# IPOG: A General Strategy for T-Way Software Testing

Y. Lei (UT Arlington), R. Kacker (NIST), D. R. Kuhn (NIST), V. Okun (NIST), J. Lawrence (GMU)

ECBS, Tucson, Arizona

3/28/2007

# Software Engineering

- Software has become pervasive in modern society
  - Directly contributes to quality of life
  - Malfunctions cost billions of dollars every year, and could have severe consequences in a safety-critical environment
- Build better software in better ways, especially for large-scale development
  - Requirements, design, coding, testing, maintenance, configuration, documentation, deployment, and etc.

# Software Testing

- A dynamic approach to detecting software faults
  - Alternatively, static analysis can be performed, which is however often intractable
- Involves sampling the input space, running the test object, and observing the runtime behavior
  - Intuitive, easy-to-use, scalable, and can be very effective for fault detection
- Perhaps the most widely used approach to ensuring software quality in practice

# The Challenge

- Testing is labor intensive and can be very costly
  - often consumes more than 50% of the development cost
- Exhaustive testing is often impractical, and is not always necessary
- How to make a good trade-off between test effort and test coverage?

# Outline

- Introduction
  - T-way testing
  - State-of-the-art
- The IPOG Strategy
  - Algorithm IPOG-Test
  - Experimental results
- Related Work on T-Way Testing
- Conclusion and Future Work

# T-Way Testing

- Given any t input parameters of a test object, every combination of values of these parameters be covered by at least one test
- Motivation: Many faults can be exposed by interactions involving a few parameters
  - Each combination of parameter values represents one possible interaction between these parameters
- Advantages
  - Light specification, requires no access to source code, automated test input generation, excellent trade-off between test effort and test coverage

# Example

Three parameters, each with values 0 and 1

| P1 | P2 | P3 |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

pairwise

| P1 | P2 | P3 |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |
| 1 | 1 | 1 |

exhaustive

# State-of-the-Art

- Greedy construction
  - Involves explicit enumeration of all possible combinations
  - tries to cover as many combinations as possible at each step
- Algebraic Construction
  - Test sets are constructed using pre-defined rules
- Most approaches focus on 2-way (or pairwise) testing

# Beyond pairwise

- Many software faults are caused by interactions involving more than two parameters
  - A recent NIST study by R. Kuhn indicates that failures can be triggered by interactions up to 6 parameters
- Increased coverage leads to a higher level of confidence
  - Safety-critical applications have very strict requirements on test coverage

# Outline

- Introduction
  - T-way testing
  - State-of-the-art
- The IPOG Strategy
  - Algorithm IPOG-Test
  - Experimental results
- Related Work on T-Way Testing
- Conclusion and Future Work

# The Framework

- Construct a t-way test set for the first t parameter
- Extend the test set to cover each of the remaining parameters one by one
  - Horizontal growth - extends each existing test by adding one value for the new parameter
  - Vertical growth – adds new tests, if needed, to make the test set complete

# Algorithm IPOG-Test

**Algorithm** *IPOG-Test* (int $t$, ParameterSet $ps$)
{
1. initialize test set $ts$ to be an empty set
2. denote the parameters in $ps$, in an arbitrary order, as $P_1$, $P_2$, …, and $P_n$
3. add into test set $ts$ a test for each combination of values of the first $t$ parameters
4. **for** (int $i = t + 1$; $i \leq n$; $i$ ++){
5. let $\pi$ be the set of $t$-way combinations of values involving parameter $P_i$ and $t$ -$1$ parameters among the first $i - 1$ parameters
6. // horizontal extension for parameter $P_i$
7. **for** (each test $\tau = (v_1, v_2, \ldots, v_{i-1})$ in test set $ts$) {
8. choose a value $v_i$ of $P_i$ and replace $\tau$ with $\tau' = (v_1, v_2, \ldots, v_{i-1}, v_i)$ so that $\tau'$ covers the most number of combinations of values in $\pi$
9. remove from $\pi$ the combinations of values covered by $\tau'$
10. }
11. // vertical extension for parameter $P_i$
12. **for** (each combination $\sigma$ in set $\pi$){
13. **if** (there exists a test that already covers $\sigma$) {
14. remove $\sigma$ from $\pi$
15. } **else** {
16. change an existing test, if possible, or otherwise add a new test to cover $\sigma$ and remove it from $\pi$
17. }
18. }
19.}
20.**return** ts;
}

# Example

- Four parameters: P1, P2, P3, and P4
- P1, P2, and P3 have 2 values
- P4 has 3 values

| P1 | P2 | P3 |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |
| 1 | 1 | 1 |

(a)

Horizontal growth

| P1 | P2 | P3 | P4 |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 2 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 2 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 |

(b)

Vertical growth

| P1 | P2 | P3 | P4 |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 2 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 2 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 2 |
| 1 | 1 | 0 | 2 |
| * | 0 | 0 | 2 |
| * | 1 | 1 | 2 |

(c)

3-way test set

# Experimental Results (1)

Question 1: How does the size of a test set generated by IPOG-Test, as well as the time taken, grow in terms of t, # of parameters, and # of values?

| t-way | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|
| size | 48 | 308 | 1843 | 10119 | 50920 |
| time | 0.11 | 0.56 | 6.38 | 63.8 | 791.35 |

**Results for 10 5-value parameters for 2- and 6-way testing**

# Experimental Results (2)

| # of params | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Size | 784 | 1064 | 1290 | 1491 | 1677 | 1843 | 1990 | 2132 | 2254 | 2378 | 2497 |
| Time | 0.19 | 0.45 | 0.92 | 1.88 | 3.58 | 6.38 | 10.83 | 17.52 | 27.3 | 41.71 | 61.26 |

Results for 5 to 15 5-value parameters for 4-way testing

| # of values | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|
| Size | 46 | 229 | 649 | 1843 | 3808 | 7061 | 11993 | 19098 | 28985 |
| Time | 0.16 | 0.547 | 1.8 | 6.33 | 16.44 | 38.61 | 83.96 | 168.37 | 329.36 |

Results for 10 parameters with 2 to 10 values for 4-way testing

# Experimental Results (3)

Question 2: How does FireEye compare to other tools, both in terms of # of tests and time to produce them?

| t-way | FireEye | | ITCH | | Jenny | | TConfig | | TVG | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Size | Time | Size | Time | Size | Time | Size | Time | Size | Time |
| 2 | 100 | 0.8 | 120 | 0.73 | 108 | 0.001 | 108 | >1 hour | 101 | 2.75 |
| 3 | 400 | 0.36 | 2388 | 1020 | 413 | 0.71 | 472 | >12 hour | 9158 | 3.07 |
| 4 | 1361 | 3.05 | 1484 | 5400 | 1536 | 3.54 | 1476 | >21 hour | 64696 | 127 |
| 5 | 4219 | 18.41 | NA | >1 day | 4580 | 43.54 | NA | >1 day | 313056 | 1549 |
| 6 | 10919 | 65.03 | NA | >1 day | 11625 | 470 | NA | >1 day | 1070048 | 12600 |

Results of different tools for the TCAS application

TCAS: Seven 2-value parameters, two 3-value parameters, one 4-value parameter, two 10-value parameters

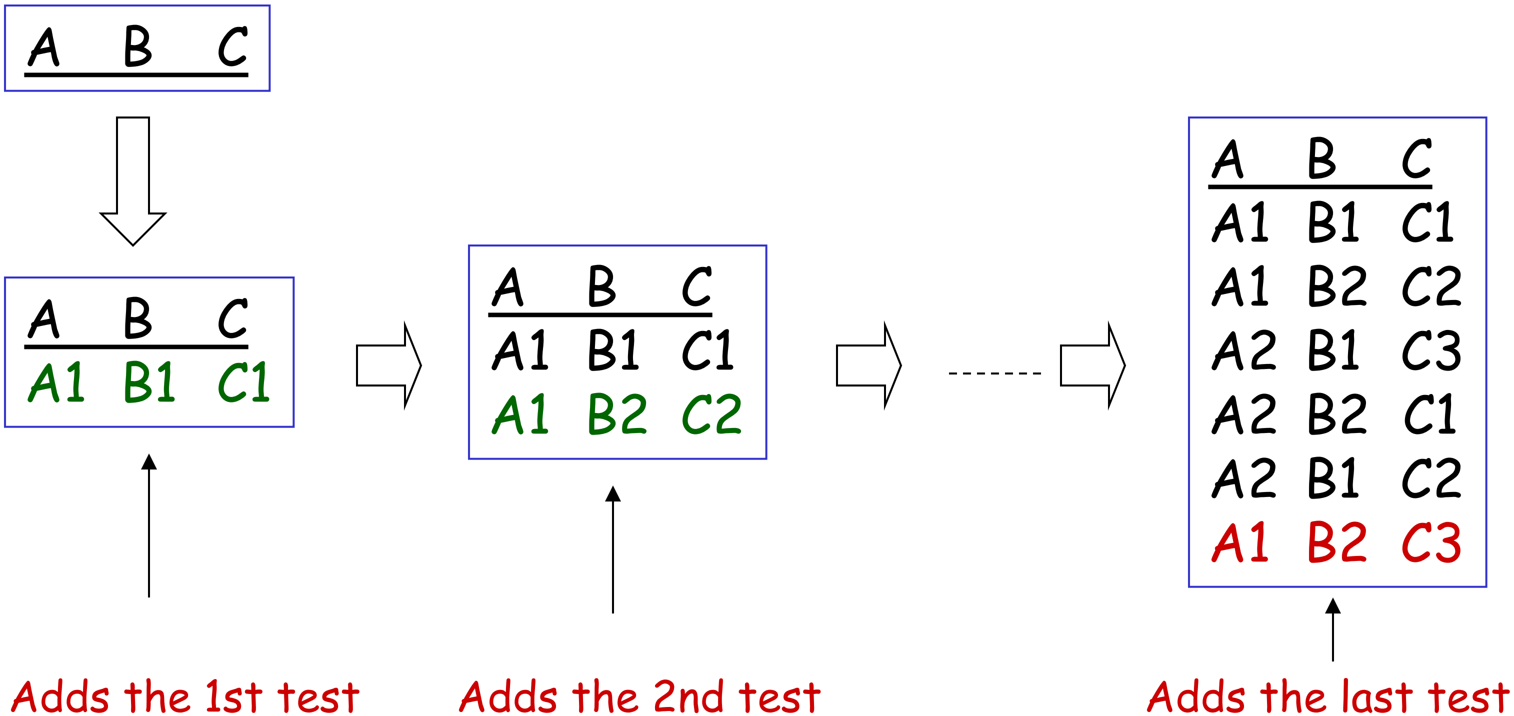
# Outline

- Introduction
  - T-way testing
  - State-of-the-art
- The IPOG Strategy
  - Algorithm IPOG-Test
  - Experimental Results
- Related Work on T-Way Testing
- Conclusion and Future Work

# AETG (1)

- Starts with an empty set and adds one (complete) test at a time
- Each test is locally optimized to cover the most number of missing pairs:
- Has a higher order of complexity, both in terms of time and space, than IPOG

# AETG (2)


A B C
A B C
A1 B1 C1
A B C
A1 B1 C1
A1 B2 C2
A B C
A1 B1 C1
A1 B2 C2
A2 B1 C3
A2 B2 C1
A2 B1 C2
A1 B2 C3
Adds the 1st test
Adds the 2nd test
Adds the last test

# Orthogonal Arrays (1)

- Given any t columns, every combination of the possible values is covered in the same number of times
  - Originally used for statistical design, which often requires a balanced coverage
  - Often computed using some pre-defined mathematical functions
- Each row can be considered as a test, and each column as a parameter
- Can be constructed extremely fast, and are optimal by definition, but do not always exist

# Orthogonal Arrays (2)

| (b0, b1) | A = b1 | B = b0 + b1 | C = b0 + 2 * b1 | D = b0 |
|---|---|---|---|---|
| (0, 0) | 0 | 0 | 0 | 0 |
| (0, 1) | 1 | 1 | 2 | 0 |
| (0, 2) | 2 | 2 | 1 | 0 |
| (1, 0) | 0 | 1 | 1 | 1 |
| (1, 1) | 1 | 2 | 0 | 1 |
| (1, 2) | 2 | 0 | 2 | 1 |
| (2, 0) | 0 | 2 | 2 | 2 |
| (2, 1) | 1 | 0 | 1 | 2 |
| (2, 2) | 2 | 1 | 0 | 2 |

# Outline

- Introduction
  - T-way testing
  - State-of-the-art
- The IPOG Strategy
  - Algorithm IPOG-Test
  - Experimental Results
- Related Work on T-Way Testing
- Conclusion and Future Work

UTA

# Conclusion

- T-way testing can substantially reduce the number of tests, while remaining effective for fault detection
- IPOG produces a t-way test set incrementally, covering one parameter at a step
- Comparing to existing tools, IPOG can produce smaller tests faster.

# Future Work

- Explicit enumeration can be very costly
  - How to reduce the number of combinations that have to enumerated?
- Support for parameter relations and constraints
  - No need to cover combinations of independent parameters
  - Invalid combinations must be excluded
- Integration of t-way testing with other tools to increase the degree of automation