Generating Constrained Random Data with Uniform Distribution

Koen Claessen Jonas Duregård Michał Pałka

Chalmers University of Technology

Our goal

We have:

- **data** $A = \ldots$
- $\mathit{predicate} :: A \rightarrow \mathit{Bool}$
- A desired size (number of construcors used)

We want to:

- Randomly generate values that satisfy *predicate*

Our goal

We have:

- **data** $A = \ldots$
- $predicate :: A \rightarrow Bool$
- A desired size (number of construcors used)

We want to:

- Randomly generate values that satisfy *predicate*
- With uniform probability distribution

Examples

Examples:

- $ordered :: [Int] \rightarrow Bool$
  To generate ordered lists
- $(typeCheck\ TInt) :: Exp \rightarrow Bool$
  To generate expressions of type Int

Application: Property Based Testing

$$insert :: Int \to [\,Int\,] \to [\,Int\,]$$

$$ordered :: [\,Int\,] \to Bool$$

$$prop\_ins :: Int \to [\,Int\,] \to Bool$$
$$prop\_ins\ x\ xs = ordered\ xs \implies ordered\ (insert\ x\ xs)$$

```
*Main> quickCheck prop_ins
+++ OK, passed 100 tests.
```

- Most tests pass without executing tested code
- Solution: generate only ordered lists

Hand written data generators are ...

- ... complicated to write
    - A generator for ordered lists is more complex than the *insert* function it tests
    - Expensive
    - Generators may contain errors

Hand written data generators are ...

- ... complicated to write
  - A generator for ordered lists is more complex than the *insert* function it tests
  - Expensive
  - Generators may contain errors
- ... not compositional

$$
\begin{aligned}
&\mathit{ordered}, \mathit{nonempty} :: \mathit{Gen}\,[\mathit{Int}]\\
&\mathit{orderedNonempty} \;:: \mathit{Gen}\,[\mathit{Int}]\\
&\mathit{orderedNonempty} = ? \quad \text{-- We start from scratch here}
\end{aligned}
$$

Random distribution

$$ordered :: Gen\,[\,Int\,]$$

- What is the probability of generating [1,2,3]?
- Do all ordered lists have a positive probability?
- What about a more complicated generator, like type correct lambda terms?

Our Method

- Start with a finite set of values and a predicate p
- Choose a value x uniformly at random
- If $p\ x \Rightarrow \mathit{True}$, we are done
- Else, exclude x from the set and any *similar* values
  'Similar' means p does not distinguish between them

Our Method

- Start with a finite set of values and a predicate p
- Choose a value x uniformly at random
- If $p\ x \Rightarrow \mathit{True}$, we are done
- Else, exclude x from the set and any *similar* values
  'Similar' means p does not distinguish between them

x and y are similar if there exists a partial value z s.t.

$p\ z \Rightarrow \mathit{False}$
$z \sqsubseteq x$ -- ("z is x with some parts undefined")
$z \sqsubseteq y$

Example: Sorted lists

- Predicate: $\mathit{ordered} :: [\mathit{Int}] \to \mathit{Bool}$
- Start with all lists of a given size
- Random list: $[2, 1, 3, 3]$ falsifies predicate
- Exclude all lists starting with 2,1
  because $\mathit{ordered}\ (2 : 1 : \bot) \Rightarrow \mathit{False}$

Example

Generating typed lambda terms

$$\begin{aligned}\textbf{data}\ E\ v = {} & Lam\ (E\ (Maybe\ v)) \\ \mid {} & App\ (E\ v)\ (E\ v) \\ \mid {} & Var\ \ v\end{aligned}$$

Example

$\textbf{data}\ \mathit{Void} \quad$ -- The empty type
$\textbf{type}\ \mathit{Closed} = E\ \mathit{Void}$

$\textbf{data}\ \mathit{Type} = \mathit{Int} \mid \mathit{Type} \mapsto \mathit{Type} \mid \ldots$

$\mathit{typeCheck} :: \mathit{Type} \to \mathit{Closed} \to \mathit{Bool}$

$p :: \mathit{Closed} \to \mathit{Bool}$
$p = \mathit{typeCheck}\ (\mathit{Int} \mapsto \mathit{Int})$

Generic datatype representation

$$T = 1 \mid T \oplus T \mid T \otimes T \mid Con\ T$$

Generic datatype representation

$$T = 1 \mid T \oplus T \mid T \otimes T \mid Con\ T$$

$$\begin{aligned}
\textbf{data}\ E\ v = {} & Lam\ (E\ (Maybe\ v)) \\
\mid {} & App\ (E\ v)\ (E\ v) \\
\mid {} & Var\ \ v
\end{aligned}$$

$$E_v = Lam\ E_{v \oplus 1} \oplus App\ (E_v \otimes E_v) \oplus Var\ v$$

Cardinalities

$|T|_k$ is the number of values in T with k constructors

$$\begin{aligned} |1|_0 &= 1 \\ |1|_{k+1} &= 0 \end{aligned}$$

Cardinalities

$|T|_k$ is the number of values in T with k constructors

$$\begin{aligned} |1|_0 &= 1 \\ |1|_{k+1} &= 0 \end{aligned}$$

$$|A \oplus B|_k = |A|_k + |B|_k$$

Cardinalities

$|T|_k$ is the number of values in T with k constructors

$$\begin{aligned} |1|_0 &= 1 \\ |1|_{k+1} &= 0 \end{aligned}$$

$$|A \oplus B|_k = |A|_k + |B|_k$$

$$|A \otimes B|_k = \sum_{a+b=k} |A|_a * |B|_b$$

Cardinalities

$|T|_k$ is the number of values in T with k constructors

$$\begin{aligned} |1|_0 &= 1 \\ |1|_{k+1} &= 0 \end{aligned}$$

$$|A \oplus B|_k = |A|_k + |B|_k$$

$$|A \otimes B|_k = \sum_{a+b=k} |A|_a * |B|_b$$

$$\begin{aligned} |\langle Con \rangle\, A|_0 &= 0 \\ |\langle Con \rangle\, A|_{k+1} &= |A|_k \end{aligned}$$

Cardinalities

$|T|_k$ is the number of values in T with k constructors

$$\begin{aligned} |1|_0 &= 1 \\ |1|_{k+1} &= 0 \end{aligned}$$

$$|A \oplus B|_k = |A|_k + |B|_k$$

$$|A \otimes B|_k = \sum_{a+b=k} |A|_a * |B|_b$$

$$\begin{aligned} |\langle Con \rangle\, A|_0 &= 0 \\ |\langle Con \rangle\, A|_{k+1} &= |A|_k \end{aligned}$$

$$\begin{aligned} |E_v|_0 &= 0 \\ |E_v|_{k+1} &= |E_{v\oplus 1}|_k + |E_v \otimes E_v|_k + |v|_k \end{aligned}$$

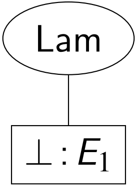
- Generating Expressions of type $Int \to Int$ and size 8
- $|E_0|_8 = 506$ (Only some of which are $Int \to Int$)

$$\boxed{\perp : E_0}$$

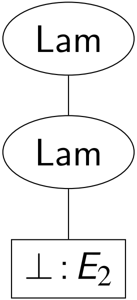
- Start with $\perp$, note that $p \perp \Rightarrow \perp$
- Probability P(Lam) of starting with a lambda = #values with head lambda / total #values

$$\boxed{\bot : E_0}$$

- Start with $\bot$, note that $p \bot \Rightarrow \bot$
- Probability P(Lam) of starting with a lambda = #values with head lambda / total #values
- $|Lam\ E_1|_8 = 464$, $P(Lam) = 0.92$
- $|App\ (E_0 \otimes E_0)|_8 = 42$, $P(App) = 0.08$
- $|Var\ 0|_8 = 0$, $P(Var) = 0$



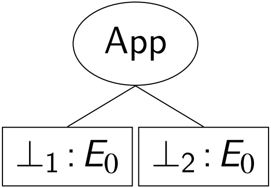
- $P(Lam) = 0.7$
- $P(App) = 0.3$
- $P(Var) = 0$



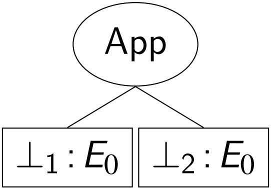
- Predicate fails $(p\,(Lam\,(Lam\,\perp)) \Rightarrow False)$
- Backtracking points:
  $App\,(E_0 \otimes E_0)$
  $Lam\,(App\,(E_1 \otimes E_1))$

$$\boxed{\perp : E'}$$

- New set: $E' = App\,(E_0 \otimes E_0) \oplus Lam\,(App\,(E_1 \otimes E_1))$
  $|E'|_8 = 182$
- New probabilities
- $|App\,(E_0 \otimes E_0)|_8 = 42,\ P(App) = 0.23$
- $|Lam\,(App\,(E_1 \otimes E_1))|_8 = 140,\ P(Lam/App) = 0.77$



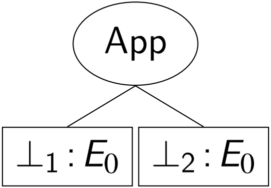
Should $\perp_1$ or $\perp_2$ be expanded next?



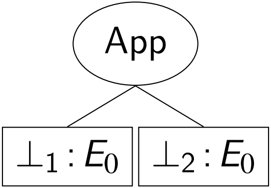
Should $\perp_1$ or $\perp_2$ be expanded next?

$$p\,(App\ \perp_1\ \perp_2) \Rightarrow \perp_1$$

So $\perp_1$ is expanded first



What is the probability of a lambda in $\perp_1$?

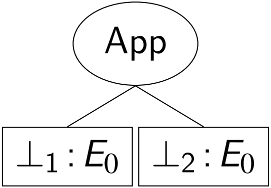


What is the probability of a lambda in $\perp_1$?

$$E_0 \otimes E_0$$

$$= \text{(definition of } E_0\text{)}$$

$$(Lam\ E_1 \oplus App\ (E_0 \otimes E_0)) \quad \otimes \quad E_0$$



What is the probability of a lambda in $\perp_1$?

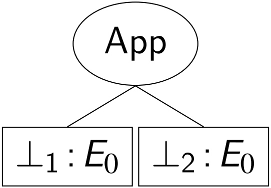
$$E_0 \otimes E_0$$

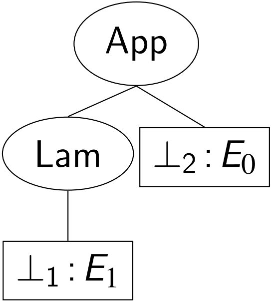
$$= \text{(definition of } E_0\text{)}$$

$$(Lam\ E_1 \oplus App\ (E_0 \otimes E_0)) \quad \otimes \quad E_0$$

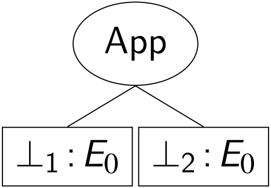
$$= \text{(distributivity)}$$

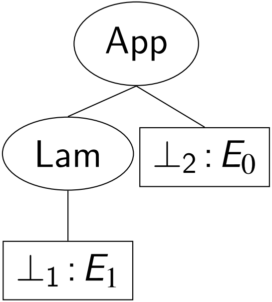
$$(Lam\ E_1 \otimes E_0) \quad \oplus \quad (App\ (E_0 \otimes E_0) \otimes E_0)$$



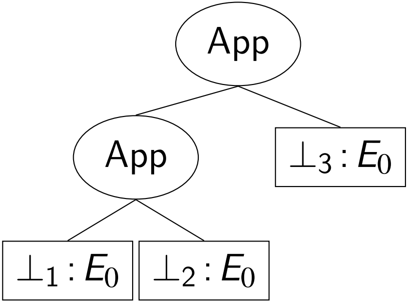


$|E_1 \otimes E_0|_6 = 41,$
$P(Lam) = 0.98$

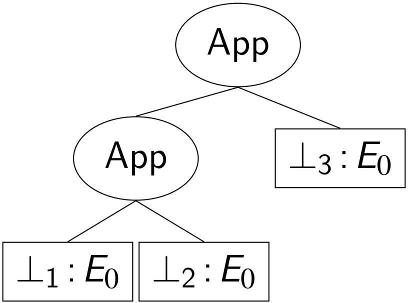




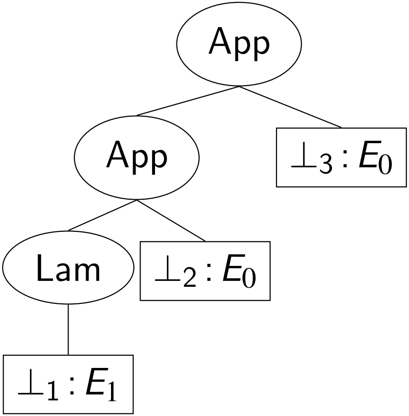
$|E_1 \otimes E_0|_6 = 41,$
$P(Lam) = 0.98$



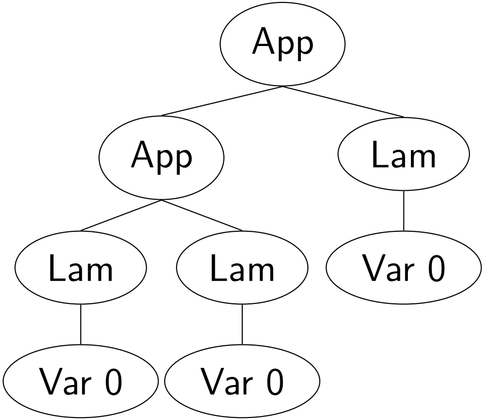
$|(E_0 \otimes E_0) \otimes E_0|_6 = 1,$
$P(App) = 0.02$



- Expanding $\perp_1$
- $|Lam\ E_1 \otimes (E_0 \otimes E_0)|_6 = 1,\ P(Lam) = 1$
- $|App\ (E_0 \otimes E_0) \otimes (E_0 \otimes E_0)|_6 = 0,\ P(App) = 0$



- Expanding $\perp_1$
- $|Var\ 1 \otimes (E_1 \otimes E_1)|_3 = |1 \otimes E_1 \otimes E_1|_2 = 1$, $P(Var\ 0) = 1$



$$((\lambda x \to x)\,(\lambda x \to x))\,(\lambda x \to x)$$

Limitations

- Relies on predicates being lazy (in the negative case)
- Memory usage is often an issue

What else is in the paper?

- Some evidence that this is practically useful: Generating Lamda terms rediscovered bugs in GHC
- Memory and speed benchmarks
- Modified algorithms that improve performance at the expense of introducing a (predictable) bias