Automatic Testing of Programs with Contracts

Alexey Kolesnichenko

Chair of Software Engineering

Nov. 25, 2015

Slides are courtesy of Yu Pei

Automatic Testing

- Many people worked on the project
- Contributors:
  - Andreas Leitner
  - Ilinca Ciupa
  - Yi Wei
  - Alexey Kolesnichenko
  - Bertrand Meyer
  - Carlo A. Furia
  - Chris Poskitt
  - Yu Pei
  - and many others

Design by contract

- Contracts

```
LINKED_LIST.index_of (v: G; i: INTEGER_32): INTEGER_32
		-- Index of `i'-th occurrence of item identical to `v'.
		-- 0 if none.
	require
		positive_occurrences: i > 0
	ensure
		non_negative_result: Result >= 0
```

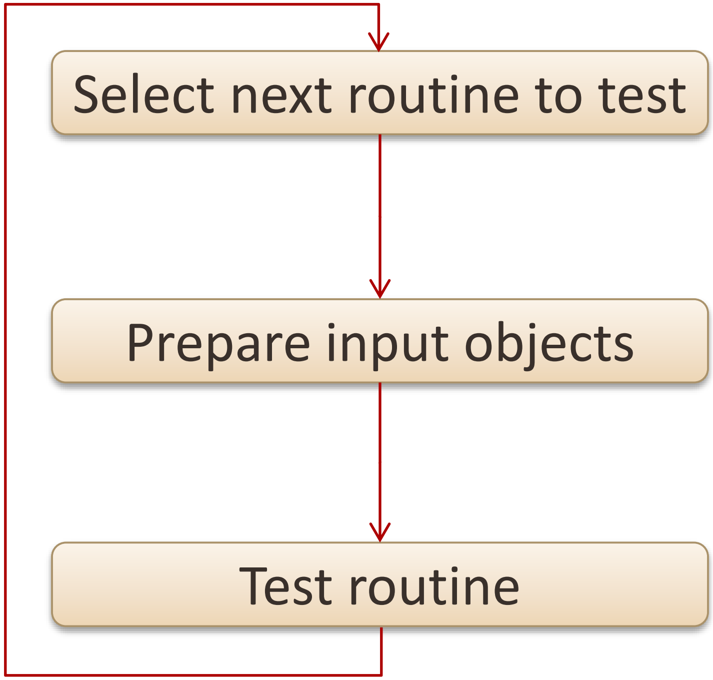
- Applications
  - Specification
  - Documentation
  - Testing & fixing

Automatic (random) testing

- Testing
  - Input
  - Oracle

- **AutoTest**: Automatic testing programs with contracts
  - Precondition of the routine under test as the valid input filter
  - Postcondition of the routine as the oracle

The select-prepare-test loop



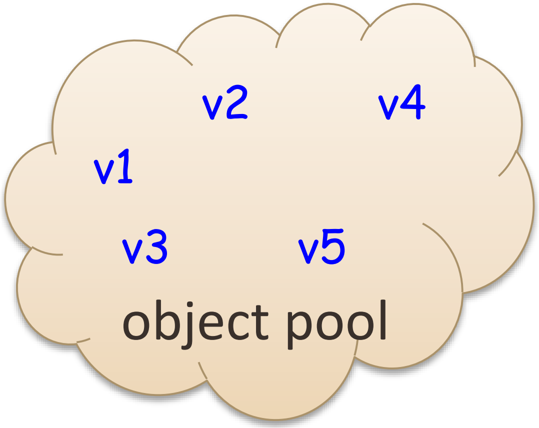
Select next routine to test

Prepare input objects

Test routine

Sample testing process

```
create {LINKED_LIST [INTEGER]} v1.make

v2 := 1

v1.extend (v2)

v1.wipe_out

v3 := 125

v4 := v1.has (v3)

v5 := v1.count
```



v2 v4

v1

v3 v5

object pool

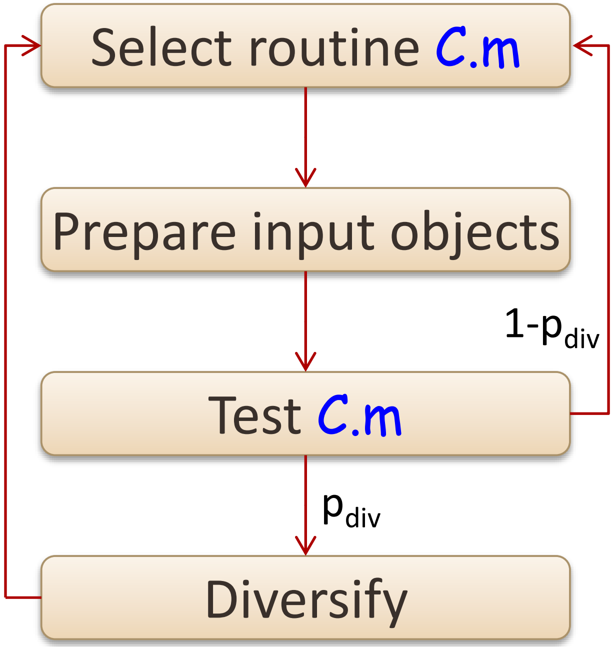
Performance evaluation

- Testing results
  - Precondition of the routine-under-test is violated
    - Invalid test case
  - Precondition of the routine-under-test is satisfied
    - Postcondition satisfied
      - Passing test case
    - Postcondition not established
      - Failing test case (detected fault)

- Evaluation criteria
  - Fault detection rate
  - Input space coverage

Random^{+} testing

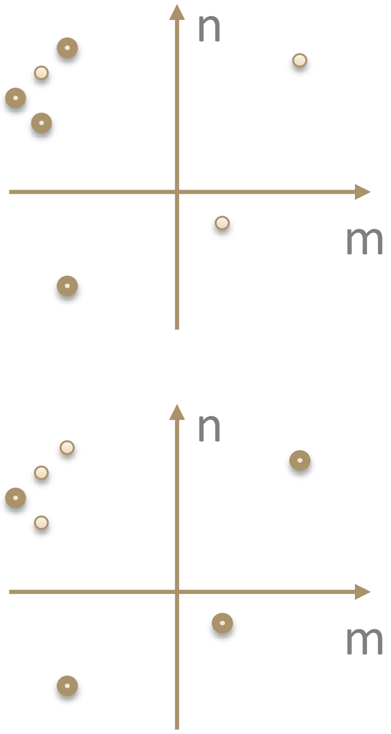
- Essentials
  - Input generation
    - Primitive types:
      random selection + boundary values
    - Reference types:
      constructor calls + random selection
  - Diversification
    - With probability p_{div} after each test



- Result
  - Find faults in widely used, industrial-grade code
  - High fault detection rate in the first a few minutes

Adaptive Random Testing

- Essentials
  - Maintain a list of objects *O* used in testing a routine `r`
  - Select the object with the highest average <u>distance</u> to *O* for the next test of `r`

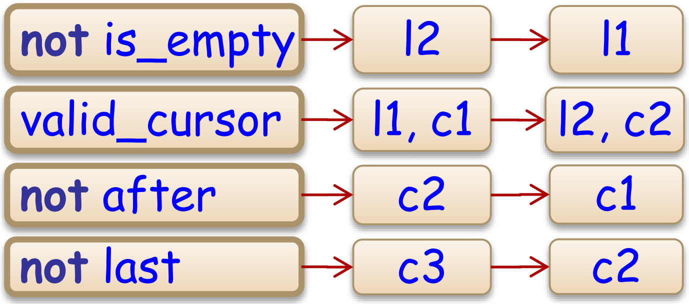


- Result
  - Takes less time and generated tests, on average by a factor of 5, to the first fault

Testing with guided object selection

```
LINKED_LIST . remove_right (cursor: CURSOR)
```

- Essentials
  - Keep track of precondition-satisfying objects
  - Use them with higher probability



not is_empty → l2 → l1

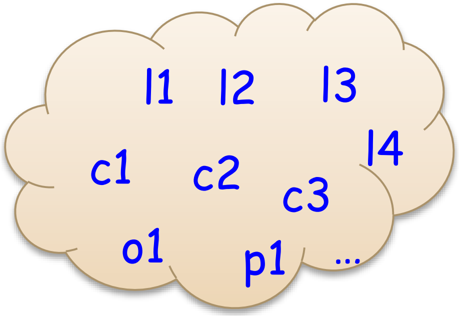
valid_cursor → l1, c1 → l2, c2

not after → c2 → c1

not last → c3 → c2

v-pool

- Results
  - 56% of the routines that cannot be tested before are now tested
  - 10% more faults detected in the same time
  - Routines tested 3.6 times more often



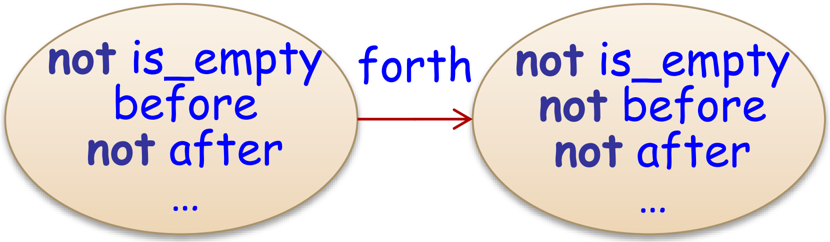
object pool

Stateful testing

- Essentials
  - Object states in Boolean expressions

```
LINKED_LIST . index_of (v: like item; i: INTEGER_32): INTEGER_32
```

  - before, after, is_empty, i > 0, …
  - Infer preconditions from existing tests
    - Boolean expressions that always hold as preconditions
  - Prepare inputs violating the inferred preconditions
    - Select objects in the object pool
    - Transit objects using object behavioral model
- Result
  - 68% more faults detected with 7% time overhead



not is_empty
before
not after
…

forth

not is_empty
not before
not after
…

What strategy is the best one?

- What do you think?

Depends on the definition of “best”!

Typically fault detection rate is the most important factor

… And Random+ beats everything else!

Summary

- Contracts promote automatic testing
  - AutoTest
    - Preconditions as input filters and postconditions as oracles
  - Project web page:

    http://se.inf.ethz.ch/research/autotest/

THANKS