Generating Constrained Random Data with Uniform Distribution

Authors: Koen Claessen, Jonas Duregard, and Michał H. Pałka
Presented By: Segev Elazar Mittelman

Meet Randy the Random Tester

```
data Nat = Z | Suc Nat
```

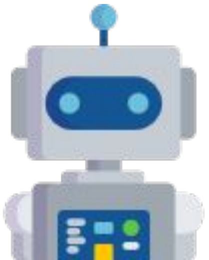
```
data ListNat = Nill | Cons Nat ListNat
```

```
sortedListNat :: ListNat -> Bool
```

I have a bunch of properties I want to test on the subset of values from my ADTs that satisfy predicates!

Good Random Testing Relies on Good Generators

Okay rad, so every time I have a new ADT to test I just write a new generator by hand! Right?



I mean you could, I guess... but you may want to reconsider.

Mr. Bot’s Reasons to Reconsider

- Writing a good generator is not trivial
  - Risk of bugs & potentially large time/effort investment in tuning generator
- Unknown value distribution ⇢? Counterexamples never generated
- New handwritten generator for each new precondition predicate?
  - Yes, sounds like fun! ⇢ Aight, see you in a month or two…
  - No, just filter values ⇢ And what if precondition is rare among values?...

All Lists of Naturals Using Exactly 17 Constructors

[[0,0,0,0,0,0,0,0],[0,0,0,0,0,0,2],[0,0,0,0,0,1,1],[0,0,0,0,0,2,0],[0,0,0,0,0,4],[0,0,0,0,1,0,1],[0,0,0,0,1,1,0],[0,0,0,0,1,3],[0,0,0,0,2,0,0],[0,0,0,0,2,2],[0,0,0,0,3,1],[0,0,0,0,4,0],[0,0,0,0,6
],[0,0,0,1,0,0,1],[0,0,0,1,0,1,0],[0,0,0,1,0,3],[0,0,0,1,1,0,0],[0,0,0,1,1,2],[0,0,0,1,2,1],[0,0,0,1,3,0],[0,0,0,1,5],[0,0,0,2,0,0,0],[0,0,0,2,0,2],[0,0,0,2,1,1],[0,0,0,2,2,0],[0,0,0,2,4],[0,0,0,
3,0,1],[0,0,0,3,1,0],[0,0,0,3,3],[0,0,0,4,0,0],[0,0,0,4,2],[0,0,0,5,1],[0,0,0,6,0],[0,0,0,8],[0,0,1,0,0,0,1],[0,0,1,0,0,1,0],[0,0,1,0,0,3],[0,0,1,0,1,0,0],[0,0,1,0,1,2],[0,0,1,0,2,1],[0,0,1,0,3,0
],[0,0,1,0,5],[0,0,1,1,0,0,0],[0,0,1,1,0,2],[0,0,1,1,1,1],[0,0,1,1,2,0],[0,0,1,1,4],[0,0,1,2,0,1],[0,0,1,2,1,0],[0,0,1,2,3],[0,0,1,3,0,0],[0,0,1,3,2],[0,0,1,4,1],[0,0,1,5,0],[0,0,1,7],[0,0,2,0,0,
0,0],[0,0,2,0,0,2],[0,0,2,0,1,1],[0,0,2,0,2,0],[0,0,2,0,4],[0,0,2,1,0,1],[0,0,2,1,1,0],[0,0,2,1,3],[0,0,2,2,0,0],[0,0,2,2,2],[0,0,2,3,1],[0,0,2,4,0],[0,0,2,6],[0,0,3,0,0,1],[0,0,3,0,1,0],[0,0,3,0
,3],[0,0,3,1,0,0],[0,0,3,1,2],[0,0,3,2,1],[0,0,3,3,0],[0,0,3,5],[0,0,4,0,0,0],[0,0,4,0,2],[0,0,4,1,1],[0,0,4,2,0],[0,0,4,4],[0,0,5,0,1],[0,0,5,1,0],[0,0,5,3],[0,0,6,0,0],[0,0,6,2],[0,0,7,1],[0,0,
8,0],[0,0,10],[0,1,0,0,0,0,1],[0,1,0,0,0,1,0],[0,1,0,0,0,3],[0,1,0,0,1,0,0],[0,1,0,0,1,2],[0,1,0,0,2,1],[0,1,0,0,3,0],[0,1,0,0,5],[0,1,0,1,0,0,0],[0,1,0,1,0,2],[0,1,0,1,1,1],[0,1,0,1,2,0],[0,1,0,
1,4],[0,1,0,2,0,1],[0,1,0,2,1,0],[0,1,0,2,3],[0,1,0,3,0,0],[0,1,0,3,2],[0,1,0,4,1],[0,1,0,5,0],[0,1,0,7],[0,1,1,0,0,0,0],[0,1,1,0,0,2],[0,1,1,0,1,1],[0,1,1,0,2,0],[0,1,1,0,4],[0,1,1,1,0,1],[0,1,1
,1,1,0],[0,1,1,1,3],[0,1,1,2,0,0],[0,1,1,2,2],[0,1,1,3,1],[0,1,1,4,0],[0,1,1,6],[0,1,2,0,0,1],[0,1,2,0,1,0],[0,1,2,0,3],[0,1,2,1,0,0],[0,1,2,1,2],[0,1,2,2,1],[0,1,2,3,0],[0,1,2,5],[0,1,3,0,0,0],[
0,1,3,0,2],[0,1,3,1,1],[0,1,3,2,0],[0,1,3,4],[0,1,4,0,1],[0,1,4,1,0],[0,1,4,3],[0,1,5,0,0],[0,1,5,2],[0,1,6,1],[0,1,7,0],[0,1,9],[0,2,0,0,0,0,0],[0,2,0,0,0,2],[0,2,0,0,1,1],[0,2,0,0,2,0],[0,2,0,0
,4],[0,2,0,1,0,1],[0,2,0,1,1,0],[0,2,0,1,3],[0,2,0,2,0,0],[0,2,0,2,2],[0,2,0,3,1],[0,2,0,4,0],[0,2,0,6],[0,2,1,0,0,1],[0,2,1,0,1,0],[0,2,1,0,3],[0,2,1,1,0,0],[0,2,1,1,2],[0,2,1,2,1],[0,2,1,3,0],[
0,2,1,5],[0,2,2,0,0,0],[0,2,2,0,2],[0,2,2,1,1],[0,2,2,2,0],[0,2,2,4],[0,2,3,0,1],[0,2,3,1,0],[0,2,3,3],[0,2,4,0,0],[0,2,4,2],[0,2,5,1],[0,2,6,0],[0,2,8],[0,3,0,0,0,1],[0,3,0,0,1,0],[0,3,0,0,3],[0
,3,0,1,0,0],[0,3,0,1,2],[0,3,0,2,1],[0,3,0,3,0],[0,3,0,5],[0,3,1,0,0,0],[0,3,1,0,2],[0,3,1,1,1],[0,3,1,2,0],[0,3,1,4],[0,3,2,0,1],[0,3,2,1,0],[0,3,2,3],[0,3,3,0,0],[0,3,3,2],[0,3,4,1],[0,3,5,0],[
0,3,7],[0,4,0,0,0,0],[0,4,0,0,2],[0,4,0,1,1],[0,4,0,2,0],[0,4,0,4],[0,4,1,0,1],[0,4,1,1,0],[0,4,1,3],[0,4,2,0,0],[0,4,2,2],[0,4,3,1],[0,4,4,0],[0,4,6],[0,5,0,0,1],[0,5,0,1,0],[0,5,0,3],[0,5,1,0,0
],[0,5,1,2],[0,5,2,1],[0,5,3,0],[0,5,5],[0,6,0,0,0],[0,6,0,2],[0,6,1,1],[0,6,2,0],[0,6,4],[0,7,0,1],[0,7,1,0],[0,7,3],[0,8,0,0],[0,8,2],[0,9,1],[0,10,0],[0,12],[1,0,0,0,0,0,1],[1,0,0,0,0,1,0],[1,
0,0,0,0,3],[1,0,0,0,1,0,0],[1,0,0,0,1,2],[1,0,0,0,2,1],[1,0,0,0,3,0],[1,0,0,0,5],[1,0,0,1,0,0,0],[1,0,0,1,0,2],[1,0,0,1,1,1],[1,0,0,1,2,0],[1,0,0,1,4],[1,0,0,2,0,1],[1,0,0,2,1,0],[1,0,0,2,3],[1,0
,0,3,0,0],[1,0,0,3,2],[1,0,0,4,1],[1,0,0,5,0],[1,0,0,7],[1,0,1,0,0,0,0],[1,0,1,0,0,2],[1,0,1,0,1,1],[1,0,1,0,2,0],[1,0,1,0,4],[1,0,1,1,0,1],[1,0,1,1,1,0],[1,0,1,1,3],[1,0,1,2,0,0],[1,0,1,2,2],[1,
0,1,3,1],[1,0,1,4,0],[1,0,1,6],[1,0,2,0,0,1],[1,0,2,0,1,0],[1,0,2,0,3],[1,0,2,1,0,0],[1,0,2,1,2],[1,0,2,2,1],[1,0,2,3,0],[1,0,2,5],[1,0,3,0,0,0],[1,0,3,0,2],[1,0,3,1,1],[1,0,3,2,0],[1,0,3,4],[1,0
,4,0,1],[1,0,4,1,0],[1,0,4,3],[1,0,5,0,0],[1,0,5,2],[1,0,6,1],[1,0,7,0],[1,0,9],[1,1,0,0,0,0,0],[1,1,0,0,0,2],[1,1,0,0,1,1],[1,1,0,0,2,0],[1,1,0,0,4],[1,1,0,1,0,1],[1,1,0,1,1,0],[1,1,0,1,3],[1,1,
0,2,0,0],[1,1,0,2,2],[1,1,0,3,1],[1,1,0,4,0],[1,1,0,6],[1,1,1,0,0,1],[1,1,1,0,1,0],[1,1,1,0,3],[1,1,1,1,0,0],[1,1,1,1,2],[1,1,1,2,1],[1,1,1,3,0],[1,1,1,5],[1,1,2,0,0,0],[1,1,2,0,2],[1,1,2,1,1],[1
,1,2,2,0],[1,1,2,4],[1,1,3,0,1],[1,1,3,1,0],[1,1,3,3],[1,1,4,0,0],[1,1,4,2],[1,1,5,1],[1,1,6,0],[1,1,8],[1,2,0,0,0,1],[1,2,0,0,1,0],[1,2,0,0,3],[1,2,0,1,0,0],[1,2,0,1,2],[1,2,0,2,1],[1,2,0,3,0],[
1,2,0,5],[1,2,1,0,0,0],[1,2,1,0,2],[1,2,1,1,1],[1,2,1,2,0],[1,2,1,4],[1,2,2,0,1],[1,2,2,1,0],[1,2,2,3],[1,2,3,0,0],[1,2,3,2],[1,2,4,1],[1,2,5,0],[1,2,7],[1,3,0,0,0,0],[1,3,0,0,2],[1,3,0,1,1],[1,3
,0,2,0],[1,3,0,4],[1,3,1,0,1],[1,3,1,1,0],[1,3,1,3],[1,3,2,0,0],[1,3,2,2],[1,3,3,1],[1,3,4,0],[1,3,6],[1,4,0,0,1],[1,4,0,1,0],[1,4,0,3],[1,4,1,0,0],[1,4,1,2],[1,4,2,1],[1,4,3,0],[1,4,5],[1,5,0,0,
0],[1,5,0,2],[1,5,1,1],[1,5,2,0],[1,5,4],[1,6,0,1],[1,6,1,0],[1,6,3],[1,7,0,0],[1,7,2],[1,8,1],[1,9,0],[1,11],[2,0,0,0,0,0,0],[2,0,0,0,0,2],[2,0,0,0,1,1],[2,0,0,0,2,0],[2,0,0,0,4],[2,0,0,1,0,1],[
2,0,0,1,1,0],[2,0,0,1,3],[2,0,0,2,0,0],[2,0,0,2,2],[2,0,0,3,1],[2,0,0,4,0],[2,0,0,6],[2,0,1,0,0,1],[2,0,1,0,1,0],[2,0,1,0,3],[2,0,1,1,0,0],[2,0,1,1,2],[2,0,1,2,1],[2,0,1,3,0],[2,0,1,5],[2,0,2,0,0
,0],[2,0,2,0,2],[2,0,2,1,1],[2,0,2,2,0],[2,0,2,4],[2,0,3,0,1],[2,0,3,1,0],[2,0,3,3],[2,0,4,0,0],[2,0,4,2],[2,0,5,1],[2,0,6,0],[2,0,8],[2,1,0,0,0,1],[2,1,0,0,1,0],[2,1,0,0,3],[2,1,0,1,0,0],[2,1,0,
1,2],[2,1,0,2,1],[2,1,0,3,0],[2,1,0,5],[2,1,1,0,0,0],[2,1,1,0,2],[2,1,1,1,1],[2,1,1,2,0],[2,1,1,4],[2,1,2,0,1],[2,1,2,1,0],[2,1,2,3],[2,1,3,0,0],[2,1,3,2],[2,1,4,1],[2,1,5,0],[2,1,7],[2,2,0,0,0,0
],[2,2,0,0,2],[2,2,0,1,1],[2,2,0,2,0],[2,2,0,4],[2,2,1,0,1],[2,2,1,1,0],[2,2,1,3],[2,2,2,0,0],[2,2,2,2],[2,2,3,1],[2,2,4,0],[2,2,6],[2,3,0,0,1],[2,3,0,1,0],[2,3,0,3],[2,3,1,0,0],[2,3,1,2],[2,3,2,
1],[2,3,3,0],[2,3,5],[2,4,0,0,0],[2,4,0,2],[2,4,1,1],[2,4,2,0],[2,4,4],[2,5,0,1],[2,5,1,0],[2,5,3],[2,6,0,0],[2,6,2],[2,7,1],[2,8,0],[2,10],[3,0,0,0,0,1],[3,0,0,0,1,0],[3,0,0,0,3],[3,0,0,1,0,0],[
3,0,0,1,2],[3,0,0,2,1],[3,0,0,3,0],[3,0,0,5],[3,0,1,0,0,0],[3,0,1,0,2],[3,0,1,1,1],[3,0,1,2,0],[3,0,1,4],[3,0,2,0,1],[3,0,2,1,0],[3,0,2,3],[3,0,3,0,0],[3,0,3,2],[3,0,4,1],[3,0,5,0],[3,0,7],[3,1,0
,0,0,0],[3,1,0,0,2],[3,1,0,1,1],[3,1,0,2,0],[3,1,0,4],[3,1,1,0,1],[3,1,1,1,0],[3,1,1,3],[3,1,2,0,0],[3,1,2,2],[3,1,3,1],[3,1,4,0],[3,1,6],[3,2,0,0,1],[3,2,0,1,0],[3,2,0,3],[3,2,1,0,0],[3,2,1,2],[
3,2,2,1],[3,2,3,0],[3,2,5],[3,3,0,0,0],[3,3,0,2],[3,3,1,1],[3,3,2,0],[3,3,4],[3,4,0,1],[3,4,1,0],[3,4,3],[3,5,0,0],[3,5,2],[3,6,1],[3,7,0],[3,9],[4,0,0,0,0,0],[4,0,0,0,2],[4,0,0,1,1],[4,0,0,2,0],
[4,0,0,4],[4,0,1,0,1],[4,0,1,1,0],[4,0,1,3],[4,0,2,0,0],[4,0,2,2],[4,0,3,1],[4,0,4,0],[4,0,6],[4,1,0,0,1],[4,1,0,1,0],[4,1,0,3],[4,1,1,0,0],[4,1,1,2],[4,1,2,1],[4,1,3,0],[4,1,5],[4,2,0,0,0],[4,2,
0,2],[4,2,1,1],[4,2,2,0],[4,2,4],[4,3,0,1],[4,3,1,0],[4,3,3],[4,4,0,0],[4,4,2],[4,5,1],[4,6,0],[4,8],[5,0,0,0,1],[5,0,0,1,0],[5,0,0,3],[5,0,1,0,0],[5,0,1,2],[5,0,2,1],[5,0,3,0],[5,0,5],[5,1,0,0,0
],[5,1,0,2],[5,1,1,1],[5,1,2,0],[5,1,4],[5,2,0,1],[5,2,1,0],[5,2,3],[5,3,0,0],[5,3,2],[5,4,1],[5,5,0],[5,7],[6,0,0,0,0],[6,0,0,2],[6,0,1,1],[6,0,2,0],[6,0,4],[6,1,0,1],[6,1,1,0],[6,1,3],[6,2,0,0]
,[6,2,2],[6,3,1],[6,4,0],[6,6],[7,0,0,1],[7,0,1,0],[7,0,3],[7,1,0,0],[7,1,2],[7,2,1],[7,3,0],[7,5],[8,0,0,0],[8,0,2],[8,1,1],[8,2,0],[8,4],[9,0,1],[9,1,0],[9,3],[10,0,0],[10,2],[11,1],[12,0],[14]

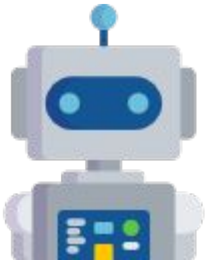
All ***SORTED*** Lists of Naturals Using Exactly 17 Constructors

```
[[0,0,0,0,0,0,0,0],[0,0,0,0,0,0,2],[0,0,0,0,0,1,1],[0,0,0,0,0,4],[0,0,0,0,1,3],[0,0,0,0,2,2],[0,0,0,0,6],[0,0,0,1,1,2],[0,0,0,1,5],[0,0,0,2,4],[0,0,0,3,3],[0,0,0,8],[0,0
,1,1,1,1],[0,0,1,1,4],[0,0,1,2,3],[0,0,1,7],[0,0,2,2,2],[0,0,2,6],[0,0,3,5],[0,0,4,4],[0,0,10],[0,1,1,1,3],[0,1,1,2,2],[0,1,1,6],[0,1,2,5],[0,1,3,4],[0,1,9],[0,2,2,4],[0
,2,3,3],[0,2,8],[0,3,7],[0,4,6],[0,5,5],[0,12],[1,1,1,1,2],[1,1,1,5],[1,1,2,4],[1,1,3,3],[1,1,8],[1,2,2,3],[1,2,7],[1,3,6],[1,4,5],[1,11],[2,2,2,2],[2,2,6],[2,3,5],[2,4,
4],[2,10],[3,3,4],[3,9],[4,8],[5,7],[6,6],[14]]
```

"You've got to ask yourself one question. Do I feel lucky? Well, do ya, [Randy]?" ~ Dirty Harry (1971)

Oh okay, that sounds kinda rough to deal with now that you say.

Glad you agree, let me show you an alternative.



Solution	Implementation
Don’t handwrite, instead derive generators from data definition.	Use common structures in ADTs to define Spaces of generated values.
Give all derived generators uniform distributions.	Convert Sized Spaces to Finite Sets and recursively index with naturals.
Don’t filter on predicate, find general subsets that all fail predicate and prune them.	Use Haskell’s Laziness to specialize indexed value step by step until predicate always true or always false.

Extract ADT Essence

```haskell
data Space a where
    Empty :: Space a
    Pure :: a -> Space a
    (:+:) :: Space a -> Space a -> Space a
    (:*:) :: Space a1 -> Space a2 -> Space (a1,a2)
    Pay :: Space a -> Space a
    (:$:) :: (a1 -> a2) -> Space a1 -> Space a2
```

```haskell
(<*>) :: Space (a -> b) -> Space a -> Space b
s1 <*> s2 = (\(f, a) -> (f a)) :$: (s1 :*: s2)
```

The Space of the Nat ADT

```haskell
data Nat = Z | Suc Nat
```

```haskell
spaceNat :: Space Nat
spaceNat = Pay (Pure Z :+: (Suc :$: spaceNat))
```

The Space of the ListNat ADT

```
data ListNat = Nill | Cons Nat ListNat
```

```
spaceListNat :: Space ListNat
spaceListNat = Pay (Pure Nill :+: (Cons :$: spaceNat Main.<*> spaceListNat))
```

The Space of the Tree ADT

```
data Tree = Leaf | Node Nat Tree Tree
```

Let's try to figure it out together.

The Space of the Tree ADT

```haskell
data Tree = Leaf | Node Nat Tree Tree
```

```haskell
spaceTree :: Space Tree
spaceTree = Pay (Pure Leaf :+: (Node :$: spaceNat Main.<*> spaceTree Main.<*> spaceTree))
```

Recursive Structure of FinSet

```haskell
data FinSet a where
    EmptySet :: FinSet a
    Single :: a -> FinSet a
    Product :: FinSet a1 -> FinSet a2 -> FinSet (a1, a2)
    Union :: FinSet a -> FinSet a -> FinSet a
    Apply :: (a1 -> a2) -> FinSet a1 -> FinSet a2
```

Measuring FinSet Cardinality

```
finSetCardinality :: FinSet a -> Integer
finSetCardinality EmptySet = 0
finSetCardinality (Single _) = 1
finSetCardinality (Product finSetA finSetB) =
    finSetCardinality finSetA * finSetCardinality finSetB
finSetCardinality (Union finSetA1 finSetA2) =
    finSetCardinality finSetA1 + finSetCardinality finSetA2
finSetCardinality (Apply _ finSet) = finSetCardinality finSet
```

Example FinSet

$\{ \text{Suc}\ x \mid x \in \{0, 1, 2\} \} \times \{A, B\}$

```haskell
example1 :: FinSet (Integer, Char)
example1 = Product
      (Apply Suc (Union (Single 0) (Union (Single 1) (Single 2))))
      (Union (Single 'A') (Union (Single 'B') EmptySet))
```

Show That Cardinality Is 6

Indexing Uniformly into FinSets

```haskell
indexFin :: FinSet a -> Integer -> Maybe a
indexFin EmptySet _ = Nothing
indexFin (Single a) 0 = Just a
indexFin (Single _) _ = Nothing
indexFin (Union fsa _) i | i < finSetCardinality fsa = indexFin fsa i
indexFin (Union fsa fsb) i = indexFin fsb (i - finSetCardinality fsa)
indexFin (Product fsa fsb) i = do
    fst <- indexFin fsa (i `div` finSetCardinality fsb)
    snd <- indexFin fsb (i `mod` finSetCardinality fsb)
    return (fst, snd)
indexFin (Apply f finSet) i = do
    val <- indexFin finSet i
    return (f val)
```

From Sized Spaces to FinSets

```haskell
sized :: Space a -> Integer -> FinSet a
sized Empty _ = EmptySet
sized (Pure a) 0 = Single a
sized (Pure _) _ = EmptySet
sized (Pay _) 0 = EmptySet
sized (Pay a) k = sized a (k - 1)
sized (a :+: b) k = Union (sized a k) (sized b k)
sized (f :$: a) k = Apply f (sized a k)
sized (a :*: b) k = sizedHelper (a :*: b) 0 k
    where
        sizedHelper :: Space a -> Integer -> Integer -> FinSet a
        sizedHelper (a :*: b) k1 k | k1 <= k =
            Union (Product (sized a k1) (sized b (k - k1))) (sizedHelper (a :*: b) (k1 + 1) k)
        sizedHelper _ _ _ = EmptySet
```

Uniformly Indexing Into ADT Spaces

```haskell
indexSized :: Space a -> Integer -> Integer -> Maybe a
indexSized space size index = indexFin (sized space size) index
```

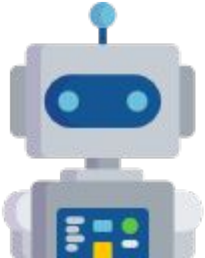
Let’s try evaluating: `indexSized spaceNat 2 0`

Let’s try this out for more interesting examples.

It’s demo time!

Sweet! But what about those PREDICATES!?!

We’ll get to them, but first, my friend, we have to learn how to get laaaaaazy.



Laziness/Call By Need Evaluation

Definition: Terms are only evaluated when needed, and only needed portions of terms are evaluated, leaving remainder of term unevaluated.

Examples:

```
const (2 + 5) undefined == 7
```

```
isS :: Nat -> Bool
isS Z = False
isS (S _) = True

isS (S undefined) == True
```

```
let x = 1 : x in
    tail x

    --vs.--

let x = 1 : x in
    length x
```

Idea: If we define predicates lazily, we can find entire sets of predicate fulfilling or failing values instead of singular values

```haskell
valid :: (a -> Bool) -> Maybe Bool
valid p | crashes (p undefined) = Nothing
valid P = Some (p undefined)
```

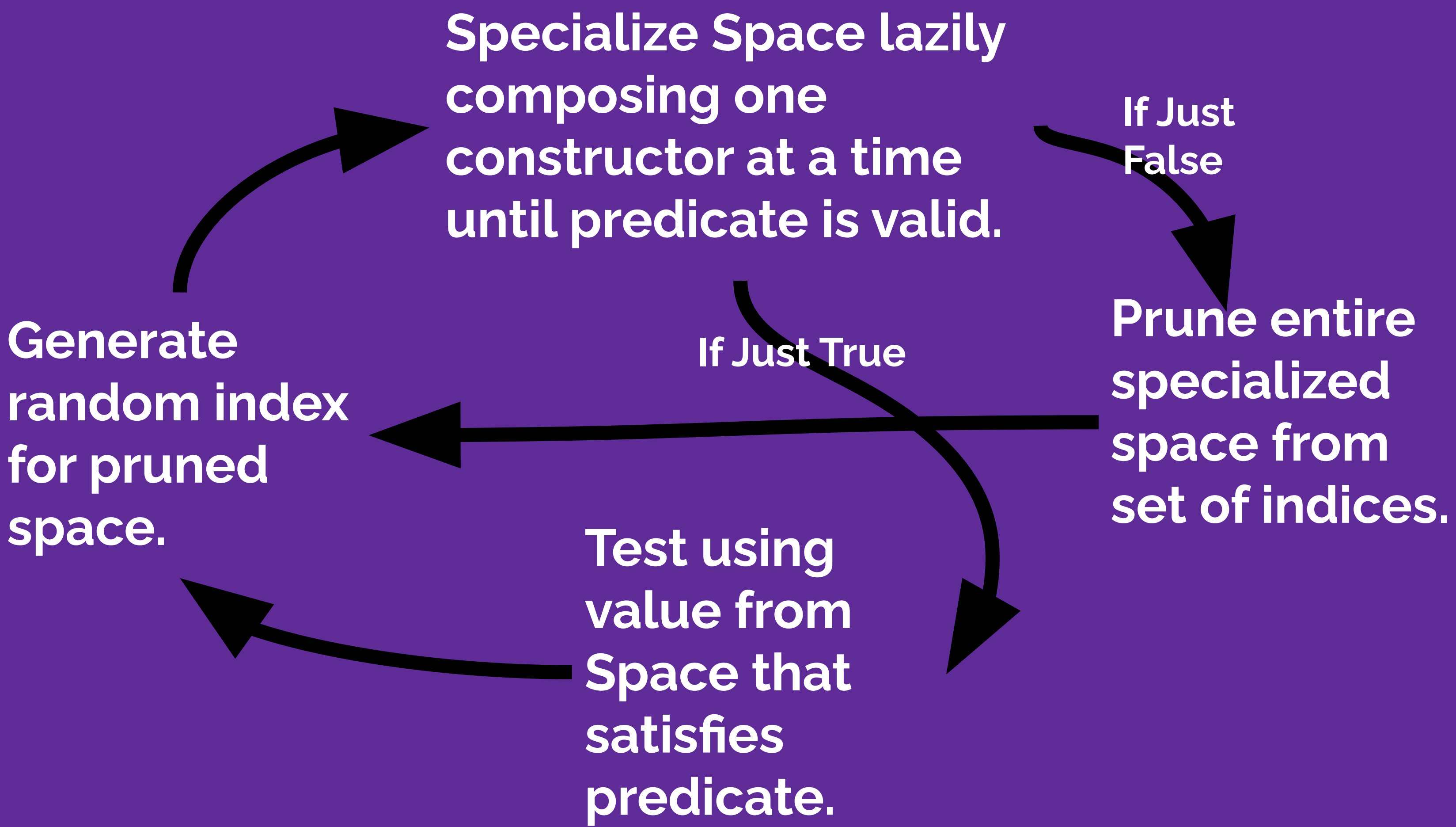
```haskell
valid isS == Nothing --needs to inspect input so fails
valid (isS . S) == Just True --(isS . S) n == True for all naturals n.
```

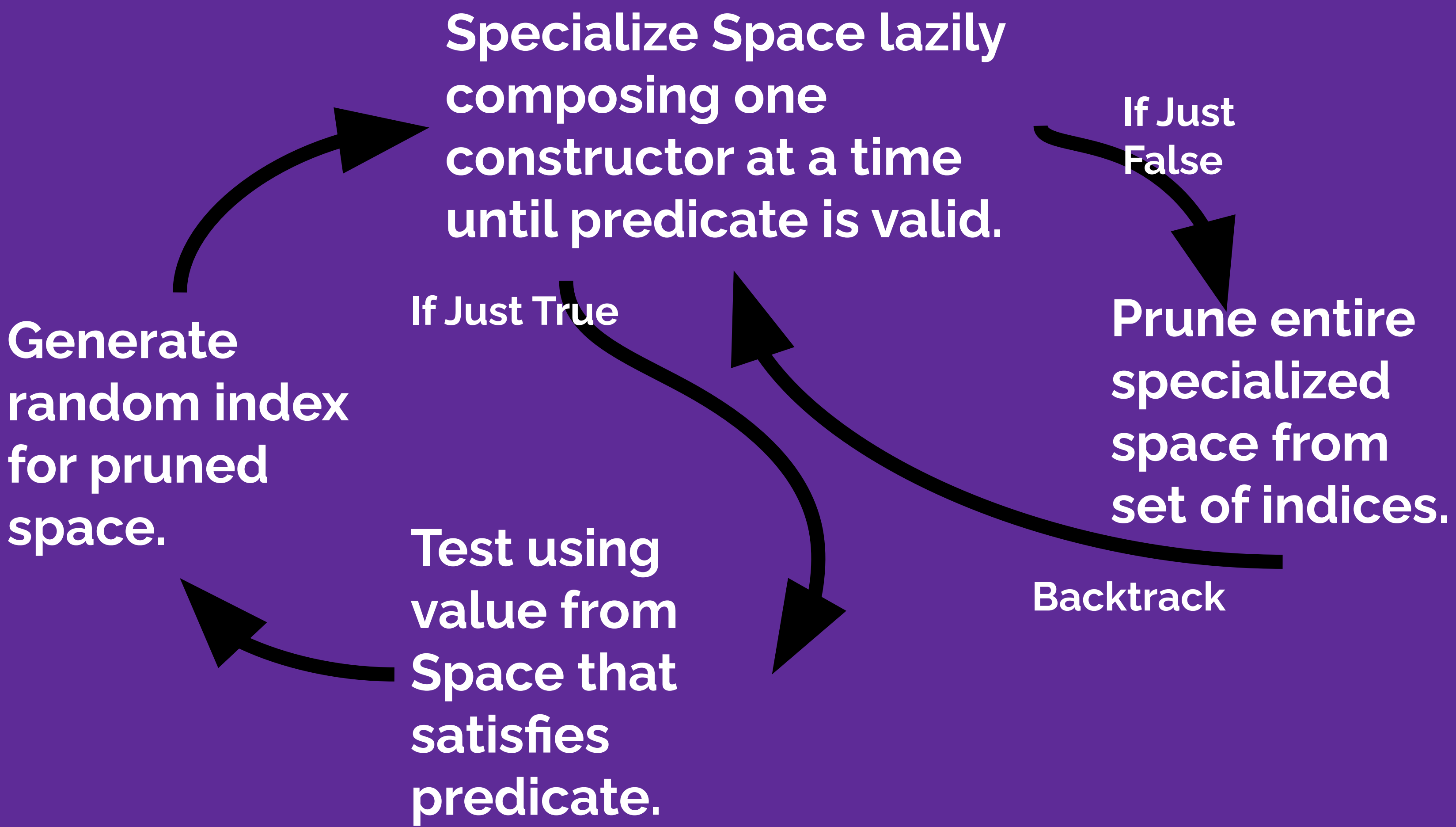
Lazy Predicate-Guided Indexing

```
index :: (a -> Bool) -> Space a -> Int -> Integer -> Space a
index predicate (constructor :$: space') size index = case valid predicate' of
    Just _ -> constructor :$: space'
    Nothing -> constructor :$: index predicate' space' size index
    where predicate' = predicate . constructor
```

```
index isS (Suc :$: spaceNat) s 0 == Suc :$: spaceNat

index is2S (Suc :$: spaceNat) s 0 == ...
    Suc :$: index (is2S . Suc) (Suc :$: spaceNat) s' 0 ==
        Suc :$: (Suc :$: spaceNat)
```

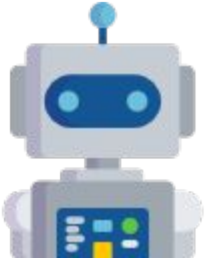

Specialize Space lazily composing one constructor at a time until predicate is valid.
If Just False
Prune entire specialized space from set of indices.
If Just True
Generate random index for pruned space.
Test using value from Space that satisfies predicate.


Specialize Space lazily composing one constructor at a time until predicate is valid.
If Just False
Prune entire specialized space from set of indices.
Backtrack
If Just True
Test using value from Space that satisfies predicate.
Generate random index for pruned space.

Let’s try to see how effective lazy pruning is.

It’s demo time!

Awesome, now does anyone have questions?



I get it now Mr. Bot, thanks a bunch!

Thank you for listening, any questions?