},
{25, 23, 34},
{47, 92, 64},
{60, 10},
{25, 96, 54,  0},
{89, 30, 39, 27},
{ 5,  7, 17},
{94, 19},
{66, 89, 77},
{57,  0, 53, 68},
{87, 18, 93, 76},
{49},
{96, 60},
{21, 92,  1},
{34, 89, 46},
{10, 68},
{34, 41, 91},
{25, 68, 41, 27},
{35, 47, 75, 66},
{27, 54, 47, 69},
{91, 84},
{10, 50, 82, 22},
{81, 73, 26, 15},
{63, 16, 74, 23},
{34},
{30, 10, 46, 16},
{40, 73, 63},
{80, 58, 67},
{47, 53, 83, 71},
{80,  8},
{47, 36},
{30, 50,  1, 44},
{23},
{35},
{35, 77, 92},
{67, 16},
{85, 90, 80, 15},
{ 4, 76, 70},
{48,  4, 33},
{81, 28, 92},
{71,  1, 56, 27},
{76, 94, 27, 15},
{56, 56, 69, 91},
{80, 60},
{40, 49, 95,  8},
{32, 12, 34},
{11, 30},
{87, 53, 59},
{62, 76},
{0},
{93, 54, 46, 23},
{15, 75},
{35, 50, 32, 37},
{50, 61, 99, 43},
{99},
{91, 51, 77},
{22, 65, 63},
{95, 82,  0},
{63},
{ 5, 58},
{ 8,  3},
{66, 80},
{87, 36, 87, 45},
{24},
{85},
{15, 44, 17, 24},
{92, 37, 47},
{ 6},
{44, 31},
{96, 20, 51},
{56},
{ 4, 27, 88, 21},
{30},
{63, 69},
{57, 34, 29, 52},
{38, 76, 90},
{96},
{86},
{34, 30, 10, 27},
{21, 22},
{57, 27, 66, 60},
{85, 73, 39},
{47, 32, 12}
```

Buggy max Example

100 random runs from input arrays

```
{61, 66, 86},        {60, 10},            {63, 16, 74, 23},    {80, 60},            {24},
{53, 37},            {25, 96, 54,  0},    {34},                {40, 49, 95, 8},     {85},
{84, 95, 53},        {89, 30, 39, 27},    {30, 10, 46, 16},    {32, 12, 34},        {15, 44, 17, 24},
{70},                { 5,  7, 17},        {40, 73, 63},        {11, 30},            {92, 37, 47},
{11, 26,  6},        {94, 19},            {80, 58, 67},        {87, 53, 59},        { 6},
{15, 41,  2, 65},    {66, 89, 77},        {47, 53, 83, 71},    {62, 76},            {44, 31},
{83, 33},            {57,  0, 53, 68},    {80,  8},            {0},                 {96, 20, 51},
{79, 89},            {87, 18, 93, 76},    {47, 36},            {93, 54, 46, 23},    {56},
{85, 28},            {49},                {30, 50,  1, 44},    {15, 75},            { 4, 27, 88, 21},
{43, 60, 59, 61},    {96, 60},            {23},                {35, 50, 32, 37},    {30},
{59, 73,  7},        {21, 92,  1},        {35},                {50, 61, 99, 43},    {63, 69},
{35, 85,  2, 79},    {34, 89, 46},        {35, 77, 92},        {99},                {57, 34, 29, 52},
{26},                {10, 68},            {67, 16},            {91, 51, 77},        {38, 76, 90},
{85, 59, 68},        {34, 41, 91},        {85, 90, 80, 15},    {22, 65, 63},        {96},
{38},                {25, 68, 41, 27},    { 4, 76, 70},        {95, 82,  0},        {86},
{35, 38, 36, 55},    [illegible] 47, 75, 66},    {48,  4, 33},        {63},                {34, 30, 10, 27},
{ 4, 78},            {27, 54, 47, 69},    {81, 28, 92},        { 5, 58},            {21, 22},
{32, 71, 22},        {91, 84},            {71,  1, 56, 27},    { 8,  3},            {57, 27, 66, 60},
{25, 23, 34},        {10, 50, 82, 22},    {76, 94, 27, 15},    {66, 80},            {85, 73, 39},
{47, 92, 64},        {81, 73, 26, 15},    {56, 56, 69, 91},    {87, 36, 87, 45},    {47, 32, 12}
```

Assistant

Buggy max Example

Validation Step (3)

Assistant

```
int max(int a[]) {                                  // a = {35, 38, 36, 55}
  //@ max1of2: a.length==2 && a[0]<=a[1];

  int m = a[0];
  for (int k=0; k < a.length; k++) {
     if ( m < a[k]) {
        m = a[k++];
     }
  }
  //@ assert max1res: m==a[1] assuming <max1of2>;

  return m;                                         // m = 55
}
```

ssistant

Buggy max Example

Validation Step (3)

Assistant

```
int max(int a[]) {                                      // a = {35, 38, 36, 55}
  //@ max1of2: a.length==2 && a[0]<=a[1];

  int m = a[0];
  for (int k=0; k < a.length; k++) {
     if ( m < a[k]) {
        m = a[k++];
     }
  }
  //@ assert max1res: m==a[1] assuming <max1of2>;

  return m;                                             // m = 55
}
```

ssistant

Assistant: Is *this run* valid?

Buggy max Example

Validation Step (3)

```
    int max(int a[]) {                        Assistant  // a = {35, 38, 36, 55}
      //@ max1of2: a.length==2 && a[0]<=a[1];

      int m = a[0];
      for (int k=0; k < a.length; k++) {
         if ( m < a[k]) {
            m = a[k++];
         }
      }
      //@ assert max1res: m==a[1] assuming <max1of2>;
      //@ assert max3res: m==a[3] assuming;
      return m;                                          // m = 55
ssistant }
```

Assistant: Is *this run* valid?

Expert: Because `a[3]` is the result (`max3res`).

Buggy max Example

Validation Step (4)

Assistant

```
int max(int a[]) {        // a = {35, 38, 36, 55}
  //@ max1of2: a.length==2 && a[0]<=a[1];

  int m = a[0];
  for (int k=0; k < a.length; k++) {
    if ( m < a[k]) {
      m = a[k++];
    }
  }
  //@ assert max1res: m==a[1] assuming <max1of2>;
  //@ assert max3res: m==a[3] assuming;
  return m;               // m = 55
}
```

Assistant

Buggy max Example

Validation Step (4)

```
int max(int a[]) {                                  // a = {35, 38, 36, 55}
  //@ max1of2: a.length==2 && a[0]<=a[1];

  int m = a[0];
  for (int k=0; k < a.length; k++) {
     if ( m < a[k]) {
        m = a[k++];
     }
  }
  //@ assert max1res: m==a[1] assuming <max1of2>;
  //@ assert max3res: m==a[3] assuming;
  return m;                                         // m = 55
}
```

Assistant

Assistant

Assistant: Why is it valid for *this run* to satisfy `max3res`?

Buggy max Example

Validation Step (4)

```
int max(int a[]) {                               Assistant   // a = {35, 38, 36, 55}
  //@ max1of2: a.length==2 && a[0]<=a[1];
  //@ max3of4: a.length==4 && a[0]<=a[3] && a[1]<=a[3] && a[2]<=a[3];
  int m = a[0];
  for (int k=0; k < a.length; k++) {
     if ( m < a[k]) {
        m = a[k++];
     }
  }
  //@ assert max1res: m==a[1] assuming <max1of2>;
Assistant  //@ assert max3res: m==a[3] assuming <max3of4>;
  return m;                                                  // m = 55
}
```

Assistant: Why is it valid for *this run* to satisfy `max3res`?

Expert: Because `max3of4`.

Buggy max Example

100 random runs from input arrays

```
{61, 66, 86},
{53, 37},
{84, 95, 53},
{70},
{11, 26,  6},
{15, 41,  2, 65},
{83, 33},
{79, 89},
{85, 28},
{43, 60, 59, 61},
{59, 73,  7},
{35, 85,  2, 79},
{26},
{85, 59, 68},
{38},
{35, 38, 36, 55},
{ 4, 78},
{32, 71, 22},
{25, 23, 34},
{47, 92, 64},
```

```
{60, 10},
{25, 96, 54,  0},
{89, 30, 39, 27},
{ 5,  7, 17},
{94, 19},
{66, 89, 77},
{57,  0, 53, 68},
{87, 18, 93, 76},
{49},
{96, 60},
{21, 92,  1},
{34, 89, 46},
{10, 68},
{34, 41, 91},
{25, 68, 41, 27},
        47, 75, 66},
{27, 54, 47, 69},
{91, 84},
{10, 50, 82, 22},
{81, 73, 26, 15},
```

Assistant

```
{63, 16, 74, 23},
{34},
{30, 10, 46, 16},
{40, 73, 63},
{80, 58, 67},
{47, 53, 83, 71},
{80,  8},
{47, 36},
{30, 50,  1, 44},
{23},
{35},
{35, 77, 92},
{67, 16},
{85, 90, 80, 15},
{ 4, 76, 70},
{48,  4, 33},
{81, 28, 92},
{71,  1, 56, 27},
{76, 94, 27, 15},
{56, 56, 69, 91},
```

```
{80, 60},
{40, 49, 95, 8},
{32, 12, 34},
{11, 30},
{87, 53, 59},
{62, 76},
{0},
{93, 54, 46, 23},
{15, 75},
{35, 50, 32, 37},
{50, 61, 99, 43},
{99},
{91, 51, 77},
{22, 65, 63},
{95, 82,  0},
{63},
{ 5, 58},
{ 8,  3},
{66, 80},
{87, 36, 87, 45},
```

```
{24},
{85},
{15, 44, 17, 24},
{92, 37, 47},
{ 6},
{44, 31},
{96, 20, 51},
{56},
{ 4, 27, 88, 21},
{30},
{63, 69},
{57, 34, 29, 52},
{38, 76, 90},
{96},
{86},
{34, 30, 10, 27},
{21, 22},
{57, 27, 66, 60},
{85, 73, 39},
{47, 32, 12}
```

Buggy max Example

Validation Step (4)

Assistant

```
int max(int a[]) {                                   // a = {56, 56, 69, 91}
  //@ max1of2: a.length==2 && a[0]<=a[1];
  //@ max3of4: a.length==4 && a[0]<=a[3] && a[1]<=a[3] && a[2]<=a[3];
  int m = a[0];
  for (int k=0; k < a.length; k++) {
     if ( m < a[k]) {
        m = a[k++];
     }
  }
  //@ assert max1res: m==a[1] assuming <max1of2>;
  //@ assert max3res: m==a[3] assuming <max3of4>;
  return m;                                          // m = 69
}
```

Assistant: Why is it valid for *this run* to satisfy `max3res`?

Expert: Because `max3of4`.

Buggy max Example

Validation Step (4)

```
int max(int a[]) {                                  // a = {56, 56, 69, 91}
  //@ max1of2: a.length==2 && a[0]<=a[1];
  //@ max3of4: a.length==4 && a[0]<=a[3] && a[1]<=a[3] && a[2]<=a[3];
  int m = a[0];
  for (int k=0; k < a.length; k++) {
     if ( m < a[k]) {
        m = a[k++];
     }
  }
  //@ assert max1res: m==a[1] assuming <max1of2>;
  //@ assert max3res: m==a[3] assuming <max3of4>;
  return m;                                         // m = 69
}
```

Checking failed, bug detected!

Assistant

Assistant: Why is it valid for *this run* to satisfy `max3res`?

Expert: Because `max3of4`.

General Description

Validation Step

1. Tool chooses existing assertion *a*, run *r*
2. Expert judges *a* correct for *r*
   - ☞ Otherwise: Bug found!
3. Expert adds assertion references `<`L_1`,...,`L_n`>` to **`assuming`** of *a*

3a Expert also adds all referenced assertions

4. Verification tools check *a* with new assumptions

(WIP) Program Run Debugging & Visualization in the IDE

(Student Software Project)

Status Quo

Conservative Debugging in IDEs

- Set breakpoint
- Debug ’Program’
- Visualization:
  1. Current line
  2. Current variable values

Better

Visualization of the whole program run

1. Control Flow Path
2. All variable values

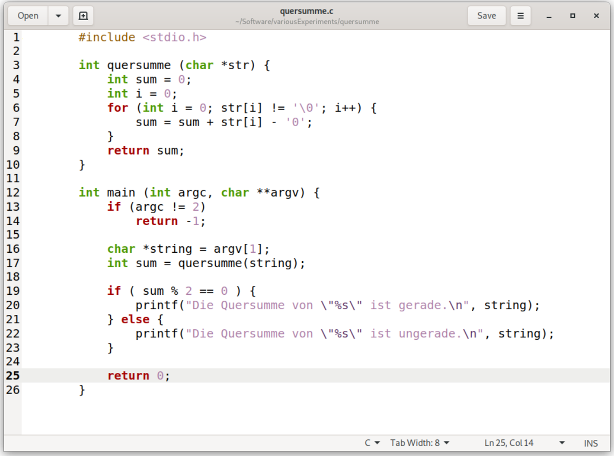
Better

Visualization of the whole program run

1. Control Flow Path
2. All variable values

☞ Simple idea, isn't it?

1. Control Flow Path

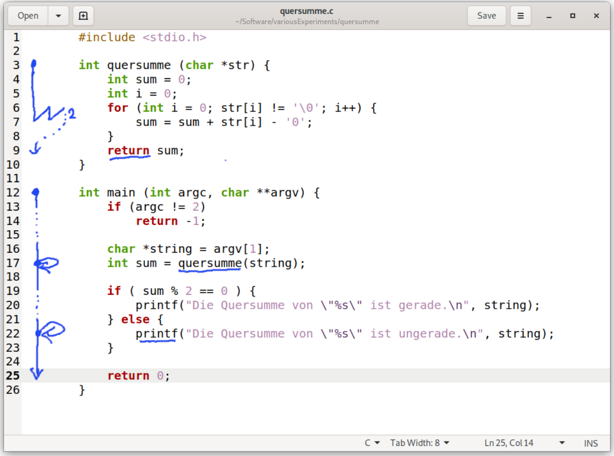


```c
#include <stdio.h>

int quersumme (char *str) {
    int sum = 0;
    int i = 0;
    for (int i = 0; str[i] != '\0'; i++) {
        sum = sum + str[i] - '0';
    }
    return sum;
}

int main (int argc, char **argv) {
    if (argc != 2)
        return -1;

    char *string = argv[1];
    int sum = quersumme(string);

    if ( sum % 2 == 0 ) {
        printf("Die Quersumme von \"%s\" ist gerade.\n", string);
    } else {
        printf("Die Quersumme von \"%s\" ist ungerade.\n", string);
    }

    return 0;
}
```

☞ Control flow of whole program run

- *Arrow:* Code line executed
- *Pointed:* Conditioned code skipped
- *Zickzack:* Loop multiple times
- *Circle arrow:* Function call
- *Link:* Jump (call/return)

1. Control Flow Path



```c
#include <stdio.h>

int quersumme (char *str) {
    int sum = 0;
    int i = 0;
    for (int i = 0; str[i] != '\0'; i++) {
        sum = sum + str[i] - '0';
    }
    return sum;
}

int main (int argc, char **argv) {
    if (argc != 2)
        return -1;

    char *string = argv[1];
    int sum = quersumme(string);

    if ( sum % 2 == 0 ) {
        printf("Die Quersumme von \"%s\" ist gerade.\n", string);
    } else {
        printf("Die Quersumme von \"%s\" ist ungerade.\n", string);
    }

    return 0;
}
```

☞ Control flow of whole program run

- *Arrow:* Code line executed
- *Pointed:* Conditioned code skipped
- *Zickzack:* Loop multiple times
- *Circle arrow:* Function call
- *Link:* Jump (call/return)

Record, Replay, Validate

Record and Deterministic Replay

- Omniscient recording/debugging (gdb)
  - ✗ High Overhead

Record and Deterministic Replay

- Omniscient recording/debugging (gdb)
  - ✗ High Overhead

- Manual interface instrumentation (liblog, R2)

Record and Deterministic Replay

- Omniscient recording/debugging (gdb)
  - ✗ High Overhead

- Manual interface instrumentation (liblog, R2)

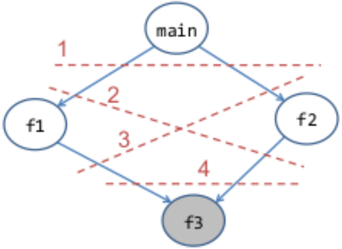
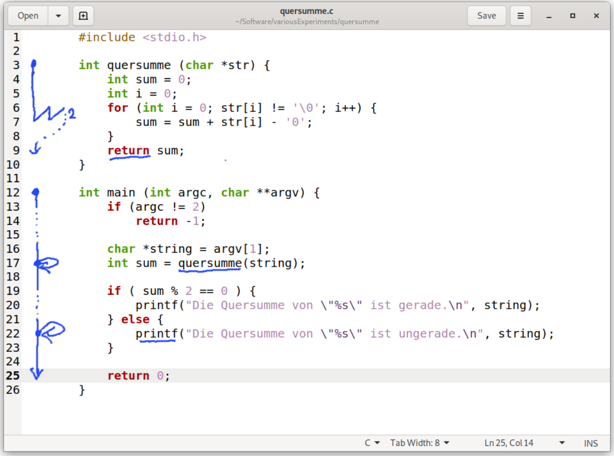


Figure 2: Four cuts in a call graph for record and replay. The function `f3` interacts with the environment.

Record and Deterministic Replay

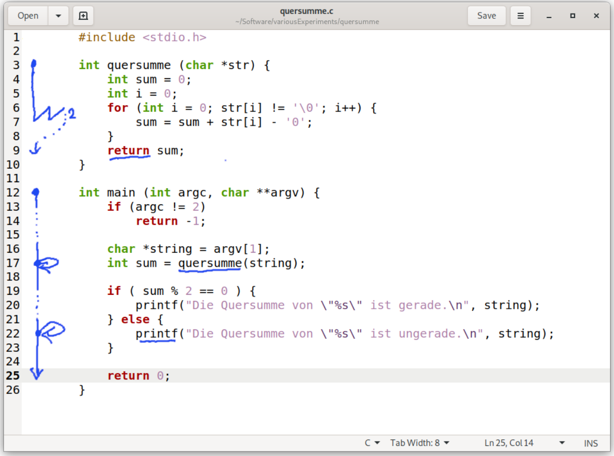
- Omniscient recording/debugging (gdb)
  - ✗ High Overhead
- Manual interface instrumentation (liblog, R2)
- OS interface (RR debugger)
  - ✓ Low Overhead
  - ? Stability, Security
  - ✗ Portability

Control Flow Recording



```c
#include <stdio.h>

int quersumme (char *str) {
    int sum = 0;
    int i = 0;
    for (int i = 0; str[i] != '\0'; i++) {
        sum = sum + str[i] - '0';
    }
    return sum;
}

int main (int argc, char **argv) {
    if (argc != 2)
        return -1;

    char *string = argv[1];
    int sum = quersumme(string);

    if ( sum % 2 == 0 ) {
        printf("Die Quersumme von \"%s\" ist gerade.\n", string);
    } else {
        printf("Die Quersumme von \"%s\" ist ungerade.\n", string);
    }

    return 0;
}
```

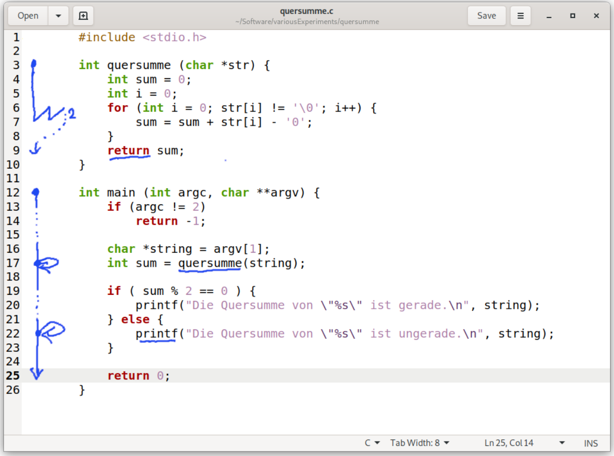
Control Flow Recording



```c
#include <stdio.h>

int quersumme (char *str) {
    int sum = 0;
    int i = 0;
    for (int i = 0; str[i] != '\0'; i++) {
        sum = sum + str[i] - '0';
    }
    return sum;
}

int main (int argc, char **argv) {
    if (argc != 2)
        return -1;

    char *string = argv[1];
    int sum = quersumme(string);

    if ( sum % 2 == 0 ) {
        printf("Die Quersumme von \"%s\" ist gerade.\n", string);
    } else {
        printf("Die Quersumme von \"%s\" ist ungerade.\n", string);
    }

    return 0;
}
```

- ☞ Recorded trace (simplified):
  0 110 0

Control Flow Recording



```
#include <stdio.h>

int quersumme (char *str) {
    int sum = 0;
    int i = 0;
    for (int i = 0; str[i] != '\0'; i++) {
        sum = sum + str[i] - '0';
    }
    return sum;
}

int main (int argc, char **argv) {
    if (argc != 2)
        return -1;

    char *string = argv[1];
    int sum = quersumme(string);

    if ( sum % 2 == 0 ) {
        printf("Die Quersumme von \"%s\" ist gerade.\n", string);
    } else {
        printf("Die Quersumme von \"%s\" ist ungerade.\n", string);
    }

    return 0;
}
```

☞ Recorded trace (simplified):
0 110 0

Retrospective Assertion Checking

	Approach	Replay	Assertion Checking
1	Record all inputs	Concrete re-execution	*Run-time assertion checking*
2	Deterministic replay	Using a *debugger* (*RR* debugger)	Interactive in debugger
3	Control flow recording	Symbolic execution	Check symbolic path condition + negated assertion

Retrospective Assertion Checking

	Approach	Replay	Assertion Checking
1	Record all inputs	Concrete re-execution	*Run-time assertion checking*
2	Deterministic replay	Using a *debugger* (*RR* debugger)	Interactive in debugger
3	Control flow recording	Symbolic execution	Check symbolic path condition + negated assertion

☞ Approaches old, except for assertion checking

✓ We implemented approach 3

(WIP) A Theory of Sound Dependent Assertions

Problem

- Dependent assertions (our 2022 paper)
- Semantics unclear

Problem

- Dependent assertions (our 2022 paper)
- Semantics unclear
- Solution: Use a meta logic
- Different syntax

Box & Diamond

Assertion in line 2 valid *for all* iterations of block in line 1

```
1  while (...) {
2      assert expr ;
3  }
```

Assertion in line 2 valid *for at least one* iteration of block in line 1

```
1  while (...) {
2      assert expr ;
3  }
```

Box & Diamond

Assertion in line 2 valid *for all* iterations of block in line 1

```
1  while (...) {
2      assert expr with [1]2;
3  }
```

Assertion in line 2 valid *for at least one* iteration of block in line 1

```
1  while (...) {
2      assert expr with <1>2;
3  }
```

Meta-Logic Pattern—“Postcondition” I

```
int bar () {
    assert pre with ...;


    CODE;


    assert post with [1]( 2 → 7);
    return 42;
}
```

Meta-Logic Pattern—“Postcondition” II

```
 1  int bar () {
 2      assert pre with ...;
 3
 4      if (...) {
 5          assert post with [1]( 2 → ⟨4⟩5 );
 6          return 17;
 7      }
 8
 9      return 42;
10  }
```

Meta-Logic Pattern—“Loop-Invariant” I

```
int bar () {
    assert pre with ...;

    for (...) {
        CODE;
        assert linv with [1]( 2 → [3]5 );
    }

    return 42;
}
```

Meta-Logic Pattern—“Postcondition” III

```
int bar () {


    for (...) {
        CODE;
        assert ... with ...;
    }
    assert post with [1](  [3]5 → 7 );
    return 42;
}
```

Meta-Logic Pattern—“Precondition”

```
 1  int bar () {
 2      assert pre with [11]( 12 → [12][1] 2 )
 3
 4
 5      CODE;
 6
 7
 8
 9      return 42;
10  }
11
12  caller () {
13      assert ...;
14      bar ();
15  }
```

Meta Logic Calculus

- Can use standard calculus of modal logic

☞ Propositional logic + axiom K + rule of necessity

Meta Logic Calculus

- Can use standard calculus of modal logic

☞ Propositional logic + axiom K + rule of necessity

- Conjecture: Our meta problems in are $O(n)$

Meta Logic Calculus

- Can use standard calculus of modal logic

☞ Propositional logic + axiom K + rule of necessity

- Conjecture: Our meta problems in are $O(n)$

☞ Some derived rules

$$\begin{array}{rrcrr}
[x]A, & [x](\ A \rightarrow B\ ) & \Longrightarrow & [x]B & \text{(apply post)} \\
[y]A, & [x](\ [y]A \rightarrow B\ ) & \Longrightarrow & [x]B & \text{(apply post-rec)} \\
[y][x]A, & [x](\ A \rightarrow B\ ) & \Longrightarrow & [y][x]B & \text{(apply pre)}
\end{array}$$

Meta Logic Calculus

- Can use standard calculus of modal logic

☞ Propositional logic + axiom K + rule of necessity

- Conjecture: Our meta problems in are $O(n)$

☞ Some derived rules

$$\begin{array}{rrcrr} [x]A, & [x](\ A \to B\ ) & \Longrightarrow & [x]B & \text{(apply post)} \\ [y]A, & [x](\ [y]A \to B\ ) & \Longrightarrow & [x]B & \text{(apply post-rec)} \\ [y][x]A, & [x](\ A \to B\ ) & \Longrightarrow & [y][x]B & \text{(apply pre)} \end{array}$$

☞ Name of meta logic: *modal horn logic*

Insertion Sort

```
 1  void insert(int pos, int array[len]) {
 2     int value = array[pos];
 3     for (int j = pos-1; j ≥ 0; j--) {
 4        if (array[j] ≤ value) {
 5           break;
 6        }
 7        array[j+1] = array[j];
          assert ∀ k; 0 ≤ k ≤ j;   array[ k ] == \old(array[k])
             with [13][14, 1][3]7_1;
          assert ∀ k; j ≤ k ≤ pos; array[k+1] == \old(array[k])
             with [13][14, 1][3]7_3;
 8     }
 9     array[j+1] = value;
       assert ∀ k; 0 ≤ k < pos; array[k] ≤ array[k+1]
          with [13]([14, 1][3](7_1 ∧ 7_3) → ⟨14, 1⟩9_1);
       assert ∀ i; (\count k; 0 ≤ k < len; i ==      array[k] )
                == (\count k; 0 ≤ k < len; i == \old(array[k]))
          with [12][13]([14, 1][3](7_1 ∧ 7_3) → ⟨14, 1⟩9_3);
10  }
```

Insertion Sort

```
12  void insertionSort(int array[len]) {
13     for (int pos = 1; pos < len; pos++) {
14        insert(pos, array);
15     }
       assert ∀ i; (\count k; 0 ≤ k < len; i ==      array[k] )
                == (\count k; 0 ≤ k < len; i == \old(array[k]))
          with [12]([13]⟨14, 1⟩9_3 → 15_1);
       assert ∀ k; 0 ≤ k < len-1; array[k] ≤ array[k+1]
          with [12]([13]⟨14, 1⟩9_1 → 15_4);
16  }
```

Insertion Sort—Meta Logic

$$
\begin{aligned}
&[13][14, 1][3]7_1 \\
&[13][14, 1][3]7_3 \\
&[13]([14, 1][3](7_1 \wedge 7_3) \rightarrow \langle 14, 1\rangle 9_1) \\
[12]&[13]([14, 1][3](7_1 \wedge 7_3) \rightarrow \langle 14, 1\rangle 9_3) \\
[12]&([13]\langle 14, 1\rangle 9_3 \rightarrow 15_1) \\
[12]&([13]\langle 14, 1\rangle 9_1 \rightarrow 15_4)
\end{aligned}
$$

Insertion Sort—Meta Logic

$$
\begin{aligned}
&[13][14,1][3]7_1\\
&[13][14,1][3]7_3\\
&[13]([14,1][3](7_1 \wedge 7_3) \to \langle 14,1\rangle 9_1)\\
&[12][13]([14,1][3](7_1 \wedge 7_3) \to \langle 14,1\rangle 9_3)\\
&[12]([13]\langle 14,1\rangle 9_3 \to 15_1)\\
&[12]([13]\langle 14,1\rangle 9_1 \to 15_4)
\end{aligned}
$$

Derivable: $[12]15_1$ and $[12]15_4$ (easy exercise!)

Note On Verification

- Meta logic inferences $O(n)$
- Verification?

Note On Verification

- Meta logic inferences $O(n)$
- Verification?
- Verification idea

  0. Automatic assignable/accessible clause generation
  1. Start symbolic execution somewhere
  2. Show that the assertion holds (once or never reached, depending on assertion)
  3. Do either:

     3.1 Unroll loop and continue from 2

     3.2 Continue from 1

Note On Verification

- Meta logic inferences $O(n)$
- Verification?
- Verification idea
  0. Automatic assignable/accessible clause generation
  1. Start symbolic execution somewhere
  2. Show that the assertion holds (once or never reached, depending on assertion)
  3. Do either:
     - 3.1 Unroll loop and continue from 2
     - 3.2 Continue from 1

☞ Verification strategies depend on classification as pre-condition, loop-invariant, post-condition, ...

Conclusion

- ✓ Incremental specifications with assertions
- ✓ Retrospective assertion checking
- ✓ Better debugging visualization
- ✓ Modularity
- ✓ Generalization of method contracts*

Initial Goals

TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ✓ Design by contract   less specification effort
- ✓ Post-hoc static verification   less specification effort
- ✓ Regression test cases   better coverage
- ✓ Code review   more systematic
- ✓ (Trace) debugging   better visualization/navigation

Backup

Careful With Normal Assert/Assume in KeY

```
/*@ requires true;
    ensures anything; */
foo () {
   //@ assume false;
}
```

```
/*@ requires true;
    ensures true; */
bar () {
   //@ assert anything;
}
```

Careful With Normal Assert/Assume in KeY

```
/*@ requires true;
    ensures anything; */
foo () {
   //@ assume false;
}
```

- Inconstent specification!
- **ensures** does not follow from **requires** alone!
- Soundness issue: caller is not required to establish any **assume**!

```
/*@ requires true;
    ensures true; */
bar () {
   //@ assert anything;
}
```

- KeY might not verify specification!
- But **ensures** follows trivial!
- Modularity issue: caller cannot use **assert** result!