

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

- 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 the selected node with it
- “C-i R” – Like “C-i r”, but normalizes as it looks for matches (same as  $\rho+$ )



## 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 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

# 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 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 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: 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 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

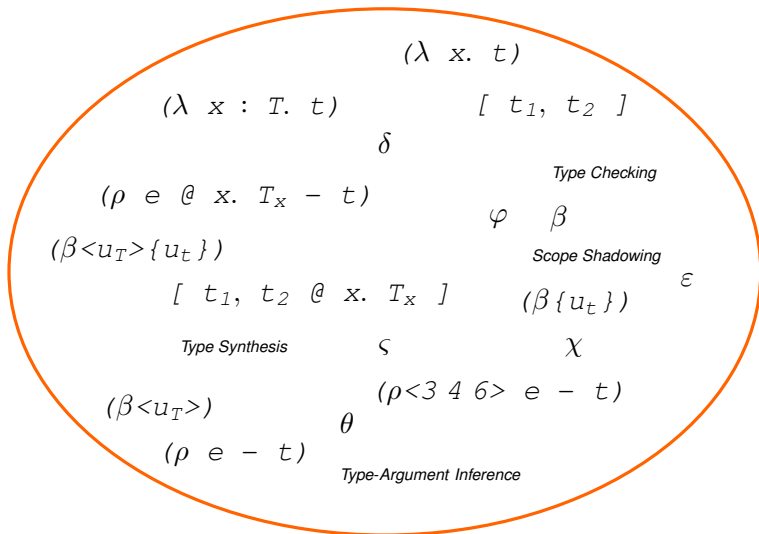
$\rightsquigarrow$  cedille<sub>core</sub>

Elaboration to Cedille Core

# What is Cedille Core?

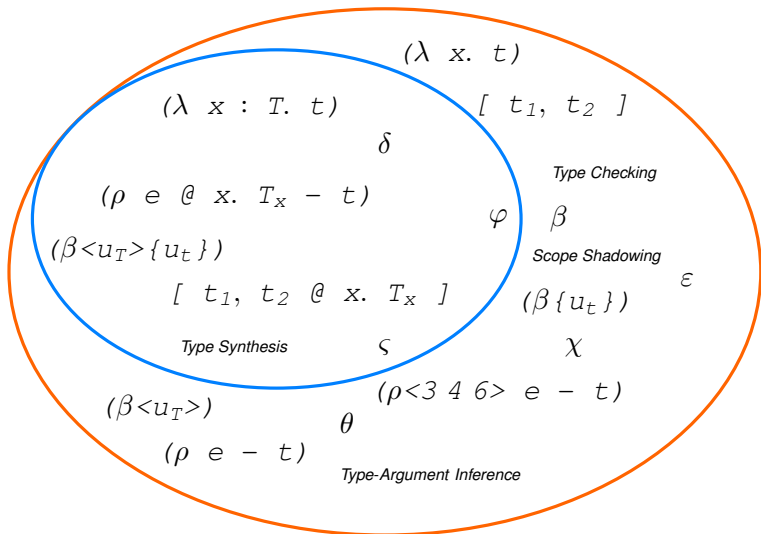
- Independent implementation of CDLE
- Full annotations required
  - No type inference
  - No bidirectionality
- More verbose, much easier to check
- Fewer than 1000 lines of Haskell code

# Cedille vs. Cedille Core





# Cedille vs. Cedille Core



# Cedille $\rightsquigarrow$ Cedille Core

- Classify  $\lambda$ -terms

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

# Cedille $\rightsquigarrow$ Cedille Core

- Classify  $\lambda$ -terms

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

- Insert inferred type arguments

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

# Cedille $\rightsquigarrow$ Cedille Core

- Classify  $\lambda$ -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  $\iota$ -pairs

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

# Cedille $\rightsquigarrow$ Cedille Core

- Classify  $\lambda$ -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  $\iota$ -pairs

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

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

$$\beta \rightsquigarrow \beta <u> \{ \lambda x. x \}$$

# Cedille $\rightsquigarrow$ Cedille Core

- Classify  $\lambda$ -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  $\iota$ -pairs

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

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

$$\beta \rightsquigarrow \beta <u> \{ \lambda x. x \}$$

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

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

# Cedille $\rightsquigarrow$ Cedille Core

- Classify  $\lambda$ -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  $\iota$ -pairs

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

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

$$\beta \rightsquigarrow \beta <u> \{ \lambda x. x \}$$

- Convert  $\delta$ -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  $\rho$ -terms

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

# Cedille $\rightsquigarrow$ Cedille Core

- Classify  $\lambda$ -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  $\iota$ -pairs

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

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

$$\beta \rightsquigarrow \beta <u> \{ \lambda x. x \}$$

- Convert  $\delta$ -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  $\rho$ -terms

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

- Rewriting is not guaranteed to preserve typeability!



# Cedille $\rightsquigarrow$ Cedille Core

- Classify  $\lambda$ -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  $\iota$ -pairs

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

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

$$\beta \rightsquigarrow \beta <u> \{ \lambda x. x \}$$

- Convert  $\delta$ -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  $\rho$ -terms

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

- Rewriting is not guaranteed to preserve typeability!

# Type-Preserving Rewrites: Problem A

*Context*

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

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

$\rightsquigarrow$

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

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

# Type-Preserving Rewrites: Solution A

*Context*

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

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

$\rightsquigarrow$

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

# Type-Preserving Rewrites: Problem B

*Context*

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

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

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

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

$\rightsquigarrow$

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

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

# Type-Preserving Rewrites: Solution B

Context

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

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

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

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

$\rightsquigarrow$

$\rho \ p \ @ \ x. (B \ x \ a' \ e \rightarrow B \ x \ a' \ e) -$

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

$(\rho \varsigma \ p \ @ \ x. \{ x \simeq a' \ (\lambda x. x) \} - e).$

$b$

# Cedille $\rightsquigarrow$ Cedille Core

- Classify  $\lambda$ -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  $\iota$ -pairs

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

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

$$\beta \rightsquigarrow \beta <u> \{ \lambda x. x \}$$

- Convert  $\delta$ -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  $\rho$ -terms

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

- Rewriting is guaranteed to preserve typeability!