Sabicu

Automatic Identification of Common and Special Object-Oriented Unit Tests

Tao Xie

Advisor: David Notkin

Dept. of Computer Science & Engineering
University of Washington, Seattle

Oct. 2004

Motivation

- Human loves writing unit tests!
  - Human capability is limited

- Machine comes to rescue!
  - Automated tools generate many test inputs

Commercial: [Parasoft Jtest, Agitar Agitator, …]

Academic: [JCrasher@Gatech, Eclat@MIT, Rostra@UW, Symstra@UW, …]

Problem

- Automated tools generate many test inputs
  - Infeasible to inspect all (6777 tests for LinkedList)
  - Select test inputs that throw exceptions or achieve new structural coverage
    [Parasoft Jtest, Agitar Agitator, JCrasher, …]

- Any "gold" left in the generated tests?
  - Need new gold mining devices

Sabicu: Automatic Identification of Common and Special Tests

- Intuition:
  common tests exercise common cases
  special tests exercise special cases
- Key:
  ways to characterize common and special cases
- Device:
  observe runtime behavior and infer
  universal properties: true all the time
  common properties: true most of the time

Statistical properties

- Gold:
  - common tests: satisfy universal/common properties
  - special tests: violate common properties

Examples of Inferred Statistical Properties

- Syntactically identical to algebraic specs
- Universal property:

```
size(clear(S).state).retval == 0
```

Satisfying count: 121 Violating count: 0

- Common property:

```
remove(removeLast(S).state, m0_2).state
==removeLast(remove(S, m0_2).state).state
```

Satisfying count: 318 Violating count: 42

Inferring Statistical Properties

During development of Sabicu

- Looked into human-written algebraic specs
- Predefined 25 abstraction templates [UW-CSE -04-08-03]

```
size(clear(S).state).retval == 0
```

g(f(S, args1).state, args2).retval == const

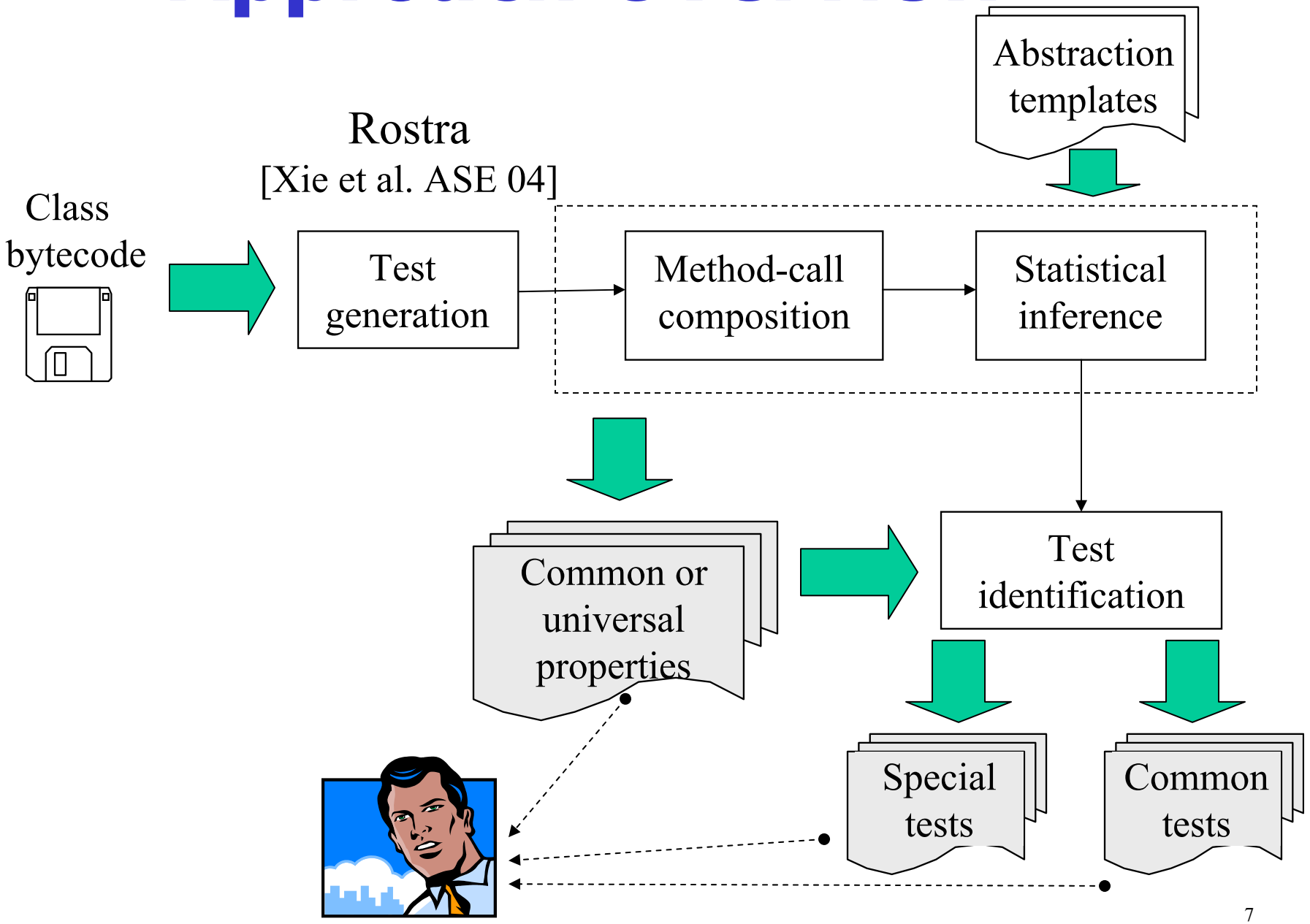
```
remove(removeLast(S).state, m0_2).state
==removeLast(remove(S, m0_2).state).state
```

g(f(S, args1).state, args2).state
== f(g(S, args1).state, args2).state

At runtime

- Instantiate templates with instances (tests)
- Obtain statistical properties

Approach Overview



Subjects and Quantitative Results

http://www.cs.washington.edu/homes/taoxie/sabicu/

Table 1. Quantitative results for identifying special and common tests

subject	meth	axiom space	iter	axioms consd	time (sec)	properties univ	properties c-univ	properties common	generated	tests special	tests common	tests both
BinSearchTree	4	240	3	75	0.93	6	7	7	91	6	14	3
			4	75	1.32	6	7	6	136	5	14	3
			5	75	1.32	6	7	6	136	5	14	3
BinomialHeap	12	2364	3	501	44.36	20	3	52	5272	42	57	1
			4	501	119.78	17	4	51	12440	44	56	1
			5	502	371.23	16	4	53	19888	43	54	1
FibonacciHeap	9	1242	3	287	1.97	21	5	45	173	32	53	4
			4	287	3.59	18	5	53	341	37	55	4
			5	287	7.02	17	4	56	677	39	55	4
HashMap	13	2022	3	381	16.25	73	8	19	2213	15	79	5
			4	381	65.11	73	8	21	7533	17	86	9
			5	381	157.59	73	8	22	15345	18	86	10
HashSet	8	792	3	211	1.85	39	11	17	157	15	45	7
			4	211	2.65	39	11	16	235	14	46	9
			5	211	3.12	39	11	18	261	15	47	10
LinkedList	21	6048	3	796	6.80	44	14	21	729	20	68	3
			4	797	21.88	43	14	33	2241	30	80	9
			5	797	76.71	43	14	31	6777	29	79	8
SortedList	24	7827	3	877	10.19	45	10	24	820	21	70	4
			4	878	33.14	44	10	23	2521	20	70	3
			5	878	110.78	44	9	31	7624	23	74	3
TreeMap	15	1968	3	409	20.31	74	8	20	2911	16	81	6
			4	409	79.94	74	8	21	9421	17	87	9
			5	409	314.46	74	8	20	15991	16	87	9
IntStack	4	252	3	33	0.57	2	0	2	76	2	3	1
			4	33	1.21	2	0	5	241	4	5	2
			5	33	2.84	2	0	5	766	4	5	2
UBStack	10	1077	3	87	0.78	11	1	6	183	6	17	1
			4	87	1.01	11	1	6	274	6	17	3
			5	87	1.23	11	1	5	365	5	16	1

More to be Done

More applications:

- Focused testing on universal properties
- Applied in software evolution
- Testing different implementations of the same interface

More evaluations

- Measure the fault detection capability and structural coverage of identified tests
- Case studies on programmers

Conclusion

- **Statistical properties are useful too**
  - Daikon: axiomatic spec inference [Ernst et al. TOSE 01]
  - Algebraic spec inference [Henkel&Diwan ECOOP 03]

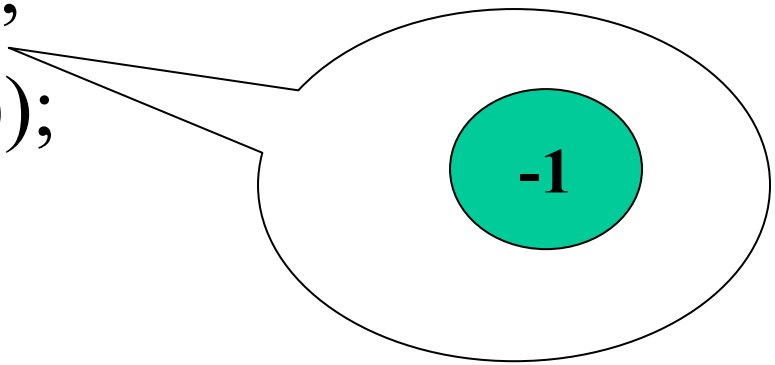
- **Gold mining is just starting**:
  making the most out of generated tests!
  - Test selection based on operational violations [Xie&Notkin ASE 03]
  - Relating to industry: Agitar Agitator, Parasoft Jtest [Xie et al. ASE 04]

Examples of Common and Special Tests

```
removeLast(addFirst(S, m0_1).state).state
          == addFirst(removeLast(S).state, m0_1).state
```

Satisfying count: 117 **(common test)**

```
LinkedList m = new LinkedList( );
m.add(0, new Integer(-1));
m.addFirst(new Integer(0));
m.removeLast ( );
```

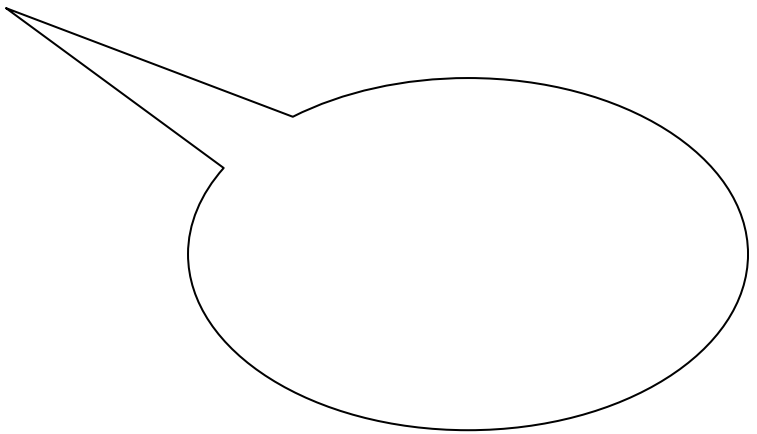


Examples of Common and Special Tests

```
removeLast(addFirst(S, m0_1).state).state
          == addFirst(removeLast(S).state, m0_1).state
```

Satisfying count: 3 **(special test)**

```
LinkedList m = new LinkedList( );
m.addFirst(new Integer(0));
m.removeLast ( );
```



Examples of Common and Special Tests

```
remove(removeLast(S).state, m0_2).state
 ==removeLast(remove(S, m0_2).state).state
```

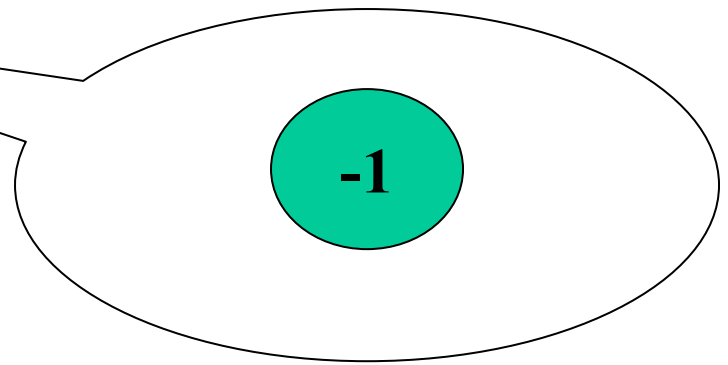
- Satisfying count: 318 **(common test)**

  LinkedList m = new LinkedList( );

  m.add (0, new Integer(-1));

  m.removeLast( );

  m.remove(new Integer(-1));

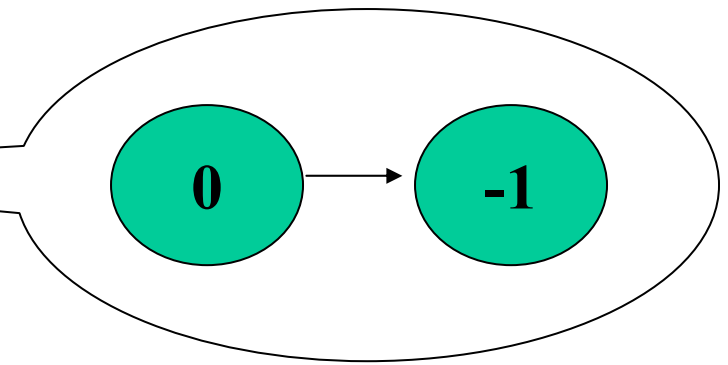


Examples of Common and Special Tests

```
remove(removeLast(S).state, m0_2).state
 ==removeLast(remove(S, m0_2).state).state
```

- Violating count: 42 **(special test)**

```
LinkedList  m = new LinkedList( );
m.add (0, new Integer(-1));
m.add (0, new Integer(0));
m.removeLast( );
m.remove(new Integer(-1));
```



Examples of Common and Special Tests

```
lastIndexOf(addFirst(S, m0_1).state, m0_2).retval
      == (lastIndexOf(S, m0_2).retval + 1)
                                    [where (m0_1==m0_2)]
```

- Satisfying count: 120 **(special test)**

  LinkedList m = new LinkedList( );

  m.addFirst(new Integer(-1));

  m.lastIndexOf(new Integer(-1));