SM2-TES: Functional Programming and Property-Based Testing, Day 10

Jan Midtgaard

MMMI, SDU

SDU

Last time. . .

- Inference rules
  for formally specifying a type system
- The Curry-Howard correspondence:
  A type system as a logic
- Typed program generation
  by bottom-up reading of typing rules
- Differential testing
  of OCaml's compiler backends against each other

Outline

Race Condition Testing with Parallel State Machines

Stack-Driven Program Generation of WebAssembly

Integrated vs. Type-Directed Shrinking

Race Condition Testing with Parallel State Machines

SDU

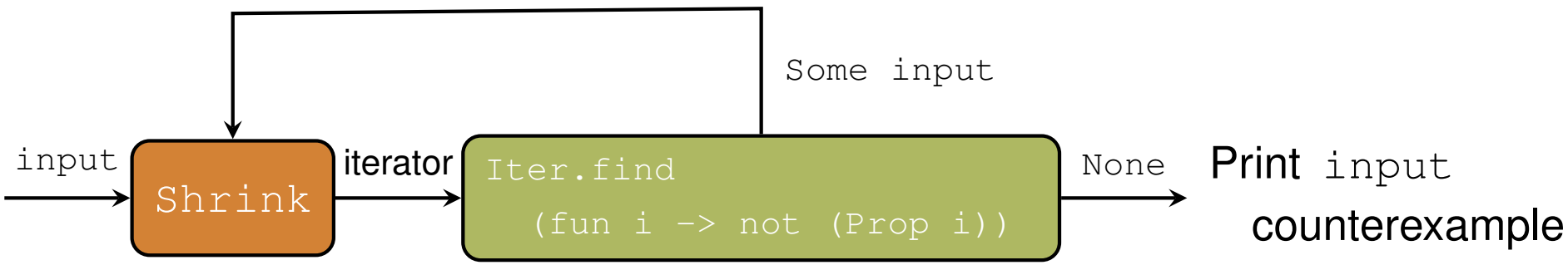
John Hughes:
Testing the Hard Stuff and Staying Sane

`https://www.youtube.com/watch?v=zi0rHwfiX1Q`

Parallel state machines: The fine print (1/2)

Concurrency bugs are extra hard because they are non-deterministic:

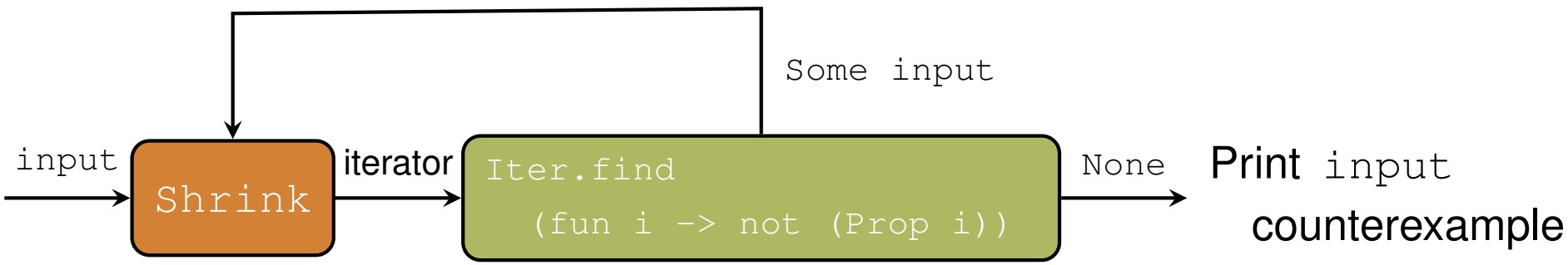
- you don't get the same behaviour in each run
- this complicates PBT, as testing and shrinking assume determinism, e.g., in our shrinking loop:



Parallel state machines: The fine print (1/2)

Concurrency bugs are extra hard because they are non-deterministic:

- you don't get the same behaviour in each run
- this complicates PBT, as testing and shrinking assume determinism, e.g., in our shrinking loop:



This fails to shrink much if a concurrency bug only shows in, e.g., 1/10 runs!

But Hughes's shrunk examples were so small?

Parallel state machines: The fine print (2/2)

Idea: Just run 100s or 1000s of random tests as usual.

Some sequences may not trigger the bug because of unfortunate scheduling. . .

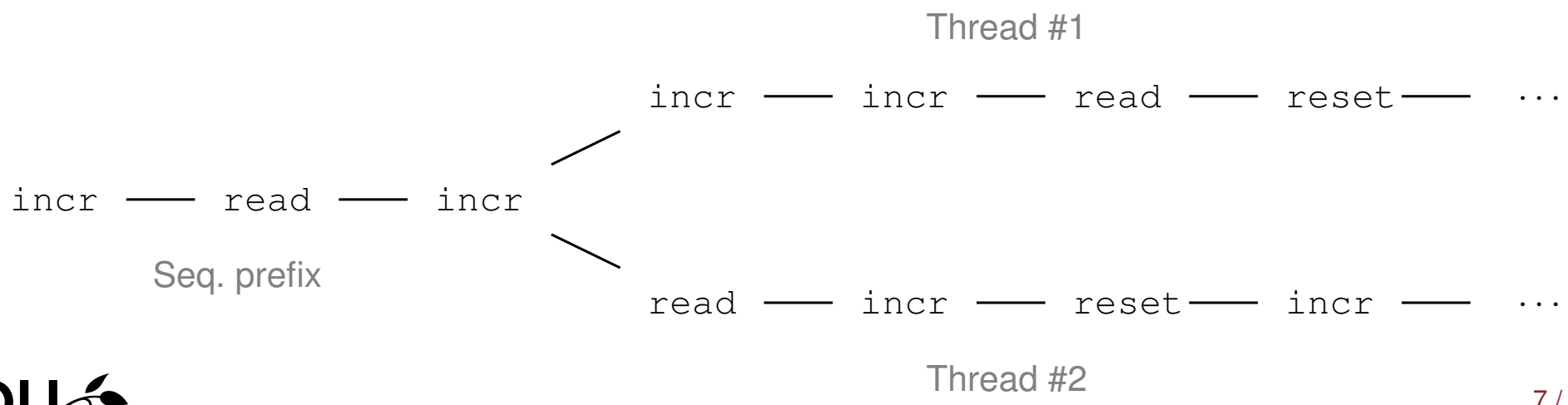
Parallel state machines: The fine print (2/2)

Idea: Just run 100s or 1000s of random tests as usual.

Some sequences may not trigger the bug because of unfortunate scheduling. . .

When a parallel test fails (no sequential interleaving) shrinking needs a little extra work:

- Rerun property 1000 times for each shrunk command sequence
- If just 1/1000 of the reruns fail the bug is still there!
- Shrink tip: move commands to seq. prefix to reduce concurrency:



Stack-Driven Program Generation of WebAssembly

SDU

Perényi and Midtgaard:
Stack-Driven Program Generation of WebAssembly, APLAS 2020

Integrated vs. Type-Directed Shrinking

Integrated shrinking motivation

For built-in data types (`int`, `string`, pairs, lists, . . .)
QCheck provides shrinking for free via `arbitrary`.

These are type-directed: `list string`, `pair int int`

However for user-defined data types

- JSON
- `aexp`
- ASTs
- . . .

we have to write both a generator and a shrinker ourselves (using the `Gen`, `Shrink`, `Iter` modules). . .

This is tedious!

Integrated shrinking: Shrinking for free!?

In a number of frameworks

- Hedgehog (Haskell, Scala, F#, R)
- Quviq QuickCheck (Erlang)
- Hypothesis (Python)
- ...

when you write a generator you get a (not necessarily optimal) shrinker derived automatically!

Such a shrinker does not utilize domain knowledge

- `(fun x -> e') e` $\equiv$ `let x = e in e'`
- `close-db; open-db` cancel each other
- ...

However: it may be good enough!

Integrated shrinking: How?

In QCheck, `'a Gen.t = Random.State.t -> 'a`
Separately, `'a Shrink.t = 'a -> 'a Iter.t`
Two phases naturally follow: generation and shrinking

Integrated shrinking: How?

In QCheck, `'a Gen.t = Random.State.t -> 'a`
Separately, `'a Shrink.t = 'a -> 'a Iter.t`
Two phases naturally follow: generation and shrinking

Instead, one approach is to generate a combined tree with the input data as root and children as shrinking candidates:

```
'a Tree.t = Node of 'a * 'a Tree.t Seq.t
'a Gen.t = Random.State.t -> 'a Tree.t
```

Such a tree with a variable and unbounded children is called a Rose tree.

To keep it manageable we compute children on-demand (lazily), only exploring sub-trees when we need to...

Jacob Stanley:
Gens N'Roses: Appetite for Reduction
`https://www.youtube.com/watch?v=AIv_9T0xKEo`
YOW! Lambda Jam 2017

Challenge: Combining shrink trees

Suppose we have generated **`true`** and `3` along with their shrink trees and want to generate pairs:

```
true
`- false
```

```
3
+- 1
|  `- 0
`- 2
   `- 1
      `- 0
```

Challenge: Combining shrink trees

Suppose we have generated **`true`** and `3` along with their shrink trees and want to generate pairs:

```
true
`- false
```

```
3
+- 1
|  `- 0
`- 2
   `- 1
      `- 0
```

```
(true,3)
+- (false,3)
|  +- (false,1)
|  |  `- (false,0)
|  `- (false,2)
|     `- (false,1)
|        `- ...
+- (true,1)
|  +- (false,1)
|  |  `- (false,0)
|  `- (true,0)
|     `- ...
`- (true,2)
   +- (false,2)
   |  `- ...
   `- ...
```

A `pair` generator does this behind the scenes for you...

Integrated shrinking prototype (1/2)

I've built a little proof-of-concept library with integrated shrinking in OCaml:

```
open Intqc

let nt2 =
  make_test "lessthan 12" Print.int Gen.int (fun i -> i < 12)

;;
test_runner [nt2]
```

Integrated shrinking prototype (1/2)

I've built a little proof-of-concept library with integrated shrinking in OCaml:

```
open Intqc

let nt2 =
  make_test "lessthan␣12" Print.int Gen.int (fun i -> i < 12)

;;
test_runner [nt2]
```

This produces:

```
Test "lessthan␣12": #
  Failed! Property failed after 1 tests
  Initial counterexample: 348632671
  Shrunk counterexample: 12
```

Integrated shrinking prototype (2/2)

Another example, not immediately expressive in QCheck:

```
open Intqc

let nt13 =
  let dep_gen = Gen.((int_bound 10) >>= fun i ->
                  map (fun xs -> (i,xs)) (listlen int i)) in
  make_test "map␣dep␣gen" Print.(pair int (list int))
    dep_gen (fun (i,xs) -> i = List.length xs && i<8)

;;
test_runner [nt13]
```

Integrated shrinking prototype (2/2)

Another example, not immediately expressive in QCheck:

```
open Intqc

let nt13 =
  let dep_gen = Gen.((int_bound 10) >>= fun i ->
                  map (fun xs -> (i,xs)) (listlen int i)) in
  make_test "map_dep_gen" Print.(pair int (list int))
    dep_gen (fun (i,xs) -> i = List.length xs && i<8)

;;
test_runner [nt13]
```

This produces:

```
Test "map_dep_gen": #####
  Failed! Property failed after 5 tests
  Initial counterexample: (10,[101240475; 53486851; 185017518;
560607193; 38350177; 334796179; 366297528; 57891145; 672811027;
382524160])
  Shrunk counterexample: (8,[0; 0; 0; 0; 0; 0; 0; 0])
```

Summary

We have studied

- how PBT can be used to find race conditions
- a case study: generating and testing WebAssembly
- type-directed vs. integrated shrinking