(How) Did you specify your test suite?

Helmut Veith
Formal Methods in Systems Engineering
Vienna University of Technology

Joint work with
Andreas Holzer, Christian Schallhart, and Michael Tautschnig
The purpose of abstraction is *not* to be vague, but to create a new semantic level in which one can be absolutely precise. We are not there yet.
Software Testing

Random Testing?

Directed Testing?

Manual Testing?

There is no general purpose formalism for white box test case specification!
Standard Coverage Criteria

```
1 int example(int a, int d)
2 {
3   if (a)
4     d = 0;
5   else
6     unimplemented();
7   return d*2;
8 }
```

Basic Block Coverage
Test suite such that each basic block is executed at least once
Standard Coverage Criteria

```c
1  int example(int a, int b, int c, int d) 2  { 3     if ((a || b) && c) 4       d = 0; 5     else 6       unimplemented(); 7     return d*2; 8  }
```

**Decision Coverage**

Conditional statements evaluate to true/false
Standard Coverage Criteria

```c
int example(int a, int b, int c, int d)
{
    if ((a || b) && c)
        d = 0;
    else
        unimplemented();
    return d*2;
}
```

Condition Coverage
Atomic conditions evaluate to true/false
Standard Coverage Criteria

1 int example(int a, int b, int c, int d)
2 {
3   if ((a || b) && c)
4       d = 0;
5   else
6     unimplemented();
7   return d*2;
8 }

Condition/Decision Coverage
Union of condition and decision coverage
1  int example(int a, int b, int c, int d) 2  { 3    if ((a || b) && c) 4      d = 0; 5    else 6      unimplemented(); 7    return d*2; 8  }

Multiple Condition Coverage
All Boolean combinations of atomic conditions in complex conditions
Standard Coverage Criteria

Def-Use Coverage
Cover all definition/use pairs of a variable \( d \)
New Coverage Criteria

```c
1 void insert(int * a, int pos) {
2   ...
3 }
4
5 int main(int argc, char * argv[]) {
6   int i;
7   int A[100];
8
9   for (i=0; i<100; ++i)
10      insert(A, i);
11
12   return 0;
13 }
```

Loop-Bounded Path Coverage
All paths through `main` and `insert`
Pass each statement two times
New Coverage Criteria

```c
1 int partition(int a[], int left, int right) {
2   int v = a[right], i = left - 1, j = right, t;
3   for (;;) {
4     ... ;
5     while (j > left && a[--j] > v) ;
6     if (i >= j) break;
7     t = a[i]; a[i] = a[j]; a[j] = t;
8   }
9   t = a[i]; a[i] = a[right]; a[right] = t;
10  return i;
11 }
```

Cartesian Block Coverage
All pairs and triples of basic blocks in function `partition`
Specs for Working Programmers

```c
1  int example(int a, int b, int c, int d)
2  {
3    if ((a || b) && c)
4      d = 0;
5    else
6      unimplemented();
7    return d*2;
8  }
```

**Cover Specific Lines of Code**

Lines 4 and 6
void eval(int * a, int first, int last) {
  if (first > last) return;
  printf("a[%d]=%d\n", first, a[first]);
  eval(a+1, first+1, last);
  return;
}

int main(int argc, char * argv[]) {
  int i;
  int A[100];
  for (i=0; i<100; ++i)
    insert(A, i);
  eval(A, 0, 100);
  return 0;
}

Restricted Scope of Analysis
Basic block coverage in function eval only
Specs for Working Programmers

```c
1 void sort(int * a, int len) {
  2   int i, t;
  3   for (i=1; i<len; ++i) {
  4     if (compare(a[i-1], a[i])) continue;
  5     t=a[i]; a[i]=a[i-1]; a[i-1]=t;
  6   }
  7 }
  8
  9 void eval(int * a, int len) {
 10   int i;
 11   for (i=0; i < 3; ++i)
 12     printf("a[%d]=%d\n", i, a[i]);
 13 }
 14
 15 int main(int argc, char * argv[]) {
 16   int i; int A[100];
 17   for (i=0; i < 100; ++i) sort(A, 100);
 18   eval(A, 100);
 19   return 0;
 20 }
```

**Interaction Coverage**

Cover all pairs of conditions in `sort` and basic blocks in `eval`
Specs for Working Programmers

```c
1 void sort(int * a, int len) {
2     int i, t;
3     for (i=1; i<len; ++i) {
4         if (compare(a[i-1], a[i])) continue;
5         t=a[i];
6         a[i]=a[i-1];
7         a[i-1]=t;
8     }
9     return;
10 }
```

**Constrained Inputs**

Basic block coverage in *sort*

Each test case shall use list of 2 to 15 elements

How did you specify your test suite?
Specs for Working Programmers

```c
1  int example(int a, int d) 2  { 3    if (a) 4      d = 0; 5    else 6      unimplemented(); 7    return d*2; 8  }
```

Avoid Unfinished Code

Basic block coverage in `example`

Never call `unimplemented`
Simple Coverage Criteria?

- "Condition Coverage" cover all program conditions.

```c
void foo(int x) {
    int a = x > 2 && x < 5;
    if (a) { 0; } else { 1; }
}
```

Test suite:
- `x = 1;`
- `x = 4;`

Condition coverage?

Commercial Tools:
- Coverage Meter, CTC++
- BullseyeCoverage

Software Considerations in Airborne Systems and Equipment Certification

DOCUMENT NO. RTCA/DO-178B
December 1, 1992
Prepared by: SC-167

There is no general purpose formalism for white box test case specification!
Related Work

- Coverage formalizations
  - Using Z (Vilkomir, Bowen FORTEST’08)
  - Temporal logics (Hong et al. TACAS’02)

- Test case generation techniques
  - Random testing, fuzzing
  - Directed testing (Godefroid et al. PLDI’05)
  - Model-checking based (Beyer et al. ICSE’04)

- BLAST query language (Beyer et al. SAS’04)
Challenge: Systematic Approach to White Box Testing?

- Precise Semantics

- Expressive Power
  small number of orthogonal concepts suffice to express large classes of specifications

- Simplicity and Code Independence
  tool for the working programmer
  simple specs easily expressible
  relative stability during code refactoring

- Encapsulation of Language Specifics
  easily adaptable to a large class of imperative programming languages

- Tool Support for Real World Code
  test case generation engines

Database Analogy?
Precise and simple formalism to describe test suites

Precise semantics

High expressive power

Simple specifications easily expressible

Suitable for working programmer
Algorithmic Solution

Tool for the working programmer

Applicability to real world C code (embedded systems)

Efficient test input generation engines
if (ctrl_io_in[SENS_IMPULS]) {
    if (!lastImp) {
        if (ctrl_io_out[MOTOR_ON]) {
            if (directionUp) {
                ++cnt;
            } else {
                --cnt;
            }
        } else if (timImp > 0) {
            if (directionUp) {
                ++cnt;
            } else {
                --cnt;
            }
        }
    }
}

if (ctrl_io_in[SENS_BOTTOM]) {
    cnt = 0;
    cntValid = TRUE;
}

lastImp = ctrl_io_in[SENS_IMPULS];

if (timImp > 0) {
    --timImp;
    if (timImp == 0) {
        if (cmd != CMD_NONE) {
            cmd = CMD_NONE;
        }
    }
}
Query-Driven Program Testing

Algorithmic approach: CAV’08, VMCAI’09
FQL Language Concept: ASE 2010

Existing engines:

- FShell: Based on CBMC (C Bounded Model Checker)
- FilleSh: Uses CPAchecker (Configurable Program Analysis)

http://code.forsyte.de/fshell
Applications of FQL/FShell

- Test case generation
- Certification
- Requirement-driven testing
- Coverage Evaluation
- Reasoning about test specifications

How did you specify your test suite?
FQL Challenge:
Designing a Coverage Specification Language

- Syntax of the program
- Semantics of the program
- Reasonably language independent

- Easy to write
- Easy to understand
- Natural to use
- Predictable performance

- High expressive power
- Tractable to evaluate
FQL Challenge: Designing a Coverage Specification Language

Specification for basic block coverage

„for each basic block in the program
there is a test case in the test suite
which covers the basic block

1. Specifies a test suite, i.e., multiple test cases
2. Contains a universal quantifier
3. Assumes knowledge about programs.
   What IS a basic block for a logic?
4. Has a meaning independent of the program under test
   Can be translated into concrete specifications for a fixed program
FQL Challenge:
Designing a Coverage Specification Language

Universal quantification?

Programming language support?

CFG view versus trace view?

Language independence?

→ Evolve path patterns / regular expressions into a more complex formalism.
FQL Key Concept
Quoted Regular Expressions

A regular expression matches a word
A quoted regular expression matches a finite set of words

„a*“ + „ab|cd“

matches all sets \{X,Y\} where
- X matches „a*“
- Y matches „ab|cd“

\{aaa, ab\}, \{a,cd\}, \{aaaaaaaaaaa,ab\} etc.

Unquoted part without Kleene star → finite sets
FQL Language Concept

- Program Representation: Control Flow Automata
- Single Test Case Specification
- Test Suite Specification
Control Flow Automata (CFA)

```c
int example(int a, int d)
{
    if (a)
        d = 0;
    else
        unimplemented();
    return d*2;
}
```

Henzinger et al. (POPL’02)

Edges
- assume (conditional branches)
- assignment
- function call
- function return
- skip

Nodes: program locations

How did you specify your test suite?
FQL Language Concept

- Program Representation: Control Flow Automata
  - Single Test Case Specification
  - Test Suite Specification
1 int example(int a, int d)  
2 {  
3   if (a)  
4     d = 0;  
5   else  
6     unimplemented();  
7   return d*2;  
8  }

**Cover line 4**  
cover “@4”  
IN: a=2, d=0

**Cover some statement**  
cover “ID”  
IN: a=8192, d=0

How did you specify your test suite?
Single Test Case Specification

```c
int example(int a, int d) {
    if (a)
        d = 0;
    else
        unimplemented();
    return d*2;
}
```

Cover line 4 or line 6
cover “@4+@6”

Cover a basic block
cover “@basicblockentry”

Here:
@basicblockentry tantamount to @4+@6+@7
Single Test Case Specification

```c
1  int example(int a, int d)
2  {
3    if (a)
4      d = 0;
5    else
6      unimplemented();
7    return d*2;
8  }
```

Test case = execution (program path)

“cover line 4” means
- execute some statement(s)
- reach line 4
- execute until program exit

cover “ID*.@4.ID*”

IN:  a=2,  d=0

How did you specify your test suite?
Single Tests by Path Patterns

```c
1  int example(int a, int d)  
2  {  
3    if (a)  
4      d = 0;  
5    else  
6      unimplemented();  
7    return d*2;  
8  }
```

cover “ID* . @4 . ID* + ID* . @6 . ID*”

Path patterns are regular expressions
- Kleene star (*)
- Concatenation (.)
- Alternative/Union (+)

(Could go beyond regular expressions …)
Specification by Path Patterns

Path Pattern Alphabet: Program Syntax and Semantics

- Filter functions: interface to programming language syntax
- Filter expressions
- Assertions over program variables
- CFA edges, nodes and paths

How did you specify your test suite?
Alphabet: Filter Functions

1 int example(int a, int b, int c, int d) 
2 { 
3    if ((a || b) && c) 
4      d = 0; 
5    else 
6      unimplemented(); 
7    return d*2; 
8 } 

cover "@conditionedge"
cover "@conditiongraph"
cover "@decisionedge"
cover "@call(unimplemented)"
cover "@func(example)"

→ Program independence
Alphabet: Filter Expressions

```c
int example(int a, int b, int c, int d) {
    if ((a || b) && c)
        d = 0;
    else
        unimplemented();
    return d*2;
}
```

Graph operations on filters

cover "@7 | @decisionedge"
cover "not(@3)"
cover "setminus(@decisionedge,@6)"
cover "@decisionedge & @4"

How did you specify your test suite?
Alphabet: Program Assertions

```c
int example(int a, int d) {
    if (a)
        d = 0;
    else
        unimplemented();
    return d*2;
}
```

- cover "{a==42}.@3"
  - IN: a=42, d=0

- cover "(@4+@6).{d!=0}"
  - IN: a=0, d=65536

How did you specify your test suite?
FQL Language Concept

- Program Representation: Control Flow Automata
- Single Test Case Specifications
- Test Suite Specifications
Coverage Patterns

```c
int example(int a, int d) {
    if (a)
        d = 0;
    else
        unimplemented();
    return d*2;
}
```

- Cover if and else branch?
- Need more than one test case - a test suite

How did you specify your test suite?
Coverage Patterns

Coverage patterns = **quoted** path patterns

*Each generated path pattern must be covered*

Path patterns may describe an **infinite** language

\[ a^* \cdot b \cdot a^* + cd^* = \{ b, c, ab, cd, aba, \\
  \quad cdd, aab, aaba, aabaa, \ldots \} \]

But quoting operator **blocks expansion**: 

- “\( a^* \cdot b \cdot a^* + cd^* \)” = \( a^*ba^* + cd^* \}
- “\( a^* \cdot b \cdot a^* \)” + “\( cd^* \)” = \( a^*ba^*, cd^* \}

Coverage patterns: Kleene star only within quotes

How did you specify your test suite?
Cover if and else branch

cover "ID*.@4.ID*" + "ID*.@6.ID*"

2 words - 2 test goals

IN:  a=2, d=0

IN:  a=0, d=0
Coverage Patterns

```c
int example(int a, int d) {
    if (a)
        d = 0;
    else
        unimplemented();
    return d*2;
}
```

Can we cover pairs of lines?

“ID*”.(@4+@6+@7).”ID*”.(@4+@6+@7).”ID*”

Coverage pattern describes 9 test goals

IN:  a=67108864, d=0
IN:  a=0, d=0

How did you specify your test suite?
Coverage Patterns

```c
int example(int a, int d) {
    if (a)
        d = 0;
    else
        unimplemented();
    return d*2;
}
```

Recall
@basicblockentry tantamount to @4+@6+@7

cover “ID*”.(@4+@6+@7).”ID*”.(@4+@6+@7).”ID*”
tantamount to
cover “ID*”.@basicblockentry.”ID*”.@basicblockentry.”ID*”

How did you specify your test suite?
```c
int example(int a, int b, int c, int d) {
    if ((a || b) && c)
        d = 0;
    else
        unimplemented();
    return d*2;
}
```

Specs with Edges, Nodes, Paths

How did you specify your test suite?

- $a = 1, c = 1$
- $a = 0, b = 0$
- $a = 0, b = 1, c = 0$
```c
int example(int a, int b, int c, int d)
{
    if ((a || b) && c)
        d = 0;
    else
        unimplemented();
    return d*2;
}
```

Specs with Edges, Nodes, Paths

cover NODES(@conditiongraph)

5 test goals, 2 test cases suffice:
- a = 1, c = 1
- a = 0, b = 0

How did you specify your test suite?
Specs with Edges, Nodes, Paths

```c
int example(int a, int b, int c, int d) {
    if ((a || b) && c)
        d = 0;
    else
        unimplemented();
    return d*2;
}
```

cover PATHS(@conditiongraph, 1)

1 = repetition bound to assure finiteness

5 test goals, 5 test cases required:
- a = 1, c = 1
- a = 0, b = 0
- a = 0, b = 1, c = 0
- a = 0, b = 1, c = 1
- a = 1, c = 0

How did you specify your test suite?
FQL Coverage Specifications

General form of coverage specification

- **in** [scope]
  - Where?
- **cover** [test goals]
  - What?
- **passing** [constraints]
  - How?

"For each test goal inside scope, provide a test case which matches the passing condition."

"Restricted Scope of Analysis"
Condition coverage in function partition with test cases that reach line 7 at least once.

**in** @FUNC(partition)
**cover** @CONDITIONEDGE
**passing** _*.@7._*
FQL Summary

Filter functions
@FILE(), @LINE(), @BASICBLOCKENTRY, @CONDITONEEDGE,
@STMTYPE(), UNION, INTERSECT, SETMINUS, etc.
Encapsulate the interface to the programming language

Filter expressions
union, intersect, compose, not, id, setminus, etc.

Path Patterns
Regular expressions (or more) over filter functions, assertions,
{cond}, >=k, etc.

Coverage Patterns
Quoted regular expressions

CFA view of program

Trace view of program
Full Format FQL Queries

in scope cover goals passing constraints

scope: CFA transformer/filter function
goals: coverage pattern
constraints: path pattern

“For each test goal inside scope, provide a test case which satisfies the passing constraints.”

in @func(partition) cover @conditionedge passing @7
Condition coverage in function partition with test cases that reach line 7 at least once.
FQL Challenge: Designing a Coverage Specification Language

Specification for basic block coverage

"for each basic block in the program
there is a test case in the test suite
which covers the basic block"

1. Specifies a test suite, i.e., multiple test cases
2. Contains a universal quantifier
3. Assumes knowledge about programs.
   What IS a basic block for a logic?
4. Has a meaning independent of the program under test
   Can be translated into set of path patterns for a fixed program
Standard Coverage Criteria

1  int example(int a, int d)
2  {
3    if (a)
4      d = 0;
5    else
6      unimplemented();
7    return d*2;
8  }

Basic Block Coverage
Requires a test suite such that each basic block is executed at least once

cover @basicblockentry
→ (at runtime)
cover @4+@6+@7

How did you specify your test suite?
Standard Coverage Criteria

```c
int partition(int a[], int left, int right) {
    int v = a[right], i = left - 1, j = right, t;
    for (;;) {
        while (a[++i] < v);
        while (j > left && a[--j] > v);
        if (i >= j) break;
        t = a[i]; a[i] = a[j]; a[j] = t;
    }
    t = a[i]; a[i] = a[right]; a[right] = t;
    return i;
}
```

Basic Block Coverage

Requires a test suite such that each basic block is executed at least once

cover @basicblockentry

How did you specify your test suite?
Standard Coverage Criteria

```c
1  int example(int a, int b, int c, int d)
2  {
3      if ((a || b) && c)
4          d = 0;
5      else
6          unimplemented();
7      return d*2;
8  }
```

Decision Coverage
Conditional statements evaluate to true/false

cover  @decisionedge

How did you specify your test suite?
Standard Coverage Criteria

1 int example(int a, int b, int c, int d)  
2 {  
3   if ((a || b) && c)  
4     d = 0;  
5   else  
6     unimplemented();  
7   return d*2;  
8 }  

Condition Coverage  
Atomic conditions evaluate to true/false

cover @conditionedge
Standard Coverage Criteria

```
int example(int a, int b, int c, int d)
{
    if ((a || b) && c)
        d = 0;
    else
        unimplemented();
    return d*2;
}
```

Condition/Decision Coverage
Union of condition and decision coverage

cover @conditionedge + @decisionedge
```c
int example(int a, int b, int c, int d) {
    if ((a || b) && c)
        d = 0;
    else
        unimplemented();
    return d*2;
}
```

**Multiple Condition Coverage**

All Boolean combinations of atomic conditions in complex conditions

`cover paths(@conditiongraph, 1)`

How did you specify your test suite?
Standard Coverage Criteria

```
int example(int a, int d) {
    if (a)
        d = 0;
    else
        unimplemented();
    return d*2;
}
```

**Def-Use Coverage**
Cover all definition/use pairs of a variable \( d \)

```
cover @def(d)."not(@def(d))".@use(d)
```
New Coverage Criteria

```
void insert(int * a, int pos) {
    ...
}

int main(int argc, char * argv[]) {
    int i;
    int A[100];
    for (i=0; i<100; ++i)
        insert(A, i);
    return 0;
}
```

Loop-Bounded Path Coverage
- All paths through `main` and `insert`
- Pass each statement two times

```
cover paths(@func(main) | @func(insert),2)
```
New Coverage Criteria

```c
int partition(int a[], int left, int right) {
    int v = a[right], i = left - 1, j = right, t;
    for (;;) {
        while (a[++i] < v);
        while (j > left && a[--j] > v);
        if (i >= j) break;
        t = a[i]; a[i] = a[j]; a[j] = t;
    }
    t = a[i]; a[i] = a[right]; a[right] = t;
    return i;
}
```

Cartesian Block Coverage

All pairs and triples of basic blocks in function `partition`

cover @basicblockentry->@basicblockentry+
@basicblockentry->@basicblockentry-> @basicblockentry

How did you specify your test suite?
1  int example(int a, int b, int c, int d)  
2  {  
3    if ((a || b) && c)  
4      d = 0;  
5    else  
6      unimplemented();  
7    return d*2;  
8  }

Cover Specific Lines of Code

Lines 4 and 6

cover @4 + @6

How did you specify your test suite?
Specs for Working Programmers

```c
1 void eval(int * a, int first, int last) {
2   if (first > last) return;
3   printf("a[%d]=%d\n", first, a[first]);
4   eval(a+1, first+1, last);
5   return;
6 }
7
8 int main(int argc, char * argv[]) {
9   int i;
10  int A[100];
11
12  for (i=0; i<100; ++i)
13    insert(A, i);
14
15  eval(A, 0, 100);
16  return 0;
17 }
```

**Restricted Scope of Analysis**

Basic block coverage in function `eval` only

cover @basicblockentry & @func(eval)
void sort(int * a, int len) {
    int i, t;
    for (i=1; i<len; ++i) {
        if (compare(a[i-1], a[i])) continue;
        t=a[i]; a[i]=a[i-1]; a[i-1]=t;
    }
}

void eval(int * a, int len) {
    int i;
    for (i=0; i < 3; ++i)
        printf("a[%d]=%d\n", i, a[i]);
}

int main(int argc, char * argv[]) {
    int i; int A[100];
    for (i=0; i < 100; ++i) sort(A, 100);
    eval(A, 100);
    return 0;
}

Interaction Coverage
Cover all pairs of conditions in sort and basic blocks in eval

cover (@conditionedge & @func(sort)) ->
(@basicblockentry & @func(eval))
void sort(int *a, int len) {
  int i, t;
  for (i=1; i<len; ++i) {
    if (compare(a[i-1], a[i])) continue;
    t=a[i];
    a[i]=a[i-1];
    a[i-1]=t;
  }
  return;
}

Constrained Inputs
Basic block coverage in sort
Each test case shall use list of 2 to 15 elements

cover {len>=2} . {len<=15} . @basicblockentry & @func(sort)
Specs for Working Programmers

```c
1  int example(int a, int d)
2  {
3    if (a)
4      d = 0;
5    else
6      unimplemented();
7    return d*2;
8  }
```

Avoid Unfinished Code
Basic block coverage in example
Never call unimplemented

cover @basicblockentry & @func(example)
passing ^not(@call(unimplemented))*$
Conclusions and Current Work

Test specification language based on quoted regular expressions
Two test input generation engines

Current work
• Evaluation with partners from automotive and avionics industry
• Model-based testing / DO 178B
• Directed Software Verification

How did you specify your test suite?

http://code.forsyte.de/fshell