Looking Back: A Probabilistic Inverse Perspective on Test Generation

Joachim Tilsted Kristensen [1]
Tobias Reinhard [2,3]
Michael Kirkedal Thomsen [1,4]

[1]University of Oslo

[2]KU Leuven

[3]Technical University of Darmstadt

[4]University of Copenhagen

TYPES, June 11, 2024

Inverse Programs

Definition (Invertible)

A function `f :: A -> B` is called invertible, if there exists another function `unf :: B -> A`, that satisfies the equations

```
unf (f a) == a
f (unf b) == b
```

Examples

```
dec :: Int -> Int
dec   suc_n = suc_n - 1
undec     n =     n + 1

fib, unfib :: (Int, Int) -> (Int, Int)
fib   ( a, b) = (a +  b, a)
unfib (ab, a) = (a, ab - a)

fib_pair :: Int -> (Int, Int)
fib_pair 0 = (1, 1)
fib_pair n = fib (fib_pair (dec n))

unfib_pair :: (Int, Int) -> Int
unfib_pair (1, 1) = 0
unfib_pair  p     = undec (unfib_pair (unfib p))
```

Motivation

Why?

- `decode (encode x) = x`

Motivation

Why?

- `decode (encode x) = x`
- `decrypt key (encrypt key x) = x`

Motivation

Why?

- `decode (encode x) = x`
- `decrypt key (encrypt key x) = x`
- `test (untest False) = False`

Motivation

Why?

- `decode (encode x) = x`
- `decrypt key (encrypt key x) = x`
- `test (untest False) = False`
- Static guarantees that follow from information preservation.

Motivation

Why?

- `decode (encode x) = x`
- `decrypt key (encrypt key x) = x`
- `test (untest False) = False`
- Static guarantees that follow from information preservation.

How?

- McCarthy's generate and test

Motivation

Why?

- `decode (encode x) = x`
- `decrypt key (encrypt key x) = x`
- `test (untest False) = False`
- Static guarantees that follow from information preservation.

How?

- McCarthy’s generate and test
- Bennett’s reversible Turing Machine (`f' x = (x, f x)`)

Motivation

Why?

- `decode (encode x) = x`
- `decrypt key (encrypt key x) = x`
- `test (untest False) = False`
- Static guarantees that follow from information preservation.

How?

- McCarthy's generate and test
- Bennett's reversible Turing Machine (`f' x = (x, f x)`)
- Reversible by construction (Janus, Theseus, RFun, π-calculus)

Motivation

Why?

- `decode (encode x) = x`
- `decrypt key (encrypt key x) = x`
- `test (untest False) = False`
- Static guarantees that follow from information preservation.

How?

- McCarthy's generate and test
- Bennett's reversible Turing Machine (`f' x = (x, f x)`)
- Reversible by construction (Janus, Theseus, RFun, π-calculus)
- Relational intermediate languages (Mogensen, Kirkeby/Glück)

Motivation

Why?

- `decode (encode x) = x`
- `decrypt key (encrypt key x) = x`
- `test (untest False) = False`
- Static guarantees that follow from information preservation.

How?

- McCarthy's generate and test
- Bennett's reversible Turing Machine (`f' x = (x, f x)`)
- Reversible by construction (Janus, Theseus, RFun, π-calculus)
- Relational intermediate languages (Mogensen, Kirkeby/Glück)
- Explicit inverse program semantics (Jeopardy)

Motivation

Why?

- `decode (encode x) = x`
- `decrypt key (encrypt key x) = x`
- `test (untest False) = False`
- Static guarantees that follow from information preservation.

How?

- McCarthy's generate and test
- Bennett's reversible Turing Machine (`f' x = (x, f x)`)
- Reversible by construction (Janus, Theseus, RFun, π-calculus)
- Relational intermediate languages (Mogensen, Kirkeby/Glück)
- Explicit inverse program semantics (Jeopardy)

Jeopardy (Syntax)

$$
\begin{aligned}
x &\in \textbf{Name} && \text{(Well-formed variable names).}\\
c &\in \textbf{Name} && \text{(Well-formed constructor names).}\\
\tau &\in \textbf{Name} && \text{(Well-formed datatype names).}\\
f &\in \textbf{Name} && \text{(Well-formed function names).}\\
p &::= x \mid [c\ p_i] && \text{(Patterns).}\\
v &::= [c\ v_i] && \text{(Values).}\\
\Delta &::= f\ (p : \tau_p) : \tau_t \ = \ t\ .\ \Delta && \text{(Function definition).}\\
&\mid \texttt{data}\ \tau \ = \ [c\ \tau_i]_j\ .\ \Delta && \text{(Data type definition).}\\
&\mid \texttt{main}\ g\ . && \text{(Main function declaration).}\\
g &::= f \mid (\texttt{invert}\ g) && \text{(Function).}\\
t &::= p && \text{(Patterns in terms).}\\
&\mid g\ p && \text{(Function application).}\\
&\mid \texttt{case}\ t : \tau\ \texttt{of}\ p_i \to t_i && \text{(Case statement).}
\end{aligned}
$$

Jeopardy (Semantics)

$$\downarrow\text{Variable} : \frac{}{\Delta\Gamma \vdash x \downarrow v} \quad (\Gamma(x) = v) \qquad \downarrow\text{Constructor} : \frac{\Delta\Gamma \vdash p_i \downarrow v_i}{\Delta\Gamma \vdash [c\ p_i] \downarrow [c\ v_i]}$$

$$\downarrow\text{Cases} : \frac{\Delta\Gamma \vdash t \downarrow v \quad \Delta(\Gamma \circ \texttt{unify}(v, p_i)) \vdash t_i \downarrow v_i}{\Delta\Gamma \vdash \texttt{case}\ t : \tau\ \texttt{of}\ p_i \to t_i \downarrow v_i} \quad (\psi)$$

$$\downarrow\text{Application} : \frac{\Delta\Gamma \vdash p \downarrow v' \quad \Delta(\texttt{unify}(v', p')) \vdash t' \downarrow v}{\Delta[f\ (p' : \cdot) : \cdot = t']\Gamma \vdash f\ p \downarrow v}$$

$$\downarrow\text{Inversion} : \frac{\Delta\Gamma \vdash g\ p \uparrow v}{\Delta\Gamma \vdash (\texttt{invert}\ g)\ p \downarrow v}$$

Jeopardy (Semantics)

$$\downarrow\text{Variable} : \frac{}{\Delta\Gamma \vdash x \downarrow v} \quad (\Gamma(x) = v) \qquad \downarrow\text{Constructor} : \frac{\Delta\Gamma \vdash p_i \downarrow v_i}{\Delta\Gamma \vdash [c\ p_i] \downarrow [c\ v_i]}$$

$$\downarrow\text{Cases} : \frac{\Delta\Gamma \vdash t \downarrow v \quad \Delta(\Gamma \circ \texttt{unify}(v, p_i)) \vdash t_i \downarrow v_i}{\Delta\Gamma \vdash \texttt{case}\ t : \tau\ \texttt{of}\ p_i \to t_i \downarrow v_i} \quad (\psi)$$

$$\downarrow\text{Application} : \frac{\Delta\Gamma \vdash p \downarrow v' \quad \Delta(\texttt{unify}(v', p')) \vdash t' \downarrow v}{\Delta[f\ (p' : \cdot) : \cdot = t']\Gamma \vdash f\ p \downarrow v}$$

$$\downarrow\text{Inversion} : \frac{\Delta\Gamma \vdash g\ p \uparrow v}{\Delta\Gamma \vdash (\texttt{invert}\ g)\ p \downarrow v}$$

$$\uparrow\text{Application} : \frac{\Delta\Gamma \vdash p \downarrow v' \quad \Delta[\![t' \downarrow v']\!] \rightsquigarrow \Gamma' \quad \Delta\Gamma' \vdash p' \downarrow v}{\Delta[f\ (p' : \cdot) : \cdot = \ t']\Gamma \vdash f\ p \uparrow v}$$

$$\uparrow\text{Inversion} : \frac{\Delta\Gamma \vdash g\ p \downarrow v}{\Delta\Gamma \vdash (\texttt{invert}\ g)\ p \uparrow v}$$

Jeopardy (Environment Inference)

$\boxed{\Delta[\![t \downarrow v]\!] \rightsquigarrow \Gamma}$ (for linear terms t)

$$\rightarrow\text{Variable} : \frac{}{\Delta[\![x \downarrow v]\!] \rightsquigarrow \{x \mapsto v\}} \qquad \rightarrow\text{Constructor} : \frac{\Delta[\![p_i \downarrow v_i]\!] \rightsquigarrow \Gamma_i}{\Delta[\![[c\ p_i] \downarrow [c\ v_i]]\!] \rightsquigarrow \circ\Gamma_i}$$

$$\rightarrow\text{Cases} : \frac{\Delta[\![t_i \downarrow v_i]\!] \rightsquigarrow \Gamma_i \quad \Delta\Gamma_i \vdash p_i \downarrow v \quad \Delta[\![t \downarrow v]\!] \rightsquigarrow \Gamma}{\Delta[\![\texttt{case}\ t : \tau\ \texttt{of}\ p_i \rightarrow t_i \downarrow v_i]\!] \rightsquigarrow \Gamma_i \circ \Gamma}\ (\psi)$$

$$\rightarrow\text{Application} : \frac{\Delta[\![t' \downarrow v]\!] \rightsquigarrow \Gamma' \quad \Delta\Gamma' \vdash p' \downarrow v' \quad \Delta[\![p \downarrow v']\!] \rightsquigarrow \Gamma}{\Delta[f\ (p' : \cdot) : \cdot =\ t'][\![f\ p \downarrow v]\!] \rightsquigarrow \Gamma}$$

$$\rightarrow\text{Inversion} : \frac{\Delta[\![g\ p \uparrow v]\!] \leftsquigarrow \Gamma}{\Delta[\![(\texttt{invert}\ g)\ p \downarrow v]\!] \rightsquigarrow \Gamma}$$

Jeopardy (Environment Inference)

$\boxed{\Delta[\![t \downarrow v]\!] \rightsquigarrow \Gamma}$ (for linear terms t)

$$\rightarrow\text{Variable}: \frac{}{\Delta[\![x \downarrow v]\!] \rightsquigarrow \{x \mapsto v\}} \qquad \rightarrow\text{Constructor}: \frac{\Delta[\![p_i \downarrow v_i]\!] \rightsquigarrow \Gamma_i}{\Delta[\![[c\ p_i] \downarrow [c\ v_i]]\!] \rightsquigarrow \circ\Gamma_i}$$

$$\rightarrow\text{Cases}: \frac{\Delta[\![t_i \downarrow v_i]\!] \rightsquigarrow \Gamma_i \quad \Delta\Gamma_i \vdash p_i \downarrow v \quad \Delta[\![t \downarrow v]\!] \rightsquigarrow \Gamma}{\Delta[\![\texttt{case}\ t : \tau\ \texttt{of}\ p_i \to t_i \downarrow v_i]\!] \rightsquigarrow \Gamma_i \circ \Gamma} \quad (\psi)$$

$$\rightarrow\text{Application}: \frac{\Delta[\![t' \downarrow v]\!] \rightsquigarrow \Gamma' \quad \Delta\Gamma' \vdash p' \downarrow v' \quad \Delta[\![p \downarrow v']\!] \rightsquigarrow \Gamma}{\Delta[f\ (p' : \cdot) : \cdot =\ t'][\![f\ p \downarrow v]\!] \rightsquigarrow \Gamma}$$

$$\rightarrow\text{Inversion}: \frac{\Delta[\![g\ p \uparrow v]\!] \leftsquigarrow \Gamma}{\Delta[\![(\texttt{invert}\ g)\ p \downarrow v]\!] \rightsquigarrow \Gamma}$$

$$\leftarrow\text{Application}: \frac{\Delta[\![p' \downarrow v]\!] \rightsquigarrow \Gamma' \quad \Delta\Gamma' \vdash t' \downarrow v' \quad \Delta[\![p \downarrow v']\!] \rightsquigarrow \Gamma}{\Delta[f\ p'\ =\ t'][\![f\ p \uparrow v]\!] \leftsquigarrow \Gamma}$$

$$\leftarrow\text{Inversion}: \frac{\Delta[\![g\ p \downarrow v]\!] \rightsquigarrow \Gamma}{\Delta[\![(\texttt{invert}\ g)\ p \uparrow v]\!] \leftsquigarrow \Gamma}$$

Explicit Program Inversion

Problems

- Reversibility (local invertibility) Vs (global) Invertibility

```
swap :: (a, b) -> (b, a)
swap p = (snd p, fst p)
```

- Non-injective functions

```
sorted :: Ord a => [a] -> Bool
sorted [      ] = True
sorted (a : as) = all (a<=) as && sorted as
```

Explicit Program Inversion

Problems

- Reversibility (local invertibility) Vs (global) Invertibility

```
swap :: (a, b) -> (b, a)
swap p = (snd p, fst p)
```

- Non-injective functions

```
sorted :: Ord a => [a] -> Bool
sorted [      ] = True
sorted (a : as) = all (a<=) as && sorted as
```

Approach (WIP)

- Probabilistic program inversion
  ((invert f) : (b : B) -> Generator (A b))
- Stochastic types
  ((invert f) : B -> A_δ)

Probabilistic program inversion

Computable distribution

- Consider the term (`even  x`) to be *open*.

Probabilistic program inversion

Computable distribution

- Consider the term (`even x`) to be *open*.
- Assume a distribution $\mathcal{D}$ over x

Probabilistic program inversion

Computable distribution

- Consider the term (even x) to be *open*.
- Assume a distribution $\mathcal{D}$ over x
- $\mathcal{D}$ cannot be the uniform distribution.

Probabilistic program inversion

Computable distribution

- Consider the term (`even x`) to be *open*.
- Assume a distribution $\mathcal{D}$ over x
- $\mathcal{D}$ cannot be the uniform distribution.

Possible Solutions

- Size types

```
m : fin 5 = (invert even)(true)
```

Probabilistic program inversion

Computable distribution

- Consider the term (`even x`) to be *open*.
- Assume a distribution $\mathcal{D}$ over x
- $\mathcal{D}$ cannot be the uniform distribution.

Possible Solutions

- Size types

```
m : fin 5 = (invert even)(true)
```

- Stochastic Choice

Stocastic Types (WIP)

Example syntax

$$t := x \mid \ldots \mid \texttt{let}\, x\, \texttt{from}\, \tau_\delta\, \texttt{in}\, t$$
$$\delta := \texttt{uniform} \mid \ldots$$
$$\tau := \ldots$$

Intended rule for let

$$\delta\text{-let} : \frac{\Gamma[x \mapsto \tau_{x\delta}] \vdash t : \tau_\delta}{\Gamma \vdash \texttt{let}\, x\, \texttt{from}\, \tau_{x\delta}\, \texttt{in}\, t : \tau_\delta}$$

Thank you `{~_^}`.