cedille

Tooling: Interactive Commands

Interactive Commands

- Backend sends all data in one batch

Interactive Commands

- Backend sends all data in one batch
- Sometimes we want more information than was given, but wouldn't be realistic to include for all nodes

Interactive Commands

- Backend sends all data in one batch
- Sometimes we want more information than was given, but wouldn't be realistic to include for all nodes
- Hence the need for "Interactive" Commands

Interactive Commands

- Backend sends all data in one batch
- Sometimes we want more information than was given, but wouldn't be realistic to include for all nodes
- Hence the need for "Interactive" Commands
- Node Commands

Interactive Commands

- Backend sends all data in one batch
- Sometimes we want more information than was given, but wouldn't be realistic to include for all nodes
- Hence the need for "Interactive" Commands
- Node Commands
    - "C-i n" – Normalize

Interactive Commands

- Backend sends all data in one batch
- Sometimes we want more information than was given, but wouldn't be realistic to include for all nodes
- Hence the need for "Interactive" Commands
- Node Commands
  - "C-i n" – Normalize
  - "C-i h" – Head-normalize

Interactive Commands

- Backend sends all data in one batch
- Sometimes we want more information than was given, but wouldn't be realistic to include for all nodes
- Hence the need for "Interactive" Commands
- Node Commands
  - "C-i n" – Normalize
  - "C-i h" – Head-normalize
  - "C-i e" – Erase

Interactive Commands

- Backend sends all data in one batch
- Sometimes we want more information than was given, but wouldn't be realistic to include for all nodes
- Hence the need for "Interactive" Commands
- Node Commands
  - "C-i n" – Normalize
  - "C-i h" – Head-normalize
  - "C-i e" – Erase
- Beta-Reduction Commands

Interactive Commands

- Backend sends all data in one batch
- Sometimes we want more information than was given, but wouldn't be realistic to include for all nodes
- Hence the need for "Interactive" Commands
- Node Commands
  - "C-i n" – Normalize
  - "C-i h" – Head-normalize
  - "C-i e" – Erase
- Beta-Reduction Commands
  - "C-i b" – Prompt for expression to open in beta-reduction buffer

Interactive Commands

- Backend sends all data in one batch
- Sometimes we want more information than was given, but wouldn't be realistic to include for all nodes
- Hence the need for "Interactive" Commands
- Node Commands
  - "C-i n" – Normalize
  - "C-i h" – Head-normalize
  - "C-i e" – Erase
- Beta-Reduction Commands
  - "C-i b" – Prompt for expression to open in beta-reduction buffer
  - "C-i B" – Use selected node in beta-reduction buffer

Interactive Commands

- Backend sends all data in one batch
- Sometimes we want more information than was given, but wouldn't be realistic to include for all nodes
- Hence the need for "Interactive" Commands
- Node Commands
  - "C-i n" – Normalize
  - "C-i h" – Head-normalize
  - "C-i e" – Erase
- Beta-Reduction Commands
  - "C-i b" – Prompt for expression to open in beta-reduction buffer
  - "C-i B" – Use selected node in beta-reduction buffer
  - "C-i t" – Use selected node's type in beta-reduction buffer

Interactive Commands: Beta-Reduction Buffer

- Navigation commands (“f”, “b”, “n”, “p”, etc...)

Interactive Commands: Beta-Reduction Buffer

- Navigation commands (“f”, “b”, “n”, “p”, etc...)
- “C-i n” – Normalize and replace the selected node
- “C-i h” – Head-normalize and replace the selected node

Interactive Commands: Beta-Reduction Buffer

- Navigation commands (“f”, “b”, “n”, “p”, etc...)
- “C-i n” – Normalize and replace the selected node
- “C-i h” – Head-normalize and replace the selected node
- “C-i r” – Prompt for proof of equation, and rewrite and replace the selected node with it
- “C-i R” – Like “C-i r”, but normalizes as it looks for matches (same as ρ+)

Interactive Commands: Beta-Reduction Buffer

- Navigation commands (“f”, “b”, “n”, “p”, etc...)
- “C-i n” – Normalize and replace the selected node
- “C-i h” – Head-normalize and replace the selected node
- “C-i r” – Prompt for proof of equation, and rewrite and replace the selected node with it
- “C-i R” – Like “C-i r”, but normalizes as it looks for matches (same as ρ+)
- “C-i =” – Prompt for expression, and replace the selected node with it if convertible

Interactive Commands: Beta-Reduction Buffer

- Navigation commands (“f”, “b”, “n”, “p”, etc...)
- “C-i n” – Normalize and replace the selected node
- “C-i h” – Head-normalize and replace the selected node
- “C-i r” – Prompt for proof of equation, and rewrite and replace the selected node with it
- “C-i R” – Like “C-i r”, but normalizes as it looks for matches (same as $\rho+$)
- “C-i =” – Prompt for expression, and replace the selected node with it if convertible
- “C-i ,” – Undo
- “C-i .” – Redo

Interactive Commands: Beta-Reduction Buffer

- Navigation commands ("f", "b", "n", "p", etc...)
- "C-i n" – Normalize and replace the selected node
- "C-i h" – Head-normalize and replace the selected node
- "C-i r" – Prompt for proof of equation, and rewrite and replace the selected node with it
- "C-i R" – Like "C-i r", but normalizes as it looks for matches (same as $\rho+$)
- "C-i =" – Prompt for expression, and replace the selected node with it if convertible
- "C-i ," – Undo
- "C-i ." – Redo
- "C-i p" – Reconstruct a proof from each step taken so far

Interactive Commands: One More

- "E" – Elaborate to Cedille Core

$\rightsquigarrow$ $\texttt{cedille}_{\texttt{core}}$

Elaboration to Cedille Core

What is Cedille Core?

- Independent implementation of CDLE
- Full annotations required
  - No type inference
  - No bidirectionality
- More verbose, much easier to check
- Fewer than 1000 lines of Haskell code

Cedille vs. Cedille Core

$(\lambda\ x.\ t)$

$(\lambda\ x : T.\ t)$

$[\ t_1,\ t_2\ ]$

δ

Type Checking

$(\rho\ e\ @\ x.\ T_x\ -\ t)$

φ β

$(\beta<u_T>\{u_t\})$

Scope Shadowing

ε

$[\ t_1,\ t_2\ @\ x.\ T_x\ ]$

$(\beta\{u_t\})$

Type Synthesis

ς

χ

$(\rho<3\ 4\ 6>\ e\ -\ t)$

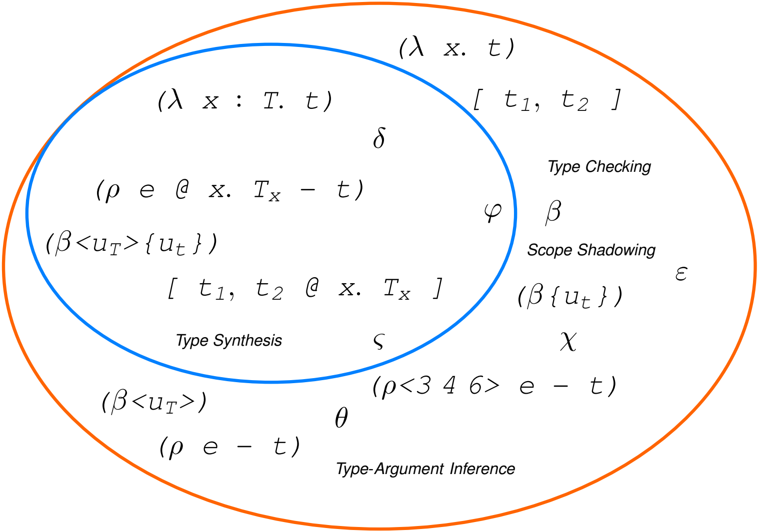
$(\beta<u_T>)$

θ

$(\rho\ e\ -\ t)$

Type-Argument Inference

Cedille vs. Cedille Core



Cedille $\rightsquigarrow$ Cedille Core

- Classify λ-terms

  $\lambda\ x.\ t \rightsquigarrow \lambda\ x : T.\ t$

Cedille $\leadsto$ Cedille Core

- Classify λ-terms

  $\lambda\ x.\ t \leadsto \lambda\ x : T.\ t$

- Insert inferred type arguments

  $id\ zero \leadsto id \cdot Nat\ zero$

Cedille $\rightsquigarrow$ Cedille Core

- Classify λ-terms

  $\lambda\ x.\ t \rightsquigarrow \lambda\ x : T.\ t$

- Insert inferred type arguments

  $id\ zero \rightsquigarrow id \cdot Nat\ zero$

- Provide dependent type of second projection in ι-pairs

  $[\ zero',\ zeroi\ ] \rightsquigarrow [\ zero',\ zeroi\ @\ x.\ NatI\ x\ ]$

Cedille $\leadsto$ Cedille Core

- Classify λ-terms

  $\lambda\ x.\ t \leadsto \lambda\ x : T.\ t$

- Insert inferred type arguments

  $id\ zero \leadsto id \cdot Nat\ zero$

- Provide dependent type of second projection in ι-pairs

  $[\ zero',\ zeroi\ ] \leadsto [\ zero',\ zeroi\ @\ x.\ NatI\ x\ ]$

- Ensure β-terms supply both a term u for proving $\{\ u \simeq u\ \}$ and one for erasure

  $\beta \leadsto \beta\text{<}u\text{>}\{\lambda\ x.\ x\}$

Cedille $\leadsto$ Cedille Core

- Classify λ-terms

  $\lambda\ x.\ t \leadsto \lambda\ x : T.\ t$

- Insert inferred type arguments

  $id\ zero \leadsto id \cdot Nat\ zero$

- Provide dependent type of second projection in ι-pairs

  $[\ zero',\ zeroi\ ] \leadsto [\ zero',\ zeroi\ @\ x.\ NatI\ x\ ]$

- Ensure β-terms supply both a term u for proving $\{\ u \simeq u\ \}$ and one for erasure

  $\beta \leadsto \beta\text{<}u\text{>}\{\lambda\ x.\ x\}$

- Convert δ-contradictions to Church-encoded $\{\ tt \simeq ff\ \}$

  $\{suc\ n \simeq zero\} \leadsto \{suc\ n\ ff\ (\lambda\ \_.\ tt) \simeq zero\ ff\ (\lambda\ \_.\ tt)\}$

Cedille $\rightsquigarrow$ Cedille Core

- Classify λ-terms

  $\lambda\ x.\ t \rightsquigarrow \lambda\ x : T.\ t$
- Insert inferred type arguments

  $id\ zero \rightsquigarrow id \cdot Nat\ zero$
- Provide dependent type of second projection in ι-pairs

  $[\ zero',\ zeroi\ ] \rightsquigarrow [\ zero',\ zeroi\ @\ x.\ NatI\ x\ ]$
- Ensure β-terms supply both a term u for proving $\{\ u \simeq u\ \}$ and one for erasure

  $\beta \rightsquigarrow \beta\langle u\rangle\{\lambda\ x.\ x\}$
- Convert δ-contradictions to Church-encoded $\{\ tt \simeq ff\ \}$

  $\{suc\ n \simeq zero\} \rightsquigarrow \{suc\ n\ ff\ (\lambda\ \_.\ tt) \simeq zero\ ff\ (\lambda\ \_.\ tt)\}$
- Specify where rewrites occur in ρ-terms

  $\rho\ e - t \rightsquigarrow \rho\ e\ @\ x.\ T_x - t$

Cedille $\leadsto$ Cedille Core

- Classify λ-terms

  $\lambda\ x.\ t \leadsto \lambda\ x : T.\ t$

- Insert inferred type arguments

  $id\ zero \leadsto id \cdot Nat\ zero$

- Provide dependent type of second projection in ι-pairs

  $[\ zero',\ zeroi\ ] \leadsto [\ zero',\ zeroi\ @\ x.\ NatI\ x\ ]$

- Ensure β-terms supply both a term u for proving $\{\ u \simeq u\ \}$ and one for erasure

  $\beta \leadsto \beta\text{<}u\text{>}\{\lambda\ x.\ x\}$

- Convert δ-contradictions to Church-encoded $\{\ tt \simeq ff\ \}$

  $\{suc\ n \simeq zero\} \leadsto \{suc\ n\ ff\ (\lambda\ \_.\ tt) \simeq zero\ ff\ (\lambda\ \_.\ tt)\}$

- Specify where rewrites occur in ρ-terms

  $\rho\ e - t \leadsto \rho\ e\ @\ x.\ T_x - t$

  - Rewriting is <u>not</u> guaranteed to preserve types' kindability!

Type-Preserving Rewrites: Problem A

Context

$\underline{A} : \star \quad \underline{B} : A \to \star \quad \underline{a} : A \quad \underline{e} : \{ \ a \simeq \lambda\ x.\ x\ x \ \}$

$$\rho\ e - \lambda\ b.\ b \Leftarrow (B\ a \to B\ a)$$

$$\rightsquigarrow$$

$$\rho\ e\ @\ x.\ (B\ x \to B\ x) -$$
$$\lambda\ b : B\ (\lambda\ x.\ x\ x).\ b$$

$$(\lambda\ x.\ x\ x) \Rightarrow ???$$

Type-Preserving Rewrites: Solution A

$$\mathtt{Context}$$

$$\underline{A} : \star \quad \underline{B} : A \to \star \quad \underline{a} : A \quad \underline{e} : \{\ a \simeq \lambda\ x.\ x\ x\ \}$$

$$\rho\ e\ -\ \lambda\ b.\ b \Leftarrow (B\ a \to B\ a)$$

$$\rightsquigarrow$$

$$\rho\ e\ @\ x.\ (B\ x \to B\ x)\ -$$
$$\lambda\ b : B\ (\underline{\varphi\ e\ -\ a\ \{\lambda\ x.\ x\ x\}}).\ b$$

Type-Preserving Rewrites: Problem B

$$\mathtt{Context}$$

$$\underline{A} : \star \quad \underline{B} : (\Pi\ y : A.\ \Pi\ z : A.\ \{\ y \simeq z\ (\lambda\ x.\ x)\ \} \to \star)$$

$$\underline{a} : A \quad \underline{a'} : A$$

$$\underline{e} : \{\ a \simeq a'\ (\lambda\ x.\ x)\ \} \quad \underline{p} : \{\ a \simeq \lambda\ x.\ x\ \}$$

$$\rho\ p - \lambda\ b.\ b \Leftarrow (B\ a\ a'\ e \to B\ a\ a'\ e)$$

$$\leadsto$$

$$\rho\ p\ @\ x.\ (B\ x\ a'\ e \to B\ x\ a'\ e)\ -$$
$$\lambda\ b : B\ (\varphi\ p - a\ \{\lambda\ x.\ x\})\ a'\ e.\ b$$

$$e \Rightarrow \{\ a \simeq a'\ (\lambda\ x.\ x)\ \}$$

$$e \not\Rightarrow \{\ \lambda\ x.\ x \simeq a'\ (\lambda\ x.\ x)\ \}$$

Type-Preserving Rewrites: Solution B

$$\texttt{Context}$$

$$\underline{A} : \star \quad \underline{B} : (\Pi\ y : A.\ \Pi\ z : A.\ \{\ y \simeq z\ (\lambda\ x.\ x)\ \} \to \star)$$

$$\underline{a} : A \quad \underline{a'} : A$$

$$\underline{e} : \{\ a \simeq a'\ (\lambda\ x.\ x)\ \} \quad \underline{p} : \{\ a \simeq \lambda\ x.\ x\ \}$$

$$\rho\ p - \lambda\ b.\ b \Leftarrow (B\ a\ a'\ e \to B\ a\ a'\ e)$$

$$\rightsquigarrow$$

$$\begin{array}{l} \rho\ p\ @\ x.\ (B\ x\ a'\ e \to B\ x\ a'\ e)\ - \\ \quad \lambda\ b : B\ (\varphi\ p - a\ \{\lambda\ x.\ x\})\ a' \\ \qquad\qquad (\underline{\rho\ \varsigma\ p\ @\ x.\ \{\ x \simeq a'\ (\lambda\ x.\ x)\ \}\ -\ e).} \\ \quad\ b \end{array}$$