SM2-TES: Functional Programming and Property-Based Testing, Day 7

Jan Midtgaard

MMMI, SDU

SDU

Outline

Exercises

Talk: Growing and Shrinking Polygons . . .

More on Properties

Talk: QuickChecking Google's LevelDB

Project Ideas

Exercises

Exercises from last time

Talk: Growing and Shrinking Polygons . . .

SDU

Ilya Sergey, ICFP 2016: Growing and Shrinking Polygons for Random Testing of Computational Geometry

More on Properties

Properties

What properties should one test for?

- In an unsafe language a first property could simply be *"doesn't crash"*.

  In C/C++/. . . code this can find many errors

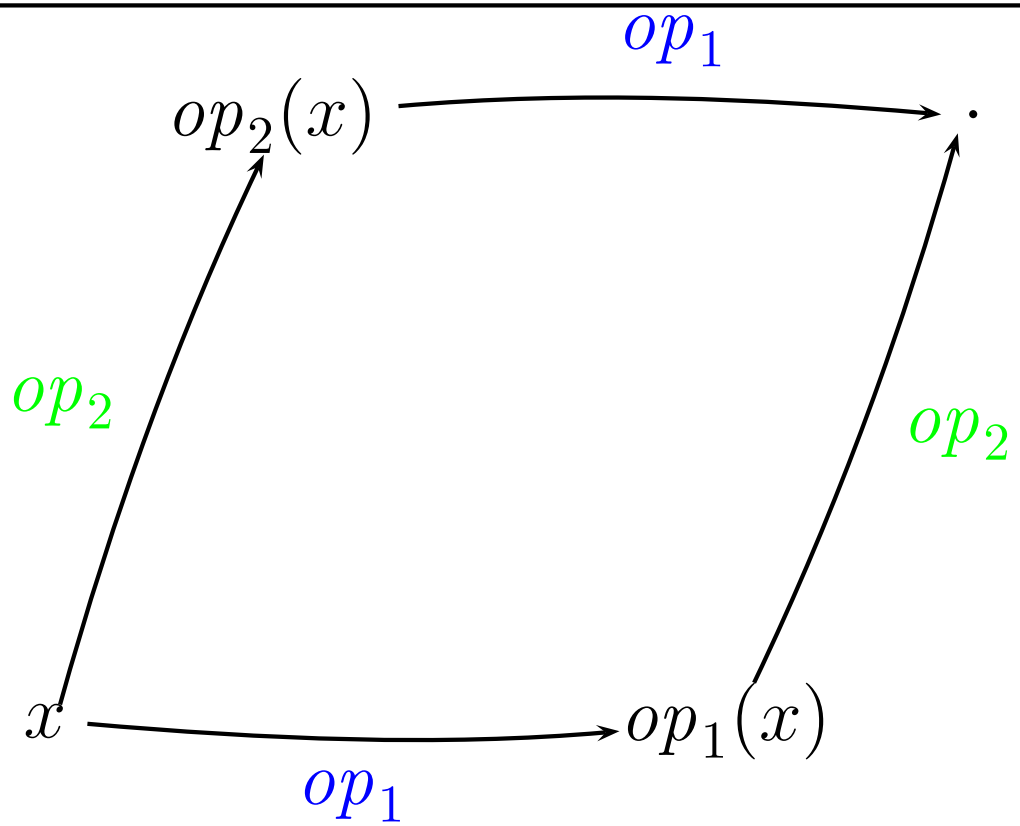
- Sometimes you have an oracle which you can test against.

  For example, testing an advanced data structure against a simpler, naive implementation

These are two simple guidelines for coming up with properties

But there are more general patterns (Scott Wlaschin)

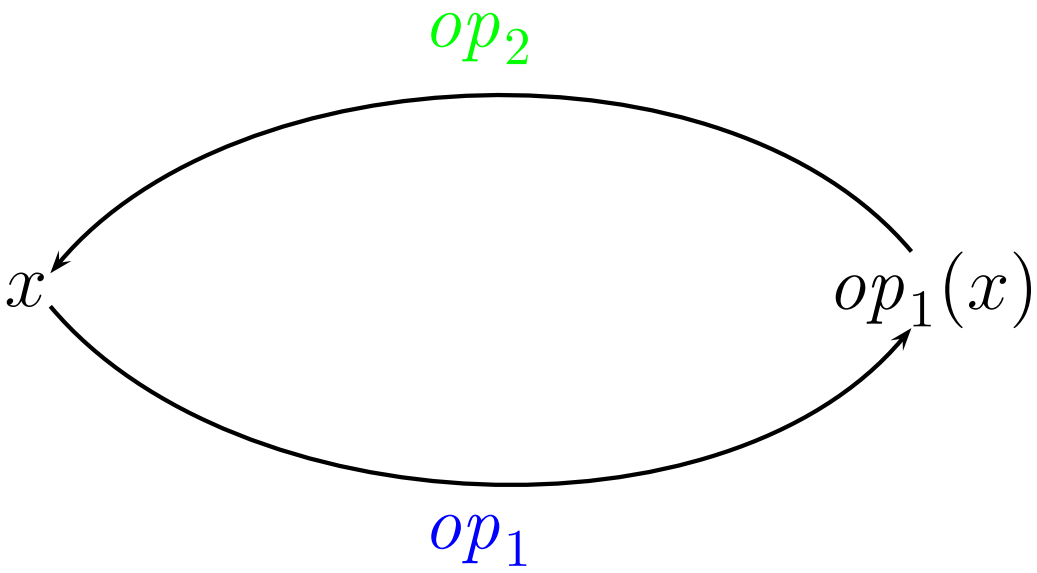
Commuting diagram ("different paths, same destination")



A common property is that two different sequences of operations should yield the same result.

Examples:
model-impl. agreement in model-based approach,
rev-concat vs concat-rev, interp. vs compile-run, . . .

Inverses (“there and back again”)



Another common property is that

two operations act as inverses

Examples:
encryption+decryption, prettyprint-parse, add-subtract, exp-log, reverse-reverse, serialize-deserialize, insert-remove, add-lookup

Related inputs lead to related outputs

Another common (relational) property is that two related inputs given to an operation should give rise to two related outputs

Examples:

- Congruence $i \sim i' \implies f(i) \sim f(i')$

  Two equal/equivalent sets/data structures repr. differently in memory, should produce equivalent results

- Monotonicity/anti-tonicity $i \leq i' \implies f(i) \leq f(i')$
  (data-flow analysis, shortests paths, . . .)

  In general, “bigger input” should lead to “bigger result” (for suitable ordering, e.g., interpreting $\texttt{false} \leq \texttt{true}$)

Invariants (“some things never change”)

Common to many data structures (but also many programs) is an invariant

(something that doesn’t change or vary)

Examples:

- Red-black invariant,
- search-tree invariant,
- sorting preserves length,
- sorting preserves elements,
- “counter represents number of elements in database”

Idempotency ("The more things change, the more they stay the same")

Another common property is that several invocations of the same operation does not change the outcome.

Examples:

- sorting $\texttt{sort}\ l = \texttt{sort}\ (\texttt{sort}\ l)$,
- `String.lowercase`, `String.uppercase`,
- member?-functions
- ...

Structural induction ("Solve a smaller problem first")

Some properties lend themselves to be broken up into a property for a sub-problem, akin to how we prove a property using structural induction.

Examples:

- Sorting: a list is sorted if it has
  - zero or one element (base cases)
  - two or more elements, the first two are sorted, and the list's tail is sorted (inductive hypothesis)

```
let rec sorted xs = match xs with
  | []  -> true
  | [x] -> true
  | x::y::xs' -> x <= y && sorted (y::xs')
```

- Reverse: a list `ys` is the reverse of `xs` iff ...?

Easier to verify (“hard to prove, easy to verify”)

A number of problems in CS are hard to solve, but much easier to check.

Examples:

- prime-number factorization,
- sorting vs. check sorted,
- path finding,
- tokenization,
- any NP-complete problem (SAT, traveling salesman, graph colouring, . . .),
- fixed-point computation vs. checking,
- . . .

Blackbox or whitebox testing?

- The red-black trees is an example of a data structure invariants — a whitebox property
- The algebraic and model-based approaches in yesterday's `Queue` example are typically geared towards blackbox properties

  "How should the individual API interact / operate abstractly?"

Working out properties

Sometimes you have half an idea for a relevant property

It can be useful to play devil's advocate:

"Which malicious implementation could escape these tests?"

Working out properties

Sometimes you have half an idea for a relevant property

It can be useful to play devil's advocate:

"Which malicious implementation could escape these tests?"

"How can I patch my property such that it doesn't?"

Working out properties

Sometimes you have half an idea for a relevant property

It can be useful to play devil's advocate:

"Which malicious implementation could escape these tests?"

"How can I patch my property such that it doesn't?"

Example:

```
(fun xs -> xs = List.rev (List.rev xs))
```

Q: can you think of an implementation flying under this radar?

Talk: QuickChecking Google's LevelDB

SDU

Joseph W. Norton, Lambda Jam 2013:
QuickCheck A Silver Bullet for testing?
(QuickChecking Google's LevelDB)

Project Ideas

Project ideas (1/3)

Here is first a list of generic proposals:

- Test the last program you wrote
- Test the last library you used

Project ideas (1/3)

Here is first a list of generic proposals:

- Test the last program you wrote
- Test the last library you used
- Is there a program/library/module at work which could benefit from testing?

Project ideas (1/3)

Here is first a list of generic proposals:

- Test the last program you wrote
- Test the last library you used
- Is there a program/library/module at work which could benefit from testing?
- Test software from the last course you took

Project ideas (1/3)

Here is first a list of generic proposals:

- Test the last program you wrote
- Test the last library you used
- Is there a program/library/module at work which could benefit from testing?
- Test software from the last course you took
- Test the implementation of the last non-trivial algorithm you wrote/reused

Project ideas (2/3)

- Test properties of csv or sexp library (a generator of csv or sexp, which properties?)

Project ideas (2/3)

- Test properties of csv or sexp library (a generator of csv or sexp, which properties?)
- Test properties of XML tools (a generator of random XML, which properties?)

Project ideas (2/3)

- Test properties of csv or sexp library (a generator of csv or sexp, which properties?)
- Test properties of XML tools (a generator of random XML, which properties?)
- Test properties of JSON tools (a generator of random JSON, which properties?)

Project ideas (2/3)

- Test properties of csv or sexp library (a generator of csv or sexp, which properties?)
- Test properties of XML tools (a generator of random XML, which properties?)
- Test properties of JSON tools (a generator of random JSON, which properties?)
- Test graphical properties of a 2D/3D engine (a generator of random vectors, matrices, . . . which properties?)

Project ideas (2/3)

- Test properties of csv or sexp library (a generator of csv or sexp, which properties?)
- Test properties of XML tools (a generator of random XML, which properties?)
- Test properties of JSON tools (a generator of random JSON, which properties?)
- Test graphical properties of a 2D/3D engine (a generator of random vectors, matrices, . . . which properties?)
- Test properties of a 2D/3D physical engine (a generator of random 2D/3D shapes, . . . which properties?)

Project ideas (3/3)

- model-based GUI testing: a language of 'graphical actions' (push button, select item in dropdown menu, ...) and an state machine generator to produce them

Project ideas (3/3)

- model-based GUI testing: a language of 'graphical actions' (push button, select item in dropdown menu, ...) and an state machine generator to produce them
- Test (parts of) a standard library (Sets, maps, hashtables, queues, . . .)

Project ideas (3/3)

- model-based GUI testing: a language of 'graphical actions' (push button, select item in dropdown menu, ...) and an state machine generator to produce them
- Test (parts of) a standard library (Sets, maps, hashtables, queues, . . .)
- Test a math library (which properties?)

Project ideas (3/3)

- model-based GUI testing: a language of 'graphical actions' (push button, select item in dropdown menu, ...) and an state machine generator to produce them
- Test (parts of) a standard library (Sets, maps, hashtables, queues, . . .)
- Test a math library (which properties?)
- Test a big-int library (which properties?)

Project ideas (3/3)

- model-based GUI testing: a language of 'graphical actions' (push button, select item in dropdown menu, ...) and an state machine generator to produce them
- Test (parts of) a standard library (Sets, maps, hashtables, queues, . . .)
- Test a math library (which properties?)
- Test a big-int library (which properties?)
- Test a bdd library (which properties?)

Project ideas (3/3)

- model-based GUI testing: a language of 'graphical actions' (push button, select item in dropdown menu, ...) and an state machine generator to produce them
- Test (parts of) a standard library (Sets, maps, hashtables, queues, . . .)
- Test a math library (which properties?)
- Test a big-int library (which properties?)
- Test a bdd library (which properties?)
- Test MIRs REST API – or another one (which properties? general or domain-specific?)

Summary and conclusion

- We've seen two widely different example uses
- We've covered general patterns of reusable QuickCheck properties
- Properties can be either whitebox or blackbox
- It can be useful to play devil's advocate ("which impl. could escape these tests?")
- We discussed various project ideas
- Perhaps you got an idea for a project from one of the above?