Generating Good Generators for Inductive Relations

POPL 2018

Leonidas Lampropoulos[1] Zoe Paraskevopoulou[2] Benjamin Pierce[1]

[1]University of Pennsylvania [2]Princeton University

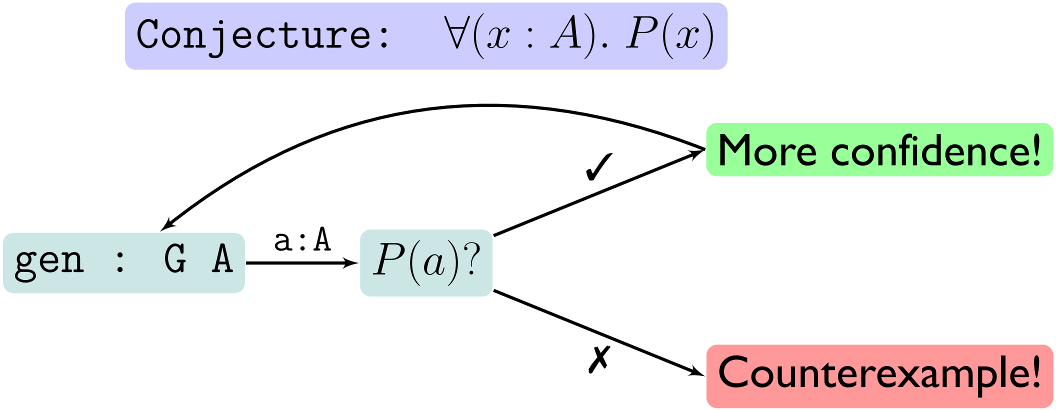
Generating Good Generators for Inductive Relations

in

QuickChick

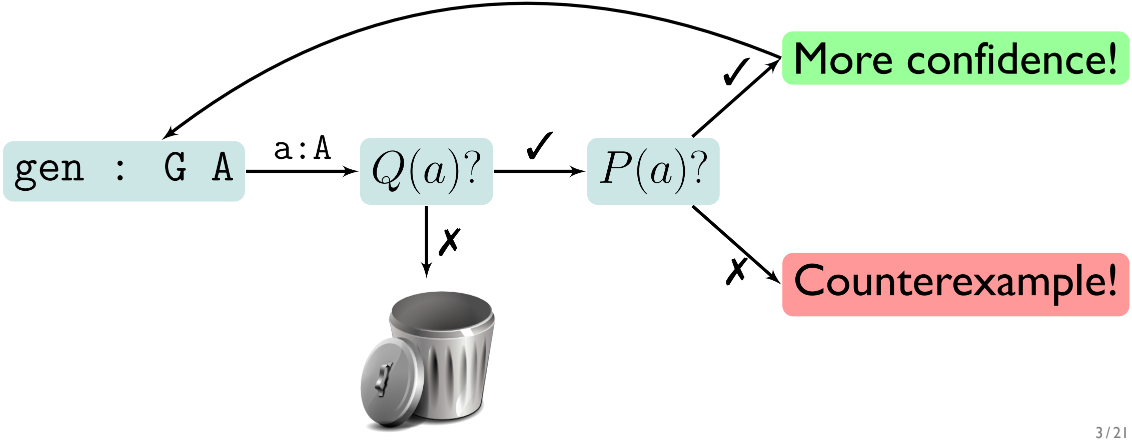
a property-based random testing tool for Coq

Testing with QuickChick

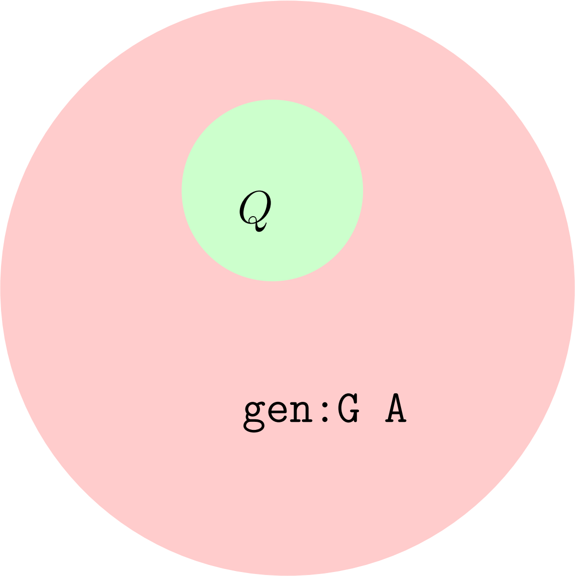


Testing with QuickChick

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



Sparse Preconditions



```
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 :
  ∀ (e1 e2 : term) (t : type),
    [] |- e1 : t →
    e1 ===> e2 →
    [] |- e2 : t.
Proof.
  quickchick.

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

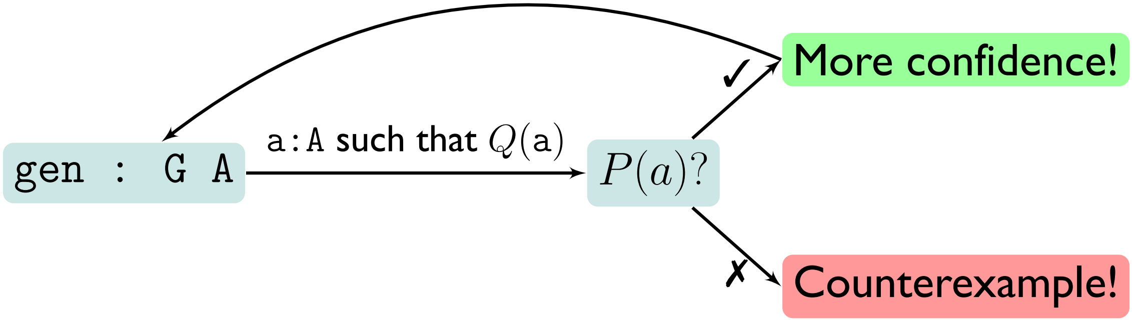
More confidence?

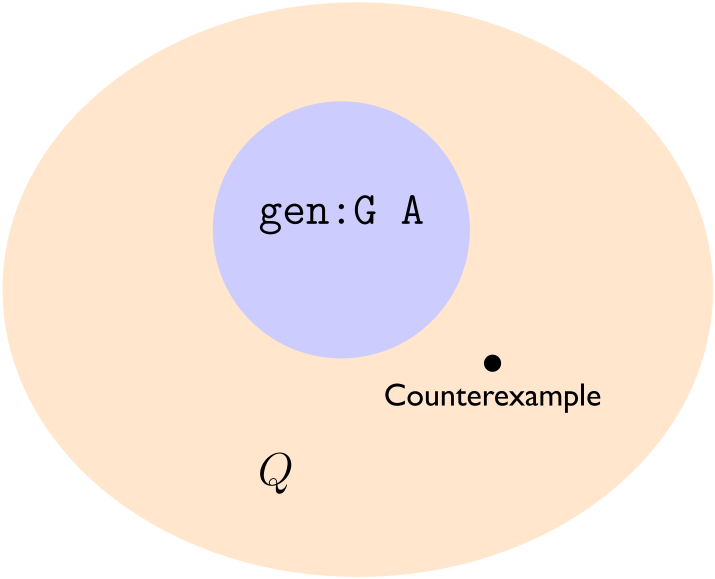
$4\ x$	1	$5\ 4$
$(\lambda\ z^{\mathbb{N}}.x)\ 1$	$\lambda\ v^{(\mathbb{N}\to\mathbb{N})\to\mathbb{N}}.x_6$	$((\lambda\ x^{\mathbb{N}}.z)\ (3\ (\lambda\ v^{\mathbb{N}}.6)))\ 3$
42	$\lambda\ x^{\mathbb{N}}.2$	$(\lambda\ z^{(\mathbb{N}\to\mathbb{N})\to\mathbb{N}}.y)\ (\lambda\ y^{\mathbb{N}}.y)$
$5\ y$	w	$\lambda\ x^{\mathbb{N}}.1\ 5$
$z\ (1\ y)$	$\lambda\ z^{N}.(1\ (\lambda\ y^{N}.4))$	$\lambda\ y^{(\mathbb{N}\to\mathbb{N})\to(\mathbb{N}\to\mathbb{N})}.(\lambda\ y^{\mathbb{N}}.1\ y)$
$\lambda\ y^{\mathbb{N}\to\mathbb{N}}\ (\lambda\ y^{\mathbb{N}}\ x)$	$\lambda\ y^{\mathbb{N}\to\mathbb{N}}.y\ (3\ 2)$	3
$(\lambda\ w^{\mathbb{N}}.v)\ x$	z	$\lambda\ z^{\mathbb{N}}.(\lambda\ y^{\mathbb{N}}.(\lambda\ z^{\mathbb{N}\to\mathbb{N}}.2))$

4 x	1	5 4
$(\lambda\ z^{\mathbb{N}}.x)$ 1	$\lambda\ v^{(\mathbb{N}\to\mathbb{N})\to\mathbb{N}}.x_6$	$((\lambda\ x^{\mathbb{N}}.z)\ (3\ (\lambda\ v^{\mathbb{N}}.6)))\ 3$
42	$\lambda\ x^{\mathbb{N}}.2$	$(\lambda\ z^{(\mathbb{N}\to\mathbb{N})\to\mathbb{N}}.y)\ (\lambda\ y^{\mathbb{N}}.y)$
5 y	w	$\lambda\ x^{\mathbb{N}}.1\ 5$
z (1 y)	$\lambda\ z^{N}.(1\ (\lambda\ y^{N}.4))$	$\lambda\ y^{(\mathbb{N}\to\mathbb{N})\to(\mathbb{N}\to\mathbb{N})}.(\lambda\ y^{\mathbb{N}}.1\ y)$
$\lambda\ y^{\mathbb{N}\to\mathbb{N}}\ (\lambda\ y^{\mathbb{N}}\ x)$	$\lambda\ y^{\mathbb{N}\to\mathbb{N}}.y\ (3\ 2)$	3
$(\lambda\ w^{\mathbb{N}}.v)\ x$	z	$\lambda\ z^{\mathbb{N}}.(\lambda\ y^{\mathbb{N}}.(\lambda\ z^{\mathbb{N}\to\mathbb{N}}.2))$

Testing with Good Generators

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




gen:G A
Counterexample
Q

$\texttt{Theorem:} \quad \forall(x : A).Q(x) \Rightarrow P(x)$

`gen:G A` is *good* if

Soundness

$x \in \texttt{range}(\texttt{gen}) \Rightarrow Q(x)$

and

Completeness

$x \in Q(x) \Rightarrow \texttt{range}(\texttt{gen})$

Generate *only* well-typed terms!

$$\texttt{gen\_term : env} \rightarrow \texttt{type} \rightarrow \texttt{G term}$$

such that

$$\texttt{e} \in \texttt{range}(\texttt{gen\_term}\,\Gamma\,\texttt{t}) \Rightarrow \Gamma \vdash \texttt{e} : \texttt{t}$$

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

$$\Gamma \vdash \texttt{e} : \texttt{t} \Rightarrow \texttt{e} \in \texttt{range}(\texttt{gen\_term}\,\Gamma\,\texttt{t})$$

```
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
```

Testing an Optimising Compiler by Generating Random Lambda Terms.
Michal H. Palka, Koen Claessen, Alejandro Russo, and John Hughes. AST '11

```
             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.
```

Generating Good Generators

$$\texttt{Inductive } \texttt{Q} : \texttt{A}_1 \rightarrow \texttt{A}_2 \rightarrow \texttt{Prop}$$

$$\texttt{gen\_A}_2 : \texttt{A}_1 \rightarrow \texttt{G term}$$

and

$$\forall\, \texttt{a}_1,\ \texttt{range}(\texttt{gen\_A}_2\ \texttt{a}_1) \equiv \{\texttt{a}_2 \mid Q\ \texttt{a}_1\ \texttt{a}_2\}$$

Generating Good Generators

$\texttt{Inductive has\_type : env} \rightarrow \texttt{term} \rightarrow \texttt{type} \rightarrow \texttt{Prop}$

$\texttt{gen\_term : env} \rightarrow \texttt{type} \rightarrow \texttt{G term}$

```
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 :
  ∀ (e1 e2 : term) (t : type),
    [] |- e1 : t →
    e1 ===> e2 →
    [] |- e2 : t.
Proof.
  quickChick.
```

```
Arrow N N
Some App (Abs (Arrow N N) (Abs N (Id 0))) (Abs N (Id 0))
*** Failed after 8 tests and 0 shrinks. (31 discards)
```

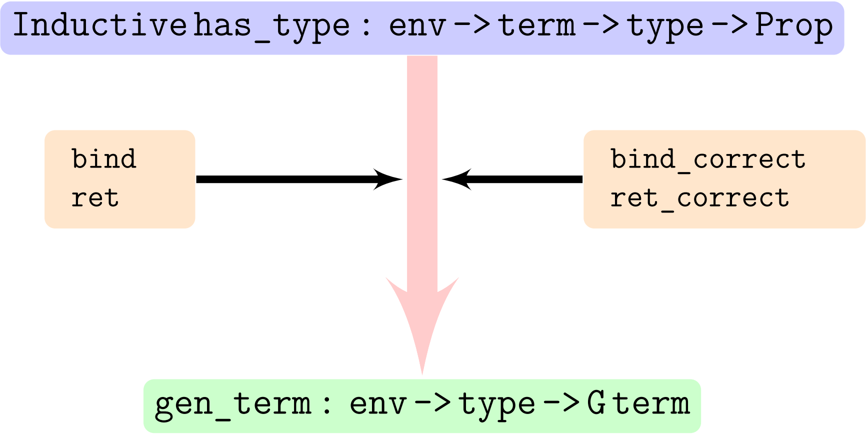
Generating Provably Good Generators

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

`bind`
`ret`

`gen_term : env -> type -> G term`

Generating Provably Good Generators



Generating Provably Good Generators

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

`bind`
`ret`

`bind_correct`
`ret_correct`

$\texttt{gen\_term\_correct} : \forall \Gamma\, \texttt{t},\ \texttt{range}(\texttt{gen\_term}\ \Gamma\ \texttt{t}) \equiv \{\texttt{e} \mid \Gamma \vdash \texttt{e} : \texttt{t}\}$

Evaluation

Is the class of inductive definitions large/general/useful?

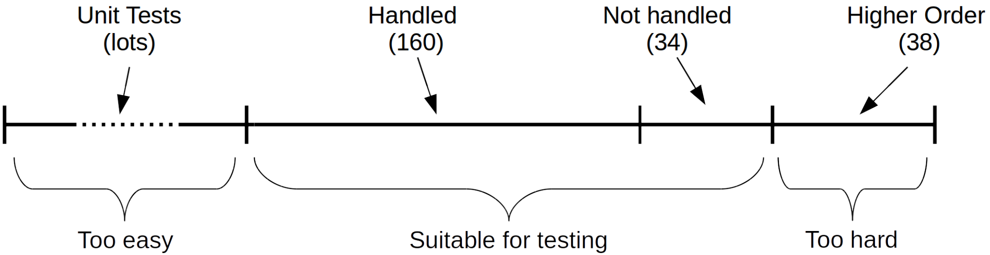
Are the generators efficient?

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 :
  ∀ P Q R c1 c2,
    { Q } c2 { R } →
    { P } c1 { Q }→
    { 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 that 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`