Finding Julia Bugs Automatically

Robert Feldt, Simon Poulding

BLEKINGE TEKNISKA HÖGSKOLA · BTH ·

DEPARTMENT
OF SOFTWARE
ENGINEERING

Vision: Support spectrum from test to specification

Unit testing à la JUnit et al is great but we can do even better!

Automated test (data) generation

Parameterised Unit Testing

Property-based testing (QuickCheck)

Fuzz Testing

Library of Data Generators

Performance Testing

Distributed Learning

Support “normal” Base.Test syntax

Concrete Tests …Properties… DesignByContract Formal Specification

A set of Julia packages supporting each other

DataGenerators.jl

Generate complex/structured data

Optimize generator to find specific datums

Mix/match/combine generators

DataMutators.jl

Shrink data while it still “fails”

But can also grow data => explore boundaries

BaseTestAuto.jl

Extend Base.Test (in 0.5-dev) => drop-in

Add (adaptive) repeats => auto testing

Assertions can have state and be dynamic

More detailed sub-expression info on fail

BaseTestAuto.jl

Repeated execution of `Base.Test` `@testset`'s

```
1  using BaseTestAuto
2
3  @testset repeats=100 "sign takes values in [-1, 0, 1]" begin
4      i = rand(-10:10)
5      @test_many sign(i) takes_values([-1, 0, 1])
6  end
```

```
8  @testset repeats=true "sign takes values in [-1, 0, 1]" begin
9      i = rand(-10:10)
10     @test_many sign(i) takes_values([-1, 0, 1])
11 end
```

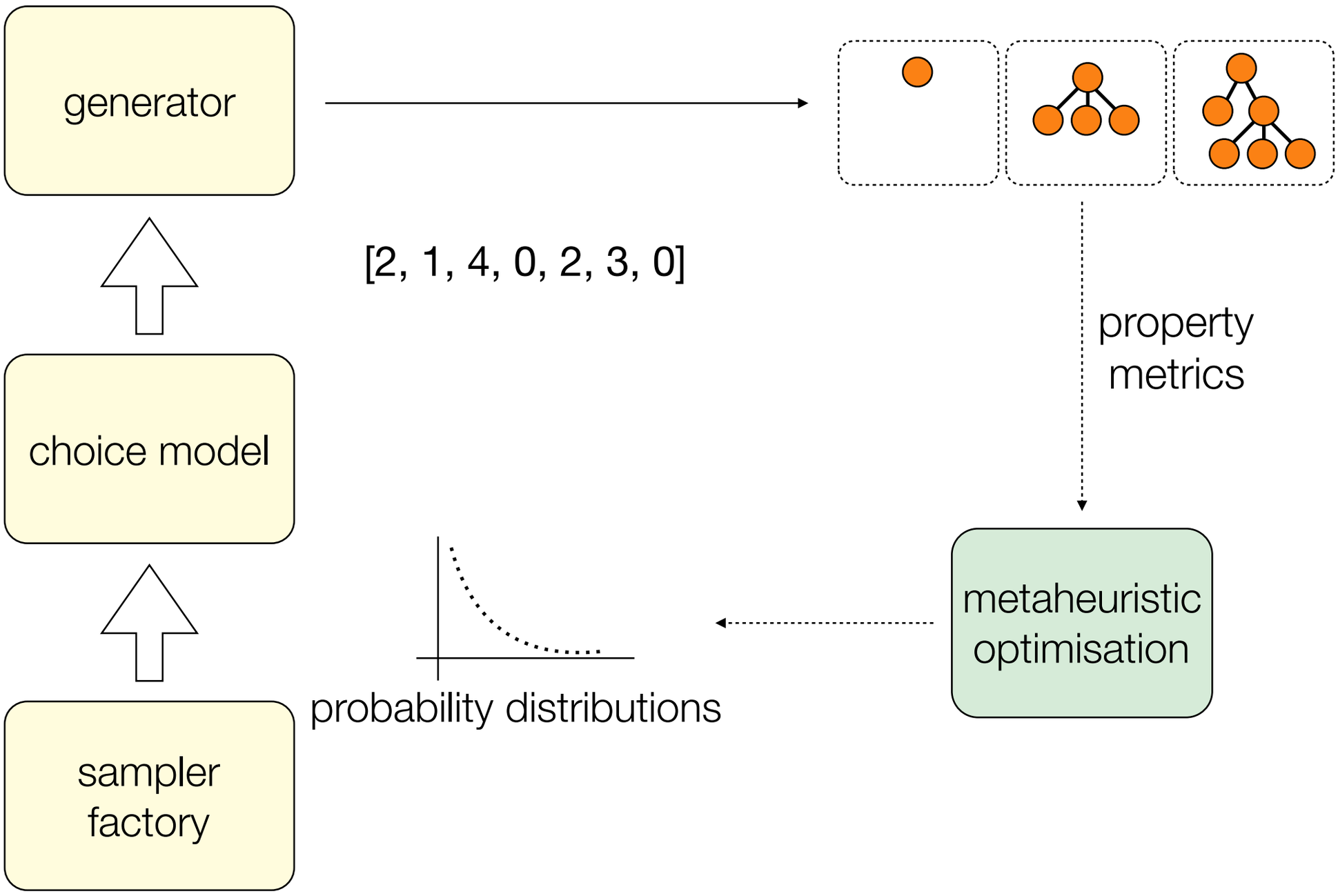
DataGenerators.jl

Generating Trees

```
25  using DataGenerators
26
27  @generator TreeGen begin
28      start = treenode
29      treenode = TreeNode(label, mult(treenode))
30      label = choose(Int, 1, 9)
31  end
32
```

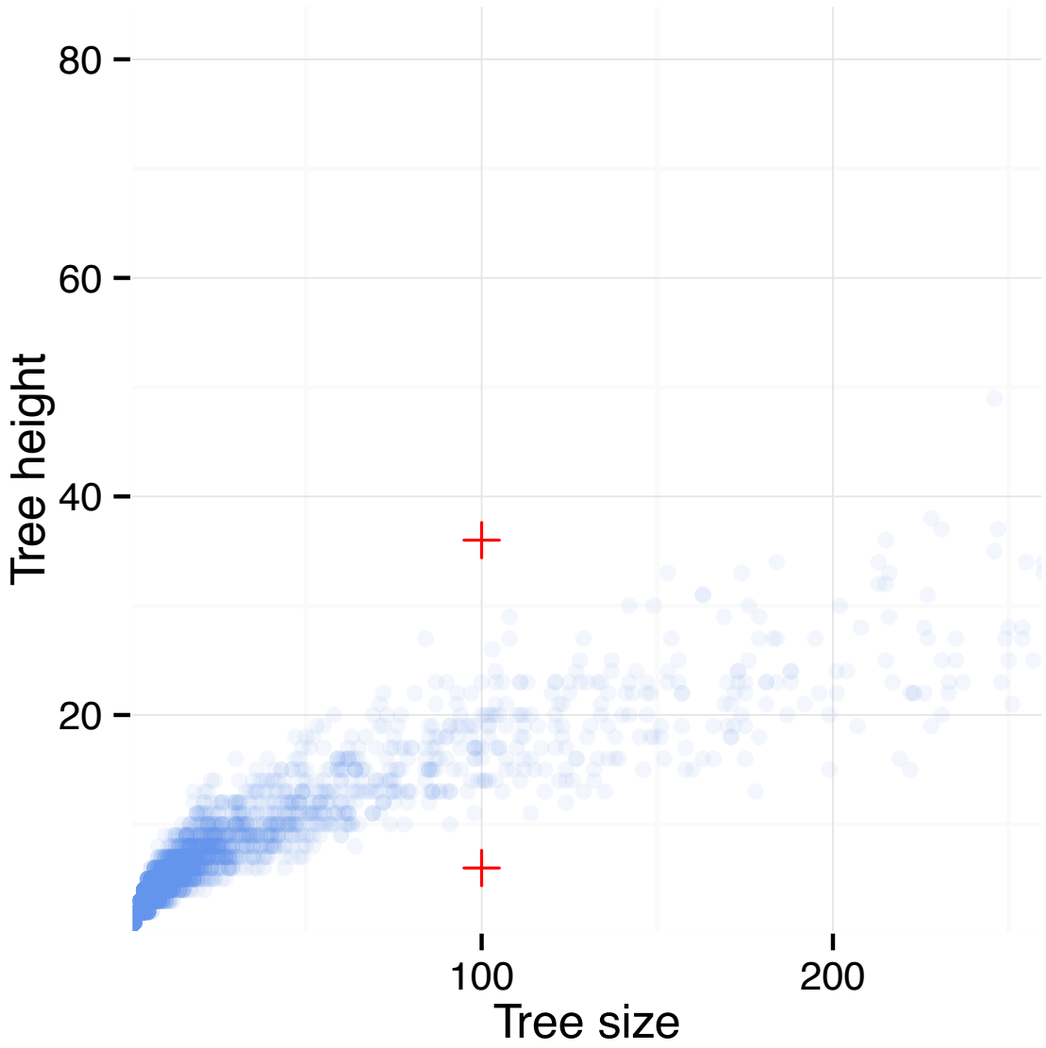
DataGenerators.jl design

Extracts a model of choice points from a non-deterministic generator; optimises the choice model using metaheuristic optimisation to met bias objectives

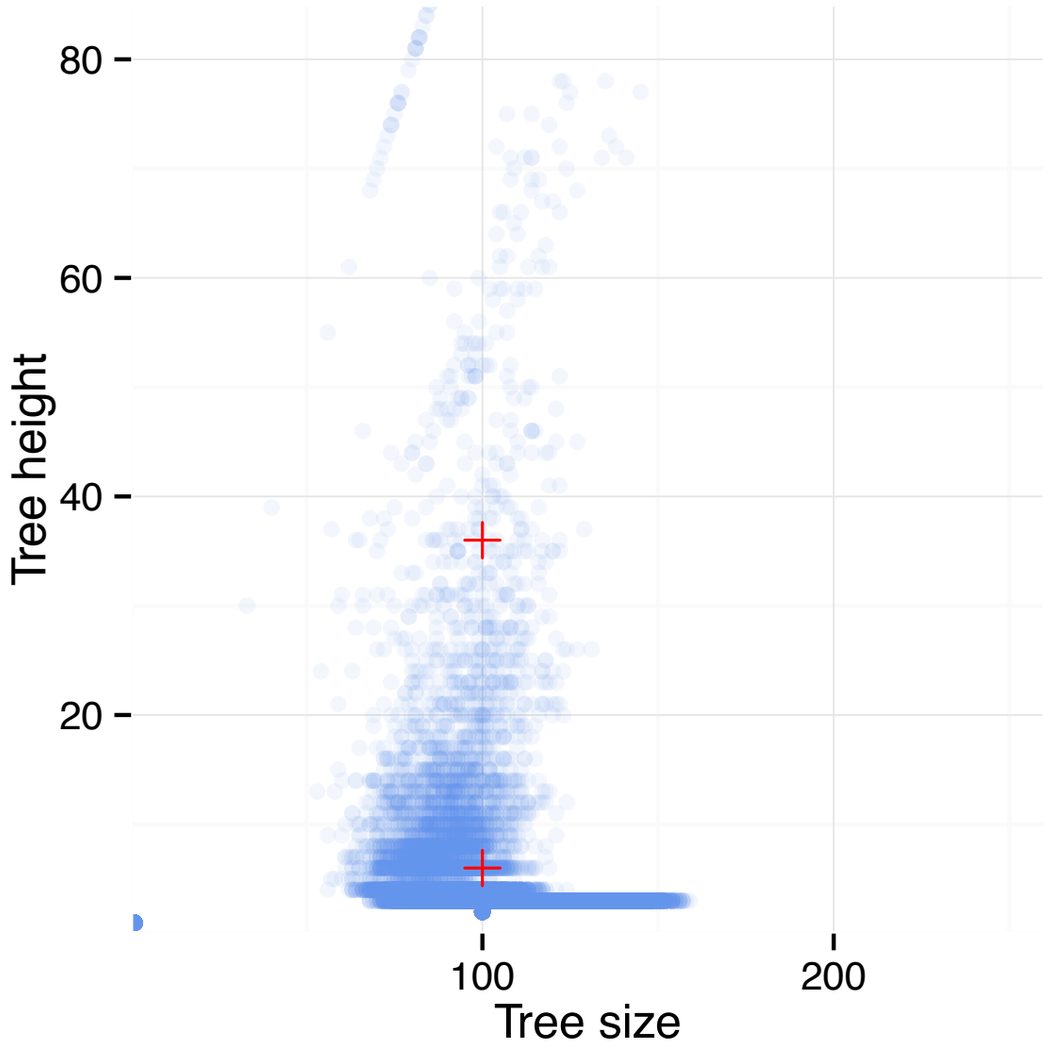


The Competition: Random “Searching” for specific trees

Scatter plots show the distribution of tree sizes and heights; target objectives are indicated by crosses



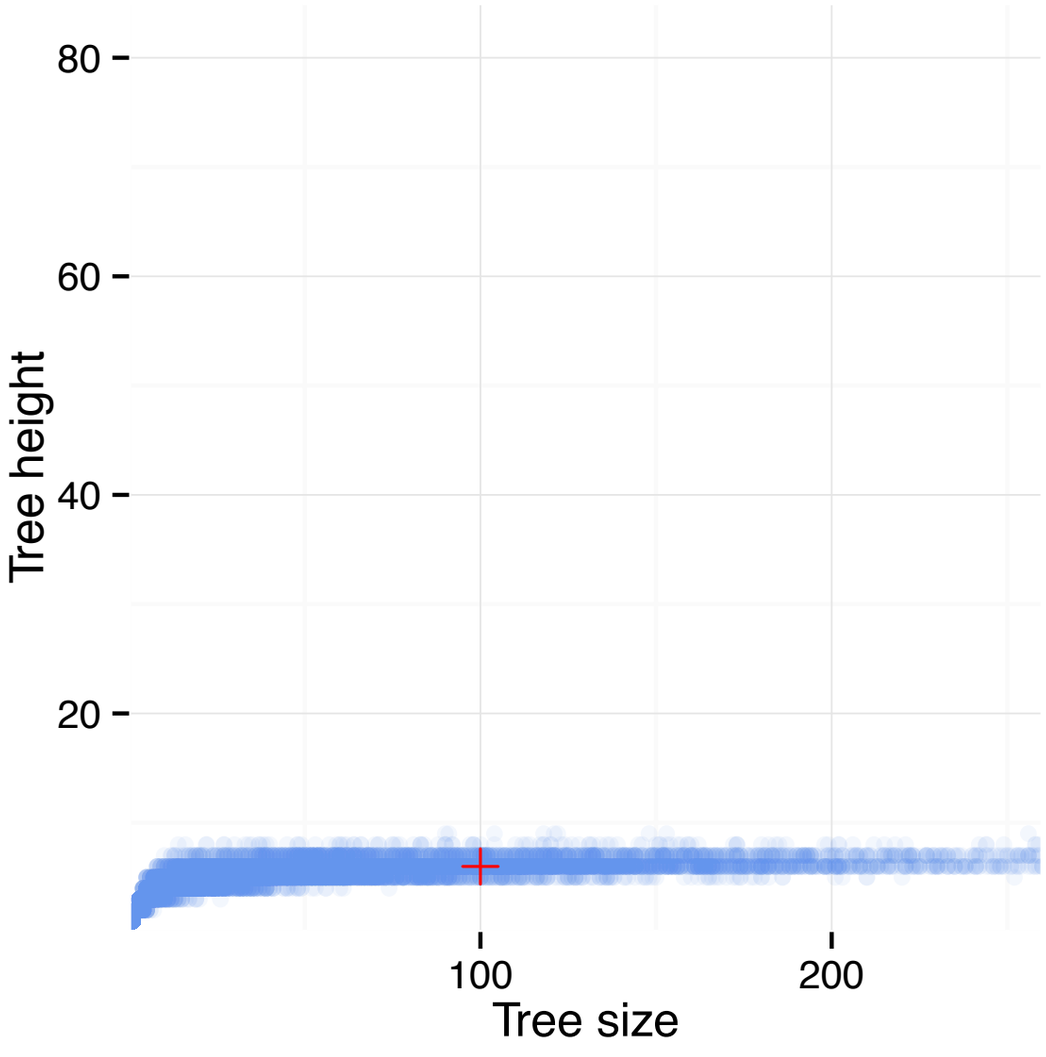
Boltzmann Sampler



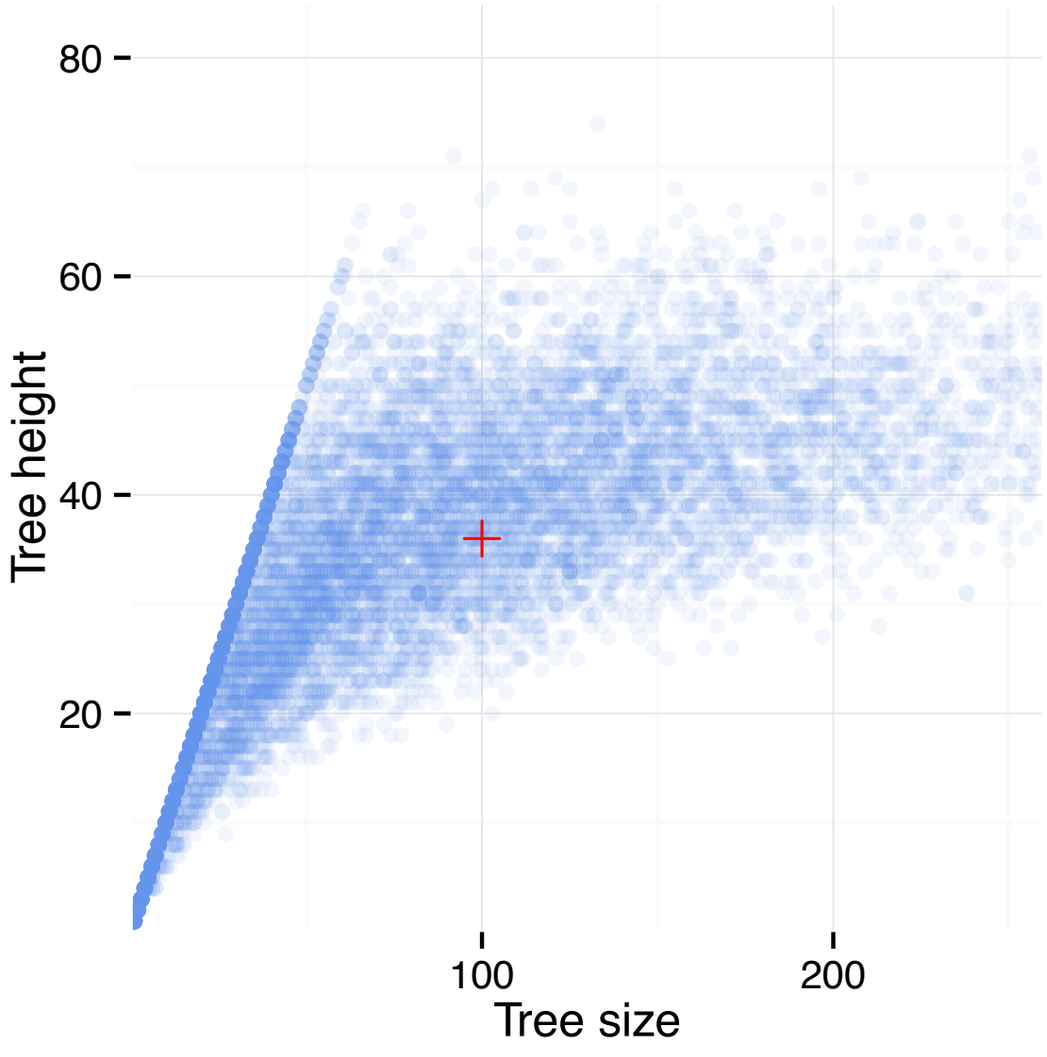
QuickCheck

DataGenerators: Searching for specific trees

Scatter plots show the distribution of tree sizes and heights; target objectives are indicated by crosses

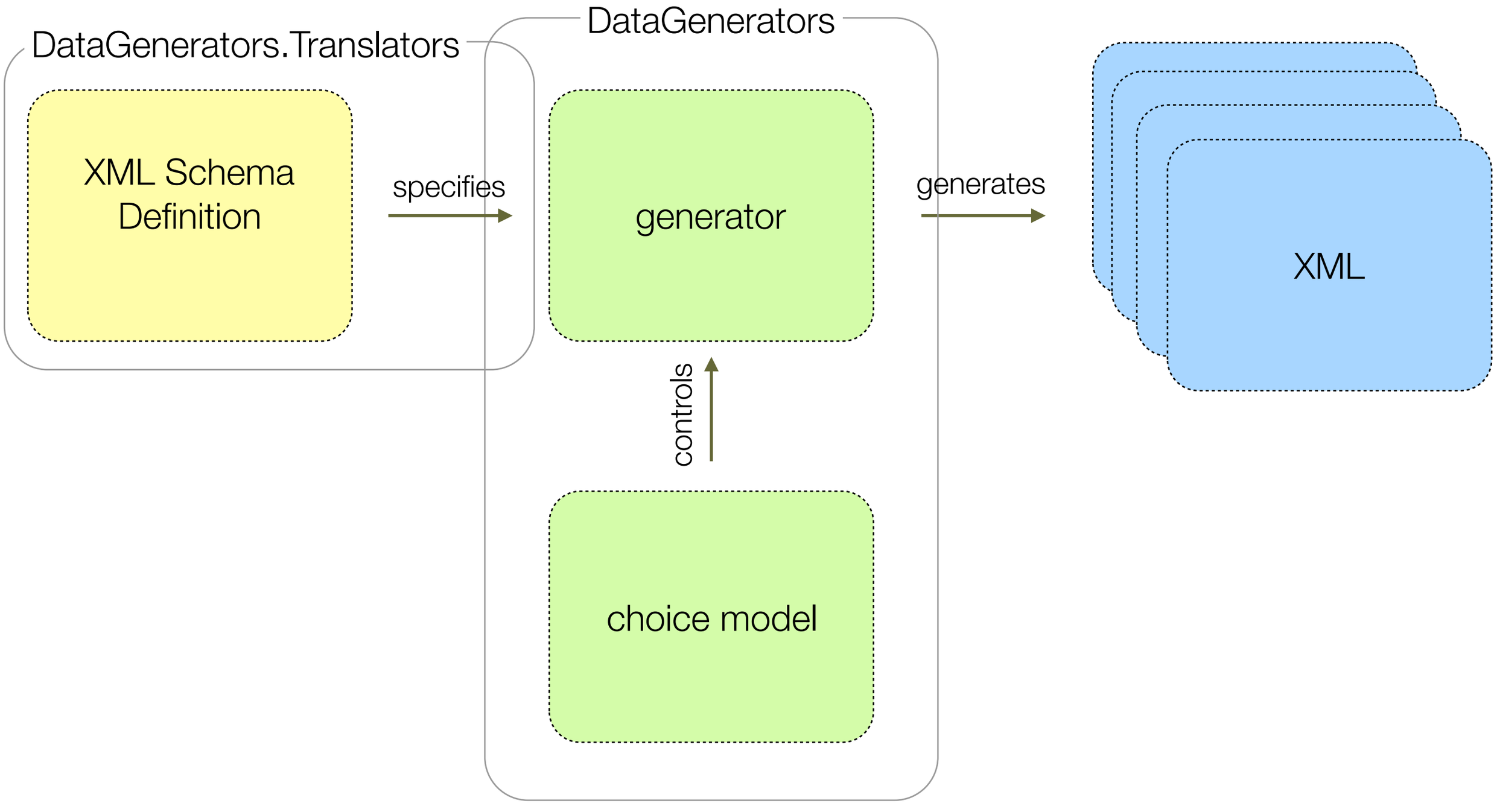


DataGenerators Target 1



DataGenerators Target 2

Automatically creating generators from specifications



Automatically creating generators from XSD

```
1  using DataGenerators.Translators
2
3  translate_from_xsd("mathml2.xsd","math", "MathMLGen","mathmlgen.jl")
4
5  include("mathmlgen.jl")
6
7  choose(MathMLGen)
8
```

DataMutators.jl

DataMutators.jl - Shrinking test data

```
#
# 1. Function under test: myrev
# (Contrived example with strange (seeded) bug)
#
#
# 2b. Basic random testing with a recursive array generator
#
# 3. Shrink the failing test datum
#
a = DataMutators.shrink(ary, props_myrev)

end
arraygen = ArrayOfMixedElements()
ary = first(filter(a -> !props_myrev(a), Any[choose(arraygen) for i in 1:100]))

end

```

Summary

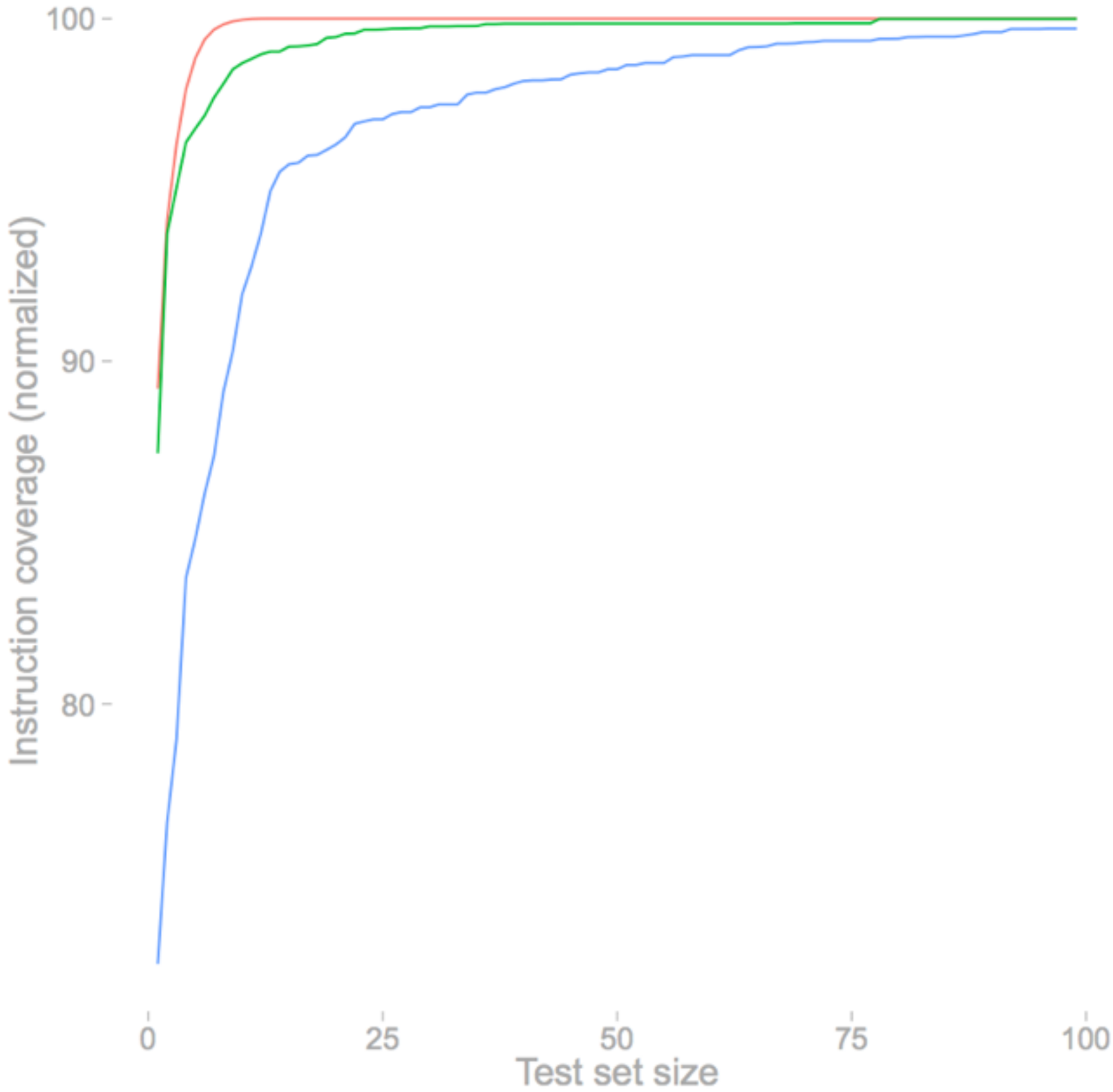
- Introduced 3 Julia packages suppoting automated testing
  - BaseTestAuto - add auto test support to Base.Test 0.5-dev
  - DataGenerators - generate/optimize structured data
  - DataMutators - shrink for understanding, grow for exploration
  - All 3 packages will be announced in the coming weeks
- Upcoming features
  - Web front end to continuous/auto testing (Escher?)
  - Reorder test cases based on code changes
  - Generate diverse test data, Integrate search-based testing
- We need your advice and input!
  - Should we also support FactCheck syntax?
  - Special testing needs for numerical code?
  - How extend Base.Test longer-term?

Extras as needed

Diverse Test Data: Normalised Compression Distance (NCD)

```
using Libz
compress(str) = readbytes(ZlibDeflateInputStream(takebuf_array(IOBuffer(str))))
C(str) = length(compress(str))
lexorder(strs) = join(sort(strs), "")
ncd(x, y, c = C) = ( c(lexorder([x, y])) - min(c(x), c(y)) ) / max(c(x), c(y))
```

Higher code coverage if selecting test data based on NCD



NCD for multisets (aka “bags”, “lists”, …)

$$\mathbf{NCD}_1(X) = \frac{C(X) - \min_{x \in X}\{C(x)\}}{\max_{x \in X}\{C(X \setminus \{x\})\}}$$

$$\mathbf{NCD}(X) = \max\left\{\mathbf{NCD}_1(X), \max_{Y \subset X}\{\mathbf{NCD}(Y)\}\right\}$$

The algorithm starts from the multiset $Y_0 = X = \{x_1, x_2, \ldots, x_n\}$, and proceeds as:

1) Find index i that maximizes $C(Y_k \setminus \{x_i\})$.
2) Let $Y_{k+1} = Y_k \setminus x_i$.
3) Repeat from step 1 until the subset contains only two strings.
4) Calculate $\mathbf{NCD}(X)$ as: $\max_{0 \le k \le n-2}\{\mathbf{NCD}_1(Y_k)\}$.