Formalising Luck
Improved Probabilistic Semantics for Property-Based Generators

Diane Gallois-Wong
supervised by Cătălin Hrițcu, INRIA Paris

September 5th 2016
**Property-Based Testing**: running a program on many random inputs to test a property about it (QuickCheck)

Testing insertion function for binary search trees $\text{insertBST}$:

$$\forall (tr : \text{int tree})(x : \text{int}). \text{isBST}(tr) \Rightarrow \text{isBST}(\text{insertBST} \ tr \ x)$$
Property-Based Generators

\[\forall (tr : int\ tree)(x : int).\ isBST(tr) \Rightarrow isBST(insertBST\ tr\ x)\]

Problem: \(isBST\) is sparse, most of the tests are irrelevant

Solution: Property-Based Generators (PBGs)

Problem: hard to implement, easily unbalanced/incomplete (e.g. generate almost only small trees)

**Luck:** programming language dedicated to writing PBGs
Luck: principle

Property-Based Generator in Luck:

- Form of the program: predicate (boolean function) representing the property
- Running it: give unknowns as arguments, generate valuation $\forall$ for them so that predicate evaluates to True
- (Can also be interpreted as a usual predicate by giving concrete values as arguments)

Mixing constraint solving and instantiation / backtracking: constraint solving by default, user-controlled instantiation
Luck: standard basis

Standard lambda calculus with

- pairs
- binary sums:
  \[ \text{L}_{T_1+T_2} e \quad \text{R}_{T_1+T_2} e \quad \text{case } e \text{ of } (\text{L } x \rightarrow e_1) (\text{R } y \rightarrow e_2) \]
- recursive types

Plus unknowns and special constructs to control instantiation
fun (bst : int -> int -> int -> int tree -> bool)
    size low high tree =
    if size == 0 then tree == Empty
    else case tree of
        | 1 % Empty -> True
        | size % Node x l r ->
            ((low < x && x < high) !x)
            && bst (size / 2) low x l
            && bst (size / 2) x high r

Apply bst to given integers size, low, high to obtain a BSTs generator (predicate int tree -> bool)
Choice-recording semantics

- Big-step operational probabilistic semantics

- Derivations for a generator $gen$ produce a possible output but also the choices made to reach it, recorded in trace $t$
  
  $gen \uparrow_t output$

- Doesn’t handle backtracking: $output$ is either an actual valuation $\mathcal{V}$, or need to backtrack $\emptyset$ (error monad)
Traces of choices

**Choice:** $(m, n, q)$
- $n$: number of possibilities
- $m$: index of the one actually taken $(0 \leq m < n)$
- $q$: probability of making this choice $(q \in \mathbb{Q}, 0 < q \leq 1)$

**Trace:** sequence of choices

**Probability** $P(t)$ of a trace $t$:
- product of the probabilities of its choices
(Sub)probability distribution

(P(t): product of the probabilities of the choices in t)

\[ \pi_{\text{gen}} = [ \text{output} \mapsto \sum_{t \mid \text{gen} \uparrow_t \text{output}} P(t) ] \]

gen \uparrow [ (0,2,\frac{1}{2}); (0,2,\frac{1}{3}); (0,2,\frac{2}{5}) ]

gen \uparrow [ (0,2,\frac{1}{2}); (0,2,\frac{1}{3}); (1,2,\frac{3}{5}) ]

gen \uparrow [ (0,2,\frac{1}{2}); (1,2,\frac{2}{3}) ]

gen \uparrow [ (1,2,\frac{1}{2}) ]

\emptyset \quad \mathcal{N}_1 \quad \mathcal{V}_2

[ \mathcal{N}_1 \mapsto \frac{1}{10}; \mathcal{V}_2 \mapsto \frac{1}{2}; \emptyset \mapsto \frac{1}{15} + \frac{1}{3} ]
Advantages of this new semantics

Former semantics: directly derives whole probability distribution in a collecting style \( \text{gen} \uparrow \pi_{\text{gen}} \)

Our new choice-recording semantics is:

- simpler: proofs with Coq
- more expressive:
  - still able to produce \( \pi_{\text{gen}} \)
  - better handles not always terminating programs
  - provides more detailed information, used in next part
Integrating backtracking strategies: motivation

$\pi_{gen}$ ranges over valuations $\mathcal{V}$ but also need to backtrack $\emptyset$

Running a generator in practice:
apply a backtracking strategy to always output a valuation

Objective: determine final distribution $\rho^{bstrat}_{gen}$ over valuations of generator $gen$ when using backtracking strategy $bstrat$
(Assume all executions terminate)

\[\text{gen} \uparrow \{(0,\frac{2}{5}); \ (0,\frac{1}{3}); \ (0,\frac{2}{5})\}\]
\[\text{gen} \uparrow \{(0,\frac{2}{5}); \ (0,\frac{1}{3}); \ (1,\frac{3}{5})\}\]
\[\text{gen} \uparrow \{(0,\frac{2}{5}); \ (1,\frac{2}{3})\}\]
\[\text{gen} \uparrow \{(1,\frac{1}{2})\}\]
Markov chains to model application of a strategy

Time-homogeneous Markov chains with discrete time and finite state space

Restart-from-scratch strategy

Absorbing Markov chain

\[
\begin{align*}
\mathcal{V}_1 & \mapsto \frac{1}{6}; \mathcal{V}_2 \mapsto \frac{5}{6}
\end{align*}
\]
Example of non absorbing Markov chain

Backtrack-from-leaf-to-parent strategy

Markov chain is not absorbing: no probability distribution
More complex strategy I

- Remember hopeless nodes that lead only to $\emptyset$ leaves
- Backtrack to parent from $\emptyset$ leaf or node with only hopeless children
- Restart execution when more than a set number $B_{max}$ of $\emptyset$ leaves have been encountered

Set of states:

\[ \text{Nodes} \times ( \mathcal{P}(\text{Nodes}) \times \{0, 1, ..., B_{max}\} ) \]

\[ ( \text{node}, ( \text{hopeless nodes, number of } \emptyset \text{ leaves seen} ) ) \]
More complex strategy II

\[(D, S, n) \xrightarrow{1} (C, S \cup \{D\}, n + 1)\]
\[\text{if } n < B_{\text{max}}\]

\[(D, S, B_{\text{max}}) \xrightarrow{1} (A, S \cup \{D\}, 0)\]

\[\begin{align*}
(C, S, n) & \xrightarrow{2/5} (D, S, n) \quad \text{if } D, E \not\in S \\
(C, S, n) & \xrightarrow{3/5} (E, S, n) \quad \text{if } D, E \not\in S \\
(C, S, n) & \xrightarrow{1} (E, S, n) \quad \text{if } D \in S, E \not\in S \\
(C, S, n) & \xrightarrow{1} (B, S \cup \{C\}, n) \quad \text{if } D, E \in S \\
& \quad \text{(impossible, yet transition exists)}
\end{align*}\]

\[(G, S, n) \xrightarrow{1} (G, S, n)\]
Definition of a strategy

**Backtracking strategy**: computable function associating to a choice tree a Markov chain verifying:

- it is time-homogeneous with discrete time
- it has a finite set of states of form $\text{Nodes} \times \mathcal{M}$
- it has a single initial state of form $(\text{root}, M_0)$
- absorbing states are exactly those with a non-$\emptyset$ leaf
- it is absorbing
Computability of the final probability distribution

**Theorem**

*Final distribution* $\rho_{\text{bstrat}}^{\text{gen}}$ *is computable from* $\text{gen}$ *and* $\text{bstrat}$.  

- $\text{gen} \rightarrow$ choice tree: computable (finite)
- choice tree $\rightarrow$ Markov chain: $\text{bstrat}$ computable (def)
- Markov chain $\rightarrow$ $\rho_{\text{gen}}^{\text{bstrat}}$:
  
  Transition matrix of absorbing Markov chain

\[
\begin{pmatrix}
Q & R \\
0 & I
\end{pmatrix}
\]

Probability to be absorbed in $j$ from non-absorbing $i$:

\[
\sum_{s \geq 0} Q^s R = (I - Q)^{-1} R
\]
A strategy is “activated only upon backtracking” if for any Markov chain in its image:

for any state that is reachable from the initial state without visiting a $\emptyset$ leaf, the transitions follow the choice tree

\[
\begin{array}{c}
A \\
\frac{2}{3} \quad \frac{1}{3} \\
B & C
\end{array}
\]

Consequence: $\forall \mathcal{V}. \rho_{\text{gen}}^{bstrat}(\mathcal{V}) \geq \pi_{\text{gen}}(\mathcal{V})$
Restriction about computation of MC from CT

A strategy is a computable function from choice trees to Markov chains: very permissive

Possible restriction: transitions from a state \((node, M)\) computable only from \(node, M, \) children of \(node\) and edges to them in choice tree

Examples need minor adaptation:
- restart-from-scratch: add memorised information which always contains \(root\)
- more complex strategy: add path of ancestors of the current node to memorised information
Evaluation of backtracking strategies

\( \pi_{gen} \) usually more intuitive for the user than \( \rho_{gen}^{bstrat} \)
so ideally \( bstrat \) should keep \( \rho_{gen}^{bstrat} \) close to \( \pi_{gen} \)

Extremality result: restart-from-scratch most conservative, according to two functions evaluating closeness of distributions

\[\begin{align*}
    &Bhattacharyya distance \quad D_B(P, Q) = -\log \sum_i \sqrt{P(i)Q(i)} \\
    &Kullback-Leibler divergence \quad D_{KL}(P \| Q) = \sum_i P(i) \log \frac{P(i)}{Q(i)}
\end{align*}\]

Future work: sum with other terms to minimise, such as time complexity term (e.g. expected number of steps in the Markov chain before being absorbed)
Choice-recording semantics style: simple (Coq proofs) yet more expressive than the previous one.

Formalisation of backtracking strategies. Integration of them into the semantics to obtain the final probability distribution of a generator. Computability result.

Future work:
- identify and study narrower subclasses of strategies
- further explore evaluation of strategies
to characterise viable strategies.