A Reasonably Gradual Dependent Type Theory

Kenji Maillard

Inria Nantes, team Gallinette
j.w.w.


Meven Lennon-Bertrand


Nicolas Tabareau


Éric Tanter

ICFP'22, Ljubljana
Monday, 12th of September, 2022

Prototyping with dependent types


```
Inductive vect (A : Type) : ℕ → Type :=
| nil : vect A 0
| _::_ {n : ℕ} (head : A) (tail : vect A n) : vect A (1+n)
```

Prototyping with dependent types


```
Inductive vect (A : Type) : ℕ → Type :=
| nil : vect A 0
| _::_ {n : ℕ} (head : A) (tail : vect A n) : vect A (1+n)
```

```
Equations filter {A : Type} (P : A → 𝔹) {len : ℕ} (l : vect A len) : vect A ? :=
```

Prototyping with dependent types


```
Inductive vect (A : Type) : ℕ → Type :=
| nil : vect A 0
| _::_ {n : ℕ} (head : A) (tail : vect A n) : vect A (1+n)
```

```
Equations filter {A : Type} (P : A → 𝔹) {len : ℕ} (l : vect A len) : vect A ? :=
filter P nil := nil
filter P (head :: tail) when not (P head) := filter tail
filter P (head :: tail) := head :: (filter tail) : vect A ?
```

Prototyping with dependent types


```
Inductive vect (A : Type) : ℕ → Type :=
| nil : vect A 0
| _::_ {n : ℕ} (head : A) (tail : vect A n) : vect A (1+n)
```

```
Equations filter {A : Type} (P : A → 𝔹) {len : ℕ} (l : vect A len) : vect A ? :=
filter P nil := nil
filter P (head :: tail) when not (P head) := filter tail
filter P (head :: tail) := head :: (filter tail) : vect A ?
                           ─────────────────────
                           vect A (1+?)
```

Prototyping with dependent types


```
Inductive vect (A : Type) : ℕ → Type :=
| nil : vect A 0
| _::_ {n : ℕ} (head : A) (tail : vect A n) : vect A (1+n)
```

```
Equations filter {A : Type} (P : A → 𝔹) {len : ℕ} (l : vect A len) : vect A ? :=
filter P nil := nil
filter P (head :: tail) when not (P head) := filter tail
filter P (head :: tail) := head :: (filter tail) : vect A ?
```

vect A (1+ ?)

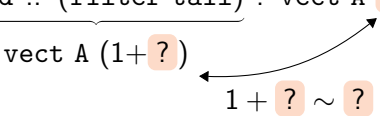
1 + ? ∼ ?

Prototyping with dependent types


```
head: forall (A: Type) (n: ℕ), vect A (1+n) → A

head ℕ ? (filter even (1 :: 2 :: 4 :: nil))
head ℕ ? (filter even nil)

Equations filter {A : Type} (P : A → 𝔹) {len : ℕ} (l : vect A len) : vect A ? :=
filter P nil := nil
filter P (head :: tail) when not (P head) := filter tail
filter P (head :: tail) := head :: (filter tail) : vect A ?
```



Prototyping with dependent types


```
head: forall (A: Type) (n: ℕ), vect A (1+n) → A

head ℕ ? (filter even (1 :: 2 :: 4 :: nil))
head ℕ ? (filter even nil)
```

vect A (1+ ?)

```
Equations filter {A : Type} (P : A → 𝔹) {len : ℕ} (l : vect A len) : vect A ? :=
filter P nil := nil
filter P (head :: tail) when not (P head) := filter tail
filter P (head :: tail) := head :: (filter tail) : vect A ?
```

vect A (1+ ?)

1 + ? ∼ ?

Prototyping with dependent types


```
head: forall (A: Type) (n: ℕ), vect A (1+n) → A
```

```
head ℕ ? (filter even (1 :: 2 :: 4 :: nil))   ⇝ head ℕ ? (2 :: 4 :: nil)
head ℕ ? (filter even nil)                    ⇝ head ℕ ? nil
```

```
Equations filter {A : Type} (P : A → 𝔹) {len : ℕ} (l : vect A len) : vect A ? :=
filter P nil := nil
filter P (head :: tail) when not (P head) := filter tail
filter P (head :: tail) := head :: (filter tail) : vect A ?
```

vect A (1+?)

1 + ? ∼ ?

Prototyping with dependent types


```
head: forall (A: Type) (n: ℕ), vect A (1+n) → A
```

```
head ℕ ? (filter even (1 :: 2 :: 4 :: nil))   ⇝ head ℕ ? (2 :: 4 :: nil)  ⇝  2
head ℕ ? (filter even nil)                    ⇝ head ℕ ? nil
```

```
Equations filter {A : Type} (P : A → 𝔹) {len : ℕ} (l : vect A len) : vect A ? :=
filter P nil := nil
filter P (head :: tail) when not (P head) := filter tail
filter P (head :: tail) := head :: (filter tail) : vect A ?
```

vect A (1+?)

1 + ? ∼ ?

Prototyping with dependent types


```
head: forall (A: Type) (n: ℕ), vect A (1+n) → A
```

```
head ℕ ? (filter even (1 :: 2 :: 4 :: nil))   ⇝ head ℕ ? (2 :: 4 :: nil)  ⇝ 2
head ℕ ? (filter even nil)                    ⇝ head ℕ ? nil              ⇝ err
```

```
Equations filter {A : Type} (P : A → 𝔹) {len : ℕ} (l : vect A len) : vect A ? :=
filter P nil := nil
filter P (head :: tail) when not (P head) := filter tail
filter P (head :: tail) := head :: (filter tail) : vect A ?
```

vect A (1+?)

1 + ? ∼ ?

Prototyping with dependent types


```
head: forall (A: Type) (n: ℕ), vect A (1+n) → A
```

```
head ℕ ? (filter even (1 :: 2 :: 4 :: nil))   ⇝ head ℕ ? (2 :: 4 :: nil)  ⇝  2
head ℕ ? (filter even nil)                    ⇝ head ℕ ? nil              ⇝  err
```

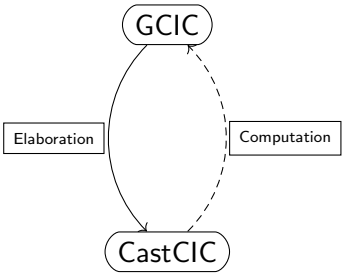
```
Equations filter {A : Type} (P : A → 𝔹) {len : ℕ} (l : vect A len) : vect A ? :=
filter P nil := nil
filter P (head :: tail) when not (P head) := filter tail
filter P (head :: tail) := head :: (filter tail) : vect A ?
```

`vect A (1+?)`

`1 + ? ~ ?`

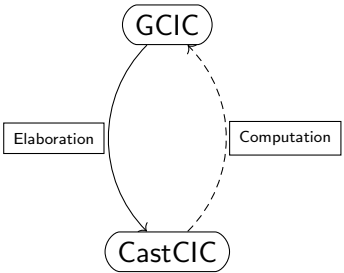
```
if l = nil
then 0
else ?
```

GCIC & CastCIC: a marriage of Gradual & Dependent Types



- Unknown terms and types $?_A : A$
- Errors $\texttt{err}_A : A$
- Casts $t : A \implies \langle B \Leftarrow A \rangle\, t : B$

GCIC & CastCIC: a marriage of Gradual & Dependent Types

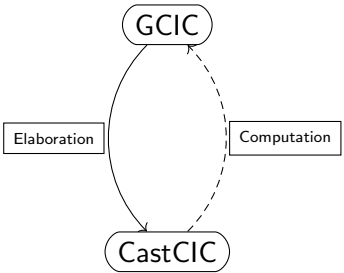


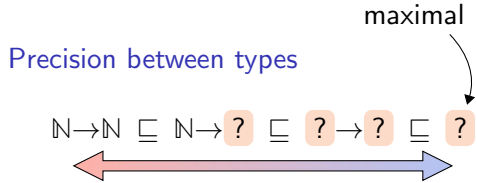
Precision between types

$\mathbb{N} \to \mathbb{N} \sqsubseteq \mathbb{N} \to ? \sqsubseteq ? \to ? \sqsubseteq ?$

- Unknown terms and types $?_A : A$
- Errors $\texttt{err}_A : A$
- Casts $t : A \implies \langle B \Leftarrow A \rangle\, t : B$

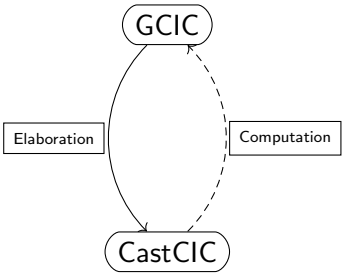
GCIC & CastCIC: a marriage of Gradual & Dependent Types





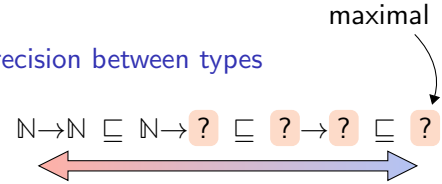
- Unknown terms and types $?_A : A$
- Errors $\texttt{err}_A : A$
- Casts $t : A \implies \langle B \Leftarrow A \rangle\, t : B$

GCIC & CastCIC: a marriage of Gradual & Dependent Types



- Unknown terms and types $?_A : A$
- Errors $\texttt{err}_A : A$
- Casts $t : A \implies \langle B \Leftarrow A \rangle\, t : B$

Precision between types



Precision enforces cast validity:

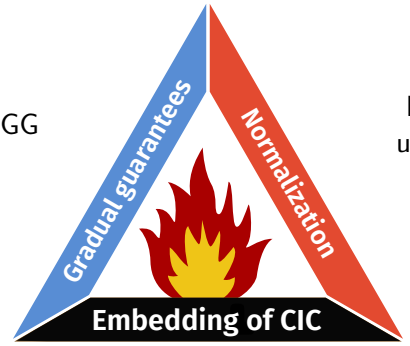
$$A \sqsubseteq B \implies A \underset{\langle A \Leftarrow B \rangle}{\overset{\langle B \Leftarrow A \rangle}{\rightleftarrows}} B$$

Graduality [New & Ahmed 2018]

The Fire Triangle of Graduality

From Gradualizing the CIC [TOPLAS 2022]

Graduality
Static & Dynamic GG



$?_\square \to ?_\square \not\sqsubseteq ?_\square$
No embedding of
untyped λ-calculus

Conservativity: t, A ? -free

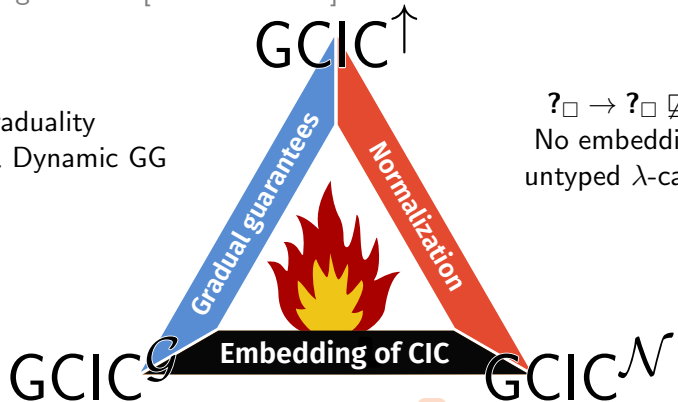
t has type A in GCIC iff t has type A in CIC

$\vdash_? t : A \iff \vdash_{\text{CIC}} t : A$

The Fire Triangle of Graduality

From Gradualizing the CIC [TOPLAS 2022]

Graduality
Static & Dynamic GG



$?_\square \to ?_\square \not\sqsubseteq ?_\square$
No embedding of
untyped λ-calculus

Conservativity: t, A ? -free

t has type A in GCIC iff t has type A in CIC

$\vdash_? t : A \iff \vdash_{\text{CIC}} t : A$

Internal precision in GRIP

$\mathsf{CastCIC}^{\mathcal{N}}$: Normalizing, conservative over CIC, not gradual

How far is $\mathsf{CastCIC}^{\mathcal{N}}$ from actually being gradual ?

Internal precision in GRIP

$\mathsf{CastCIC}^{\mathcal{N}}$: Normalizing, conservative over CIC, not gradual

How far is $\mathsf{CastCIC}^{\mathcal{N}}$ from actually being gradual ?

Make precision internal so that we can reason on it!

$$\mathsf{GRIP} = \mathsf{CastCIC}^{\mathcal{N}} + \sqsubseteq + \mathbb{P}$$

Internal precision in GRIP

$\mathsf{CastCIC}^{\mathcal{N}}$: Normalizing, conservative over CIC, not gradual

$$\mathsf{GRIP} = \mathsf{CastCIC}^{\mathcal{N}} + \sqsubseteq + \mathbb{P}$$

$$\frac{\Gamma \vdash A, B : \square_i}{\Gamma \vdash A \sqsubseteq_i B : \mathbb{P}} \qquad \frac{\Gamma \vdash A, B : \square_i \qquad \Gamma \vdash a : A \qquad \Gamma \vdash b : B}{\Gamma \vdash a \ {}_A\sqsubseteq_B \ b : \mathbb{P}}$$

$\mathbb{P}$: Pure universe of propositions

Internal precision in GRIP

$\mathsf{CastCIC}^{\mathcal{N}}$: Normalizing, conservative over CIC, not gradual

$$\mathsf{GRIP} = \mathsf{CastCIC}^{\mathcal{N}} + \sqsubseteq + \mathbb{P}$$

$$\frac{\Gamma \vdash A, B : \square_i}{\Gamma \vdash A \sqsubseteq_i B : \mathbb{P}} \qquad \frac{\Gamma \vdash A, B : \square_i \qquad \Gamma \vdash a : A \qquad \Gamma \vdash b : B}{\Gamma \vdash a \ {}_A\sqsubseteq_B \ b : \mathbb{P}}$$

$\mathbb{P}$: Pure universe of propositions for sound reasoning

- Pure: no error, no casts, no ?
- propositions: definitionally proof-irrelevant

Proofs of precision – External meaning

Dynamic Gradual Guarantee

$$\texttt{DGG} : \forall (A : \square)(C : A \to \mathbb{B})(x\ y : A), x\ {}_A\sqsubseteq_A\ y \to C\ x\ {}_{\mathbb{B}}\sqsubseteq_{\mathbb{B}}\ C\ y.$$

$\texttt{DGG}\ A\ C$ holds $\iff$ $C\ {}_{A\to\mathbb{B}}\sqsubseteq_{A\to\mathbb{B}}\ C$ $\iff$ C monotone wrt. $\sqsubseteq$

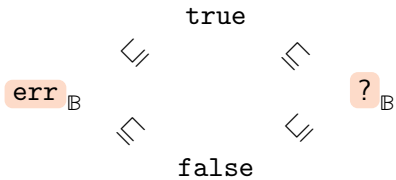
Proofs of precision – External meaning

Dynamic Gradual Guarantee

$$\texttt{DGG} : \forall(A : \square)(C : A \to \mathbb{B})(x\ y : A), x\ {}_A\sqsubseteq_A\ y \to C\ x\ {}_{\mathbb{B}}\sqsubseteq_{\mathbb{B}}\ C\ y.$$

$$\texttt{DGG}\ A\ C \text{ holds} \iff C\ {}_{A\to\mathbb{B}}\sqsubseteq_{A\to\mathbb{B}}\ C \iff C \text{ monotone wrt. } \sqsubseteq$$

DGG correspond to error-approximation:



Proofs of precision – An example

$$\texttt{add1} : \mathbb{N} \to \mathbb{N} := \lambda x : \mathbb{N}.x + 1$$
$$\texttt{add?} : \textbf{?}_{\square} \to \mathbb{N} := \lambda x : \textbf{?}_{\square}.(\langle \mathbb{N} \Leftarrow \textbf{?}_{\square} \rangle\, x) + 1$$

$\texttt{add1} \;{}_{\mathbb{N}\to\mathbb{N}}\!\sqsubseteq_{\textbf{?}_{\square}\to\mathbb{N}} \texttt{add?} \quad \Longrightarrow \quad$ add? does not error on good input

Proofs of precision – An example

$$\texttt{add1} : \mathbb{N} \to \mathbb{N} := \lambda x : \mathbb{N}.x + 1$$
$$\texttt{add?} : \textbf{?}_{\square} \to \mathbb{N} := \lambda x : \textbf{?}_{\square}.(\langle \mathbb{N} \Leftarrow \textbf{?}_{\square} \rangle\, x) + 1$$

$$\texttt{add1} \;\; {}_{\mathbb{N}\to\mathbb{N}}{\sqsubseteq}_{\textbf{?}_{\square}\to\mathbb{N}} \;\; \texttt{add?}$$

Proofs of precision – An example

$$\texttt{add1} : \mathbb{N} \to \mathbb{N} := \lambda x : \mathbb{N}.x + 1$$
$$\texttt{add?} : \textbf{?}_\square \to \mathbb{N} := \lambda x : \textbf{?}_\square.(\langle \mathbb{N} \Leftarrow \textbf{?}_\square \rangle\, x) + 1$$

$$\frac{\forall (x : \mathbb{N})(z : \textbf{?}_\square), x \;{}_{\mathbb{N}}\sqsubseteq_{\textbf{?}_\square}\; z \quad \to \quad \texttt{add1}\; x \;{}_{\mathbb{N}}\sqsubseteq_{\mathbb{N}}\; \texttt{add?}\; z}{\texttt{add1} \;{}_{\mathbb{N}\to\mathbb{N}}\sqsubseteq_{\textbf{?}_\square\to\mathbb{N}}\; \texttt{add?}} \quad \sqsubseteq \text{ on } \to$$

Proofs of precision – An example

$$\texttt{add1} : \mathbb{N} \to \mathbb{N} := \lambda x : \mathbb{N}.x + 1$$
$$\texttt{add?} : \texttt{?}_\square \to \mathbb{N} := \lambda x : \texttt{?}_\square.(\langle \mathbb{N} \Leftarrow \texttt{?}_\square \rangle\, x) + 1$$

$$\dfrac{\dfrac{x : \mathbb{N}, z : \texttt{?}_\square, xz : x \;{}_{\mathbb{N}}\sqsubseteq_{\texttt{?}_\square}\; z \quad \vdash \quad x + 1 \;{}_{\mathbb{N}}\sqsubseteq_{\mathbb{N}}\; (\langle \mathbb{N} \Leftarrow \texttt{?}_\square \rangle\, z) + 1}{\forall (x : \mathbb{N})(z : \texttt{?}_\square), x \;{}_{\mathbb{N}}\sqsubseteq_{\texttt{?}_\square}\; z \quad \to \quad \texttt{add1}\; x \;{}_{\mathbb{N}}\sqsubseteq_{\mathbb{N}}\; \texttt{add?}\; z}\ \text{Intros, compute}}{\texttt{add1} \;{}_{\mathbb{N}\to\mathbb{N}}\sqsubseteq_{\texttt{?}_\square\to\mathbb{N}}\; \texttt{add?}}\ \sqsubseteq \text{ on } \to$$

Proofs of precision – An example

$$\texttt{add1} : \mathbb{N} \to \mathbb{N} := \lambda x : \mathbb{N}.x + 1$$
$$\texttt{add?} : \texttt{?}_\square \to \mathbb{N} := \lambda x : \texttt{?}_\square.(\langle \mathbb{N} \Leftarrow \texttt{?}_\square \rangle\, x) + 1$$

$$\dfrac{\dfrac{\dfrac{x : \mathbb{N}, z : \texttt{?}_\square, xz : x \;{}_{\mathbb{N}}\!\sqsubseteq_{\texttt{?}_\square}\; z \quad \vdash \quad x \;{}_{\mathbb{N}}\!\sqsubseteq_{\mathbb{N}}\; \langle \mathbb{N} \Leftarrow \texttt{?}_\square \rangle\, z}{x : \mathbb{N}, z : \texttt{?}_\square, xz : x \;{}_{\mathbb{N}}\!\sqsubseteq_{\texttt{?}_\square}\; z \quad \vdash \quad x + 1 \;{}_{\mathbb{N}}\!\sqsubseteq_{\mathbb{N}}\; (\langle \mathbb{N} \Leftarrow \texttt{?}_\square \rangle\, z) + 1}\ +1 \text{ monotone}}{\forall (x : \mathbb{N})(z : \texttt{?}_\square), x \;{}_{\mathbb{N}}\!\sqsubseteq_{\texttt{?}_\square}\; z \quad \to \quad \texttt{add1}\ x \;{}_{\mathbb{N}}\!\sqsubseteq_{\mathbb{N}}\; \texttt{add?}\ z}\ \text{Intros, compute}}{\texttt{add1} \;{}_{\mathbb{N}\to\mathbb{N}}\!\sqsubseteq_{\texttt{?}_\square\to\mathbb{N}}\; \texttt{add?}}\ \sqsubseteq \text{ on } \to$$

Proofs of precision – An example

$$\texttt{add1} : \mathbb{N} \to \mathbb{N} := \lambda x : \mathbb{N}.x + 1$$
$$\texttt{add?} : \texttt{?}_\square \to \mathbb{N} := \lambda x : \texttt{?}_\square.(\langle \mathbb{N} \Leftarrow \texttt{?}_\square \rangle\, x) + 1$$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{x : \mathbb{N}, z : \texttt{?}_\square, xz : x\ {}_{\mathbb{N}}\sqsubseteq_{\texttt{?}_\square}\ z \quad \vdash \quad xz : x\ {}_{\mathbb{N}}\sqsubseteq_{\texttt{?}_\square}\ z}}{x : \mathbb{N}, z : \texttt{?}_\square, xz : x\ {}_{\mathbb{N}}\sqsubseteq_{\texttt{?}_\square}\ z \quad \vdash \quad x\ {}_{\mathbb{N}}\sqsubseteq_{\mathbb{N}}\ \langle \mathbb{N} \Leftarrow \texttt{?}_\square \rangle\, z}\ \langle \mathbb{N} \Leftarrow \texttt{?}_\square \rangle \text{ right adjoint}}{x : \mathbb{N}, z : \texttt{?}_\square, xz : x\ {}_{\mathbb{N}}\sqsubseteq_{\texttt{?}_\square}\ z \quad \vdash \quad x + 1\ {}_{\mathbb{N}}\sqsubseteq_{\mathbb{N}}\ (\langle \mathbb{N} \Leftarrow \texttt{?}_\square \rangle\, z) + 1}\ +1 \text{ monotone}}{\forall (x : \mathbb{N})(z : \texttt{?}_\square), x\ {}_{\mathbb{N}}\sqsubseteq_{\texttt{?}_\square}\ z \quad \to \quad \texttt{add1}\ x\ {}_{\mathbb{N}}\sqsubseteq_{\mathbb{N}}\ \texttt{add?}\ z}\ \text{Intros, compute}}{\texttt{add1}\ {}_{\mathbb{N}\to\mathbb{N}}\sqsubseteq_{\texttt{?}_\square\to\mathbb{N}}\ \texttt{add?}}\ \sqsubseteq \text{ on } \to$$

Proofs of precision – Principles

Reasoning principles for $\sqsubseteq$

- heterogeneous transitivity

$$a \ {}_A\sqsubseteq_B \ b \wedge b \ {}_B\sqsubseteq_C \ c \to a \ {}_A\sqsubseteq_C \ c,$$

- quasi-reflexivity

$$a \ {}_A\sqsubseteq_B \ b \to a \ {}_A\sqsubseteq_A \ a \wedge b \ {}_B\sqsubseteq_B \ b,$$

- adjunction properties

$$A \sqsubseteq_i B \to \langle B \Leftarrow A \rangle \, a \ {}_B\sqsubseteq_B \ b \iff a \ {}_A\sqsubseteq_B \ b \iff a \ {}_A\sqsubseteq_A \ \langle A \Leftarrow B \rangle \, b,$$

- decomposition through upper bounds

$$A, B \sqsubseteq_i X \to a \ {}_A\sqsubseteq_B \ b \iff \langle X \Leftarrow A \rangle \, a \ {}_X\sqsubseteq_X \ \langle X \Leftarrow B \rangle \, b.$$

Proofs of precision – Principles

Reasoning principles for $\sqsubseteq$

- heterogeneous transitivity

$$a \ {}_A\sqsubseteq_B \ b \wedge b \ {}_B\sqsubseteq_C \ c \to a \ {}_A\sqsubseteq_C \ c,$$

- quasi-reflexivity

$$a \ {}_A\sqsubseteq_B \ b \to a \ {}_A\sqsubseteq_A \ a \wedge b \ {}_B\sqsubseteq_B \ b,$$

- adjunction properties

$$A \sqsubseteq_i B \to \langle B \Leftarrow A \rangle \, a \ {}_B\sqsubseteq_B \ b \iff a \ {}_A\sqsubseteq_B \ b \iff a \ {}_A\sqsubseteq_A \langle A \Leftarrow B \rangle \, b,$$

- decomposition through upper bounds

$$A, B \sqsubseteq_i X \to a \ {}_A\sqsubseteq_B \ b \iff \langle X \Leftarrow A \rangle \, a \ {}_X\sqsubseteq_X \langle X \Leftarrow B \rangle \, b.$$

However

- $A \ {}_{\square_i}\sqsubseteq_{\square_i} \ B \iff A \sqsubseteq_i B \wedge B \sqsubseteq_i \ ?_{\square_i}$

Proofs of precision – Principles

Reasoning principles for $\sqsubseteq$

- heterogeneous transitivity

$$a \ {}_A\sqsubseteq_B \ b \wedge b \ {}_B\sqsubseteq_C \ c \to a \ {}_A\sqsubseteq_C \ c,$$

- quasi-reflexivity

$$a \ {}_A\sqsubseteq_B \ b \to a \ {}_A\sqsubseteq_A \ a \wedge b \ {}_B\sqsubseteq_B \ b,$$

- adjunction properties

$$A \sqsubseteq_i B \to \langle B \Leftarrow A \rangle \, a \ {}_B\sqsubseteq_B \ b \iff a \ {}_A\sqsubseteq_B \ b \iff a \ {}_A\sqsubseteq_A \langle A \Leftarrow B \rangle \, b,$$

- decomposition through upper bounds

$$A, B \sqsubseteq_i X \to a \ {}_A\sqsubseteq_B \ b \iff \langle X \Leftarrow A \rangle \, a \ {}_X\sqsubseteq_X \langle X \Leftarrow B \rangle \, b.$$

However

- $A \ {}_{\square_i}\sqsubseteq_{\square_i} \ B \iff A \sqsubseteq_i B \wedge B \sqsubseteq_i ?_{\square_i}$
- $?_{\square_i} \to ?_{\square_i} \not\sqsubseteq_i ?_{\square_i}$

Proofs of precision – Principles

Reasoning principles for $\sqsubseteq$

- heterogeneous transitivity

$$a \ {}_A\sqsubseteq_B \ b \wedge b \ {}_B\sqsubseteq_C \ c \to a \ {}_A\sqsubseteq_C \ c,$$

- quasi-reflexivity

$$a \ {}_A\sqsubseteq_B \ b \to a \ {}_A\sqsubseteq_A \ a \wedge b \ {}_B\sqsubseteq_B \ b,$$

- adjunction properties

$$A \sqsubseteq_i B \to \langle B \Leftarrow A \rangle \, a \ {}_B\sqsubseteq_B \ b \iff a \ {}_A\sqsubseteq_B \ b \iff a \ {}_A\sqsubseteq_A \ \langle A \Leftarrow B \rangle \, b,$$

- decomposition through upper bounds

$$A, B \sqsubseteq_i X \to a \ {}_A\sqsubseteq_B \ b \iff \langle X \Leftarrow A \rangle \, a \ {}_X\sqsubseteq_X \ \langle X \Leftarrow B \rangle \, b.$$

However

- $A \ {}_{\square_i}\sqsubseteq_{\square_i} \ B \iff A \sqsubseteq_i B \wedge B \sqsubseteq_i ?_{\square_i}$
- $?_{\square_i} \to ?_{\square_i} \not\sqsubseteq_i ?_{\square_i}$
- $?_{\square_i}$ maximal for ${}_{\square_i}\sqsubseteq_{\square_i}$, not for $\sqsubseteq_i$

Proofs of precision – Principles

Reasoning principles for $\sqsubseteq$

- heterogeneous transitivity

$$a \ {}_A\sqsubseteq_B \ b \wedge b \ {}_B\sqsubseteq_C \ c \to a \ {}_A\sqsubseteq_C \ c,$$

- quasi-reflexivity

$$a \ {}_A\sqsubseteq_B \ b \to a \ {}_A\sqsubseteq_A \ a \wedge b \ {}_B\sqsubseteq_B \ b,$$

- adjunction properties

$$A \sqsubseteq_i B \to \langle B \Leftarrow A \rangle \, a \ {}_B\sqsubseteq_B \ b \iff a \ {}_A\sqsubseteq_B \ b \iff a \ {}_A\sqsubseteq_A \langle A \Leftarrow B \rangle \, b,$$

- decomposition through upper bounds

$$A, B \sqsubseteq_i X \to a \ {}_A\sqsubseteq_B \ b \iff \langle X \Leftarrow A \rangle \, a \ {}_X\sqsubseteq_X \langle X \Leftarrow B \rangle \, b.$$

However

- $A \ {}_{\Box_i}\sqsubseteq_{\Box_i} \ B \iff A \sqsubseteq_i B \wedge B \sqsubseteq_i ?_{\Box_i}$
- $?_{\Box_i} \to ?_{\Box_i} \not\sqsubseteq_i ?_{\Box_i}$
- $?_{\Box_i}$ maximal for ${}_{\Box_i}\sqsubseteq_{\Box_i}$, not for $\sqsubseteq_i$
- Mitigated by explicity cumulativity

$$\iota(?_{\Box_i} \to ?_{\Box_i}) \sqsubseteq_{i+1} ?_{\Box_{i+1}}$$

How good is GRIP? Metatheory!

Consistency $\not\vdash_{\mathsf{GRIP}} t : \bot$ (where $\bot : \mathbb{P}$)

Idea: Build a model of GRIP in MLTT+SProp using partial pre-orders.
$[\![\bot]\!]$ is empty in the model.

How good is GRIP? Metatheory!

Consistency $\not\vdash_{\mathsf{GRIP}} t : \bot$ (where $\bot : \mathbb{P}$)

Idea: Build a model of GRIP in MLTT+SProp using partial pre-orders.
$[\![\bot]\!]$ is empty in the model.

Normalization

Idea: The translation $[\![-]\!]$ preserves reduction sequences into a normalizing target (MLTT + SProp, Gilbert et al. [POPL'19]).

How good is GRIP? Metatheory!

Consistency $\not\vdash_{\mathsf{GRIP}} t : \bot$ (where $\bot : \mathbb{P}$)

Idea: Build a model of GRIP in MLTT+SProp using partial pre-orders.
$[\![\bot]\!]$ is empty in the model.

Normalization

Idea: The translation $[\![-]\!]$ preserves reduction sequences into a normalizing target (MLTT + SProp, Gilbert et al. [POPL'19]).

Gradual fragment $\mathsf{GRIP}^{\uparrow}$ Automatically precise terms

Idea: Raise universe levels on Π

$$\frac{\Gamma \vdash_{\mathsf{GRIP}^{\uparrow}} A : \square_i \qquad \Gamma, x : A \vdash_{\mathsf{GRIP}^{\uparrow}} B : \square_i}{\Gamma \vdash_{\mathsf{GRIP}^{\uparrow}} \Pi\, x : A.B : \square_{i+1}}$$

Summary & Future work

GRIP = Partially gradual type theory + internal precision

- ▷ Proof of concept for GRIP in Agda
- ▷ Model in Coq based on partial pre-orders

Summary & Future work

GRIP = Partially gradual type theory + internal precision

- ▷ Proof of concept for GRIP in Agda
- ▷ Model in Coq based on partial pre-orders
- ▷ Non gradual operations, e.g. `catch` of `?` / `err` on inductives
- ▷ Adresses DGG in an exceptional world (see paper).

Summary & Future work

GRIP = Partially gradual type theory + internal precision

- ▷ Proof of concept for GRIP in Agda
- ▷ Model in Coq based on partial pre-orders
- ▷ Non gradual operations, e.g. `catch` of `?`/`err` on inductives
- ▷ Adresses DGG in an exceptional world (see paper).
- ▷ Internal reasoning to establish graduality post-hoc

Summary & Future work

GRIP = Partially gradual type theory + internal precision

- ▷ Proof of concept for GRIP in Agda
- ▷ Model in Coq based on partial pre-orders
- ▷ Non gradual operations, e.g. `catch` of `?` / `err` on inductives
- ▷ Adresses DGG in an exceptional world (see paper).
- ▷ Internal reasoning to establish graduality post-hoc

GRIP is a type theory, not yet a proof assistant:

- ▷ Elaboration from a gradual source
- ▷ General inductive types
- ▷ An actual implementation?

Thank You!

Gradualizing printf


```
Equations argOfDescr (s : stringDescr) : Type :=
  argOfDescr "" := unit ;
  argOfDescr ("%d" @ rest) := ℕ → argOfDescr rest;
  ...

Definition printf : forall (s : stringDescr), argOfDescr s := ...
```

- ▷ `printf` does not type in GRIP$^{\uparrow}$ (arbitrary arity)
- ▷ `printf` does type in GRIP
- ▷ It is not self-precise
- ▷ but its retrictions to `stringDescr` bounded by some fixed length are self-precise.