Elegant memoization

Conal Elliott

Tabula

April 3, 2014

Laziness

- Value computed only when inspected.
- Saved for reuse.
- Every part of a data structure.
- Insulate definition from use: *modularity*.
- Routinely program with infinite structures.

What about functions?

- Functions as indexed collections.
- Are "accesses" cached?

What about functions?

- Functions as indexed collections.
- Are “accesses” cached?
- Why not?

What is memoization?

What is memoization?

- Conventional story: mutable hash tables.

What is memoization?

- Conventional story: mutable hash tables.
- Ironic flaw: only correct for pure functions.

What is memoization?

- Conventional story: mutable hash tables.
- Ironic flaw: only correct for pure functions.
- My definition: *conversion of functions into data structures*

What is memoization?

- Conventional story: mutable hash tables.
- Ironic flaw: only correct for pure functions.
- My definition: *conversion of functions into data structures*
- ... without loss of information.

What is memoization?

- Conventional story: mutable hash tables.
- Ironic flaw: only correct for pure functions.
- My definition: *conversion of functions into data structures*
- ... without loss of information.
- Preferably incremental.

What is memoization?

- Conventional story: mutable hash tables.
- Ironic flaw: only correct for pure functions.
- My definition: *conversion of functions into data structures*
- ... without loss of information.
- Preferably incremental.
- How?

Convenient notation

I'll use some non-standard (for Haskell) type notation:

$$\begin{array}{l}
\textbf{type } \mathbf{1} \quad = () \\
\textbf{data } \mathbf{0} \qquad \text{-- no values} \\
\textbf{type } (\times) = (,) \\
\textbf{type } (+) = \mathit{Either}
\end{array}$$

$$\begin{array}{l}
\textbf{infixl } 7 \times \\
\textbf{infixl } 6 +
\end{array}$$

Examples

$$f :: Bool \to Int$$
$$f\ x = \textbf{if}\ x\ \textbf{then}\ 3\ \textbf{else}\ 5$$

Examples

$$f :: Bool \to Int$$
$$f\ x = \textbf{if}\ x\ \textbf{then}\ 3\ \textbf{else}\ 5$$

$$g :: \mathbf{1} \to String$$
$$g\ () = map\ toUpper\ \texttt{"memoize!"}$$

Examples

$$f :: \mathit{Bool} \rightarrow \mathit{Int}$$
$$f\ x = \textbf{if}\ x\ \textbf{then}\ 3\ \textbf{else}\ 5$$

$$g :: \mathbf{1} \rightarrow \mathit{String}$$
$$g\ () = \mathit{map}\ \mathit{toUpper}\ \texttt{"memoize!"}$$

$$h :: \mathit{Bool} \times \mathit{Bool} \rightarrow \mathit{Int}$$
$$h\ (x, y) = f\ (x \wedge y) + f\ (x \vee y)$$

Examples

$$f :: \mathit{Bool} \rightarrow \mathit{Int}$$
$$f\ x = \textbf{if}\ x\ \textbf{then}\ 3\ \textbf{else}\ 5$$

$$g :: \mathbf{1} \rightarrow \mathit{String}$$
$$g\ () = \mathit{map}\ \mathit{toUpper}\ \texttt{"memoize!"}$$

$$h :: \mathit{Bool} \times \mathit{Bool} \rightarrow \mathit{Int}$$
$$h\ (x, y) = f\ (x \wedge y) + f\ (x \vee y)$$

$$k :: \mathit{Bool} + \mathit{Bool} \rightarrow \mathit{Int}$$
$$k\ (\mathit{Left}\ \ x) = \textbf{if}\ x\ \textbf{then}\ 3\ \textbf{else}\ 5$$
$$k\ (\mathit{Right}\ y) = \textbf{if}\ y\ \textbf{then}\ 4\ \textbf{else}\ 6$$

More examples

- $Bool + Bool \times Bool \to ...$
- $Nat \to ...$
- $[\,a\,] \to ...$

What's really going on here?

- Remember: conversion of functions into data structures.
- Different shape for each domain type.
- Consider domain types systematically

What's really going on here?

- Remember: conversion of functions into data structures.
- Different shape for each domain type.
- Consider domain types systematically:
  $\mathbf{0}$, $\mathbf{1}$, $a + b$, $a \times b$, $a \to b$, **data**.

What's really going on here?

- Goal: capture all of a function's information.
- Make precise:

What's really going on here?

- Goal: capture all of a function's information.
- Make precise: ability to convert back. *Isomorphism.*

What's really going on here?

- Goal: capture all of a function's information.
- Make precise: ability to convert back. *Isomorphism.*
- Domain type drives the memo structure.

Type isomorphisms

Type isomorphisms

$$\mathbf{0} \rightarrow a \quad \cong$$

Type isomorphisms

$$\mathbf{0} \rightarrow a \;\cong\; \mathbf{1}$$

Type isomorphisms

$$
\begin{aligned}
\mathbf{0} \to a &\cong \mathbf{1} \\
\mathbf{1} \to a &\cong
\end{aligned}
$$

Type isomorphisms

$$\begin{aligned} \mathbf{0} \to a &\cong \mathbf{1} \\ \mathbf{1} \to a &\cong a \end{aligned}$$

Type isomorphisms

$$\begin{aligned} \mathbf{0} \to a &\cong \mathbf{1} \\ \mathbf{1} \to a &\cong a \\ (b + c) \to a &\cong \end{aligned}$$

Type isomorphisms

$$\begin{aligned} \mathbf{0} \rightarrow a &\cong \mathbf{1} \\ \mathbf{1} \rightarrow a &\cong a \\ (b + c) \rightarrow a &\cong (b \rightarrow a) \times (c \rightarrow a) \end{aligned}$$

Type isomorphisms

$$\begin{aligned}
\mathbf{0} \to a &\cong \mathbf{1} \\
\mathbf{1} \to a &\cong a \\
(b + c) \to a &\cong (b \to a) \times (c \to a) \\
(b \times c) \to a &\cong
\end{aligned}$$

Type isomorphisms

$$
\begin{aligned}
\mathbf{0} \rightarrow a &\cong \mathbf{1} \\
\mathbf{1} \rightarrow a &\cong a \\
(b + c) \rightarrow a &\cong (b \rightarrow a) \times (c \rightarrow a) \\
(b \times c) \rightarrow a &\cong b \rightarrow (c \rightarrow a)
\end{aligned}
$$

Type isomorphisms

$$\begin{aligned}
\mathbf{0} \to a &\cong \mathbf{1} \\
\mathbf{1} \to a &\cong a \\
(b + c) \to a &\cong (b \to a) \times (c \to a) \\
(b \times c) \to a &\cong b \to (c \to a)
\end{aligned}$$

Compare with laws of exponents:

$$\begin{aligned}
a^0 &= 1 \\
a^1 &= a \\
a^{b+c} &= a^b \times a^c \\
a^{b \times c} &= (a^c)^b
\end{aligned}$$

Type isomorphisms

$$\begin{aligned}
\mathbf{0} \to a &\cong \mathbf{1} \\
\mathbf{1} \to a &\cong a \\
(b + c) \to a &\cong (b \to a) \times (c \to a) \\
(b \times c) \to a &\cong b \to (c \to a)
\end{aligned}$$

Compare with laws of exponents:

$$\begin{aligned}
a^0 &= 1 \\
a^1 &= a \\
a^{b+c} &= a^b \times a^c \\
a^{b \times c} &= (a^c)^b
\end{aligned}$$

These rules form a memoization algorithm.

Type isomorphisms

$$\begin{aligned}
\mathbf{0} \to a &\cong \mathbf{1} \\
\mathbf{1} \to a &\cong a \\
(b + c) \to a &\cong (b \to a) \times (c \to a) \\
(b \times c) \to a &\cong b \to (c \to a)
\end{aligned}$$

Compare with laws of exponents:

$$\begin{aligned}
a^0 &= 1 \\
a^1 &= a \\
a^{b+c} &= a^b \times a^c \\
a^{b \times c} &= (a^c)^b
\end{aligned}$$

These rules form a memoization algorithm.

Catch: $\bot$.

An implementation of memoization

From *MemoTrie*:

$$\begin{array}{l} \textbf{class}\ \mathit{HasTrie}\ t\ \textbf{where} \\ \quad \textbf{type}\ t \mapsto a \\ \quad \mathit{trie} \quad\ :: (t \to a) \to (t \mapsto a) \\ \quad \mathit{untrie} :: (t \mapsto a) \to (t \to a) \end{array}$$

Law: *trie* and *untrie* are inverses, so $(t \mapsto a) \cong (t \to a)$ (modulo $\bot$).

Memoization:

An implementation of memoization

From *MemoTrie*:

$$\begin{aligned}
&\textbf{class}\ \mathit{HasTrie}\ t\ \textbf{where} \\
&\quad \textbf{type}\ t \mapsto a \\
&\quad \mathit{trie}\ \ \ \ :: (t \to a) \to (t \mapsto a) \\
&\quad \mathit{untrie} :: (t \mapsto a) \to (t \to a)
\end{aligned}$$

Law: *trie* and *untrie* are inverses, so $(t \mapsto a) \cong (t \to a)$ (modulo $\bot$).

Memoization:

$$\begin{aligned}
&\mathit{memo} :: \mathit{HasTrie}\ t \Rightarrow (t \to a) \to (t \to a) \\
&\mathit{memo} = \mathit{untrie} \circ \mathit{trie}
\end{aligned}$$

Void

Isomorphism: $\mathbf{0} \to a \cong \mathbf{1}$.

Void

Isomorphism: $\mathbf{0} \to a \cong \mathbf{1}$.

Instance:

$$
\begin{array}{l}
\textbf{instance } \mathit{HasTrie}\ \mathbf{0}\ \textbf{where} \\
\quad \textbf{type } \mathbf{0} \mapsto a = \mathbf{1} \\
\quad \mathit{trie}\ \mathit{void} = () \\
\quad \mathit{untrie}\ () = \mathit{void}
\end{array}
$$

where

$$
\begin{array}{l}
\mathit{void} :: \mathbf{0} \to z \\
\quad \text{-- empty definition}
\end{array}
$$

Unit

Isomorphism: $\mathbf{1} \to a \cong a$.

Unit

Isomorphism: $\mathbf{1} \to a \cong a$.

Instance:

$$
\begin{array}{l}
\textbf{instance}\ \mathit{HasTrie}\ \mathbf{1}\ \textbf{where} \\
\quad \textbf{type}\ \mathbf{1} \mapsto a = a \\
\quad \mathit{trie}\ f = f\ () \\
\quad \mathit{untrie}\ a = \lambda() \to a
\end{array}
$$

Boolean

Isomorphism: $Bool \to a \cong a \times a$.

Boolean

Isomorphism: $Bool \rightarrow a \cong a \times a$.

Instance:

$$
\begin{array}{l}
\textbf{instance}\ \mathit{HasTrie}\ \mathit{Bool}\ \textbf{where} \\
\quad \textbf{type}\ \mathit{Bool} \mapsto a = a \times a \\
\quad \mathit{trie}\ f = (f\ \mathit{False}, f\ \mathit{True}) \\
\quad \mathit{untrie}\ (x, y) = \lambda c \rightarrow \textbf{if}\ c\ \textbf{then}\ y\ \textbf{else}\ x
\end{array}
$$

Sums

Isomorphism: $(b + c) \to a \cong (b \to a) \times (c \to a)$.

Sums

Isomorphism: $(b + c) \to a \cong (b \to a) \times (c \to a)$.

Instance:

$$
\begin{array}{l}
\textbf{instance}\ (\mathit{HasTrie}\ b, \mathit{HasTrie}\ c) \Rightarrow \mathit{HasTrie}\ (b + c)\ \textbf{where} \\
\quad \textbf{type}\ (b + c) \mapsto a = (b \mapsto a) \times (c \mapsto a) \\
\quad \mathit{trie}\ f = (\mathit{trie}\ (f \circ \mathit{Left}), \mathit{trie}\ (f \circ \mathit{Right})) \\
\quad \mathit{untrie}\ (s, t) = \mathit{untrie}\ s \ |||\ \mathit{untrie}\ t
\end{array}
$$

where

$$
\begin{array}{l}
(g\ |||\ h)\ (\mathit{Left}\ \ \ b) = g\ b \\
(g\ |||\ h)\ (\mathit{Right}\ c) = h\ c
\end{array}
$$

Products

Isomorphism: $(b \times c) \to a \cong b \to (c \to a)$.

Products

Isomorphism: $(b \times c) \to a \cong b \to (c \to a)$.

Instance:

$$\begin{array}{l}
\textbf{instance}\ (\mathit{HasTrie}\ b, \mathit{HasTrie}\ c) \Rightarrow \mathit{HasTrie}\ (b \times c)\ \textbf{where} \\
\quad \textbf{type}\ (b \times c) \mapsto a = b \mapsto (c \mapsto a) \\
\quad \mathit{trie}\ f = \mathit{trie}\ (\mathit{trie} \circ \mathit{curry}\ f) \\
\quad \mathit{untrie}\ t = \mathit{uncurry}\ (\mathit{untrie} \circ \mathit{untrie}\ t)
\end{array}$$

where

$$\begin{array}{lll}
\mathit{curry} & g\ b\ c & = g\ (b, c) \\
\mathit{uncurry} & h\ (b, c) & = h\ b\ c
\end{array}$$

Data types

Handle other types via isomorphism:

$$(u, v, w) \cong (u \times v) \times w$$

$$[u] \cong \mathbf{1} + u \times [u]$$

$$T\ u \cong u + T\ u \times T\ u$$

$$Bool \cong \mathbf{1} + \mathbf{1}$$

Turn it around

Exponentials:

$$\begin{aligned} a^0 &= 1 \\ a^1 &= a \\ a^{b+c} &= a^b \times a^c \\ a^{b \times c} &= (a^c)^b \end{aligned}$$

Turn it around

Exponentials:

$$\begin{aligned} a^0 &= 1 \\ a^1 &= a \\ a^{b+c} &= a^b \times a^c \\ a^{b \times c} &= (a^c)^b \end{aligned}$$

Take logarithms, and flip equations:

$$\begin{aligned} \log_a 1 &= 0 \\ \log_a a &= 1 \\ \log_a(a^b \times a^c) &= b + c \\ \log_a (a^c)^b &= b \times c \end{aligned}$$

Turn it around

Exponentials:

$$
\begin{aligned}
a^0 &= 1 \\
a^1 &= a \\
a^{b+c} &= a^b \times a^c \\
a^{b \times c} &= (a^c)^b
\end{aligned}
$$

Take logarithms, and flip equations:

$$
\begin{aligned}
\log_a 1 &= 0 \\
\log_a a &= 1 \\
\log_a (u \times v) &= \log_a u + \log_a v \\
\log_a (u^b) &= b \times \log_a u
\end{aligned}
$$

Logarithms

$$
\begin{aligned}
\log_a 1 &= 0 \\
\log_a a &= 1 \\
\log_a (u \times v) &= \log_a u + \log_a v \\
\log_a (u^b) &= b \times \log_a u
\end{aligned}
$$

Logarithms

$$\begin{aligned}
\log_a 1 &= 0 \\
\log_a a &= 1 \\
\log_a (u \times v) &= \log_a u + \log_a v \\
\log_a (u^b) &= b \times \log_a u
\end{aligned}$$

Game: whose memo trie is it?

Logarithms

$$
\begin{aligned}
\log_a 1 &= 0 \\
\log_a a &= 1 \\
\log_a (u \times v) &= \log_a u + \log_a v \\
\log_a (u^b) &= b \times \log_a u
\end{aligned}
$$

Game: whose memo trie is it?

$$
\begin{aligned}
&\mathbf{data}\ P\ \ a = P\ a\ a \\
&\mathbf{data}\ S\ \ a = C\ a\ (S\ a) \\
&\mathbf{data}\ T\ a = B\ a\ (P\ (T\ a))
\end{aligned}
$$

Logarithms

$$\begin{aligned}
\log_a 1 &= 0 \\
\log_a a &= 1 \\
\log_a (u \times v) &= \log_a u + \log_a v \\
\log_a (u^b) &= b \times \log_a u
\end{aligned}$$

Game: whose memo trie is it?

data $P\ a = P\ a\ a$ **data** $S\ a = C\ a\ (S\ a)$ **data** $T\ a = B\ a\ (P\ (T\ a))$	$P\ a \cong a \times a$ $S\ a \cong a \times S\ a$ $T\ a \cong a \times P\ (T\ a)$

Logarithms

$$\begin{aligned}\log_a 1 &= 0\\ \log_a a &= 1\\ \log_a(u \times v) &= \log_a u + \log_a v\\ \log_a(u^b) &= b \times \log_a u\end{aligned}$$

Game: whose memo trie is it?

$$\begin{aligned}&\textbf{data}\ P\ a = P\ a\ a\\ &\textbf{data}\ S\ a = C\ a\ (S\ a)\\ &\textbf{data}\ T\ a = B\ a\ (P\ (T\ a))\end{aligned}$$

$$\begin{aligned}P\ a &\cong a \times a\\ S\ a &\cong a \times S\ a\\ T\ a &\cong a \times P\ (T\ a)\end{aligned}$$

$$\begin{aligned}&\textbf{data}\ LP = \mathit{False} \mid \mathit{True}\\ &\textbf{data}\ LS = \mathit{Zero} \mid \mathit{Succ}\ LS\\ &\textbf{data}\ LT = \mathit{Empty} \mid \mathit{Dig}\ LT\ LP\end{aligned}$$

$$\begin{aligned}LP &\cong \mathbf{1} + \mathbf{1}\\ LS &\cong \mathbf{1} + LS\\ LT &\cong \mathbf{1} + LT \times LP\end{aligned}$$

Memoization via higher-order types

Functor combinators:

$$
\begin{array}{lll}
\textbf{data} & \mathit{Const}\ b\ a & = \mathit{Const}\ b \\
\textbf{newtype} & \mathit{Id}\ a & = \mathit{Id}\ a \\
\textbf{data} & (f + g)\ a & = \mathit{Sum}\ (f\ a + g\ a) \\
\textbf{data} & (f \times g)\ a & = \mathit{Prod}\ (f\ a \times g\ a) \\
\textbf{newtype} & (g \circ f)\ a & = \mathit{Comp}\ (g\ (f\ a))
\end{array}
$$

Memoization via higher-order types

Functor combinators:

$$
\begin{array}{lllll}
\textbf{data} & \mathit{Const}\ b & a = & \mathit{Const}\ b & \\
\textbf{newtype} & \mathit{Id} & a = & \mathit{Id}\ a & \\
\textbf{data} & (f + g) & a = & \mathit{Sum} & (f\ a + g\ a) \\
\textbf{data} & (f \times g) & a = & \mathit{Prod} & (f\ a \times g\ a) \\
\textbf{newtype} & (g \circ f) & a = & \mathit{Comp} & (g\ (f\ a))
\end{array}
$$

Exponentials:

$$
\begin{array}{lll}
\mathit{Exp}\ \mathbf{0} & = & \mathit{Const}\ \mathbf{1} \\
\mathit{Exp}\ \mathbf{1} & = & \mathit{Id} \\
\mathit{Exp}\ (a + b) & = & \mathit{Exp}\ a \times \mathit{Exp}\ b \\
\mathit{Exp}\ (a \times b) & = & \mathit{Exp}\ a \circ \mathit{Exp}\ b
\end{array}
$$

Memoization via higher-order types

Functor combinators:

$$
\begin{array}{lll}
\textbf{data} & \mathit{Const}\ b\ a & = \mathit{Const}\ b \\
\textbf{newtype} & \mathit{Id}\ a & = \mathit{Id}\ a \\
\textbf{data} & (f + g)\ a & = \mathit{Sum}\ (f\ a + g\ a) \\
\textbf{data} & (f \times g)\ a & = \mathit{Prod}\ (f\ a \times g\ a) \\
\textbf{newtype} & (g \circ f)\ a & = \mathit{Comp}\ (g\ (f\ a))
\end{array}
$$

Exponentials:

$$
\begin{array}{ll}
\mathit{Exp}\ \mathbf{0} & = \mathit{Const}\ \mathbf{1} \\
\mathit{Exp}\ \mathbf{1} & = \mathit{Id} \\
\mathit{Exp}\ (a + b) & = \mathit{Exp}\ a \times \mathit{Exp}\ b \\
\mathit{Exp}\ (a \times b) & = \mathit{Exp}\ a \circ \mathit{Exp}\ b
\end{array}
$$

Logarithms:

$$
\begin{array}{ll}
\mathit{Log}\ (\mathit{Const}\ b) & = \mathbf{0} \\
\mathit{Log}\ \mathit{Id} & = \mathbf{1} \\
\mathit{Log}\ (f \times g) & = \mathit{Log}\ f + \mathit{Log}\ g \\
\mathit{Log}\ (g \circ f) & = \mathit{Log}\ g \times \mathit{Log}\ f
\end{array}
$$

An *almost* beautiful story

- Memoization: *conversion of functions into data structures*.
- Purely functional, directed by type isomorphisms.
- Practical in a non-strict language!
- Simple denotation *and* incremental tabulation.

However, an ironic flaw:

An *almost* beautiful story

- Memoization: *conversion of functions into data structures*.
- Purely functional, directed by type isomorphisms.
- Practical in a non-strict language!
- Simple denotation *and* incremental tabulation.

However, an ironic flaw:

The type isomorphisms only hold for a *strict* language.

Some memoization challenges

- Non-strict
- Higher-order
- Polymorphic
- Deep

References

- Ralf Hinze's paper *Memo functions, polytypically!*.
- These slides
- *MemoTrie*: Hackage, GitHub
- data-memocombinators
- Memoization blog posts

Correctness Proofs

Void

$$
\begin{array}{l}
\textbf{instance } \mathit{HasTrie}\ \mathbf{0}\ \textbf{where} \\
\quad \textbf{type } \mathbf{0} \mapsto a = \mathbf{1} \\
\quad \mathit{trie}\ \mathit{void} = () \\
\quad \mathit{untrie}\ () = \mathit{void}
\end{array}
$$

Void

$$\textbf{instance}\ \mathit{HasTrie}\ \mathbf{0}\ \textbf{where}$$
$$\quad \textbf{type}\ \mathbf{0} \mapsto a = \mathbf{1}$$
$$\quad \mathit{trie}\ \mathit{void} = ()$$
$$\quad \mathit{untrie}\ () = \mathit{void}$$

Laws:

$\mathit{untrie}\ (\mathit{trie}\ \mathit{void})$ $\equiv \mathit{untrie}\ ()$ $\equiv \mathit{void}$	$\mathit{trie}\ (\mathit{untrie}\ ())$ $\equiv \mathit{trie}\ \mathit{void}$ $\equiv ()$

Unit

instance $HasTrie\ \mathbf{1}$ **where**
  type $\mathbf{1} \mapsto a = a$
  $trie\ f = f\ ()$
  $untrie\ a = \lambda() \rightarrow a$

Unit

$$
\begin{array}{l}
\textbf{instance}\ \mathit{HasTrie}\ \mathbf{1}\ \textbf{where} \\
\quad \textbf{type}\ \mathbf{1} \mapsto a = a \\
\quad \mathit{trie}\ f = f\ () \\
\quad \mathit{untrie}\ a = \lambda() \rightarrow a
\end{array}
$$

Laws:

$$
\begin{array}{l}
\quad \mathit{untrie}\ (\mathit{trie}\ f) \\
\equiv \mathit{untrie}\ (f\ ()) \\
\equiv \lambda() \rightarrow f\ () \\
\equiv f
\end{array}
$$

$$
\begin{array}{l}
\quad \mathit{trie}\ (\mathit{untrie}\ a) \\
\equiv \mathit{trie}\ (\lambda() \rightarrow a) \\
\equiv (\lambda() \rightarrow a)\ () \\
\equiv a
\end{array}
$$

Boolean

$$
\begin{array}{l}
\textbf{instance}\ \mathit{HasTrie}\ \mathit{Bool}\ \textbf{where} \\
\quad \textbf{type}\ \mathit{Bool} \mapsto a = a \times a \\
\quad \mathit{trie}\ f = (f\ \mathit{False}, f\ \mathit{True}) \\
\quad \mathit{untrie}\ (x, y) = \lambda c \rightarrow \textbf{if}\ c\ \textbf{then}\ y\ \textbf{else}\ x
\end{array}
$$

Boolean

$$\begin{array}{l}\textbf{instance}\ \mathit{HasTrie}\ \mathit{Bool}\ \textbf{where}\\ \quad \textbf{type}\ \mathit{Bool} \mapsto a = a \times a\\ \quad \mathit{trie}\ f = (f\ \mathit{False}, f\ \mathit{True})\\ \quad \mathit{untrie}\ (x, y) = \lambda c \rightarrow \textbf{if}\ c\ \textbf{then}\ y\ \textbf{else}\ x\end{array}$$

$\begin{array}{l}\quad \mathit{untrie}\ (\mathit{trie}\ f)\\ \equiv \mathit{untrie}\ (f\ \mathit{False}, f\ \mathit{True})\\ \equiv \mathit{if}'\ (f\ \mathit{False})\ (f\ \mathit{True})\\ \equiv f\end{array}$	$\begin{array}{l}\quad \mathit{trie}\ (\mathit{untrie}\ (x, y))\\ \equiv \mathit{trie}\ (\mathit{if}'\ x\ y)\\ \equiv (\mathit{if}'\ x\ y\ \mathit{False}, \mathit{if}'\ y\ x\ \mathit{True})\\ \equiv (x, y)\end{array}$

Note:

$$\begin{array}{l}\quad \mathit{if}'\ (f\ \mathit{False})\ (f\ \mathit{True})\\ \equiv \lambda c \rightarrow \textbf{if}\ c\ \textbf{then}\ f\ \mathit{True}\ \textbf{else}\ f\ \mathit{False}\\ \equiv \lambda c \rightarrow \textbf{if}\ c\ \textbf{then}\ f\ c\ \textbf{else}\ f\ c\\ \equiv f\end{array}$$

Sums

$$\textbf{instance}\ (\mathit{HasTrie}\ b, \mathit{HasTrie}\ c) \Rightarrow \mathit{HasTrie}\ (b + c)\ \textbf{where}$$
$$\quad \textbf{type}\ (b + c) \mapsto a = (b \mapsto a) \times (c \mapsto a)$$
$$\quad \mathit{trie}\ f = (\mathit{trie}\ (f \circ \mathit{Left}), \mathit{trie}\ (f \circ \mathit{Right}))$$
$$\quad \mathit{untrie}\ (s, t) = \mathit{untrie}\ s\ |||\ \mathit{untrie}\ t$$

where

$$(g\ |||\ h)\ (\mathit{Left}\ \ b) = g\ b$$
$$(g\ |||\ h)\ (\mathit{Right}\ c) = h\ c$$

Sums

$$
\begin{array}{l}
\textbf{instance}\ (\mathit{HasTrie}\ b, \mathit{HasTrie}\ c) \Rightarrow \mathit{HasTrie}\ (b + c)\ \textbf{where} \\
\quad \textbf{type}\ (b + c) \mapsto a = (b \mapsto a) \times (c \mapsto a) \\
\quad \mathit{trie}\ f = (\mathit{trie}\ (f \circ \mathit{Left}), \mathit{trie}\ (f \circ \mathit{Right})) \\
\quad \mathit{untrie}\ (s, t) = \mathit{untrie}\ s\ |||\ \mathit{untrie}\ t
\end{array}
$$

where

$$
\begin{array}{l}
(g\ |||\ h)\ (\mathit{Left}\ \ \ b) = g\ b \\
(g\ |||\ h)\ (\mathit{Right}\ c) = h\ c
\end{array}
$$

$$
\begin{array}{l}
\quad \mathit{untrie}\ (\mathit{trie}\ f) \\
\equiv \mathit{untrie}\ (\mathit{trie}\ (f \circ \mathit{Left}), \mathit{trie}\ (f \circ \mathit{Right})) \\
\equiv \mathit{untrie}\ (\mathit{trie}\ (f \circ \mathit{Left}))\ |||\ \mathit{untrie}\ (\mathit{trie}\ (f \circ \mathit{Right})) \\
\equiv f \circ \mathit{Left}\ |||\ f \circ \mathit{Right} \\
\equiv f
\end{array}
$$

Sums

$$
\begin{array}{l}
\textbf{instance}\ (HasTrie\ b, HasTrie\ c) \Rightarrow HasTrie\ (b + c)\ \textbf{where} \\
\quad \textbf{type}\ (b + c) \mapsto a = (b \mapsto a) \times (c \mapsto a) \\
\quad trie\ f = (trie\ (f \circ Left), trie\ (f \circ Right)) \\
\quad untrie\ (s, t) = untrie\ s\ |||\ untrie\ t
\end{array}
$$

where

$$
\begin{array}{l}
(g\ |||\ h)\ (Left\ \ \ b) = g\ b \\
(g\ |||\ h)\ (Right\ c) = h\ c
\end{array}
$$

$$
\begin{array}{ll}
& trie\ (untrie\ (s, t)) \\
\equiv & trie\ (untrie\ s\ |||\ untrie\ t) \\
\equiv & (trie\ ((untrie\ s\ |||\ untrie\ t) \circ Left) \\
& , trie\ ((untrie\ s\ |||\ untrie\ t) \circ Right)) \\
\equiv & (trie\ (untrie\ s), trie\ (untrie\ t)) \\
\equiv & (s, t)
\end{array}
$$

Products

$$
\begin{array}{l}
\textbf{instance}\ (\mathit{HasTrie}\ b, \mathit{HasTrie}\ c) \Rightarrow \mathit{HasTrie}\ (b \times c)\ \textbf{where} \\
\quad \textbf{type}\ (b \times c) \mapsto a = b \mapsto (c \mapsto a) \\
\quad \mathit{trie}\ f = \mathit{trie}\ (\mathit{trie} \circ \mathit{curry}\ f) \\
\quad \mathit{untrie}\ t = \mathit{uncurry}\ (\mathit{untrie} \circ \mathit{untrie}\ t)
\end{array}
$$

where

$$
\begin{array}{lll}
\mathit{curry} & g\ b\ c & = g\ (b, c) \\
\mathit{uncurry} & h\ (b, c) & = h\ b\ c
\end{array}
$$

Products

$$\textbf{instance}\ (\mathit{HasTrie}\ b, \mathit{HasTrie}\ c) \Rightarrow \mathit{HasTrie}\ (b \times c)\ \textbf{where}$$
$$\quad \textbf{type}\ (b \times c) \mapsto a = b \mapsto (c \mapsto a)$$
$$\quad \mathit{trie}\ f = \mathit{trie}\ (\mathit{trie} \circ \mathit{curry}\ f)$$
$$\quad \mathit{untrie}\ t = \mathit{uncurry}\ (\mathit{untrie} \circ \mathit{untrie}\ t)$$

where

$$\mathit{curry}\quad g\ b\ c \quad = g\ (b, c)$$
$$\mathit{uncurry}\ h\ (b, c) = h\ b\ c$$

$$\begin{aligned}
&\quad \mathit{untrie}\ (\mathit{trie}\ f) \\
&\equiv \mathit{untrie}\ (\mathit{trie}\ (\mathit{trie} \circ \mathit{curry}\ f)) \\
&\equiv \mathit{uncurry}\ (\mathit{untrie} \circ \mathit{untrie}\ (\mathit{trie}\ (\mathit{trie} \circ \mathit{curry}\ f))) \\
&\equiv \mathit{uncurry}\ (\mathit{untrie} \circ \mathit{trie} \circ \mathit{curry}\ f) \\
&\equiv \mathit{uncurry}\ (\mathit{curry}\ f) \\
&\equiv f
\end{aligned}$$

Products

$$\mathbf{instance}\ (HasTrie\ b, HasTrie\ c) \Rightarrow HasTrie\ (b \times c)\ \mathbf{where}$$
$$\quad \mathbf{type}\ (b \times c) \mapsto a = b \mapsto (c \mapsto a)$$
$$\quad trie\ f = trie\ (trie \circ curry\ f)$$
$$\quad untrie\ t = uncurry\ (untrie \circ untrie\ t)$$

where

$$curry\ \ g\ b\ c \quad = g\ (b, c)$$
$$uncurry\ h\ (b, c) = h\ b\ c$$

$$\begin{aligned}
&\ \ trie\ (untrie\ t) \\
&\equiv trie\ (uncurry\ (untrie \circ untrie\ t)) \\
&\equiv trie\ (trie \circ curry\ (uncurry\ (untrie \circ untrie\ t))) \\
&\equiv trie\ (trie \circ untrie \circ untrie\ t) \\
&\equiv trie\ (untrie\ t) \\
&\equiv t
\end{aligned}$$