M4 - The Oracle Problem

Guillermo Polito

ECI’23 - Universidad de Buenos Aires

Goals

- The oracle problem
- Automatically separate correct from incorrect results
- Different kind of oracles

- Start discussing oracles for compilers

Remember Assertions

```
SetTest >> testAdd

   | aSet |
   "Context"
   aSet := Set new.

   “Stimuli"
   aSet add: 5.
   aSet add: 5.

   "Check"
   self assert: aSet size equals: 1.
```

in *this context*
  when *this happens*
    then *this should happen*

Remember Fuzzing Date Parser

```
f := PzRandomFuzzer new.
r := PzBlockRunner on: [ :e | e asDate ].
f run: r times: 20.
```

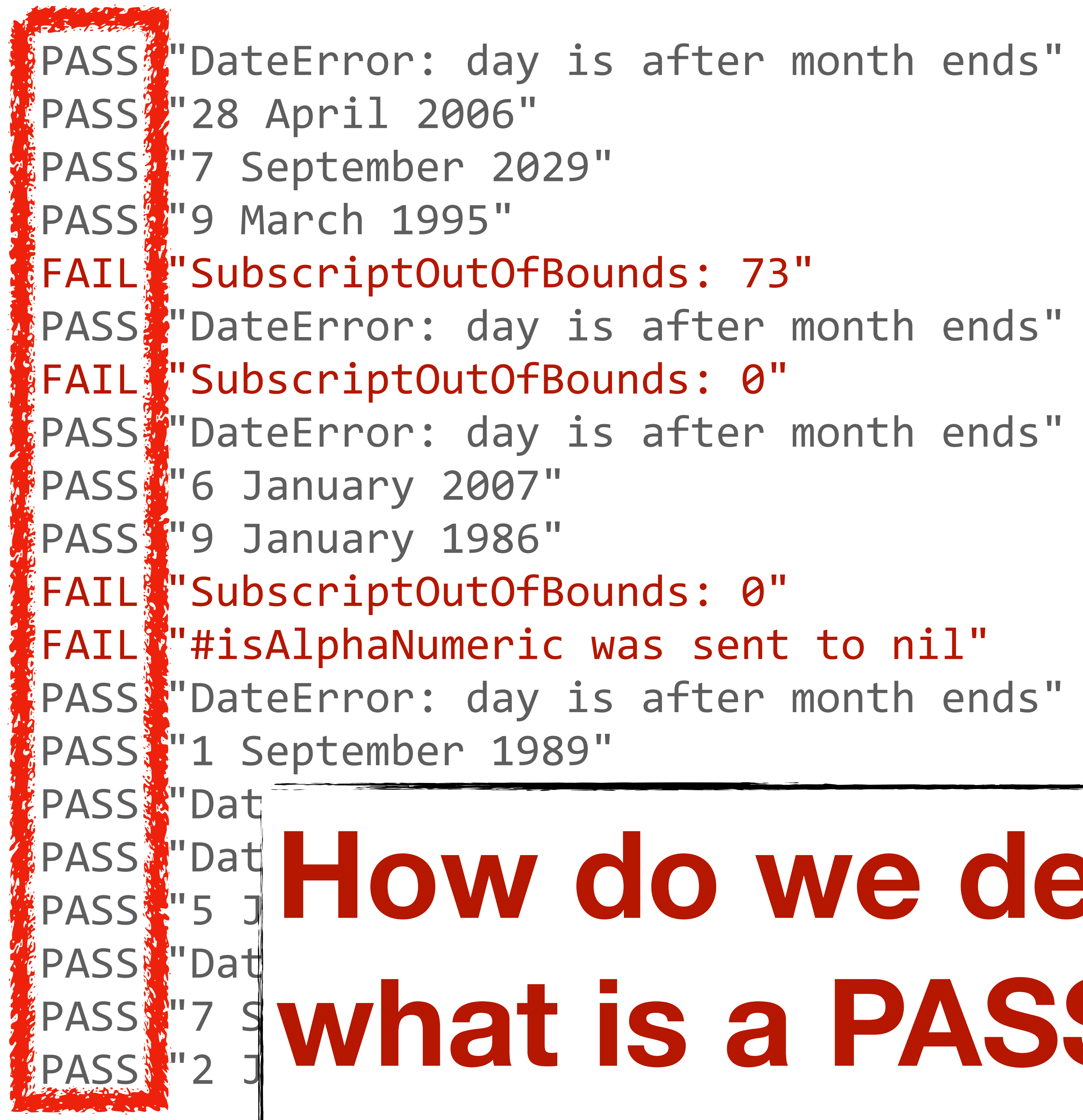
- Pharo 11
- String>>asDate

```
PASS "DateError: day is after month ends"
PASS "28 April 2006"
PASS "7 September 2029"
PASS "9 March 1995"
FAIL "SubscriptOutOfBounds: 73"
PASS "DateError: day is after month ends"
FAIL "SubscriptOutOfBounds: 0"
PASS "DateError: day is after month ends"
PASS "6 January 2007"
PASS "9 January 1986"
FAIL "SubscriptOutOfBounds: 0"
FAIL "#isAlphaNumeric was sent to nil"
PASS "DateError: day is after month ends"
PASS "1 September 1989"
PASS "DateError: day is after month ends"
PASS "DateError: day may not be zero or negative"
PASS "5 January 0228"
PASS "DateError: day may not be zero or negative"
PASS "7 September 1996"
PASS "2 January 2008"
```

Remember Fuzzing Date Parser

```
f := PzRandomFuzzer new.
r := PzBlockRunner on: [ :e | e asDate ].
f run: r times: 20.
```

- Pharo 11
- String>>asDate



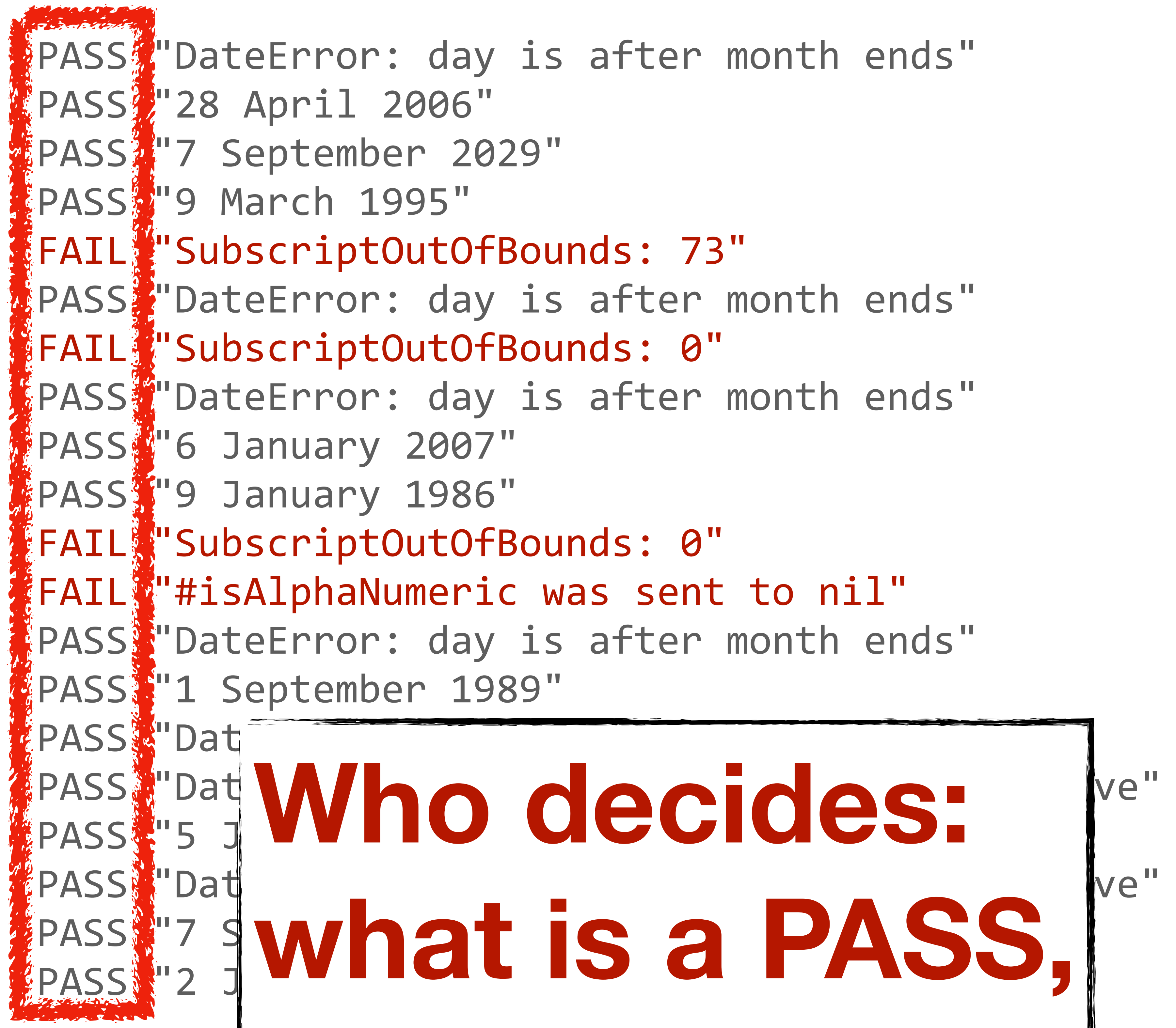
```
PASS "DateError: day is after month ends"
PASS "28 April 2006"
PASS "7 September 2029"
PASS "9 March 1995"
FAIL "SubscriptOutOfBounds: 73"
PASS "DateError: day is after month ends"
FAIL "SubscriptOutOfBounds: 0"
PASS "DateError: day is after month ends"
PASS "6 January 2007"
PASS "9 January 1986"
FAIL "SubscriptOutOfBounds: 0"
FAIL "#isAlphaNumeric was sent to nil"
PASS "DateError: day is after month ends"
PASS "1 September 1989"
PASS "Dat
PASS "Dat
PASS "5 J
PASS "Dat
PASS "7 S
PASS "2 J
```

How do we decide: what is a PASS, what is a FAIL?

Remember Fuzzing Date Parser

```
f := PzRandomFuzzer new.
r := PzBlockRunner on: [ :e | e asDate ].
f run: r times: 20.
```

- Pharo 11
- String>>asDate



```
PASS "DateError: day is after month ends"
PASS "28 April 2006"
PASS "7 September 2029"
PASS "9 March 1995"
FAIL "SubscriptOutOfBounds: 73"
PASS "DateError: day is after month ends"
FAIL "SubscriptOutOfBounds: 0"
PASS "DateError: day is after month ends"
PASS "6 January 2007"
PASS "9 January 1986"
FAIL "SubscriptOutOfBounds: 0"
FAIL "#isAlphaNumeric was sent to nil"
PASS "DateError: day is after month ends"
PASS "1 September 1989"
PASS "Dat
PASS "Dat                                  ve"
PASS "5 J
PASS "Dat                                  ve"
PASS "7 S
PASS "2 J
```

Who decides: what is a PASS, what is a FAIL?

The Date Parser Oracle

```
f := PzRandomFuzzer new.
r := PzBlockRunner on: [ :e | e asDate ].
f run: r times: 20.
```

- DateError is an expected error
- Malformed inputs should fail!
  - `.+!;/./852"%7?3720("/)"!*43<,"4@>)>'(,"0(+7?`
  - `;% *:(41)215>/1890)@ 3"@3.35+6`

```
PASS "DateError: day i
PASS "28 April 2006"
PASS "7 September 2029
PASS "9 March 1995"
FAIL "SubscriptOutOfBc
PASS "DateError: day i
FAIL "SubscriptOutOfBounds: 0"
PASS "DateError: day is after month ends"
PASS "6 January 2007"
PASS "9 January 1986"
FAIL "SubscriptOutOfBounds: 0"
FAIL "#isAlphaNumeric was sent to nil"
PASS "DateError: day is after month ends"
PASS "1 September 1989"
PASS "Dat
PASS "Dat
PASS "5 J
PASS "Dat
PASS "7 S
PASS "2 J
```

How do we decide: what is a PASS, what is a FAIL?

The Oracle Problem

Given a program and an input,

How can we **distinguish** correct from incorrect behavior?

The Oracle Problem

Given a program and an input,

How can we **automatically** **distinguish** correct from incorrect behavior?

General Solutions for the Oracle Problem

- Four kind of solutions
  - Derived oracles
  - Specified oracles
  - Implicit oracles
  - Coping with the lack of oracles

The Oracle Problem in Software Testing: A Survey. Barr et al. IEEE Trans.'15

Specified Oracles

- Formal modeling of system behavior E.g.,
  - Specification languages
    - State machines, model-checking
  - Assertions, contracts, properties

- Main challenge: model system behavior

Derived Oracles

- Oracles built from some source. E.g.,
  - grammars
  - previous versions of the system
  - documentation/comments
  - code history — git repositories

Implicit Oracles

- Aka *weak oracles*
- Look for obvious behaviors
    - crashes/errors
    - pointer sanitizing
    - Deadlocks/blockages
    - Profiler information metrics

When there is no clear Oracle

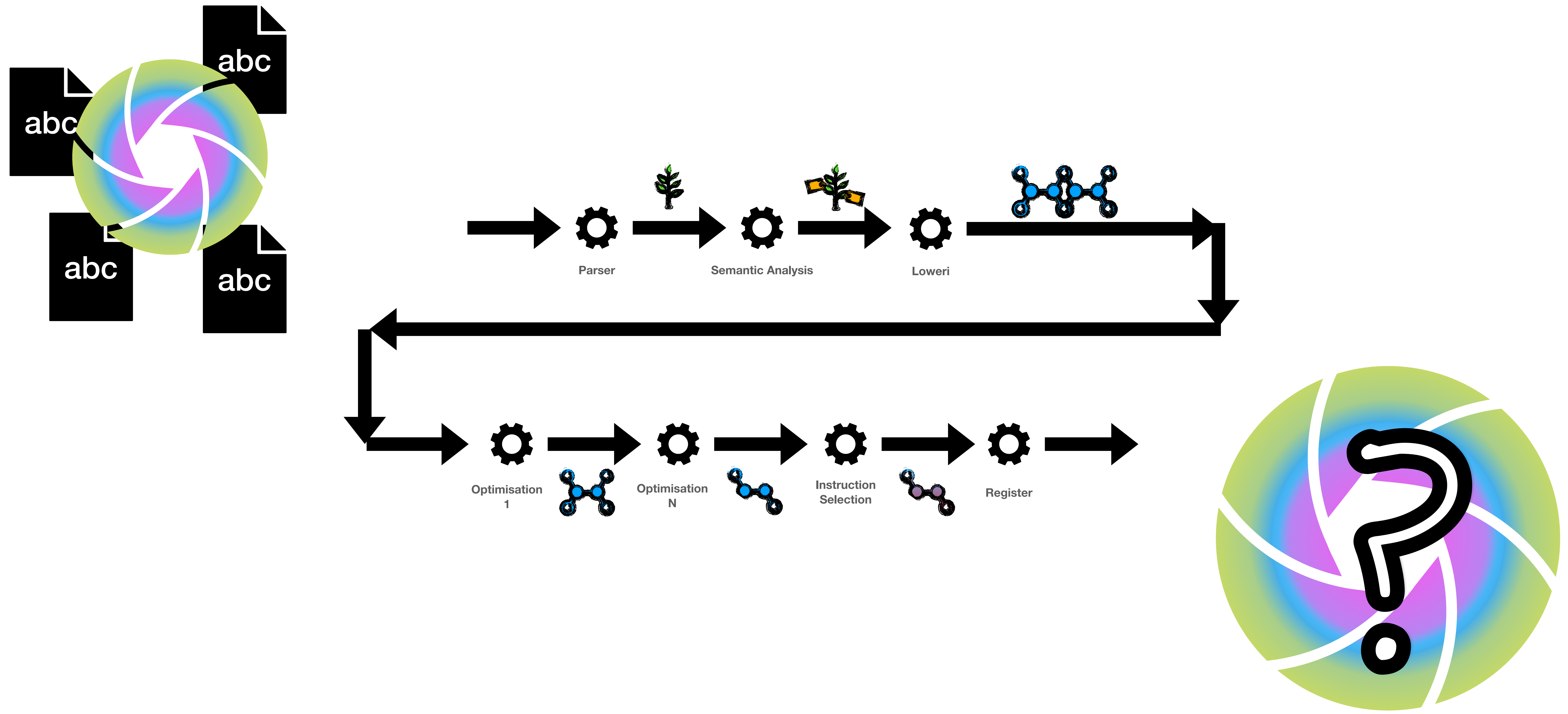
- Crowdsourcing oracles
- GPT-4?

- Or reduce the effort for human (manual) oracles
  - reduce test suites — remove redundant tests
  - reduce test cases — simplify tests

Remember Compilers

Source code
abc
AST
Annotated AST
...
Cont'd below
Parser
(Scanner/Tokeniser included)
Semantic Analysis
(Name resolution, type checking)
Lowering
Frontend
Ideally, Language Dependent
Mid-end
Ideally, Language/Machine independent
Backend
Ideally, Machine Dependent
Cont'd from above
Optimisation 1
...
Optimisation N
Instruction Selection
...
Register Allocation
Code Generation
Target Code
012

Testing Randomly Generated Programs



Takeaways

- Random inputs get random outputs
- Oracles decide when an output is expected or not
- Oracles can be implicit, derived, specified
- Alternatively, reduce the cost of manual inspection

Material

- The Oracle Problem in Software Testing: A Survey. Barr et al. IEEE Transactions.’15