A Syntactic Model of Sized Dependent Types*

•••

Jonathan Chan
University of British Columbia
18 July 2022

*MSc thesis: *Sized Dependent Types via Extensional Type Theory*

A Syntactic Model of Sized $\underbrace{\text{Dependent Types}}$

for stating and proving properties of
programs within the language itself

A Syntactic Model of Sized Dependent Types

$$\underbrace{\text{Sized}}_{\substack{\text{for checking validity}\\ \text{of recursive functions}}}\ \underbrace{\text{Dependent Types}}_{\substack{\text{for stating and proving properties of}\\ \text{programs within the language itself}}}$$

for proving consistency of the type theory

A Syntactic Model of Sized Dependent Types

$$\text{A }\overbrace{\text{Syntactic Model}}^{\text{for proving consistency of the type theory}}\text{ of}$$

$$\underbrace{\text{Sized}}_{\text{for checking validity of recursive functions}}\ \underbrace{\text{Dependent Types}}_{\text{for stating and proving properties of programs within the language itself}}$$

for checking validity
of recursive functions

for stating and proving properties of
programs within the language itself

Outline

for the next hour

① Crash course on dependent types
② Introduction to sized types
③ How to model syntactically
④ Shortcomings and future work

① Dependent types ≔ types that abstract over (“depend on”) terms & types

The Curry–Howard Correspondence (1)

Type system	Logical system
● dependent type theory	● predicate logic
● types	● propositions
● terms	● proofs
● type checking	● automated proof checking

The Curry–Howard Correspondence (2)

Type system	Logical system
`Πx: A. B x`	for all x in A, B(x) holds
`ΠP: Prop. P` (or ⊥)	falsehood
`Π_: A. B` (or A `→ B`)	A implies B
`A → ⊥`	negation of A
`ΠP: Prop. P → P` (or ⊤)	truthhood

…and existential quantification, conjunction, disjunction, equality, etc.

The Curry–Howard Correspondence (3)

Type system	**Logical system**
There exists no term e of type $\bot$	Logical consistency
(not all types are inhabited)	(not all propositions are provable)

Beyond Propositions

```
data nat: 𝒰 ≔
  zero: nat
  succ: nat → nat
```

Peano naturals (not a proposition)

```
fix _≤_: nat → nat → Prop
zero   ≤ _      ≔ ⊤
succ n ≤ succ m ≔ n ≤ m
succ _ ≤ zero   ≔ ⊥
```

Total order on `nat` (a proposition)

Recursive Functions (1)

```
fix _≤_: nat → nat → Prop
zero   ≤ _      ≔ ⊤
succ n ≤ succ m ≔ n ≤ m
succ _ ≤ zero   ≔ ⊥
```

```
fix ng: ΠP: Prop. P
ng P ≔ ng P
```

Recursive Functions (1)

```
fix _≤_: nat → nat → Prop
zero   ≤ _       ≔ ⊤
succ n ≤ succ m  ≔ n ≤ m
succ _ ≤ zero    ≔ ⊥
```

✅ recursive call on syntactic subargument

```
fix ng: ΠP: Prop → P
ng P ≔ ng P
```

❌ recursive call on own argument

Recursive Functions (2)

```
fix monus: nat → nat →
nat
monus n zero ≔ n
monus zero _ ≔ zero
monus (succ n) (succ m) ≔
  monus n m
```

monus(n, m) = max(0, n - m)

```
fix div: nat → nat → nat
div zero     _ ≔ zero
div (succ n) m ≔
  succ (div (monus n m) m)
```

div(n, m) = ⌈n/(m+1)⌉

Recursive Functions (2)

```
fix monus: nat → nat →
nat
monus n zero ≔ n
monus zero _ ≔ zero
monus (succ n) (succ m) ≔
   monus n m
```

✅ recursive call on syntactic subargument

```
fix div: nat → nat → nat
div zero       _ ≔ zero
div (succ n) m ≔
   succ (div (monus n m) m)
```

❌ recursive call on *function call on* syntactic subargument

② Sized types ≔ inductive types annotated with constr. depth (“size”)

Sized Types

additional size information

```
data nat [s]: 𝒰 ≔
  zero: ∀ α < s. nat [s]
  succ: ∀ α < s. nat [α] → nat [s]
```

larger size

Sized Types

```
data nat [s]: 𝒰 ≔
  zero: ∀ α < s. nat [s]
  succ: ∀ α < s. nat [α] → nat [s]

fix monus:              nat       → nat       → nat

fix div:             nat       → nat       → nat
div          (zero     )    _ ≔ zero
div          (succ      n) m ≔
  succ      (div          (monus          n m) m)
```

Sized Types

```
data nat [s]: 𝒰 ≔
  zero: ∀ α < s. nat [s]
  succ: ∀ α < s. nat [α] → nat [s]

fix monus: ∀α. ∀β. nat [α] → nat [β] → nat [α]

fix div: ∀α. ∀β. nat [α] → nat [β] → nat [α]
div [α] [β] (zero [γ])   _ ≔ zero [γ]
div [α] [β] (succ [γ] n) m ≔
  succ [γ] (div [γ] [β] (monus [γ] [β] n m) m)
```

recursion on smaller size γ < α

Sized Recursive Functions (Fixpoint Expressions)

$$\frac{\alpha \vdash \tau : \mathcal{U} \qquad \alpha;\ f : \forall \beta < \alpha.\ \tau[\alpha \mapsto \beta] \vdash e : \tau}{\vdash \mathtt{fix}\ f\ [\alpha] : \tau \coloneqq e : \forall \alpha.\ \tau}$$

Past Work on Sized Dependent Types

Past work
Barthe et al. (2006), Grégoire et al. (2010), Sacchini (2011, 2013), etc.
Abel et al. (2017)
MiniAgda (2010, 2012), Agda
This thesis! **STT** (2022)

Past Work on Sized Dependent Types

Past work
Barthe et al. (2006), Grégoire et al. (2010), Sacchini (2011, 2013), etc.
Abel et al. (2017)
MiniAgda (2010, 2012), Agda
This thesis! **STT** (2022)

Bounded sizes: `∀α < s. τ`

- Elegant interpretation of fixpoint expressions
- Otherwise: fixpoint types must be "semi-continuous" in recursion size
- Implemented in Agda

Past Work on Sized Dependent Types

Past work
Barthe et al. (2006), Grégoire et al. (2010), Sacchini (2011, 2013), etc.
Abel et al. (2017)
MiniAgda (2010, 2012), Agda
This thesis! **STT** (2022)

Higher-rank sizes: `(∀α. σ) → τ`

- Greater expressivity
- Can pass around size-preserving functions
- Implemented in Agda

Past Work on Sized Dependent Types

	$(\forall\ \alpha.\ \sigma) \to \tau$	$\forall\ \alpha < s.\ \tau$	$\nvdash e : \bot$
Past work	**Higher-rank**	**Bounded**	**Consistent**
Barthe et al. (2006), Grégoire et al. (2010), Sacchini (2011, 2013), etc.	❌	❌	✅
Abel et al. (2017)	✅	❌	✅
MiniAgda (2010, 2012), Agda	✅	✅	❌
This thesis! **STT** (2022)	✅	✅	✅

③ Syntactic model ≔

type-preserving translation

into a consistent language

Translation from STT to Target Type Theory

$\Phi; \Gamma \vdash e : \tau \rightsquigarrow e$: typing derivation $\to$ term

$[\![e]\!]$: (well-typed) term $\to$ term

Type Preservation and Consistency

Recall: $\bot \coloneqq \Pi P : \texttt{Prop}.\ P$

- **Postulate** (consistency).
  There is no term e such that $\vdash e : \bot$.
- **Theorem** (type preservation).
  If $\Phi;\ \Gamma \vdash e : \tau \rightsquigarrow e$, then $[\![\Phi]\!], [\![\Gamma]\!] \vdash e : [\![\tau]\!]$.
- **Corollary** (consistency).
  There is no term e such that $\vdash e : \bot$.
  Proof: If $\vdash e : \bot$, then $\vdash [\![e]\!] : \bot$ by type preservation, contradicting consistency.

Translation Details

Terms in STT		Terms in CIC_E
Size expressions		Elements of inductive type `Size`
Order on sizes	translate to $\rightsquigarrow$	`_<_: Size → Size → Prop`
Fixpoints on sizes		Uses of well-founded induction on `Size`, `<`

…Well-Founded Induction Principle

```
wfInd : ΠP: Size → 𝒰.
```

Let P be a predicate on Sizes.

```
(Πα: Size. (Πβ: Size. β < α → P β) → P α)
```

Given that P holds for every β strictly smaller than some α,

if I can show that P holds for α…

```
→
Πα: Size. P α
```

…then P holds for *any* α.

Fixpoints...

$\alpha \vdash \mathtt{P}\ \alpha : \mathcal{U}$

Let P be a predicate on sizes.

$\alpha;\ \mathtt{f}\colon \forall\beta < \alpha.\ \mathtt{P}\ \beta \vdash \mathtt{e} : \mathtt{P}\ \alpha$

Given that P holds for every β strictly smaller than some α,

if I can show that P holds for α...

$$\frac{}{\vdash \mathtt{fix}\ \mathtt{f}\ [\alpha]\colon \mathtt{P}\ \alpha \coloneqq \mathtt{e} : \forall\alpha.\ \mathtt{P}\ \alpha}$$

...then P holds for *any* α.

Fixpoints are a Well-Founded Induction Principle

```
⟦fix f [α]: τ ≔ e⟧ =
  wfInd (λα: Size. ⟦τ⟧)
        (λα: Size. λf: (Πβ: Size. β < α → ⟦τ⟧[α ↦ β]). ⟦e⟧)
```

More Technical Details

- Accessibility predicate w.r.t. size order

```
data Acc (α: Size): Prop where
  acc: (∀β: Size. β < α → Acc β) → Acc α
```

- Well-foundedness of sizes

```
fix wf: Πα: Size. Acc α
```

- Proof-irrelevance of accessibility

$\texttt{α: Size, acc}_1\texttt{: Acc α, acc}_2\texttt{: Acc α ⊢ acc}_1 \equiv \texttt{acc}_2 \texttt{ : Acc α}$

- Preservation of fixpoint reduction

```
⊢ ⟦(fix f [α]: τ ≔ e) [s]⟧ ≡
  ⟦e[α ↦ s][f ↦ Λβ < s. (fix f [α]: τ ≔ e) [β]⟧ : ⟦∀α. τ⟧
```

Inconsistent Infinity

If STT has:

$+ \qquad \infty$

$\vdash s$

$+ \qquad \dfrac{}{\vdash s \le \infty}$

$\Longrightarrow \vdash \infty < \infty$

$\therefore$ STT must not have ∞!

Then $\mathrm{CIC}_{\mathrm{E}}$ would have:

```
let ∞<∞ : ⟦∞⟧ < ⟦∞⟧

fix ¬wf∞: Acc ⟦∞⟧ → ⊥
¬wf∞ (acc p) ≔ p ⟦∞⟧ ∞<∞

let ng: ⊥
ng ≔ ¬wf∞ (wf ⟦∞⟧)
```

$\because$ Inconsistent!

④ Shortcomings and Future Work

Problem 1: Limited Size Expressions

```
fix add: ∀α. ∀β. nat [α] → nat [β] → nat [?]
add [α] [β] (zero [α']) n ≔      zero [_]
add [α] [β] m (zero [β']) ≔      zero [_]
add [α] [β] (succ [α'] n) (succ [β'] m) ≔
  let     sum  ≔ add [α'] [β'] n m
  in       succ [ _ ] (succ [_] sum)
```

- ? = ∞ (since _ < ∞; but inconsistent!)
- ? = α + β (when does it end? α × β? α^{β}? α!? f [α] [β] for any f??)

Solution 1: Existentially-Quantified Sizes

```
fix add: ∀α. ∀β. nat [α] → nat [β] → ∃γ. nat [γ]
add [α] [β] (zero [α']) n ≔ ⟨α, zero [α']⟩
add [α] [β] m (zero [β']) ≔ ⟨β, zero [β']⟩
add [α] [β] (succ [α'] n) (succ [β'] m) ≔
  let ⟨γ, sum⟩ ≔ add [α'] [β'] n m
  in ⟨γ+2, succ [γ+1] (succ [γ] sum)⟩
```

- ? = ∞ (since _ < ∞; but inconsistent!)
- ? = α + β (when does it end? α × β? $α^β$? α!? f [α] [β] for any f??)
- ? = …it doesn't matter as long as it has *some* size!

Problem 2: Infinitary Inductives

```
data ord     : 𝒰 ≔
  zero:          ord
  succ:          ord      → ord
  lim:           (    nat      → ord      ) → ord

fix natToOrd:       nat      →       ord
natToOrd       zero          ≔       zero
natToOrd      (succ       n) ≔ succ (natToOrd n)


let ω:      ord
ω ≔        lim                                natToOrd
```

Problem 2: Infinitary Inductives

```
data ord [s]: 𝒰 ≔
  zero: ∀α < s. ord [s]
  succ: ∀α < s. ord [α] → ord [s]
  lim:  ∀α < s. (∀β. nat [β] → ord [α]) → ord [s]

fix natToOrd: ∀β. nat [β] → ∃α. ord [α]
natToOrd [β] (zero [β’])    ≔ ⟨β, zero [β’]⟩
natToOrd [β] (succ [β’] n) ≔
  let ⟨α, x⟩ ≔ natToOrd [β’] n in ⟨α+1, succ [α] x⟩

let ω: ∃α. ord [α]
ω ≔ ⟨?+1, lim [?] (Λβ. λn. let ⟨α, m⟩ ≔ natToOrd [β] n in m)⟩
```

Solution 2: ???

```
let ac: (τ → ∃α. σ [α]) → ∃α. (τ → σ [α])
```

↳ requires size projection (bad) and limit sizes (worse) to implement

```
let ω: ∃α. ord [α]
ω ≔ let ⟨α, f⟩ ≔ ac natToOrd in ⟨α+1, lim [α] f⟩
```

Future Work: Missing Features and Properties

- **Inductives** in general (only $\mathbb{N}$, $\mathbb{W}$):
  straightforward, tedious, no new insights
- **Coinductives**:
  straightforward source extension,
  suspicious target translation
- **Normalization** wrt reduction,
  decidability of type checking:
  conjectured (doesn’t follow from model)
- **Infinitary constructs** (lack of ∞)

Summary

of contributions

1. Explicit, higher-rank, bounded sized dependent type theory
2. Syntactic model for sized types and proof of consistency

⑤ Questions?