Automatically Identifying Special and Common Unit Tests for Object-Oriented Programs

Tao Xie

North Carolina State University
Raleigh, NC

David Notkin

University of Washington
Seattle, WA

Automated Testing in the Absence of Specs

- Specifications help improve automated testing but they often don't exist in practice
  - JML+JUnit [CL ECOOP 02], Korat [BKM ISSTA 02], …

- Without specs, test oracles are not generated for correctness checking
  - infeasible to manually inspect
  - Insufficient to rely only on uncaught exceptions

- Solution: **infer** specs from test executions and **select** tests against inferred specs
  - select tests that violate inferred specs [ASE 03]
  - **identify special and common tests**

Automated Testing in the Absence of Specs

- Specifications help improve automated testing but they often don't exist in practice
  - JML+JUnit [CL ECOOP 02], Korat [BKM ISSTA 02], …
- Without specs, test oracles are not generated for correctness checking
  - infeasible to manually inspect
  - Insufficient to rely only on uncaught exceptions
- Solution: **infer** specs from test executions and **select** tests against inferred specs

Benefits of spec-based testing can be obtained without the pain of writing the specifications!

Synopsis

- Common and special tests
  - common tests **à** common behavior
    e.g., non-full and non-empty bounded stack
  - special tests **à** special behavior
    e.g., full or empty bounded stack
- Characterize common/special behavior with inferred statistical algebraic abstractions

Synopsis

- Common and special tests
  - common tests **à** common behavior
    e.g., non-full and non-empty bounded stack
  - special tests **à** special behavior
    e.g., full or empty bounded stack
- Characterize common/special behavior with inferred statistical algebraic abstractions
  - algebraic abstractions: in the form of axioms e.g.,
    `top(push(S, e).state).retval == e`

receiver object state of **`push`**

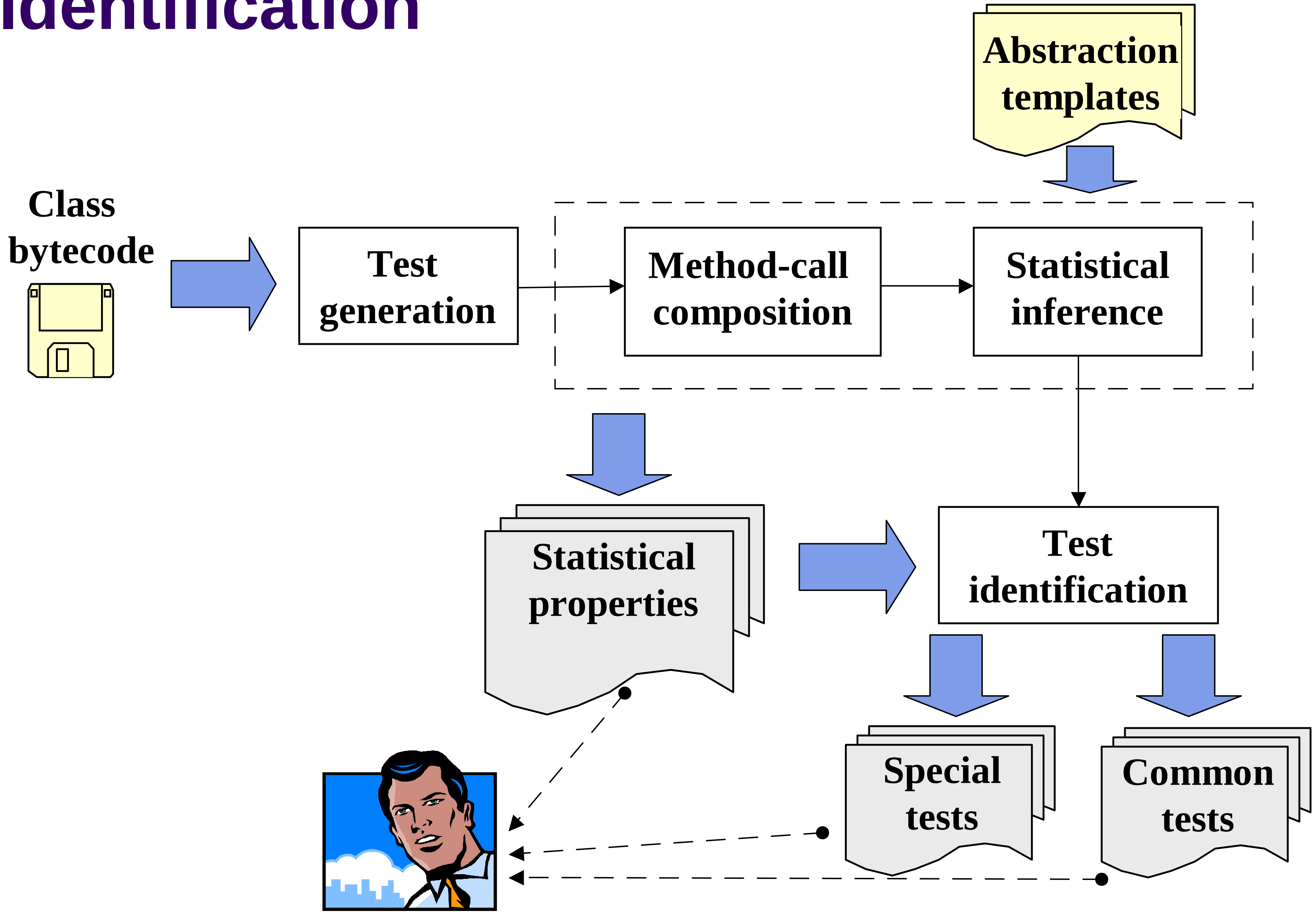
receiver object state of **`top`** (after **`push`**)

return value of **`top`**

Synopsis

- Common and special tests
  - common tests **à** common behavior
    e.g., non-full and non-empty bounded stack
  - special tests **à** special behavior
    e.g., full or empty bounded stack
- Characterize common/special behavior with inferred statistical algebraic abstractions
  - algebraic abstractions: in the form of axioms e.g.,
    `top(push(S, e).state).retval == e`
  - statistical abstractions: e.g., 6 violating tests and 47 satisfying tests,
    - ≠ universal abstractions [HD ECOOP 03][ECGN TSE 01]

Special and Common Test Identification



Sample Abstraction Templates

- f(S, args1).state != S
  - `removeFirst(S).state != S`

`LinkedList example`

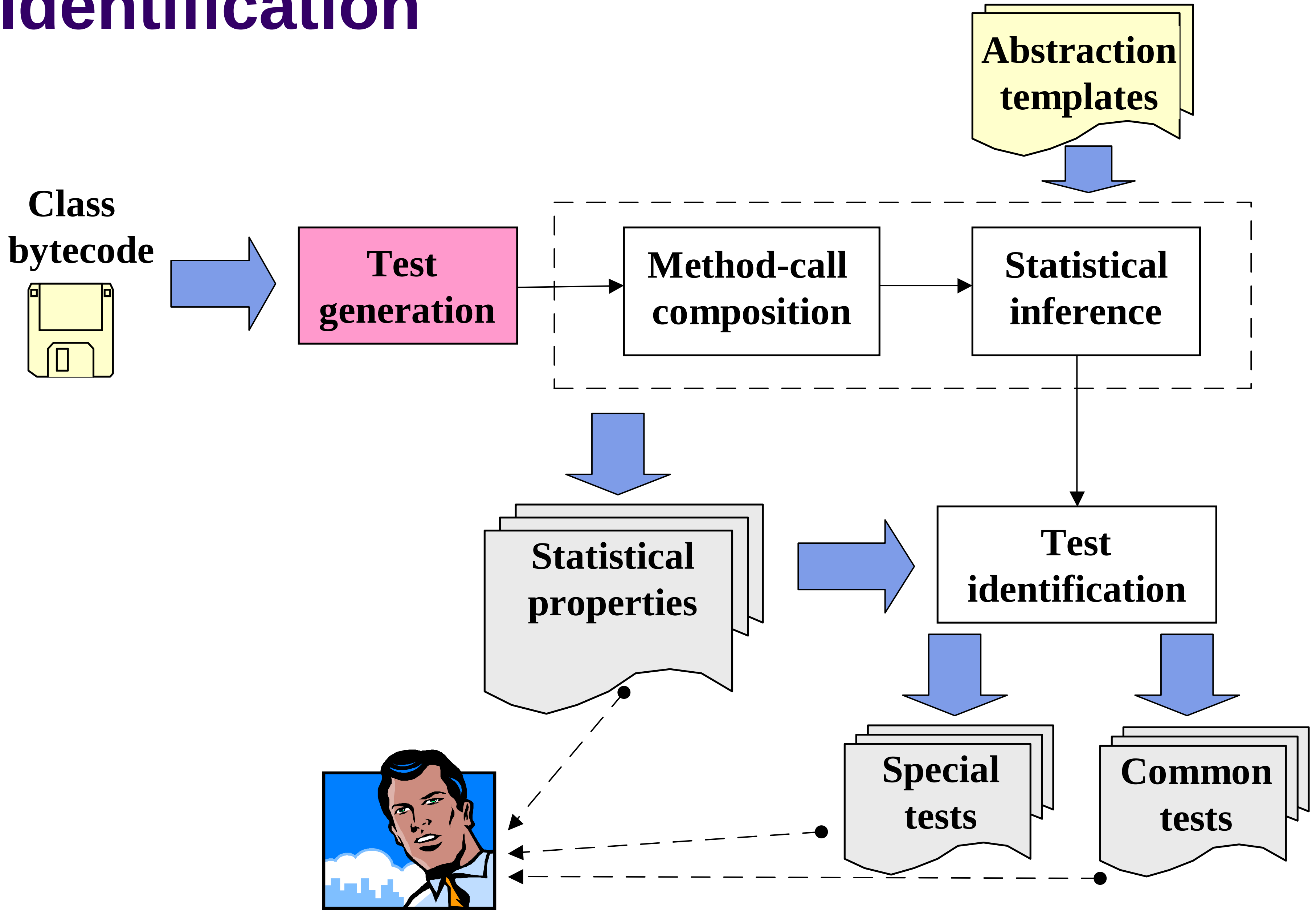
- f(S, args1).retval == const
  - `add(S, e).retval == true`
- g(f(S, args1).state, args2).retval == args1.i
  - `indexOf(add(S, i, e1).state, e2).retval == i`

Statistics of Abstraction Templates

- 13 templates for method-exit states
  - e.g., f(S, args1).state != S
- 11 templates for method returns
  - e.g., f(S, args1).retval == const
- Conditional extension to 20 templates
  - e.g., `contains(add(S, e1).state, e2).retval == true where (e1 == e2)`
- Difference extension to 11 templates
  - e.g., `size(add(S, e).state).retval == (size(S).retval + 1)`

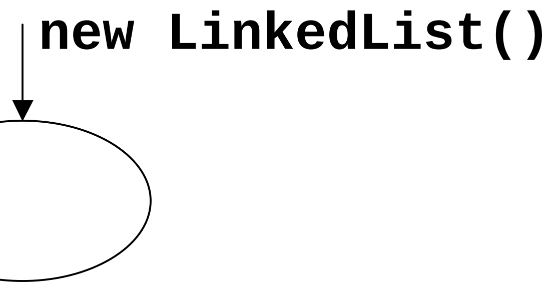
- Our templates instantiate all 146 but 2 axioms inferred by Henkel&Diwan [ECOOP 03] for **`ArrayList`**

Special and Common Test Identification



Test Generation

- Generate method arguments with JCrasher [CS SPE 04]
- Breadth-first exploration of receiver-object states with method calls with Rostra [ASE 04]

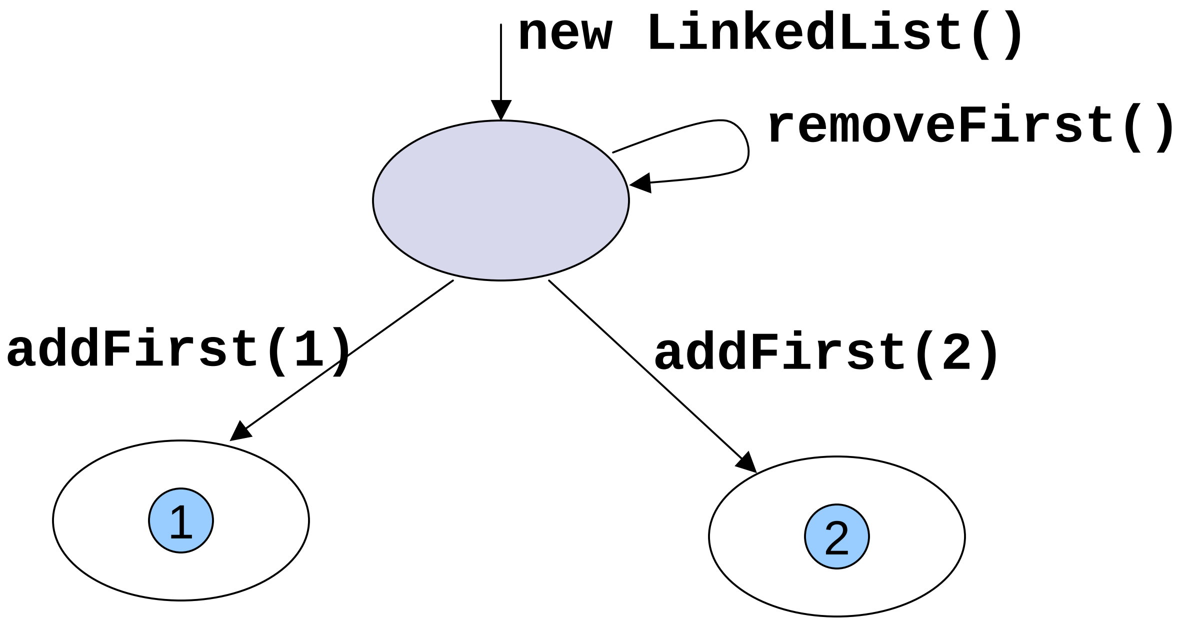


```
removeFirst()
addFirst(1)
addFirst(2)
```

Test Generation

- Generate method arguments with JCrasher [CS SPE 04]
- Breadth-first exploration of receiver-object states with method calls with Rostra [ASE 04]

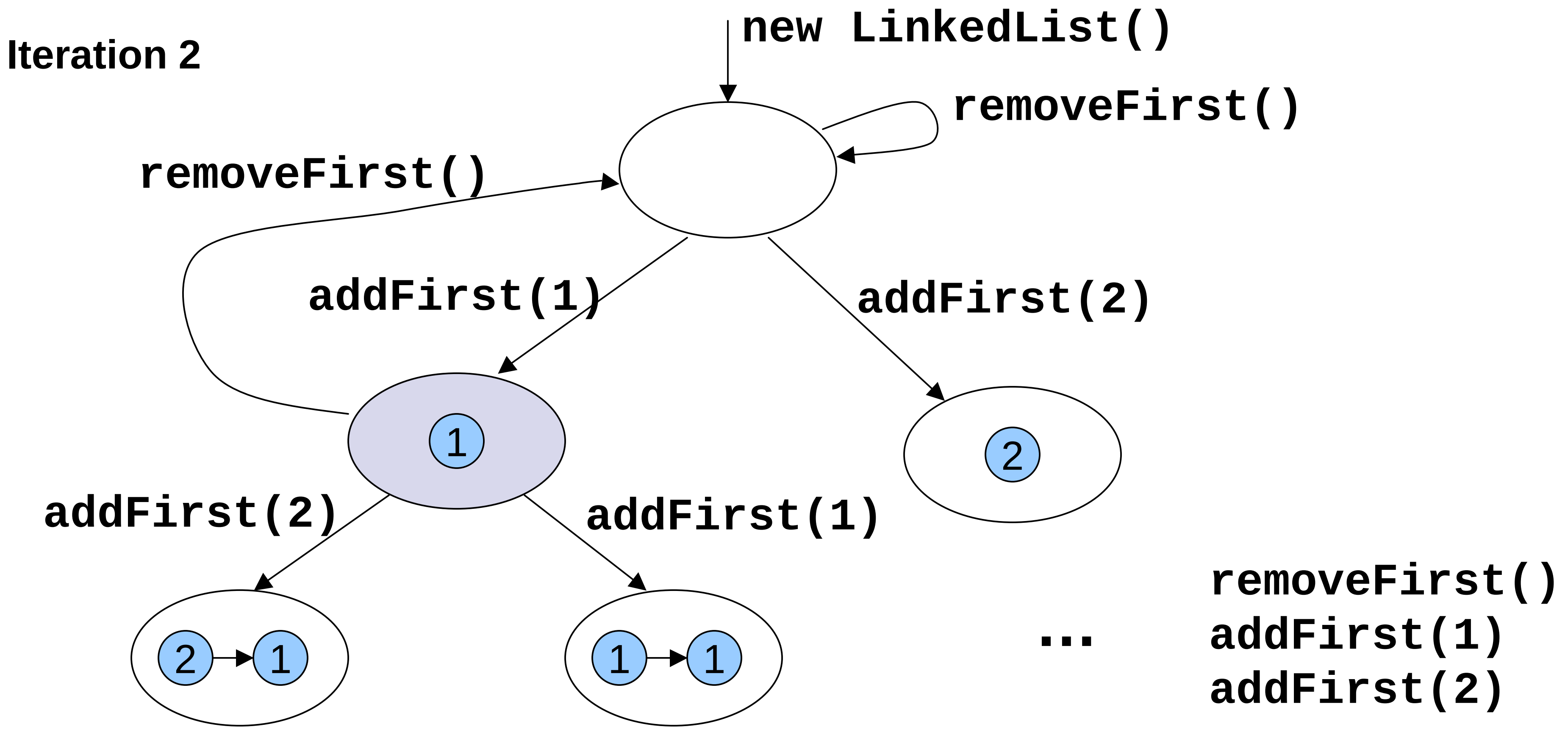
Iteration 1



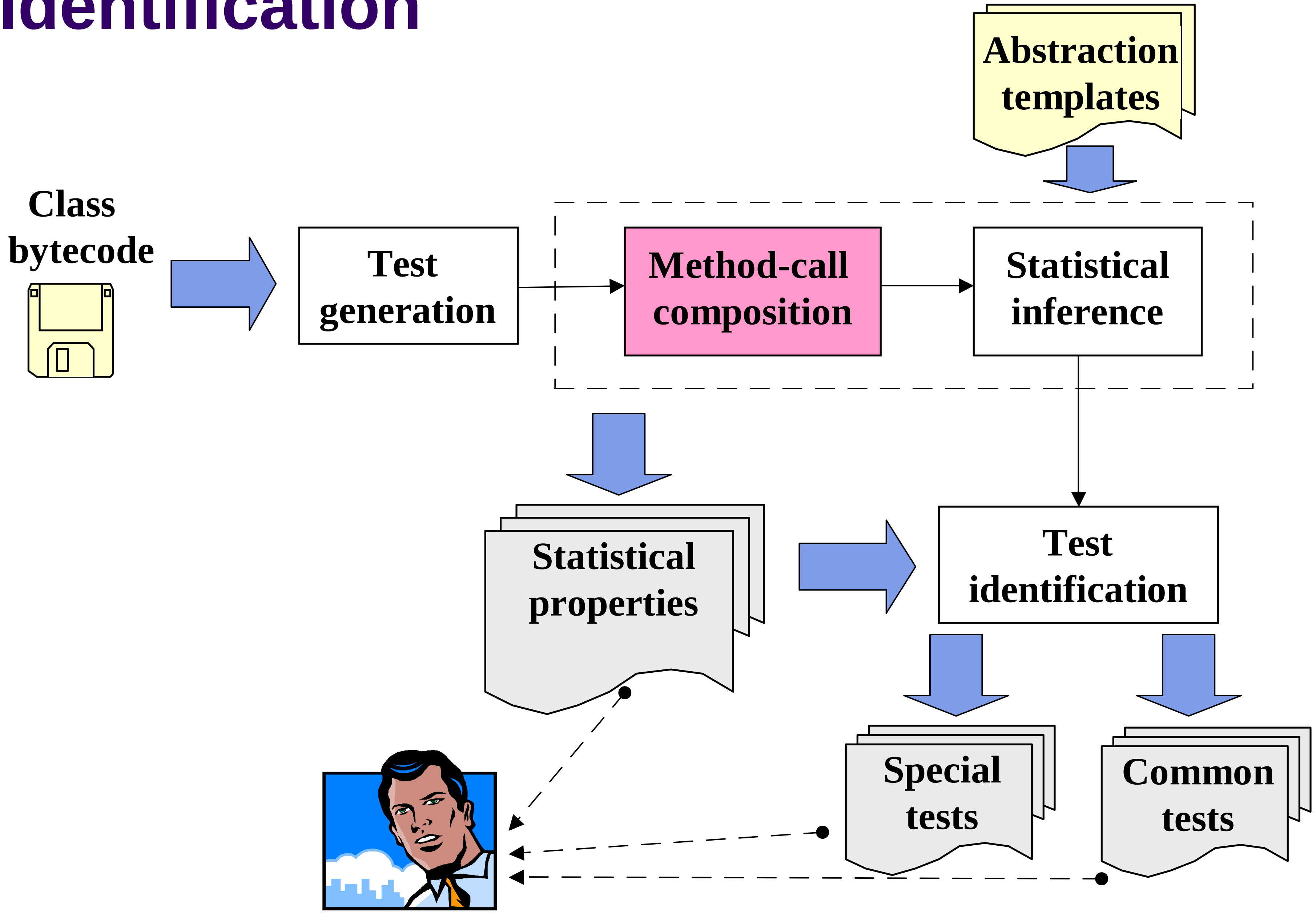
```
removeFirst()
addFirst(1)
addFirst(2)
```

Test Generation

- Generate method arguments with JCrasher [CS SPE 04]
- Breadth-first exploration of receiver-object states with method calls with Rostra [ASE 04]



Special and Common Test Identification



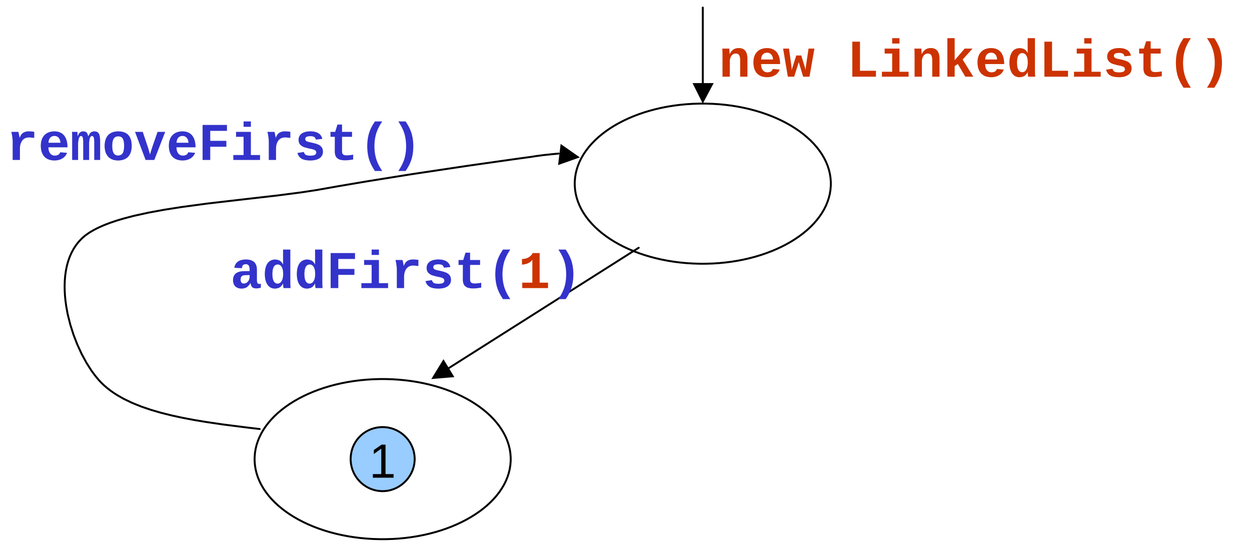
Method-Call Composition

- Goal: compose method-call pair to instantiate LHS or RHS of an abstraction template
  - template LHS:

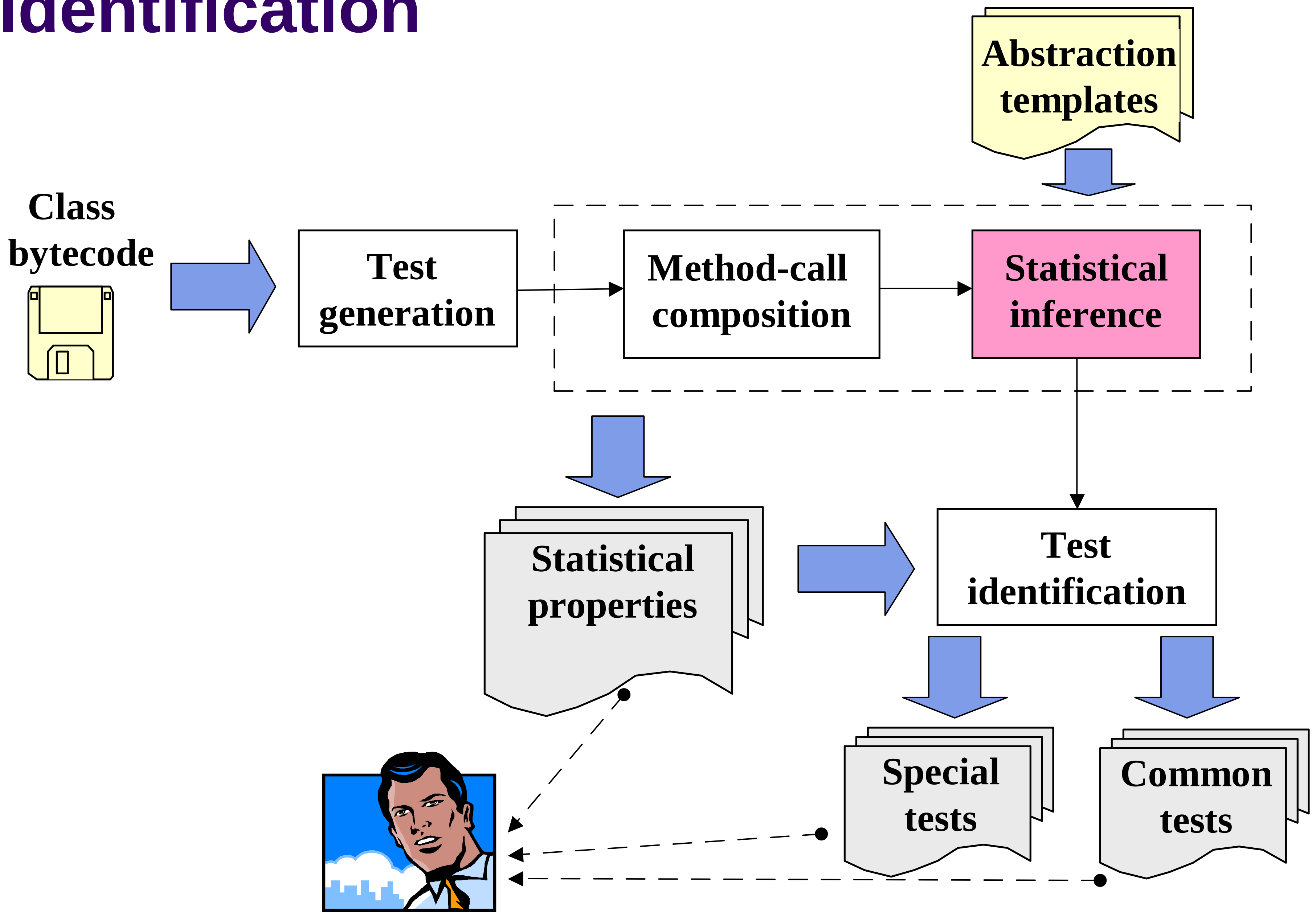
    g(f(S, args1).state, args2).state
  - abstraction LHS:

    `removeFirst(addFirst(S, e).state).state`
  - abstraction instance LHS:

`removeFirst(addFirst(new LinkedList(), 1).state).state`



Special and Common Test Identification



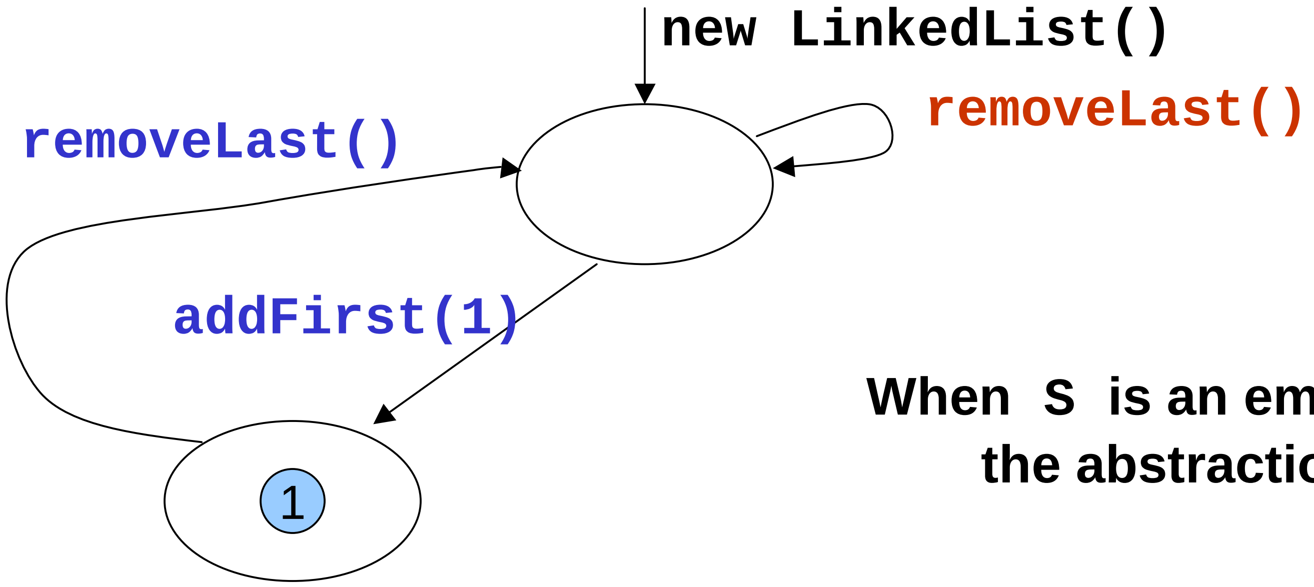
Statistical Inference

- Each statistical abstraction is associated with #satisfying instances and #violating instances
  - template: g(f(S, args1).state, args2).state == f(g(S, args2).state, args1).state
  - abstraction:

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

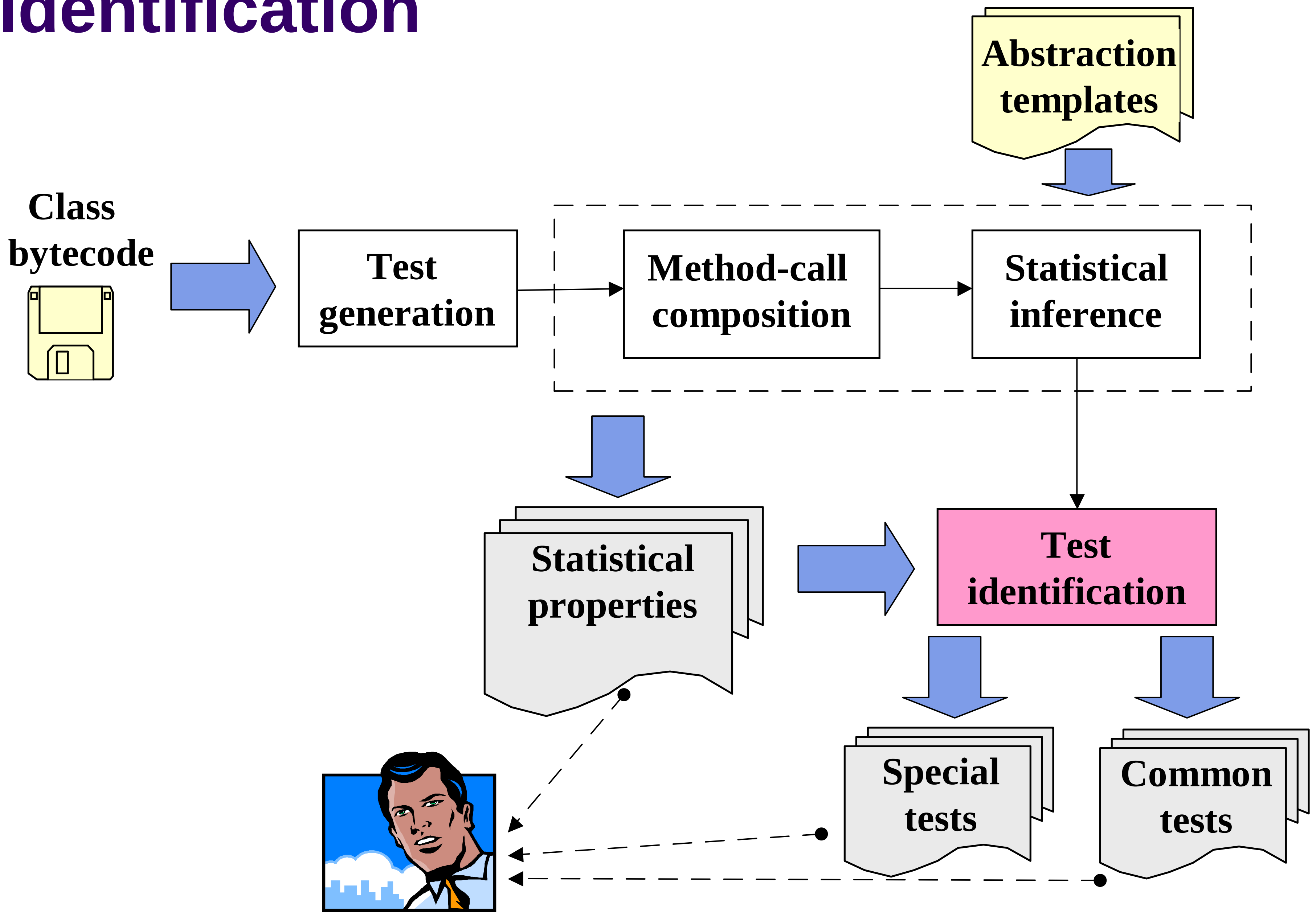
117 satisfying instances

3 violating instances



When `S` is an empty `LinkedList`, the abstraction is violated.

Special and Common Test Identification



Test Identification

- Universal property
  - no violating instances
- Common property
  - a minority of violating instances (<20% by default)

- Special test
  - a violating instance of a common property
  - a satisfying instance of a conditional universal property

  unique bounded stack

  ```
  push(push(S, e1).state, e2).state = push(S, e2).state
                                              where (e1 ==e2)
  ```
- Common test
  - a satisfying instance of a common property or universal property

Experience

- Developed the Sabicu tool for the approach
- Applied it on 10 ADT (data structures) with test generation of 5 iterations
- Inferred 3 axioms for int stack (inferred by Henkel&Diwan [ECOOP 03])
- Inferred 10 of 12 manually written axioms for unique bounded stack [SLA XP 02]
  - all 8 universal axioms
  - 2 of 4 conditional axioms
  - one inferred conditional axiom is missing from manually written ones.

Some Statistics

class	m	properties			tests		
		univ	cond-univ	comm	gen	special	comm
BinarySearchTree	4	6	10	6	136	5	14
BinomialHeap	12	21	5	51	21456	42	59
FibonacciHeap	9	12	6	80	677	52	62
HashMap	13	81	9	19	15345	15	92
HashSet	8	43	15	16	261	14	50
LinkedList	21	55	18	39	6777	37	96
SortedList	24	55	14	44	7624	33	95
TreeMap	15	84	9	17	16291	13	95
IntStack	4	5	0	5	606	4	6
UBStack	9	10	2	6	337	6	16

Related Work

- Daikon by Ernst et al [TSE 01]
  - infer axiomatic specs (universal properties)
- Tool by Henkel&Diwan [ECOOP 03]
  - infer axioms (universal properties)
- Strauss by Ammons et al. [POPL 02]
  - infer probabilistic FSMs from call sequences
- Static analysis tool by Engler et al. [SOSP 01]
  - infer common call sequence patterns and deviations from them.
- Test selection based on specs, structural info…

Conclusion

- Specs help improve automated testing but they often don’t exist in practice
- Automatically generated test inputs don’t have test oracles
- Our new approach infers statistical properties and uses them to identify special and common tests

- In future work, we plan to investigate
  - fault detection capability of selected tests
  - static/dynamic verification tools to refute inferred properties

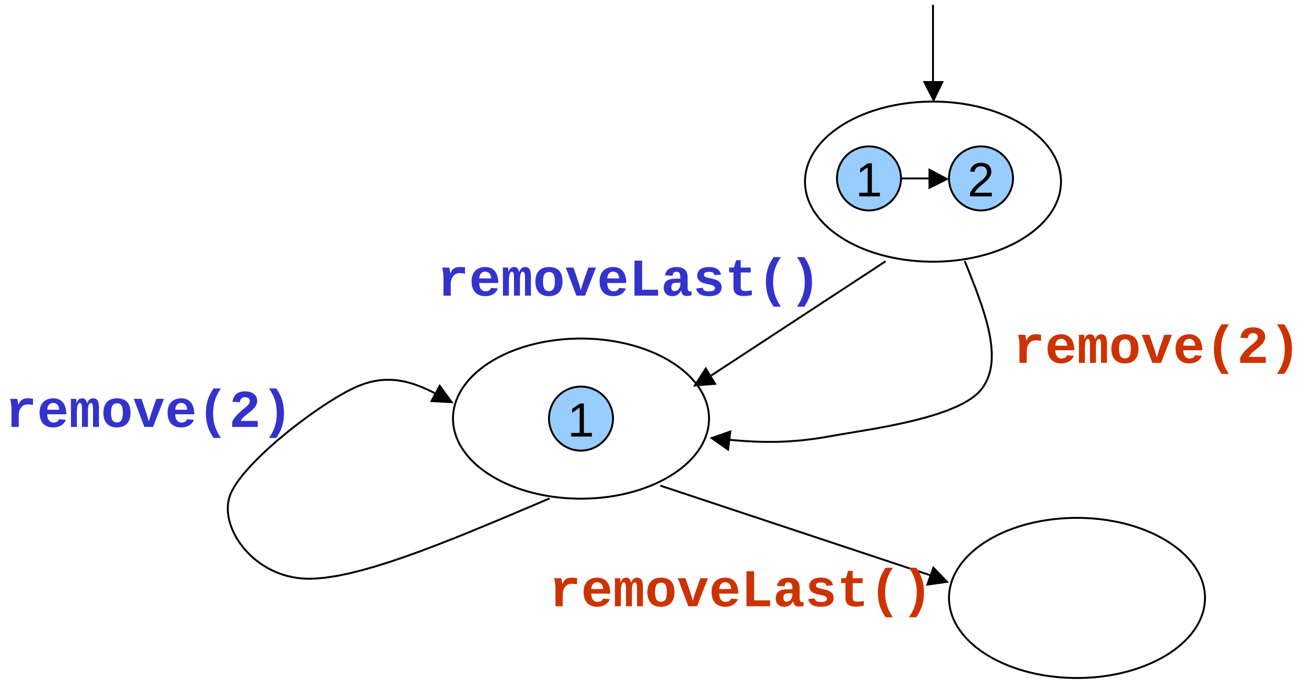
Questions?

One Common Property

```
remove(removeLast(S).state, m1).state
      = removeLast(remove(S, m1).state).state
```

408 satisfying instances

42 violating instances


1
2
removeLast()
remove(2)
remove(2)
1
removeLast()