GRADUALIZING THE CALCULUS OF INDUCTIVE CONSTRUCTIONS

Meven Lennon-Bertrand, Kenji Maillard, Nicolas Tabareau, and Éric Tanter
TOPLAS Journal-First @ POPL ’22
A BIT OF CONTEXT
BRINGING TOGETHER CIC AND GRADUAL TYPING

- The Calculus of Inductive Constructions
  - Dependent types:
    - \( \Pi (A : \text{Type}) (n : \mathbb{N}), \text{Vect} A (1 + n) \rightarrow A \)
- Gradual typing
  - Mix static and dynamic typing
  - Dynamic type \(?\) in a static system
  - Optimistic typing phase & runtime checks

Time has come for Gradual CIC!
### The Calculus of Inductive Constructions

- **Dependent types:**
  - $\Pi (A : \text{Type}) (n : \mathbb{N}), \text{Vect}_A(1+n) \rightarrow A$

---

**Gradual typing**

- Mix static and dynamic typing
- Dynamic type `?` in a static system
- Optimistic typing phase & runtime checks

Time has come for Gradual CIC!
The Calculus of Inductive Constructions

Dependent types:

\[
\text{head: } \Pi (A: \text{Type}) (n: \mathbb{N}), \text{ Vect } A (1+n) \rightarrow A
\]
The Calculus of Inductive Constructions

Dependent types:

\[ \text{head}: \Pi (A: \text{Type}) (n: \mathbb{N}), \text{Vect} A (1+n) \rightarrow A \]
**The Calculus of Inductive Constructions**

Dependent types:

\[
\text{head: } \Pi (A: \text{Type})(n: \mathbb{N}), \text{Vect } A (1+n) \rightarrow A
\]

**Gradual typing**

- Mix static and dynamic typing
- Dynamic type `?` in a static system
- Optimistic typing phase & runtime checks
The Calculus of Inductive Constructions

Dependent types:
head: \( \forall (A: \text{Type}) (n: \mathbb{N}), \text{Vect} \ A (1+n) \rightarrow A \)

Gradual typing

- Mix static and dynamic typing
- Dynamic type ? in a static system
- Optimistic typing phase & runtime checks

Time has come for Gradual c\text{ic}!
WHY GRADUAL DEPENDENT TYPES?

\[
\text{head nat} \ ? \ (\text{filter nat even} \ 3 \ [1;2;4])
\]

\[
\text{head nat} \ ? \ (\text{filter nat even} \ 0 \ [])
\]

\[
\frac{3}{11}
\]
Incremental development!
Incremental development!

\[
\text{filter} (A : \text{Type}) (p : A \rightarrow \mathbb{B}) (n : \mathbb{N}) (l : \text{Vect} A n) : \text{Vect} A \ldots
\]
WHY GRADUAL DEPENDENT TYPES?

Incremental development!

\[
\text{filter } (A : \text{Type}) (p : A \rightarrow \mathbb{B}) (n : \mathbb{N}) (l : \text{Vect} \ A \ n) : \text{Vect} \ A \ ?
\]
WHY GRADUAL DEPENDENT TYPES?

Incremental development!

\[
\text{filter} \ (A : \text{Type}) \ (p : A \to \mathbb{B}) \ (n : \mathbb{N}) \ (l : \text{Vect} \ A \ n) \\
: \text{Vect} \ A ?
\]

head nat ? (filter nat even 3 [1;2;4])
head nat ? (filter nat even 0 [])

3/11
WHY GRADUAL DEPENDENT TYPES?

Incremental development!

\[
\text{filter} \ (A : \textbf{Type})\ (p : A \to \mathbb{B})\ (n : \mathbb{N})\ (l : \text{Vect}\ A\ n)\ :
\text{Vect}\ A\ ?
\]

head nat ? (filter nat even 3 [1;2;4]) \mapsto 2
head nat ? (filter nat even 0 []) \mapsto \text{err}
Incremental development!

```haskell
filter (A : Type) (p : A → ℱ) (n : ℕ) (l : Vect A n) :
  Vect A (count A p n l)
```

```haskell
count A p n l := ?
```

```haskell
head nat ? (filter nat even 3 [1;2;4]) ↦ 2
head nat ? (filter nat even 0 []) ↦ err
```
WHY GRADUAL DEPENDENT TYPES?

Incremental development!

```plaintext
filter (A : Type) (p : A → ℙ) (n : ℕ) (l : Vect A n) :
  Vect A (count A p n l)

count A p n l :=
match l with
  | nil _ _ => 0
  | cons _ _ _ => ?
end.

head nat ? (filter nat even 3 [1;2;4]) ↦ 2
head nat ? (filter nat even 0 []) ↦ err
```
Incremental development!

\[
\text{filter}(A : \text{Type}) (p : A \rightarrow \mathbb{B}) (n : \mathbb{N}) (l : \text{Vect} A n) \\
\quad : \text{Vect} A (\text{count} A p n l)
\]

\[
\text{count} A p n l := \\
\text{match} l \text{ with} \\
\quad | \text{nil } _ _ \Rightarrow 0 \\
\quad | \text{cons } _ _ _ \Rightarrow ? \\
\text{end}.
\]

\[
\text{head} \text{ nat} ? (\text{filter} \text{ nat even} 3 \ [1;2;4]) \mapsto 2 \\
\text{head} \text{ nat} ? (\text{filter} \text{ nat even} 0 [])
\]
What’s so hard with dependent types?

- Mixing everything
  - No type/term distinction
  - No typing/runtime separation
- Effects
  - Errors
  - Divergence
- Indexed inductive types
  - Vectors need special care
  - Equality is currently out of our scope
## What’s so hard with Dependent Types?

### Mixing everything

- No type/term distinction
- No typing/runtime separation

### Effects

- Errors
- Divergence

### Indexed Inductive Types

- Vectors need special care
- Equality is currently out of our scope
### Mixing everything

- No type/term distinction
- No typing/runtime separation

### Effects

- Errors
- Divergence
### What’s so hard with dependent types?

<table>
<thead>
<tr>
<th>Mixing everything</th>
</tr>
</thead>
<tbody>
<tr>
<td>• No type/term distinction</td>
</tr>
<tr>
<td>• No typing/runtime separation</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Effects</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Errors</td>
</tr>
<tr>
<td>• Divergence</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Indexed inductive types</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Vectors need special care</td>
</tr>
<tr>
<td>• Equality is currently out of our scope</td>
</tr>
</tbody>
</table>
AN OVERVIEW OF GRADUAL CIC
NO-GO: THE FIRE TRIANGLE OF GRADUALITY

Our favorite properties

- Safety (Progress + Preservation)
- Normalization
- Conservativity (wrt. CIC)

Safety + Conservativity + Graduality
### Our favorite properties

- Safety (Progress + Preservation)
- Normalization
- Conservativity (wrt. CIC)
- Graduality
Our favorite properties

- Safety (Progress + Preservation)
- Normalization
- Conservativity (wrt. CIC)
- Graduality
Our favorite properties

- Safety (Progress + Preservation)
- Normalization
- Conservativity (wrt. CIC)
- Graduality

Safety + Conservativity + Graduality $\Rightarrow$ Pure $\lambda$-calculus $\Rightarrow$ Divergence
ONE SYSTEM, THREE VARIANTS

Controlled by universe levels of Π types:
• at reduction (cast decomposition)
• at typing

Reduction
Pure λ-calculus

Typing
CIC
Restricted

Conservativity

GCIC↑

Graduality
GCIC^G

Normalization
GCIC^N

One system, three variants

Controlled by universe levels of \( \Pi \) types:

- at reduction (cast decomposition)
- at typing
Controlled by universe levels of $\Pi$ types:
- at reduction (cast decomposition)
- at typing

<table>
<thead>
<tr>
<th></th>
<th>GCIC$^G$</th>
<th>GCIC$^N$</th>
<th>GCIC$^\uparrow$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Reduction</strong></td>
<td>Pure λ-calculus</td>
<td>×</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Eager failure</td>
<td></td>
<td>×</td>
</tr>
<tr>
<td><strong>Typing</strong></td>
<td>CIC</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td></td>
<td>Restricted</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
\[
\Gamma \vdash t : S \quad S \equiv T \\
\overline{\quad} \\
\Gamma \vdash t : T
\]
Elaboration and Consistent Conversion

\[
\Gamma \vdash t : S \quad S \equiv T \\
\hline
\Gamma \vdash t : T
\]

Issues

- transitivity
\[
\Gamma \vdash t : S \quad S \equiv T
\]

\[
\Gamma \vdash t : T
\]

<table>
<thead>
<tr>
<th>Issues</th>
<th>Solutions</th>
</tr>
</thead>
<tbody>
<tr>
<td>• transitivity</td>
<td>• bidirectional typing</td>
</tr>
</tbody>
</table>
ELABORATION AND CONSISTENT CONVERSION

\[
\Gamma \vdash t : S \quad S \equiv T \\
\hline
\Gamma \vdash t : T
\]

\[
\Gamma \vdash t \triangleright S \quad S \equiv T \\
\hline
\Gamma \vdash t \triangleleft T
\]

Issues

- transitivity

Solutions

- bidirectional typing
ELABORATION AND CONSISTENT CONVERSION

\[
\Gamma \vdash t : S \quad S \equiv T \\
\Gamma \vdash t : T
\]

\[
\Gamma \vdash t \triangleright S \quad S \equiv T \\
\Gamma \vdash t \triangleleft T
\]

**Issues**
- transitivity
- computation needs checks

**Solutions**
- bidirectional typing
ELABORATION AND CONSISTENT CONVERSION

\[
\Gamma \vdash t : S \quad S \equiv T \\
\hline
\Gamma \vdash t : T
\]

\[
\Gamma \vdash t \triangleright S \quad S \equiv T \\
\hline
\Gamma \vdash t \triangleleft T
\]

Issues

- transitivity
- computation needs checks

Solutions

- bidirectional typing
- type-based elaboration
ELABORATION AND CONSISTENT CONVERSION

\[
\begin{align*}
\Gamma \vdash t : S \quad S \equiv T \\
\frac{}{\Gamma \vdash t : T}
\end{align*}
\]

\[
\begin{align*}
\Gamma \vdash t \triangleright S \quad S \equiv T \\
\frac{}{\Gamma \vdash t \triangleleft T}
\end{align*}
\]

Issues

- transitivity
- computation needs checks

Solutions

- bidirectional typing
- type-based elaboration

CastCIC

GCIC

Elaboration

Computation
### Elaboration and Consistent Conversion

**Issues**
- transitivity
- computation needs checks

**Solutions**
- bidirectional typing
- type-based elaboration
ELABORATION AND CONSISTENT CONVERSION

\[
\frac{\Gamma \vdash t : S \quad S \equiv T}{\Gamma \vdash t : T}
\]

\[
\frac{\Gamma \vdash t \triangleright S \quad S \equiv T}{\Gamma \vdash t \triangleright T}
\]

\[
\frac{\Gamma \vdash t \rightsquigarrow t' \triangleright S \quad S \rightsquigarrow T}{\Gamma \vdash t \triangleright T \rightsquigarrow \left< T \Leftarrow S \right> t'}
\]

**Issues**
- transitivity
- computation needs checks
- decidability

**Solutions**
- bidirectional typing
- type-based elaboration
### Elaboration and Consistent Conversion

\[
\begin{align*}
\Gamma \vdash t : S \quad S \equiv T & \quad \Gamma \vdash t : T \\
\Gamma \vdash t : S \quad S \equiv T & \quad \Gamma \vdash t : T
\end{align*}
\]

\[
\begin{align*}
\Gamma \vdash t \leadsto t' \triangleright S \quad S \sim T & \quad \Gamma \vdash t \triangleleft T \leadsto (T \leftarrow S) t'
\end{align*}
\]

**Issues**
- transitivity
- computation needs checks
- decidability

**Solutions**
- bidirectional typing
- type-based elaboration
- over approximation
### New terms...

**GCIC:** ･･･ | ?

**castCIC:** ･･･ | ?_T | err_T | ⟨B ⇐ A⟩ t
New terms...

GCIC: ⋯ | ?

castCIC: ⋯ | ?_T | err_T | ⟨B ⇔ A⟩ t

... and their semantics
New terms...

GCIC: ⋯ | ?

castCIC: ⋯ | ?_T | err_T | ⟨B ⇔ A⟩ t

... and their semantics

• ? elaborates to the least precise term: Γ ⊨ ? ≈ ? □ ≥ ? □
NEW TERMS

New terms...

GCIC: ⋯ | ?

castCIC: ⋯ | ?_T | err_T | ⟨B ← A⟩ t

... and their semantics

• ? elaborates to the least precise term: Γ ⊢ ? ⇝ ??□ ▷ ?□
• ? and err as errors: if ?_B return T then t else t′ ⇝ ?_T
**New terms...**

**GCIC:** \( \cdots | ? \)

**castCIC:** \( \cdots | ?_T | \text{err}_T | \langle B \leftarrow A \rangle t \)

**... and their semantics**

- \( ? \) elaborates to the least precise term: \( \Gamma \vdash ? \rightsquigarrow ? \square \Rightarrow ? \square \)
- \( ? \) and \( \text{err} \) as errors: \( \text{if} \ ?_B \ \text{return} \ T \ \text{then} \ t \ \text{else} \ t' \ \mapsto ?_T \)
- casts compute on the type: \( \langle N \leftarrow B \rangle b \mapsto \text{err}_N \)
**New terms...**

<table>
<thead>
<tr>
<th>GCIC: ⋯</th>
<th>?</th>
</tr>
</thead>
<tbody>
<tr>
<td>castCIC: ⋯</td>
<td>?_T</td>
</tr>
</tbody>
</table>

**... and their semantics**

- ? elaborates to the least precise term: \( \Gamma \vdash ? \leadsto ?\Box \triangleright ?\Box \)
- ? and err as errors: if ?_B return T then t else t' \( \mapsto ?_T \)
- casts compute on the type: \( ⟨\textbf{N} ← \textbf{B}⟩ b \mapsto \text{err}_\textbf{N} \)
- ?\Box with casts as constructors/destructors:
  \( ⟨X ← ?\Box⟩ ⟨?\Box ← \textbf{N}⟩ n \mapsto ⟨X ← \textbf{N}⟩ n \)
THEOREMS!
THE SYNTACTIC THEOREMS

Type theory
Progress, preservation, normalization for C\textsc{ast} C\textsc{ic}.

Proof: extension of Sozeau et al., 2020 + translation back to C\textsc{ic}.

Static gradual guarantee
Elaboration is monotone with respect to loss of precision.

Syntactic precision.
Key property: precision is a simulation for reduction (Siek et al., 2015)
monotony of consistency.

Conservativity
C\textsc{ic} embeds faithfully into G\textsc{ic}.

Proof: on C\textsc{ic} consistency = conversion, using simulation again.
**Type theory**
Progress, preservation, normalization for castcIC.

Proof: extension of Sozeau et al., 2020 + translation back to CIC.

Static gradual guarantee

Elaboration is monotone with respect to loss of precision.

Syntactic precision.

Key property: precision is a simulation for reduction (Siek et al., 2015)

Monotony of consistency.

Conservativity

CIC embeds faithfully into GCIC.

Proof: on CIC consistency = conversion, using simulation again.
<table>
<thead>
<tr>
<th>Type theory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Progress, preservation, normalization for castcic.</td>
</tr>
</tbody>
</table>

Proof: extension of Sozeau et al., 2020 + translation back to CIC.
The syntactic theorems

**Type theory**
Progress, preservation, normalization for castCIC.

Proof: extension of Sozeau et al., 2020 + translation back to CIC.

**Static gradual guarantee**
Elaboration is monotone with respect to loss of precision.
\[ \Rightarrow \text{Syntactic precision.} \]
The syntactic theorems

<table>
<thead>
<tr>
<th>Type theory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Progress, preservation, normalization for castcic.</td>
</tr>
</tbody>
</table>

Proof: extension of Sozeau et al., 2020 + translation back to CIC.

<table>
<thead>
<tr>
<th>Static gradual guarantee</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elaboration is monotone with respect to loss of precision.</td>
</tr>
<tr>
<td>⇒ Syntactic precision.</td>
</tr>
</tbody>
</table>

Key property: precision is a simulation for reduction (Siek et al., 2015) ⇒ monotony of consistency.
The syntactic theorems

**Type theory**
Progress, preservation, normalization for castcic.

Proof: extension of Sozeau et al., 2020 + translation back to cic.

**Static gradual guarantee**
Elaboration is monotone with respect to loss of precision.
\[ \Rightarrow \text{Syntactic precision.} \]

Key property: precision is a simulation for reduction (Siek et al., 2015)
\[ \Rightarrow \text{monotony of consistency.} \]

**Conservativity**
cic embeds faithfully into GCIC.
### Type theory
Progress, preservation, normalization for cast\text{CIC}.

Proof: extension of Sozeau et al., 2020 + translation back to \text{CIC}.

### Static gradual guarantee
Elaboration is monotone with respect to loss of precision.
\[ \Rightarrow \text{Syntactic precision}. \]

Key property: precision is a simulation for reduction (Siek et al., 2015)
\[ \Rightarrow \text{monotony of consistency}. \]

### Conservativity
\text{CIC} embeds faithfully into \text{GCIC}.

Proof: on \text{CIC} consistency = conversion, using simulation again.
Dynamic gradual guarantee
Reduction is monotone with respect to loss of precision.

Graduality (New and Ahmed, 2018)

If $A \Rightarrow B$ then $h_A(B)$ and $h_B(A)$ form an embedding-projection pair.

Semantic precision.
Proof: build order models to interpret precision.
**Dynamic gradual guarantee**

Reduction is monotone with respect to loss of precision.

(Eremondi et al., 2019)

Graduality (New and Ahmed, 2018)

If $A \subseteq B$ then $h(A) \subseteq h(B)$ and $h(B) \subseteq h(A)$ form an embedding-projection pair.
### Dynamic gradual guarantee

Reduction is monotone with respect to loss of precision.

Allows for degenerate computation $t \mapsto ?$ (Eremondi et al., 2019).
**Dynamic gradual guarantee**

Reduction is monotone with respect to loss of precision.

Allows for degenerate computation $t \rightarrow ?$ ([Eremondi et al., 2019](#)).

---

**Graduality ([New and Ahmed, 2018](#))**

If $A \subseteq B$ then $\langle A \leftarrow B \rangle$ and $\langle B \leftarrow A \rangle$ form an embedding-projection pair.

$\Rightarrow$ Semantic precision.
Dynamic gradual guarantee
Reduction is monotone with respect to loss of precision.

Allows for degenerate computation $t \mapsto ?$ (Eremondi et al., 2019).

Graduality (New and Ahmed, 2018)
If $A \sqsubseteq B$ then $\langle A \leftrightarrow B \rangle$ and $\langle B \leftrightarrow A \rangle$ form an embedding-projection pair.
⇒ Semantic precision.

Proof: build order models to interpret precision.
## The Semantic Theorems

### Dynamic Gradual Guarantee

Reduction is monotone with respect to loss of precision.

Allows for degenerate computation $t \mapsto ?$ (Eremondi et al., 2019).

### Graduality (New and Ahmed, 2018)

If $A \subseteq B$ then $\langle A \leftarrow B \rangle$ and $\langle B \leftarrow A \rangle$ form an embedding-projection pair.

$\Rightarrow$ Semantic precision.

Proof: build order models to interpret precision.
That’s it for today
A quick summary

- No go: fire triangle of graduality
- GCIC: one system, three variants
- Type-based elaboration to castCIC
- Safety and normalization
- Conservativity, static gradual guarantee and graduality
- Equality: still in progress

Thank you!