Generating Good Generators for Inductive Relations

in

Quick Chick

a property-based random testing tool for Coq
Testing with QuickChick

Conjecture: \( \forall (x : A). P(x) \)

gen : G A \hspace{1cm} a : A \hspace{1cm} P(a) \\

✓ More confidence!

✗ Counterexample!
Testing with QuickChick

Conjecture: $\forall (x : A). Q(x) \Rightarrow P(x)$

gen : $G \ A \ a : A \ Q(a) \? \ \ P(a) \?

More confidence!

Counterexample!
Sparse Preconditions

\[ Q \]

\text{gen:G A}
Lemma STLC_preservation :
∀ (e1 e2 : term) (t : type),
[] |- e1 : t →
e1 ===> e2 →
[] |- e2 : t.

Proof.
quickchick.

- Random data generator for term
- Type inference function
- e ===> e' as a function
- Decidability for Γ |- e : t
**Lemma** STLC_preservation:
\[\forall (e1 \ e2 : \text{term}) \ (t : \text{type}),
\begin{align*}
[] & \vdash e1 : t \rightarrow \\
e1 \rightarrow e2 & \rightarrow \\
[] & \vdash e2 : t.
\end{align*}\]

**Proof.**
quickcheck.

*** Gave up! Passed only 23 tests
Discarded: 20000

More confidence?
\[
\begin{align*}
4 \times x & \quad 1 \quad 5 \times 4 \\
(\lambda z^N.x) & \quad \lambda v^{(N\to N)\to N}.x_6 \quad ((\lambda x^N.z) (3 (\lambda v^N.6))) 3 \\
42 & \quad \lambda x^N.2 \quad (\lambda z^{(N\to N)\to N}.y) (\lambda y^N.y) \\
5 \times y & \quad w \quad \lambda x^N.1 5 \\
z (1 \times y) & \quad \lambda z^N.(1 (\lambda y^N.4)) \quad \lambda y^{(N\to N)\to (N\to N)}.(\lambda y^N.1 y) \\
\lambda y^{N\to N} (\lambda y^N x) & \quad \lambda y^{N\to N}.y (3 2) \quad 3 \\
(\lambda w^N.v) x & \quad z \quad \lambda z^N.(\lambda y^N.(\lambda z^{N\to N}.2))
\end{align*}
\]
4 \ x

(\lambda \ z^N \cdot x) \ 1

\lambda \ v^{(N \rightarrow N) \rightarrow N} \cdot x_6

((\lambda \ x^N \cdot z) (3 (\lambda \ v^N \cdot 6))) \ 3

42

\lambda \ x^N \cdot 2

(\lambda \ z^{(N \rightarrow N) \rightarrow N} \cdot y) (\lambda \ y^N \cdot y)

5 \ y

w

\lambda \ x^N \cdot 1 5

z (1 \ y)

\lambda \ z^N \cdot (1 (\lambda \ y^N \cdot 4))

\lambda \ y^{(N \rightarrow N) \rightarrow (N \rightarrow N)} \cdot (\lambda \ y^N \cdot 1 \ y)

\lambda \ y^{N \rightarrow N} \cdot (\lambda \ y^N \cdot x)

\lambda \ y^N \cdot y (3 \ 2)

3

\lambda \ y^{N \rightarrow N} \cdot (\lambda \ y^N \cdot z)

z

\lambda \ z^N \cdot (\lambda \ y^N \cdot (\lambda \ z^{N \rightarrow N} \cdot 2))
Testing with Good Generators

Conjecture: $\forall (x : A). Q(x) \Rightarrow P(x)$

g $\text{gen} : G A$

a : A such that $Q(a)$

$P(a)$?

More confidence!

Counterexample!
Theorem:  \( \forall (x : A). Q(x) \Rightarrow P(x) \)

\( \text{gen:G} \quad A \text{ is good if} \)

**Soundness**
\[
x \in \text{range}(\text{gen}) \Rightarrow Q(x)
\]

and

**Completeness**
\[
x \in Q(x) \Rightarrow \text{range}(\text{gen})
\]
Generate *only* well-typed terms!

\[
\text{gen\_term} : \text{env} \to \text{type} \to \text{G term}
\]

such that

\[
e \in \text{range}(\text{gen\_term}\ \Gamma\ t) \Rightarrow \Gamma \vdash e : t
\]

Make sure that *all* of them can be generated!

\[
\Gamma \vdash e : t \Rightarrow e \in \text{range}(\text{gen\_term}\ \Gamma\ t)
\]
Fixpoint aux_arb (size : nat) (in1 : list type) (in2: type) : G (option term):=
match size with
| 0 =>
backtrack
[(1, doM! x ← genST (fun x : nat => Nth x in2 in1);
  returnGen (Some (Id x)));
(1, match in2 with
  | N => do! n ← arbitrary; returnGen (Some (Nat n))
  | Arrow _ _ => returnGen None
end)]
| size'.+1 =>
backtrack
[(1, doM! x ← genST (fun x : nat => Nth x in2 in1);
  returnGen (Some (Id x)));
(1, match in2 with
  | N => do! n ← arbitrary; returnGen (Some (Nat n))
  | Arrow _ _ => returnGen None
end);
(1, match in2 with
  | N => returnGen None
  | Arrow tau1 tau2 =>
    doM! tm ← aux_arb size' (tau2 :: in1) tau2;
    returnGen (Some (Abs tau1 tm))
end);
(1, do! tau1 ← arbitrary;
  doM! t1 ← aux_arb size' in1 (Arrow tau1 in2);
  doM! t2 ← aux_arb size' in1 tau1;
  returnGen (Some (App t1 t2)))]
end.
Fixpoint aux_arb (size : nat) (in1 : list type) (in2: type) : G (option term) :=
match size with
  | 0 =>
    backtrack
    [(1, doM! x ← genST (fun x : nat => Nth x in2 in1);
      returnGen (Some (Id x)));
    (1, match in2 with
      | N => do! n ← arbitrary; returnGen (Some (Nat n))
      | Arrow _ _ => returnGen None
    end)]
  | size'+1 =>
    backtrack
    end);
(1, match in2 with
  | N => returnGen None
  | Arrow tau1 tau2 =>
    doM! tm ← aux_arb size' (tau2 :: in1) tau2;
    returnGen (Some (Abs tau1 tm))
  end);
(1, do! tau1 ← arbitrary;
  doM! t1 ← aux_arb size' in1 (Arrow tau1 in2);
  doM! t2 ← aux_arb size' in1 tau1;
  returnGen (Some (App t1 t2)))]
end.

Testing an Optimising Compiler by Generating Random Lambda Terms.
Michał H. Palka, Koen Claessen, Alejandro Russo, and John Hughes. AST ’11
Generating Good Generators

\[
\text{Inductive } Q : A_1 \rightarrow A_2 \rightarrow \text{Prop}
\]

\[
\text{gen}_{\text{A}_2} : A_1 \rightarrow \text{Gterm}
\]

\[\forall a_1, \text{range}(\text{gen}_{\text{A}_2} a_1) \equiv \{a_2 \mid Q a_1 a_2\}\]
Generating Good Generators

Inductive has_type : env → term → type → Prop

gen_term : env → type → G term
Fixpoint gen_term (Γ : env) (t : type) : G (option term) :=
    backtrack [ ... ;

    ; ... ].
Fixpoint gen_term (Γ : env) (t : type) : G (option term) :=
  backtrack [ ... ;

; ... ].

| TAbs : ∀ Γ e t1 t2, (t1 :: Γ) |- e : t2 →
  Γ |- (Abs t1 e) : Arrow t1 t2
Fixpoint gen_term (Γ : env) (t : type) : G (option term):=
backtrack [ ... ; match t with

; ... ].

| TAbs : ∀ Γ e t1 t2, (t1 :: Γ) |- e : t2 →
Γ |- (Abs t1 e) : Arrow t1 t2
Fixpoint gen_term (Γ : env) (t : type) : G (option term) :=
  backtrack [ ... ; match t with
    | N => ret None

  ; ... ].

| TAbs : ∀ Γ e t1 t2, (t1 :: Γ) |- e : t2 →
  Γ |- (Abs t1 e) : Arrow t1 t2
Fixpoint gen_term (Γ : env) (t : type) : G (option term):=
  backtrack [ ... ; match t with
    | N => ret None
    | Arrow t1 t2 =>

end ; ... ].

| TAbs : ∀ Γ e t1 t2, (t1 :: Γ) |- e : t2 →
  Γ |- (Abs t1 e) : Arrow t1 t2
Fixpoint gen_term (Γ : env) (t : type) : G (option term) :=
  backtrack [ ... ; match t with
    | N => ret None
    | Arrow t1 t2 =>

      ret (Some (Abs t1 e))
  end ; ... ].

| TAbs : ∀ Γ e t1 t2, (t1 :: Γ) |- e : t2 →
  Γ |- (Abs t1 e) : Arrow t1 t2
Fixpoint gen_term (Γ : env) (t : type) : G (option term) :=
backtrack [ ... ; match t with
   | N => ret None
   | Arrow t1 t2 =>
     doM! e ↔ gen_term (t1 :: Γ) t2;
     ret (Some (Abs t1 e))
   end ; ... ].

| TAbs : ∀ Γ e t1 t2, (t1 :: Γ) |- e : t2 →
      Γ |- (Abs t1 e) : Arrow t1 t2
**Lemma** STLC\_preservation :
\[ \forall (e_1 \ e_2 : \text{term}) (t : \text{type}), \]
\[
\begin{array}{l}
[\] \vdash e_1 : t \to \\
\text{e1} \implies e_2 \to \\
[\] \vdash e_2 : t.
\end{array}
\]

**Proof.**
\text{quickChick.}

Arrow N N
Some App (Abs (Arrow N N) (Abs N (Id 0))) (Abs N (Id 0))
*** Failed \text{after} 8 tests and 0 shrinks. (31 discards)
Generating Provably Good Generators

\[
\forall \Gamma t, \text{range}(\text{gen\_term}_\Gamma t) \equiv \{ e | \Gamma \vdash e : t \}
\]
Generating Provably Good Generators

\[ \text{Inductive has_type : env \rightarrow \text{term} \rightarrow \text{type} \rightarrow \text{Prop}} \]

\[ \text{bind} \quad \text{ret} \]

\[ \quad \rightarrow \quad \rightarrow \quad \rightarrow \quad \rightarrow \]

\[ \text{bind\_correct} \quad \text{ret\_correct} \]

\[ \text{gen\_term : env \rightarrow \text{type} \rightarrow \text{Gterm}} \]
Generating Provably Good Generators

Inductive has_type : env -> term -> type -> Prop

bind ret

bind_correct ret_correct

\[ \text{gen
ter
correct} : \forall \Gamma t, \text{range}(\text{gen
term} \Gamma t) \equiv \{ e | \Gamma \vdash e : t \} \]
Evaluation

- Is the class of inductive definitions large/general/useful?
- Are the generators efficient?
- Do they achieve good coverage and distribution of test cases?
Evaluation

Applicability
- Tested specifications from Software Foundations textbook
- 83% of suitable-for-testing theorems could be tested with our approach
Evaluation

Applicability

• Tested specifications from Software Foundations textbook
• 83% of suitable-for-testing theorems could be tested with our approach

Example test_orb1: (orb true false) = true.
Evaluation

Applicability

- Tested specifications from Software Foundations textbook
- 83% of suitable-for-testing theorems could be tested with our approach

**Theorem** hoare_seq :
\[
\forall P \ Q \ R \ c1 \ c2, \\
\{ \ Q \} \ c2 \ { \ R \} \rightarrow \\
\{ \ P \} \ c1 \ { \ Q \} \rightarrow \\
\{ \ P \} \ c1;;c2 \ { \ R \}.
\]
Evaluation

Performance

• Compared to handwritten generators used in IFC case study by Hritcu et al. (2013, 2016)
• $1.75 \times$ slower than handwritten generators
• Same bug-finding performance (counterexamples/sec)
Conclusion

Sound and complete generators for inductive relations for free!

What’s next?
• larger class of inductive definitions
• derive decidability instances
• derive shrinkers

Find us on GitHub! github.com/QuickChick/QuickChick