A coincidence of partial and total correctness: Intrinsically Correct Sorting with a slice of Cubical Agda

Cass Alexandru[1] Vikraman Choudhury[2] Jurriaan Rot[3] Niels van der Weide[3]

[1]RPTU Kaiserslautern-Landau

[2]University of Bologna & INRIA OLAS

[3]Radboud University Nijmegen

Certified Programs and Proofs 2025

Motivation

- “Sorting with Bialgebras and Distributive Laws” (Hinze et al. 2012)

Motivation

- “Sorting with Bialgebras and Distributive Laws” (Hinze et al. 2012)
- Bialgebraic semantics (Turi and Plotkin 1997)

Motivation

- "Sorting with Bialgebras and Distributive Laws" (Hinze et al. 2012)
- Bialgebraic semantics (Turi and Plotkin 1997)
- Intrinsic Correctness: Type and specification (predicates), program and proof are intertwined, Cubical Agda: Dependent Types & Path types

Motivation

- “Sorting with Bialgebras and Distributive Laws” (Hinze et al. 2012)
- Bialgebraic semantics (Turi and Plotkin 1997)
- Intrinsic Correctness: Type and specification (predicates), program and proof are intertwined, Cubical Agda: Dependent Types & Path types
- Contribution: intrinsic verification of business logics & setting in which correctness of the dual algorithms follows

Motivation

- “Sorting with Bialgebras and Distributive Laws” (Hinze et al. 2012)
- Bialgebraic semantics (Turi and Plotkin 1997)
- Intrinsic Correctness: Type and specification (predicates), program and proof are intertwined, Cubical Agda: Dependent Types & Path types
- Contribution: intrinsic verification of business logics & setting in which correctness of the dual algorithms follows
- Key idea: Index data by the multiset of their elements

Outline

Outline

Specification of Sorting

- Totally ordered Carrier Set A

Specification of Sorting

- Totally ordered Carrier Set A
- sort : List A $\to$ List A ?

Specification of Sorting

- Totally ordered Carrier Set A
- sort : List A $\to$ List A ?
- sort : List A $\to$ Ordered List A?

Specification of Sorting

- Totally ordered Carrier Set A
- sort : List A $\to$ List A ?
- sort : List A $\to$ Ordered List A?
- "The output should be a permutation of the input"

Specification of Sorting

- Totally ordered Carrier Set A
- sort : List A $\to$ List A ?
- sort : List A $\to$ Ordered List A?
- "The output should be a permutation of the input"
  - "Mapping a list to the multiset of its elements is invariant under sorting"

$$\begin{tikzcd}
\text{List A} \arrow[rr, "\text{sort}"] \arrow[dr, "\text{elements}"'] & & \text{Ordered List A} \arrow[dl, "\text{elements}"] \\
& \text{Multiset A} &
\end{tikzcd}$$

Specification of Sorting

- Totally ordered Carrier Set A
- sort : List A $\to$ List A ?
- sort : List A $\to$ Ordered List A?
- “The output should be a permutation of the input”
  - “Mapping a list to the multiset of its elements is invariant under sorting”

$$
\begin{tikzcd}
\text{List A} \arrow[rr, "\text{sort}"] \arrow[dr, "\text{elements}"'] & & \text{Ordered List A} \arrow[dl, "\text{elements}"] \\
& \text{Multiset A} &
\end{tikzcd}
$$

  - Intrinsically: “Sorting is an index-preserving map between lists and ordered lists indexed by the finite multiset of their elements”

Specification of Sorting

- Totally ordered Carrier Set A
- sort : List A → List A ?
- sort : List A → Ordered List A?
- "The output should be a permutation of the input"
  - "Mapping a list to the multiset of its elements is invariant under sorting"

$$\begin{tikzcd} \text{List A} \arrow[rr, "\text{sort}"] \arrow[dr, "\text{elements}"'] & & \text{Ordered List A} \arrow[dl, "\text{elements}"] \\ & \text{Multiset A} & \end{tikzcd}$$

  - Intrinsically: "Sorting is an index-preserving map between lists and ordered lists indexed by the finite multiset of their elements"

$$\{g : \mathsf{FMSet}\ A\} \longrightarrow \mathsf{ElList}\ g \longrightarrow \mathsf{OElList}\ g$$

Outline

Outline

Base Functors of Recursive Datatypes

- Recursive datatypes have a shape given by a **base functor** F
- E.g. Natural numbers: $(1 + -)$. Lists of element type A: $(1 + A \times -)$.
- Recursive datatype is given by fixpoint of composition of base functor F with itself
- Least fixpoint (μF): Inductive datatype. Greatest (νF): coinductive – not neccessarily well founded

Outline

Maps Between Recursive Datatypes

- Rec F $\longrightarrow$ Rec G

Maps Between Recursive Datatypes

- Rec F ⟶ Rec G
- Algebraically: fold alg where alg : F (Rec G) ⟶ Rec G

Maps Between Recursive Datatypes

- $\mathsf{Rec\ F} \longrightarrow \mathsf{Rec\ G}$
- Algebraically: $\mathsf{fold\ alg}$ where $\mathsf{alg} : \mathsf{F\ (Rec\ G)} \longrightarrow \mathsf{Rec\ G}$
- Coalgebraically: $\mathsf{unfold\ coalg}$ where $\mathsf{coalg} : \mathsf{Rec\ F} \longrightarrow \mathsf{G\ (Rec\ F)}$

Maps Between Recursive Datatypes

- Rec F ⟶ Rec G
- Algebraically: fold alg where alg : F (Rec G) ⟶ Rec G
- Coalgebraically: unfold coalg where coalg : Rec F ⟶ G (Rec F)
- A way that gives us both...

Insertion- / Bubble Sort

(Hinze et al. 2012)

```
data L (r : Type) : Type where            -- aliasing
  []  : L r                               O = L
  _::_ : A → r → L r                      pattern _≤::_ x xs = x :: xs
```

```
swap : ∀ {x} → L (O x) → O (L x)
swap [] = []
swap (a :: []) =  a ≤:: []
swap (a :: (b ≤:: r)) with a ≤?≥ b
...| inl  a≤b  =  a  ≤:: ( b  ::  r )
...| inr  b≤a  =  b  ≤:: ( a  ::  r )
```

$$\mathsf{insertSort} = \mathsf{fold}\ (\ \mathsf{unfold}\ \ (\mathsf{swap} \circ \mathsf{L}_1\ \mathsf{out}))$$

$$\mathsf{bubbleSort} = \ \mathsf{unfold}\ \ (\mathsf{fold}\ (\mathsf{O}_1\ \mathsf{in} \circ \mathsf{swap}))$$

Insertion- / Bubble Sort

(Hinze et al. 2012)

```
data L (r : Type) : Type where
  []   : L r
  _::_ : A → r → L r

-- aliasing
O = L
pattern _≤::_ x xs = x :: xs
```

```
swap : ∀ {x} → L (O x) → O (L x)
swap [] = []
swap (a :: []) =  a ≤:: []
swap (a :: (b ≤:: r)) with a ≤?≥ b
...| inl  a≤b  =  a  ≤:: ( b  ::  r )
...| inr  b≤a  =  a  ≤:: ( b  ::  r )
```

```
insertSort = fold ( unfold  (swap ∘ L₁ out))
bubbleSort =  unfold  (fold (O₁ in ∘ swap))
```

Insertion- / Bubble Sort

(Hinze et al. 2012)

```
data L (r : Type) : Type where
  []   : L r
  _::_ : A → r → L r

-- aliasing
O = L
pattern _≤::_ x xs = x :: xs

swap : ∀ {x} → L (O x) → O (L x)
swap [] = []
swap (a :: []) = []
swap (a :: (b ≤:: r)) with a ≤?≥ b
...| inl  a≤b  =  a  ≤:: ( a  ::  r )
...| inr  b≤a  =  a  ≤:: ( b  ::  r )

insertSort = fold ( unfold  (swap ∘ L₁ out))
bubbleSort =  unfold  (fold (O₁ in ∘ swap))
```

Insertion- / Bubble Sort

(Hinze et al. 2012)

```
data L (r : Type) : Type where          -- aliasing
  []   : L r                            O = L
  _::_ : A → r → L r                    pattern _≤::_ x xs = x :: xs
```

```
swap : ∀ {x} → L (O x) → O (L x)
swap [] = []
swap (a :: []) = []
swap (a :: (b ≤:: r)) with a ≤?≥ b
...| inl  a≤b  = a ≤:: ( a :: r )
...| inr  b≤a  = a ≤:: ( b :: r )
```

```
insertSort = fold ( unfold (swap ∘ L₁ out))
bubbleSort = unfold (fold (O₁ in ∘ swap))
```

Outline

Outline

The Finite Multiset Quotient Inductive Type

(Choudhury and Fiore 2023)

```
data FMSet (A : Type ℓ) : Type ℓ where
  []     : FMSet A
  _::_   : (x : A) → (xs : FMSet A) → FMSet A
  comm : ∀ {x y xs} → x :: y :: xs ≡ y :: x :: xs
  trunc  : isSet (FMSet A)
```

The Finite Multiset Quotient Inductive Type
(Choudhury and Fiore 2023)

```
data FMSet (A : Type ℓ) : Type ℓ where
  []     : FMSet A
  _::_   : (x : A) → (xs : FMSet A) → FMSet A
  comm : ∀ {x y xs} → x :: y :: xs ≡ y :: x :: xs
  trunc  : isSet (FMSet A)

1 :: 2 :: 3 :: [] ≡⟨ cong (1 ::_) comm ⟩ 1 :: 3 :: 2 :: []
```

The Finite Multiset Quotient Inductive Type

(Choudhury and Fiore 2023)

```
data FMSet (A : Type ℓ) : Type ℓ where
  []     : FMSet A
  _::_   : (x : A) → (xs : FMSet A) → FMSet A
  comm : ∀ {x y xs} → x :: y :: xs ≡ y :: x :: xs
  trunc  : isSet (FMSet A)

1 :: 2 :: 3 :: [] ≡⟨ cong (1 ::_) comm ⟩ 1 :: 3 :: 2 :: []

pattern []𝓜 = []
pattern _::𝓜_ x xs = x :: xs
```

Base Functors for (Ordered) Element-Indexed Lists

```
data L (r : Type) : Type where
  []   : L r
  _::_ : A → r → L r
```

Base Functors for (Ordered) Element-Indexed Lists

$$
\begin{array}{llllll}
\textbf{data}\ \mathsf{L} & (r : \mathsf{Type}) & : & \mathsf{Type} & & \textbf{where} \\
\quad [] & : \mathsf{L}\ r & & & & \\
\quad \_::\_ & : A & \longrightarrow & r & \longrightarrow & \mathsf{L}\ r
\end{array}
$$

$$
\begin{array}{llllll}
\textbf{data}\ \mathsf{L} & (r : \boxed{\mathsf{FMSet}\ A \longrightarrow}\ \mathsf{Type}) & : & \boxed{\mathsf{FMSet}\ A \longrightarrow}\ \mathsf{Type} & & \textbf{where} \\
\quad [] & : \mathsf{L}\ r\ \boxed{[]\mathcal{M}} & & & & \\
\quad \_::\_ & : \boxed{\forall \{g\} \longrightarrow}\ (x : A) & \longrightarrow & (r\ \boxed{g}) & \longrightarrow & \mathsf{L}\ r\ \boxed{(x :: \mathcal{M}\ g)}
\end{array}
$$

Base Functors for (Ordered) Element-Indexed Lists

$$
\begin{array}{llllll}
\textsf{data L} & (r : \textsf{Type}) & : & \textsf{Type} & & \textsf{where} \\
\quad [] & : \textsf{L}\ r & & & & \\
\quad \_::\_ & : A & \longrightarrow & r & \longrightarrow \textsf{L}\ r &
\end{array}
$$

$$
\begin{array}{llllll}
\textsf{data L} & (r : \textsf{FMSet}\ A \longrightarrow \textsf{Type}) & : & \textsf{FMSet}\ A \longrightarrow \textsf{Type} & & \textsf{where} \\
\quad [] & : \textsf{L}\ r\ []\mathcal{M} & & & & \\
\quad \_::\_ & : \forall \{g\} \longrightarrow (x : A) & \longrightarrow & (r\ g) & \longrightarrow \textsf{L}\ r\ (x ::\mathcal{M}\ g) &
\end{array}
$$

$$
\begin{array}{llllll}
\textsf{data O} & (r : \textsf{FMSet}\ A \longrightarrow \textsf{Type}) & : & \textsf{FMSet}\ A \longrightarrow \textsf{Type} & & \textsf{where} \\
\quad [] & : \textsf{O}\ r\ []\mathcal{M} & & & & \\
\quad \_\leq::\_ & : \forall \{g\}\ (x : A) \longrightarrow (r\ g) \longrightarrow \textsf{All}\ (x \leq\_)\ g \longrightarrow \textsf{O}\ r\ (x ::\mathcal{M}\ g) & & & &
\end{array}
$$

Outline

The FMSet Index as a Termination Measure

L-coalgebras are well founded

```
pattern _::_^_ x xs g = _::_ {g = g} x xs

unfoldL : {r : FMSet A → Type} →
  ( ∀ {gᵣ } → (r  gᵣ ) → L r  gᵣ  ) → (∀ {g} → (r g) → ElList g)
unfoldL grow {_}              seed with  grow seed
```

The FMSet Index as a Termination Measure

L-coalgebras are well founded

```
pattern _::_^_ x xs g = _::_ {g = g} x xs

unfoldL : {r : FMSet A → Type} →
  ( ∀ {g_r } → (r  g_r ) → L r  g_r  ) → (∀ {g} → (r g) → ElList g)
unfoldL grow {_}             seed with  grow seed
```

- Index-preservation forces index of **seed** and **grow seed** to coincide

The FMSet Index as a Termination Measure

L-coalgebras are well founded

```
pattern _::_^_ x xs g = _::_ {g = g} x xs


unfoldL : {r : FMSet A → Type} →
  ( ∀ {gr } → (r  gr ) → L r  gr  ) → (∀ {g} → (r g) → ElList g)
unfoldL grow {_}             seed with  grow seed
unfoldL grow  .{[]M}          _     |  []                =
unfoldL grow  .{x ::M g'} _     |  x :: seed′ ^  g'  =
```

- Index-preservation forces index of **seed** and **grow seed** to coincide
- **with**-abstraction: Pattern matching refines earlier arguments, propagates information about the indexee back to the index

The FMSet Index as a Termination Measure

L-coalgebras are well founded

```
pattern _::_^_ x xs g = _::_ {g = g} x xs


unfoldL : {r : FMSet A → Type} →
  (∀ {gᵣ} → (r  gᵣ) → L r  gᵣ ) → (∀ {g} → (r g) → ElList g)
unfoldL grow {_}              seed with grow seed
unfoldL grow .{[]ℳ}           _     | []                 = []
unfoldL grow .{x ::ℳ g'}      _     | x :: seed′ ^ g'   =
                                 x :: unfoldL grow {g'} seed′
```

- Index-preservation forces index of **seed** and **grow seed** to coincide
- **with**-abstraction: Pattern matching refines earlier arguments, propagates information about the indexee back to the index
- Index of recursive argument is smaller

Well Founded Recursion

- Syntactic termination checking based on dot-patterns of HITs inconsistent (Pitts 2020)

Well Founded Recursion

- Syntactic termination checking based on dot-patterns of HITs inconsistent (Pitts 2020)
- Define a family of maps indexed by FMSet A by well founded induction on the length of the index

Well Founded Recursion

- Syntactic termination checking based on dot-patterns of HITs inconsistent (Pitts 2020)
- Define a family of maps indexed by FMSet A by well founded induction on the length of the index
- length defined by eliminating from FMSet A as the free commutative monoid to $(\mathbb{N}\ ,\ +)$ by $\lambda\ a \longrightarrow 1$

swap, Revisited

```
swap : {r : FMSet A → Type} {g : FMSet A} →
  L (O r) g → O (L r) g
swap [] = []
swap (a ∷ []) = (a ≤∷ []) []-A
swap (a ∷ (b ≤∷ r) a≤#r) with a ≤?≥ b
...| inl a≤b = (a ≤∷ (b ∷ r)) $ a≤b ≤∷# a≤#r
...| inr b≤a = (b ≤∷ (a ∷ r)) $ b≤a ∷-A a≤#r €
  subst (O (L _)) comm
```

swap, Revisited

```
swap : {r : FMSet A → Type} {g : FMSet A} →
  L (O r) g → O (L r) g
swap [] = []
swap (a :: []) = (a ≤:: []) []-A
swap (a :: (b ≤:: r) a≤#r) with a ≤?≥ b
...| inl a≤b = (a ≤:: (b :: r)) $ a≤b ≤::# a≤#r
...| inr b≤a = (b ≤:: (a :: r)) $ b≤a ::-A a≤#r €
  subst (O (L _)) comm
```

- Evaluation of subst ...?

swap, Revisited

```
swap : {r : FMSet A → Type} {g : FMSet A} →
  L (O r) g → O (L r) g
swap [] = []
swap (a ∷ []) = (a ≤∷ []) []-A
swap (a ∷ (b ≤∷ r) a≤#r) with a ≤?≥ b
...| inl a≤b = (a ≤∷ (b ∷ r)) $ a≤b ≤∷# a≤#r
...| inr b≤a = (b ≤∷ (a ∷ r)) $ b≤a ∷-A a≤#r €
  subst (O (L _)) comm
```

- Evaluation of subst ...?
- transpX-O (λ n → ...) i0 ... (Cavallo and Harper 2019)

swap, Revisited

```
swap : {r : FMSet A → Type} {g : FMSet A} →
  L (O r) g → O (L r) g
swap [] = []
swap (a :: []) = (a ≤:: []) []-A
swap (a :: (b ≤:: r) a≤#r) with a ≤?≥ b
...| inl a≤b = (a ≤:: (b :: r)) $ a≤b ≤::# a≤#r
...| inr b≤a = (b ≤:: (a :: r)) $ b≤a ::-A a≤#r €
  subst (O (L _)) comm
```

- Evaluation of subst ...?
- transpX-O (λ n → ...) i0 ... (Cavallo and Harper 2019)
- Discard index with toList : {g : FMSet A} → OEIList g → List A

Outline

Conclusion

- Indexing by FMSet allowed expressing orderedness, element-preservation & acted as termination measure

Conclusion

- Indexing by FMSet allowed expressing orderedness, element-preservation & acted as termination measure
- Intrinsically correct algorithms from correct distr. law

Conclusion

- Indexing by FMSet allowed expressing orderedness, element-preservation & acted as termination measure
- Intrinsically correct algorithms from correct distr. law
- For verified quick/treesort & heapsort following (Hinze et al. 2012), semantics via slice category $\rightarrow$ see paper

Future Work

- Conditions under which coalgebras are recursive in an indexed/fibered setting

Future Work

- Conditions under which coalgebras are recursive in an indexed/fibered setting
- More algorithms to verify with a distributive law as business logic

Well-foundedness of O-Coalgebras

$$
\begin{aligned}
c &: (X, g) \to O(X, g) \\
c^n &: X \to 1 + X \\
c^0(x) &:= \mathsf{inr}(x) \\
c^{n+1}(x) &:= \begin{cases} \mathsf{inl}(\star) & c^n(x) = \mathsf{inl}(\star) \\ c(y) & c^n(x) = \mathsf{inr}(y) \end{cases}
\end{aligned}
$$

$$
\begin{tikzcd}
X \arrow[rd, "g"] \arrow[d, "c"'] & \\
OX \arrow[r, "{[\emptyset, \cup \circ (\eta \times g)]}"'] & \mathcal{M}(A)
\end{tikzcd}
$$

- c well-founded: $\forall x \in X.\, \exists n.\, c^n(x) = \mathsf{inl}(\star)$
- Idea: Use g as a ranking function into well-order $(\mathcal{M}(A), \subset)$
- Case $c(x) = \mathsf{inr}(a, r)$: $g(x) = \eta(a) \cup g(r) \Rightarrow g(r) \subset g(x)$ $\square$

Slice Topos over a Monoid object

$L_{\mathsf{elt}}(X, g)$:

$$\begin{tikzcd} 1 + A \times X \arrow[d, "Lg"] \\ 1 + A \times \mathcal{M}(A) \arrow[d, "{[\emptyset, \cup\circ(\eta\times\mathrm{id})]}"] \\ \mathcal{M}(A) \end{tikzcd}$$

$$\begin{tikzcd} 1 \arrow[ddr, "\emptyset"'] & \oplus & A \arrow[d, "\eta"'] & \otimes & X \arrow[dll, "g"] \\ & & \mathcal{M}(A)^2 \arrow[dl, "\cup"] & & \\ & \mathcal{M}(A) & & & \end{tikzcd}$$

$$\hat{1},\ \hat{A} : \mathsf{Set}/\mathcal{M}(A)$$
$$\hat{1} := (1, \emptyset)$$
$$\hat{A} := (A, \eta)$$
$$L_{\mathsf{elt}}(g) \simeq \hat{1} \oplus \hat{A} \otimes g$$

$$\begin{aligned} \oplus \quad &: (\mathsf{Set}/\mathcal{M}(A))^2 \to \mathsf{Set}/\mathcal{M}(A) \\ g \oplus h \quad &:= [g, h] \\ \otimes \quad &: (\mathsf{Set}/\mathcal{M}(A))^2 \to \mathsf{Set}/\mathcal{M}(A) \\ g \otimes h \quad &:= \cup \circ (g \times h) \end{aligned}$$