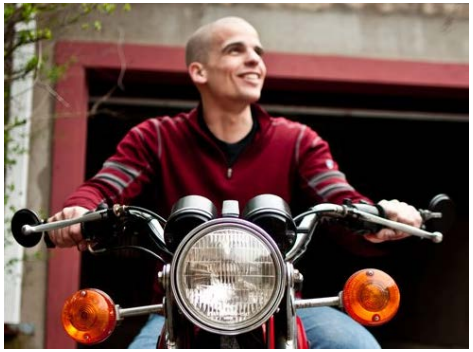
Automating Programming Assessments

Things I Learned Porting 15-150 to Autolab

Iliano Cervesato

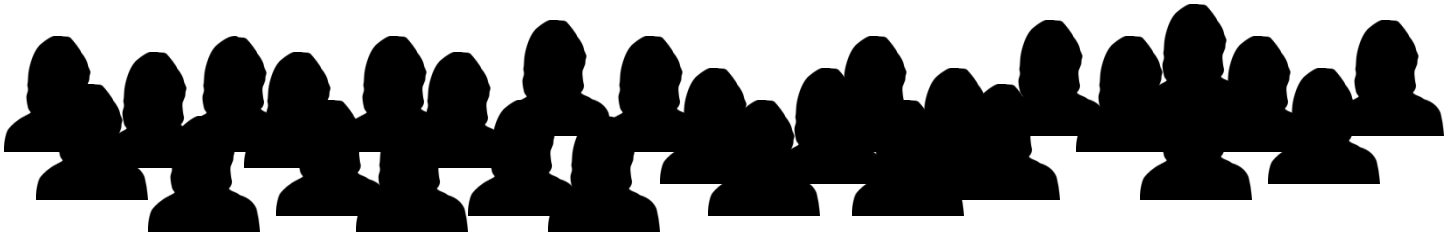
Thanks!

Bill Maynes



Ian Voysey

Jorge Sacchini



Generations of 15-150, 15-210 and 15-212 teaching assistants

Outline

- Autolab
- The challenges of 15-150
- Automating Autolab
  - Test generation
- Lessons learned and other thoughts

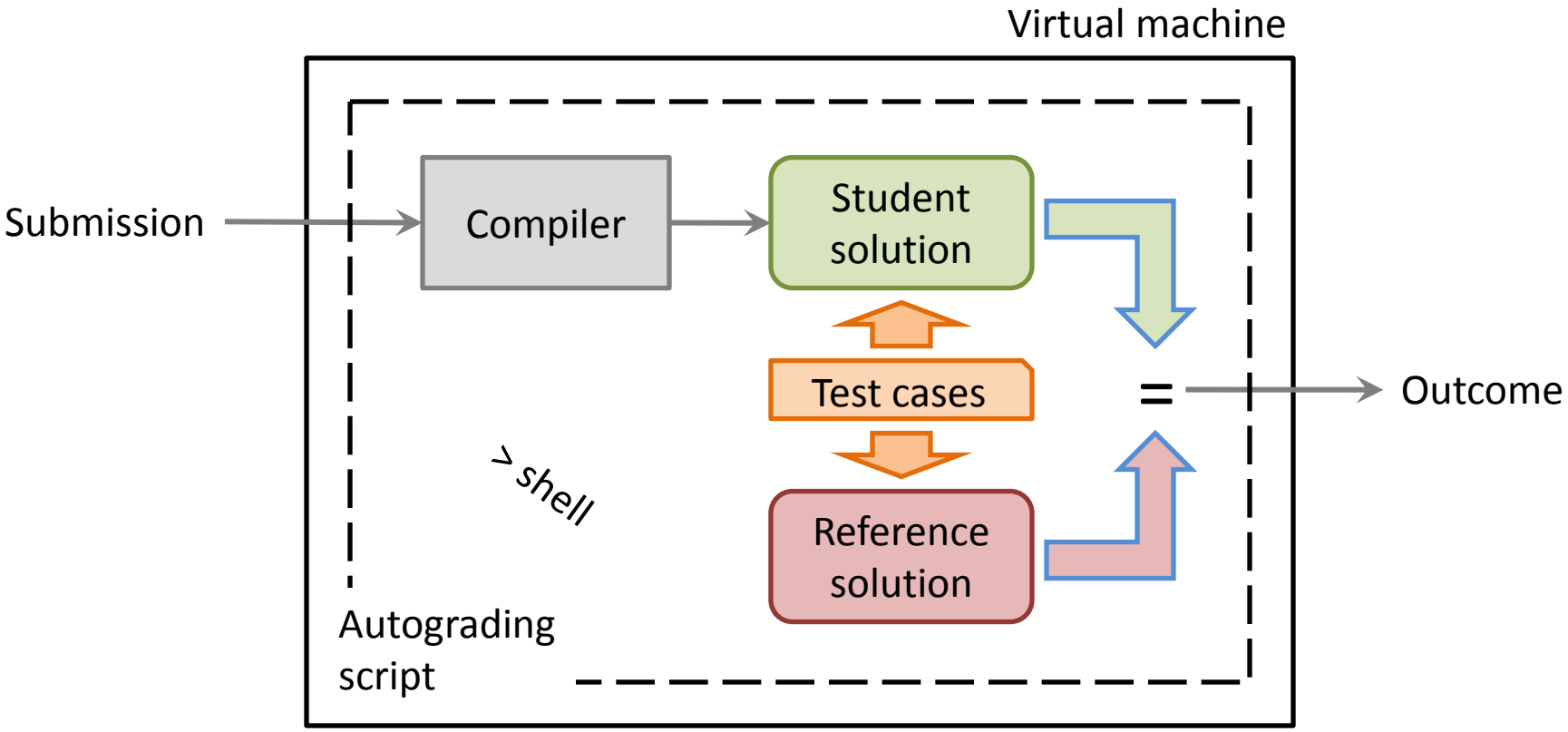
AUTØLAB

- Tool to automate assessing programming assignments
  - Student submits solution
  - Autolab runs it against reference solution
  - Student gets immediate feedback
    - » Learns from mistakes while on task
- Used in 80+ editions of 30+ courses
- Customizable

The promises of Autolab

- Enhance learning
  - By pointing out errors while students are on task
  - *Not when the assignment is returned*
    - » *Students are busy with other things*
    - » *They don't have time to care*
- Streamline the work of course staff … maybe
  - Solid solution must be in place from day 1
  - Enables automated grading
    - » Controversial

How Autolab works, typically



The Challenges of 15-150

15-150

Use the mathematical structure of a problem to program its solution

- Core CS course
- Programming and theory assignments

- Pittsburgh (x 2)
  - 150-200 students
  - 18-30 TAs
- Qatar
  - 20-30 students
  - 0-2 TAs

Autolab in 15-150q

- Used as
  - Submission site
  - Immediate feedback for coding components
  - Cheating monitor via MOSS integration
- Each student has 5 to 10 submissions
  - Used 50.1% in Fall 2014
- Grade is *not* determined by Autolab
  - All code is read and commented on by staff

The Challenges of 15-150

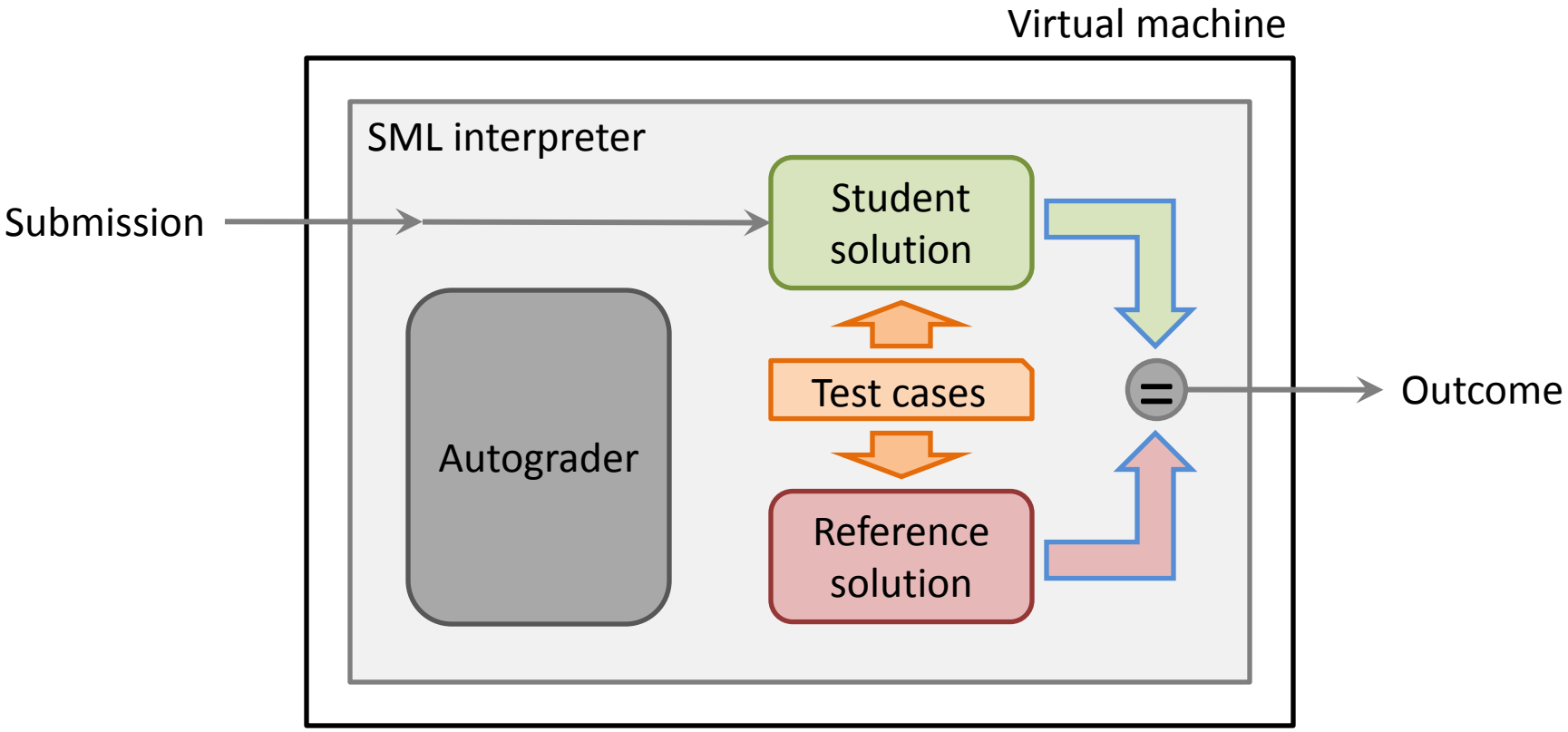
- 15-150 relies on Standard ML (common to 15-210, 15-312, 15-317, …)
  - Used as an *interpreted* language
    - » no I/O
  - Strongly typed
    - » No "eval"
  - Strict module system
    - » Abstract types
- 11, very diverse, programming assignments
  - Grader for hw-*(x+1)* very different from hw-*x*

Autograding SML code

- Traditional model does not work well
  - Requires students to write unnatural code
  - Needs complex parsing and other infrastructure
    - » But SML interpreter already comes with a parser for SML
- Instead, make everything happen *within* SML
  - running test cases
  - establishing outcome
  - dealing with errors

Student and reference code become modules

Running Autolab with SML



Making it work is non-trivial

- Done for 15-210
  - But 15-150 has much more assignment diversity
- No documentation
  - Initiation rite of TAs by older TAs
    - » Cannot work on the Qatar campus!
  - Demanding on the course staff
- TA-run
  - Divergent code bases

Too important to be left to rotating TAs

What's in a typical autograder?

```
grader.cm
handin.cm
handin.sml
autosol.cm
autosol.sml
HomeworkTester.sml
xyz-test.sml
aux/
   allowed.sml
   xyz.sig
   sources.cm
   support.cm
```

- A working autograder took 3 days to write
  - Tedious, ungrateful job
  - Proceed by trial and error
  - Lots of repetitive parts
  - Cognitively complex
  - Each assignment brings new challenges
- Time taken away from helping students
- Discourages developing new assignments

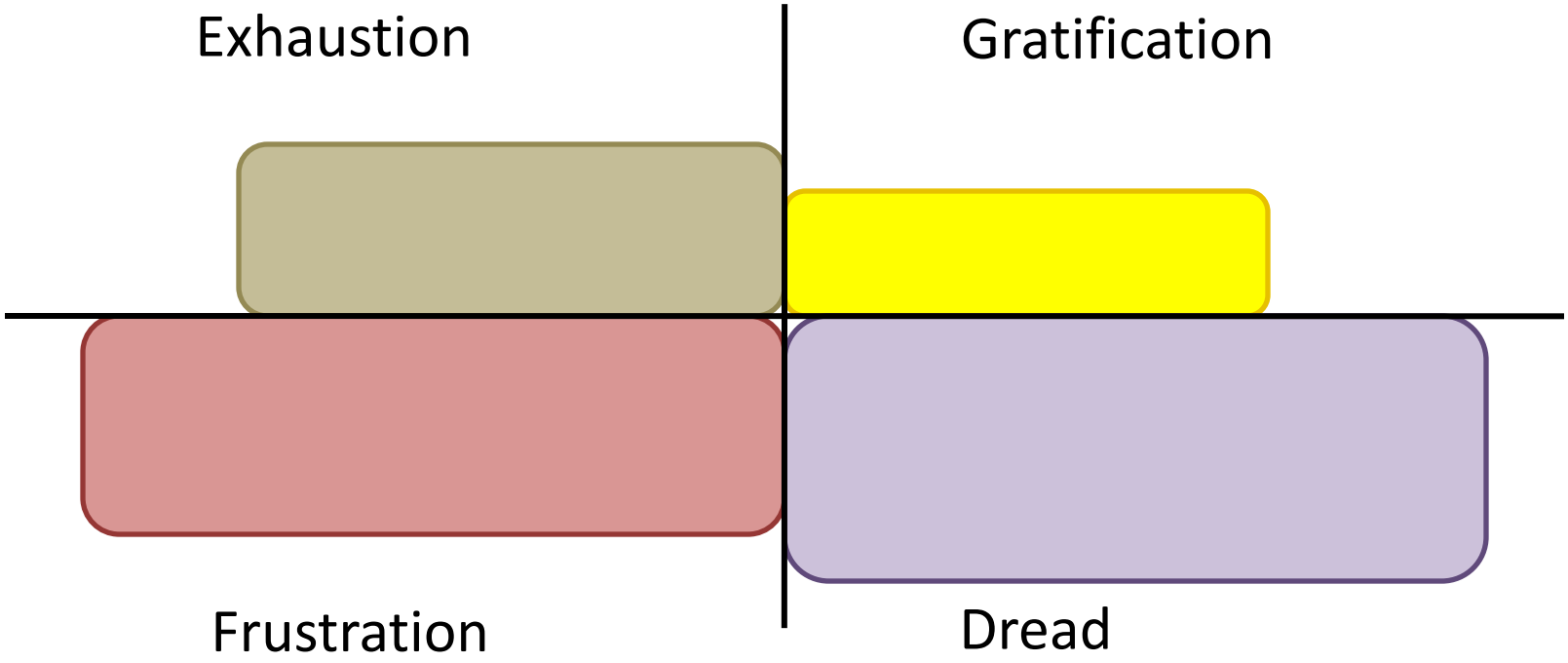
(*simplified*)

HomeworkTester.sml – *Fall 2013*

```
structure HomeworkTester =
struct
  exception FatalError of string

  structure Stu = StuHw04Code
  structure Our = Hw04Tests (Hw04 (Stu))

  fun bool_toString true  = "true"
    | bool_toString false = "false"

  fun pair_toString fst_ts snd_ts (x,y) =
      "(" ^ (fst_ts x) ^ ", " ^ (snd_ts y) ^ ")"

  fun triple_toString ts snd_ts trd_ts (x,y,z) =
     "(" ^ (fst_ts x) ^ ", " ^ (snd_ts y) ^ ", " ^ (trd_ts z) ^ ")"

  fun list_toString toString l =
    let fun lts [] = ""
          | lts [x] = toString x
          | lts (x::l) = toString x ^ ",\n " ^ lts l
    in "[" ^ lts l ^ "]" end

  fun compareReal (x: real, y: real): bool = Real.abs (x-y) < 0.0001

  val studTests_traverseS = Our.treeSList1
  val studTests_canonical = Our.treeSList1
  val studTests_simplify = Our.treeSList1
  val studTests_simplify_safe = studTests_simplify

  val studTests_traverseC = Our.treeCList1
  val studTests_convert = Our.treeCList1
  val studTests_convertCan = studTests_convert
  val studTests_convertCan_safe = studTests_convertCan
  val studTests_convertSloppy = Our.treeSList1

  val studTests_convert_safe = studTests_convert
  val studTests_splitN = Our.treeIntList1
  val studTests_leftmost = Our.treeList3
  val studTests_halves = Our.treeList3
  val studTests_rebalance = Our.treeList1

  fun test_traverseS () = OurTester.testFromRef
      (Our.treeS_toString) (list_toString Char.toString)
      (op =)
      (Stu.traverseS) (Our.traverseS)
      (studTests_traverseS)

  fun test_canonical () = OurTester.testFromRef
      (Our.treeS_toString) (bool_toString)
      (op =)
      (Stu.canonical) (Our.canonical)
      (studTests_canonical)

  fun test_simplify () = OurTester.testFromRef
      (Our.treeS_toString) (Our.treeS_toString)
      (op =)
      (Stu.simplify) (Our.simplify)
      (studTests_simplify)

  fun test_simplify_safe () = OurTester.testFromRef
      (Our.treeS_toString) (Our.treeS_toString)
      (op =)
      (Stu.simplify_safe) (Our.simplify_safe)
      (studTests_simplify_safe)

  fun test_traverseC () = OurTester.testFromRef
      (Our.treeC_toString) (list_toString Char.toString)
      (op =)
      (Stu.traverseC) (Our.traverseC)
      (studTests_traverseC)

  fun test_convertCan () = OurTester.testFromRef
      (Our.treeC_toString) (Our.treeC_toString)
      (op =)
      (Stu.convertCan) (Our.convertCan)
      (studTests_convertCan)

  fun test_convertCan_safe () = OurTester.testFromRef
      (Our.treeS_toString) (Our.treeC_toString)
      (op =)
      (Stu.convertCan_safe) (Our.convertCan_safe)
      (studTests_convertCan_safe)

  fun test_convertSloppy () = OurTester.testFromRef
      (Our.treeS_toString) (Our.treeC_toString)
      (op =)
      (Stu.convertSloppy) (Our.convertSloppy)
      (studTests_convertSloppy)

  fun test_convert () = OurTester.testFromRef
      (Our.treeC_toString) (Our.tree_toString)
      (Our.tree_eq)
      (Stu.convert) (Our.convert)
      (studTests_convert)

  fun test_convert_safe () = OurTester.testFromRef
      (Our.treeC_toString) (Our.tree_toString)
      (Our.tree_eq)
      (Stu.convert_safe) (Our.convert_safe)
      (studTests_convert_safe)

  fun test_splitN () = OurTester.testFromRef
      (pair_toString Our.tree_toString Int.toString)
      (pair_toString Our.tree_toString Our.tree_toString)
      (op =)
      (Stu.splitN) (Our.splitN)
      (studTests_splitN)

  fun test_leftmost () = OurTester.testFromRef
      (Our.tree_toString)
      (pair_toString Char.toString Our.tree_toString)
      (op =)
      (Stu.leftmost) (Our.leftmost)
      (studTests_leftmost)

  fun test_halves () = OurTester.testFromRef
      (Our.tree_toString)
      (triple_toString Our.tree_toString Char.toString Our.tree_toString)
      (op =)
      (Stu.halves) (Our.halves)
      (studTests_halves)

  fun test_rebalance () = OurTester.testFromRef
      (Our.tree_toString) (Our.tree_toString)
      (op =)
      (Stu.rebalance) (Our.rebalance)
      (studTests_rebalance)
end
```

```
fun test_traverseS () = OurTester.testFromRef
    (Our.treeS_toString)
    (list_toString Char.toString)
    (op =)
    (Stu.traverseS) (Our.traverseS)
    (studTests_traverseS)
```

Autograder development cycle



Work of course staff hardly streamlined

Automating Autolab for 15-150

However …

```
grader.cm
handin.cm
handin.sml
autosol.cm
autosol.sml
HomeworkTester.sml
xyz-test.sml
aux/
   allowed.sml
   xyz.sig
   sources.cm
   support.cm
```

- Most files can be generated automatically from function types
- Some files stay the same
- Others are trivial
  - given a working solution

(*simplified*)

Significant opportunity for automation

- Summer 2013:
  - Hired a TA to deconstruct 15-210 infrastructure
- Fall 2013:
  - Ran 15-150 with Autolab
  - Early automation
- Fall 2014:
  - Full automation of large fragment
  - Documentation
- Summer 2015:
  - Further automation
  - Automated test generation
  - Fall 2015 was loaded on Autolab by first day of class

Autograder Generator

```
structure HwTest =
struct
open MkGrader

val sloppy = mkPbset ("Sloppy",
    ["datatype treeS = emptyS | leafS of string | nodeS of treeS * treeS",
     "val traverseS:     treeS -> string list",
     "val canonical:     treeS -> bool",
     "val simplify:      treeS -> treeS",
     "val simplify_safe: treeS -> treeS"
    ])

val canonical = mkPbset ("Canonical",
    ["datatype treeS = emptyS | leafS of string | nodeS of treeS * treeS",
     "datatype treeC' = leafC of string | nodeC of treeC' * treeC'",
     "datatype treeC = emptyC | T of treeC'",
     "val traverseS:     treeS -> string list",
     "val convertCan:    treeS -> treeC",
     "val convertCan_safe: treeS -> treeC",
     "val convertMap:    treeS -> treeC"
    ])

val balanced = mkPbset ("Balanced",
    ["datatype treeS = leafS of string | nodeS of treeC' * treeC'",
     "datatype treeC  = emptyC | T of treeC'",
     "datatype tree = empty | node of tree * string * tree",
     "val convert: treeC -> tree",
     "val convert_safe: treeC -> tree",
     "val splitN: tree * int -> tree * tree",
     "val rightMost: tree -> string * tree",
     "val halves: tree -> tree * string * tree",
     "val rebalance: tree -> tree"
    ])

val homework = [sloppy, canonical, balanced]

val _ = writeAllFiles homework

end (* structure HwTest *)

(* Short name *)
structure H = HwTest
val _ = OS.Process.exit OS.Process.success
```

```
val sloppy = mkPbset ("Sloppy",
    ["datatype treeS = emptyS | leafS of string | nodeS of treeS * treeS",
     "val traverseS:     treeS -> string list",
     "val canonical:     treeS -> bool",
     "val simplify:      treeS -> treeS",
     "val simplify_safe: treeS -> treeS"
    ])
```

However …

```
mkTester.sml
grader.cm
handin.cm
handin.sml
autosol.cm
autosol.sml
HomeworkTester.sml
xyz-test.sml
aux/
   allowed.sml
   xyz.sig
   sources.cm
   support.cm
```

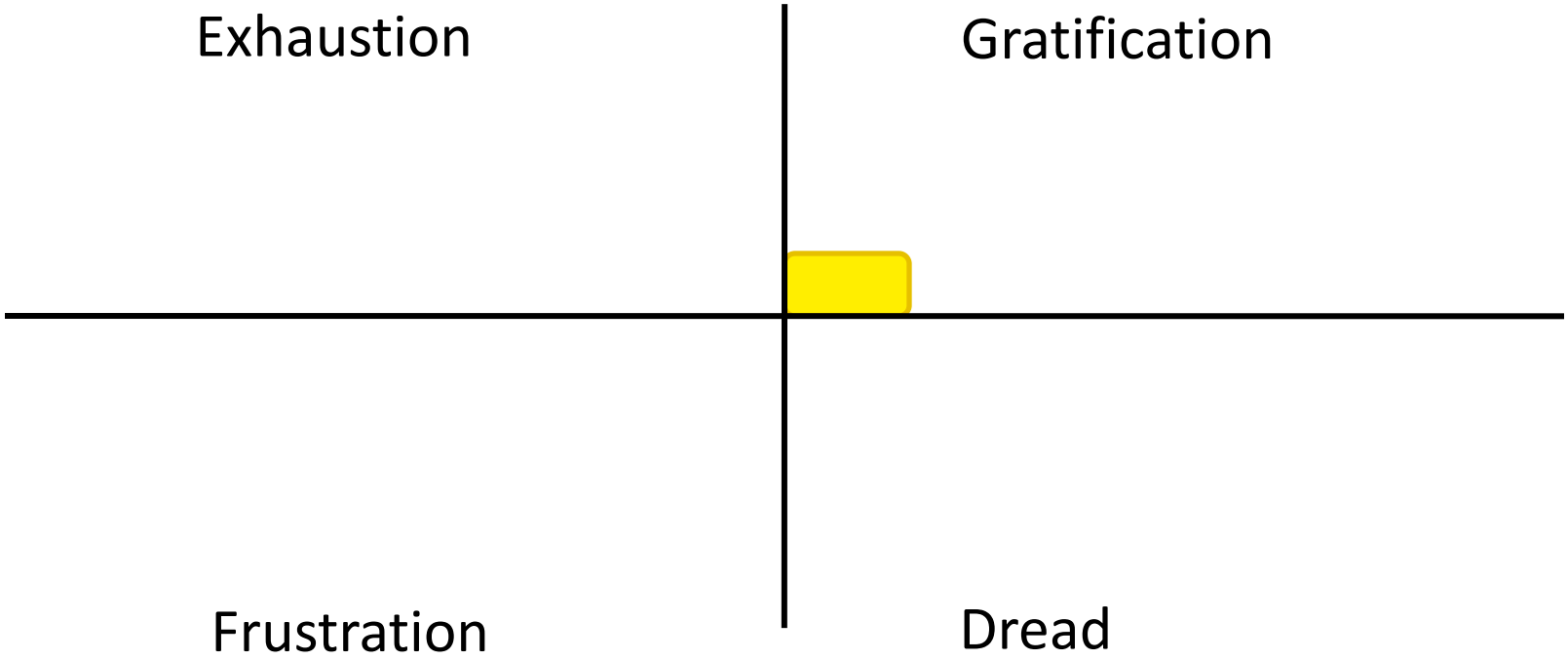
- Most files can be generated automatically from function types
- Some files stay the same
- Others are trivial
  - given a working solution

(*simplified*)

HomeworkTester.sml – *Fall 2015*

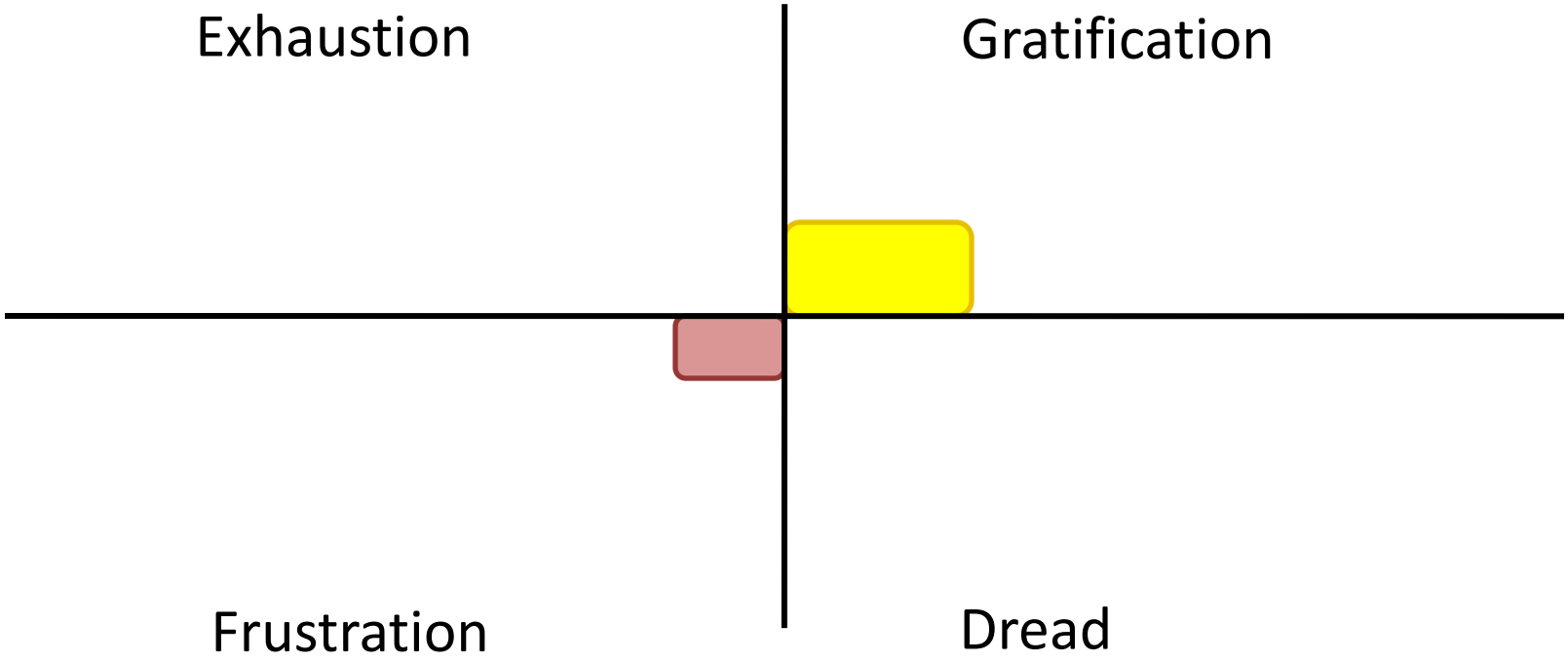
```
(* val canonical:      treeS -> bool *)
fun test_canonical () = OurTester.testFromRef
(*  Input to string      *)  Our.treeS_toString
(*  Output to string     *)  U.bool_toString
(*  output equality      *)  op=
(*  Student solution     *)  (Stu.canonical)
(*  Reference solution   *)  (Our.canonical)
(*  List of test inputs  *)  (studTests_canonical @
                               (extra moreTests_canonical))
```

Is Autolab effortless for 15-150?



Not quite …

… but definitely streamlined



Automate what?

```
(* val fibonacci: int -> int *)
fun test_fibonacci () = OurTester.testFromRef
(* Input to string     *)  Int.toString
(* Output to string    *)  Int.toString
(* output equality     *)  op=
(* Student solution    *)  (Stu.fibonacci)
(* Reference solution  *)  (Our.fibonacci)
(* List of test inputs *)  (studTests_fibonacci @
                             (extra moreTests_fibonacci))
```

Printing

Equality

Tests

Automatically generated

- For each function to be tested,
  - Test cases
  - Equality function
  - Printing functions

Equality and Printing Functions

- Assembled automatically for primitive types
- Generated automatically for user-defined types
  - New ➢ Trees, regular expressions, game boards, …
- Placeholders for abstract types
  - ➢ Good idea to export them!

- Handles automatically
  - ➢ Polymorphism, currying, exceptions, …
  - ➢ Non-modular code

Example

```
(* datatype tree = empty | node of tree * string * tree *)
fun tree_toString (empty: tree): string = "empty"
  | tree_toString (node x) =
     "node" ^ ((U.prod3_toString (tree_toString,
                                  U.string_toString,
                                  tree_toString))     x)

(* datatype tree = empty | node of tree * string * tree *)
fun tree_eq (empty: tree, empty: tree): bool =  true
  | tree_eq (node x1, node x2) =
     (U.prod3_eq (tree_eq, op=, tree_eq)) (x1,x2)
  | tree_eq _ = false
```

Automatically generated

Test case generation

- Defines randomized test cases based on function input type
  - Handles functions as arguments too
- Relies on QCheck library
- Fully automated
  - Works great!

Example

```
(* datatype tree = empty | node of tree * int * tree *)
fun tree_gen (0: int): tree Q.gen =
      Q.choose [Q.lift empty ]
  | tree_gen n =
      Q.choose'[(1, tree_gen  0),
                (4, Q.map node (Q.prod3 (tree_gen (n-1),
                                         Q.intUpto 10000,
                                         tree_gen (n-1)))) ]

(* val Combine : tree * tree -> tree *)
fun Combine_gen n = (Q.prod2 (tree_gen n, tree_gen n))

val Combine1 = Q.toList (Combine_gen 5)
```

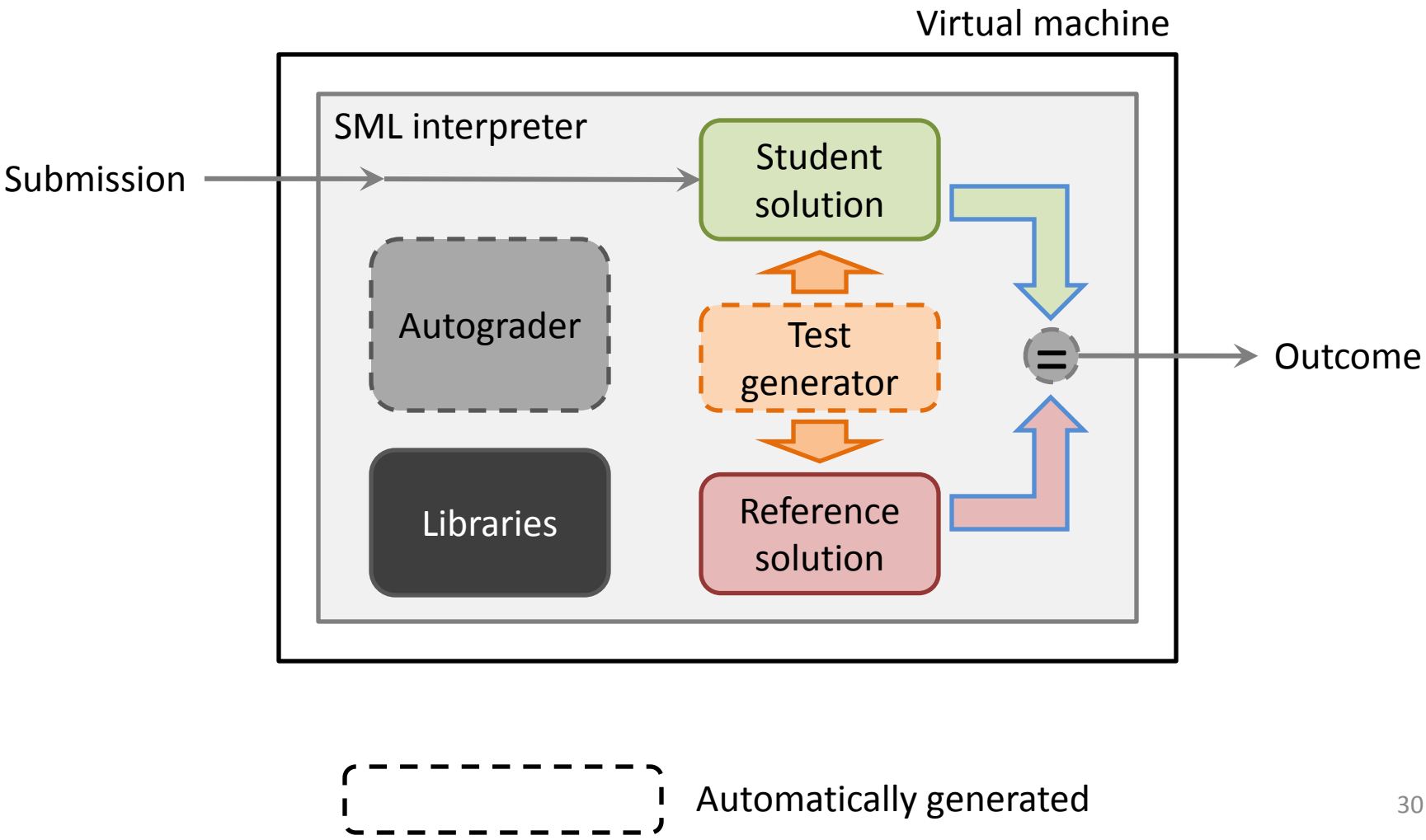
Mostly automatically generated

A more complex example

```
(* val permoPartitions: 'a list -> ('a list * 'a list) list *)
fun test_permoPartitions (a_ts) (a_eq) = OurTester.testFromRef
(* Input to string     *)  (U.list_toString a_ts)
(* Output to string    *)  (U.list_toString
                              (U.prod2_toString
                                (U.list_toString a_ts,
                                 U.list_toString a_ts)))
(* output equality     *)  (U.list_eq
                              (U.prod2_eq
                                (U.list_eq a_eq,
                                 U.list_eq a_q)))
(* Student solution    *)  (Stu.permoPartitions)
(* Reference solution  *)  (Our.permoPartitions)
(* List of test inputs *)  (studTests_permoPartitions @
                             (extra moreTests_permoPartitions))
```

Automatically generated

Current Architecture



Status

- Developing an autograder now takes from 5 minutes to a few hours
  - 3 weeks for all Fall 2015 homeworks, including selecting/designing the assignments, and writing new automation libraries
- Used also in 15-312 and 15-317
- Some manual processes remain

Manual interventions

- Type declarations
  - Tell the autograder they are shared
- Abstract data types
  - Marshalling functions to be inserted by hand
- Higher-order functions in return type
    - E.g., streams
  - Require special test format

- Could be further automated
  - Appear in minority of assignments
  - Cost/reward tradeoff

Example

```
(* val map : (''a -> ''b) -> ''a set -> ''b set *)
fun test_map (a_ts, b_ts) (b_eq) = OurTester.testFromRef
(* Input to string     *)  (U.prod2_toString
                             (U.fn_toString a_ts b_ts,
                              (Our.toString a_ts) o Our.fromList))
(* Output to string    *)  ((Our.toString b_ts) o Our.fromList)
(* output equality     *)  (Our.eq o (mapPair Our.fromList))
(* Student solution    *)  (Stu.toList o (U.uncurry2 Stu.map)
                              o (fn (f,s) => (f, Stu.fromList s)))
(* Reference solution  *)  (Our.toList o (U.uncurry2 Our.map)
                              o (fn (f,s) => (f, Our.fromList s)))
(* List of test inputs *)  (studTests_map @
                             (extra moreTests_map))
```

Mostly automatically generated

Tweaking test generators

- Readability
  - » E.g., avoid finding mistake in 10,000 node tree
- Invariants
  - ➢ Default test generator is unaware of invariants
    - » E.g., factorial: input should be non-negative
- Overflows
  - » E.g., factorial: input should be less than 43
- Complexity
  - » E.g., full tree better not be taller than 20-25

- Still: much better than writing tests by hand!

About testing

- Writing tests by hand is tedious
  - Students hate it
    - » Often skip it even when penalized for it
  - TAs/instructors do a poor job at it
- Yet, testing reveals bugs
  - Pillar of current software development
- Manual tests are skewed
  - Few, small test values
  - Edge cases not handled exhaustively
  - Subconscious bias
    - » Mental invariants

Thoughts

Lessons learned

- Automated grading support helps me run a better course
- Writing an autograder generator is a lot more fun than writing an autograder
- Room for further automation
  - Worked really hard to do less work in the future
- Automated test generation is great!

Future Developments

- Better test generation through annotations
  - E.g., 15-122 style contracts
- Automate a few more manual processes
- Overall architecture can be used with other languages
- Let students use the test generators
  - Currently too complex

To autograde or not to autograde?

- So far, Autolab has be an aid to grading
- Could be used to determine grades automatically in programming assignments
  - Impact on student learning?
  - Cheating?
  - Enable running 15-150 with fewer resources

15-150 beyond programming

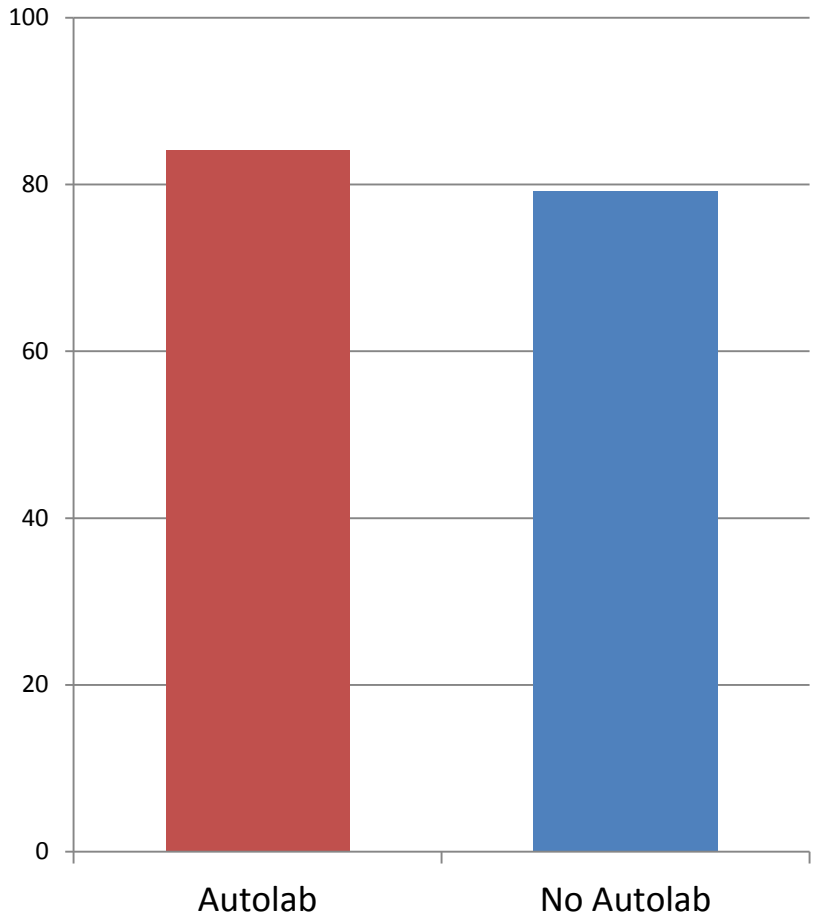
- Proofs
  - Students don't like induction, but don't mind coding
  - Modern theorem provers turn writing a proof into a programming exercise
    - » Can be autograded
- Complexity bounds
  - Same path?

Questions?

Other pedagogic devices

- Bonus points for early submissions
  - Encourages good time management
  - Lowers stress
- Corrected assignments returned individually
  - Helps correct mistakes
  - Assignments graded within 2 days
- Grade forecaster
  - Student knows exactly standing in the course
  - What-if scenarios

Effects on Learning in 15-150



- Insufficient data for accurate assessment
  - ➢ Too many other variables
- Average of the normalized median grade in programming assignments