

# Programming with proofs and explicit contexts

Joshua Dunfield

Joint work with Brigitte Pientka

McGill University

*PPDP'08*, Valencia, Spain

16 July 2008

# First-Order Data

- Types and constructors

nat : **type**.

Zero : nat.

Suc : nat  $\rightarrow$  nat.

case e of

Zero  $\Rightarrow$  ...

| Suc u  $\Rightarrow$  ...

# First-Order Data

- Types and constructors

$\text{nat} : \mathbf{type}.$	$\text{case } e \text{ of}$
$\text{Zero} : \text{nat}.$	$\text{Zero} \Rightarrow \dots$
$\text{Suc} : \text{nat} \rightarrow \text{nat}.$	$  \text{Suc } u \Rightarrow \dots$

- No support for variable binding

# Higher-Order Abstract Syntax

- Object-level binders as meta-level binders

$o$  : **type**.

$eq$  :  $\text{nat} \rightarrow \text{nat} \rightarrow o$ .

$imp$  :  $o \rightarrow o \rightarrow o$ .

$forall$  :  $(\text{nat} \rightarrow o) \rightarrow o$ .

- $\forall x. (x = x)$  as  $forall \lambda x. (eq\ x\ x)$
- Built-in  $\alpha$ -equivalence and substitution
- Twelf,  $\lambda$ Prolog, Elphin, Delphin, Abella, Bedwyr, ...

## ... Beluga



- Data layer (LF) + Computation layer
- A language for programming and writing proofs

# Issues



- Analyzing higher-order data:  
case on forall  $\lambda x. (eq\ x\ x)$
- Recursion into open terms
- Manipulation of variables

# Issues



- Analyzing higher-order data:  
case on `forall`  $\lambda x. (eq\ x\ x)$
- Recursion into open terms
- Manipulation of variables

Our solution:

**Contextual** open data

Explicit contexts characterized by schemas

# Contextual Open Data

- Contextual Modal Type Theory  
[Nanevski, Pfenning, Pientka 2008]
- Type  $o[\Psi]$  of formulas referring to variables in  $\Psi$ :

Object  $(\text{forall } \lambda x. (\text{eq } x \ x))$  :  $o[\cdot]$

Object  $(\text{eq } x \ x)$  :  $o[x:\text{nat}]$

Meta-variable  $u[x]$  :  $o[x:\text{nat}]$

- $A[\Psi]$  closed with respect to  $\Psi$ 
  - Prevents extrusion

# Explicit Contexts

$$\Psi ::= \cdot \mid \psi \mid \Psi, x:A$$

- Context variables  $\psi$  abstract over contexts

Context	element	
$x:\text{nat}, y:\text{nat}$	$x$	Concrete parameter
$\psi, x:\text{nat}, y:\text{nat}$	$p[\text{id}_\psi]$	Parameter variable

- Contexts are ordered

# Counting Variables in `nats`

$\text{rec } cntVN : \Pi \psi : (\text{nat})^* . \text{nat}[\psi, x:\text{nat}] \rightarrow \text{nat}[\cdot]$   
 $= \Lambda \psi . \text{fn } n . \text{case } n \text{ of}$

$\text{box}(\psi, x . x) \Rightarrow \text{box}(. \text{Suc } \text{Zero})$	} Variable cases
$  \text{box}(\psi, x . p[\text{id}_\psi]) \Rightarrow \text{box}(. \text{Zero})$	
$  \text{box}(\psi, x . \text{Zero}) \Rightarrow \text{box}(. \text{Zero})$	} Constructor cases
$  \text{box}(\psi, x . \text{Suc } u[\text{id}_\psi, x]) \Rightarrow$ $cntVN [\psi] \text{box}(\psi, x . u[\text{id}_\psi, x])$	

- Context variables bound by  $\Lambda \psi . \dots$
- $p[\sigma]$ : Instantiated with a variable
- $u[\sigma]$ : Instantiated with any object

# Counting Variables in Formulas

rec  $cntV : \Pi \psi : (\text{nat})^* . o[\psi, x:\text{nat}] \rightarrow \text{nat}[\cdot]$   
 $= \Lambda \psi . \text{fn } f . \text{case } f \text{ of}$

...

|  $\text{box}(\psi, x . \text{forall } \lambda y . u[\text{id}_\psi, x, y]) \Rightarrow$

$cntV \ [\psi, y:\text{nat}] \ \text{box}(\psi, y, x . u[\text{id}_\psi, x, y])$

|  $\text{box}(\psi, x . \text{eq } u[\text{id}_\psi, x] \ v[\text{id}_\psi, x]) \Rightarrow$

$add \ (cntVN \ [\psi] \ \text{box}(\psi, x . u[\text{id}_\psi, x]))$   
 $\ (cntVN \ [\psi] \ \text{box}(\psi, x . v[\text{id}_\psi, x]))$

Contexts are ordered

# Programming with Proofs

- Higher-order dependent data
- Hilbert-style axioms

$\text{hil} : \circ \rightarrow \mathbf{type}.$

$K : \text{hil} (\text{imp } A (\text{imp } B A)).$

$S : \text{hil} (\text{imp} (\text{imp } A (\text{imp } B C)) (\text{imp} (\text{imp } A B) (\text{imp } A C))).$

$MP : \text{hil} (\text{imp } A B) \rightarrow \text{hil } A \rightarrow \text{hil } B.$

- Inductive proofs ~ recursive programs...

# Bracket Abstraction

$\text{hil} : \circ \rightarrow \mathbf{type}.$

$K : \text{hil} (\text{imp } A (\text{imp } B A)).$

$S : \text{hil} (\text{imp} (\text{imp } A (\text{imp } B C)) (\text{imp} (\text{imp } A B) (\text{imp } A C))).$

$\text{MP} : \text{hil} (\text{imp } A B) \rightarrow \text{hil } A \rightarrow \text{hil } B.$

- **Lemma.**

If  $\mathcal{D} :: \Gamma, x:\text{hil } A \vdash \text{hil } B$

then  $\mathcal{D}' :: \Gamma \vdash \text{hil} (\text{imp } A B)$

# Bracket Abstraction

$\text{hil} : \circ \rightarrow \mathbf{type}.$

$K : \text{hil} (\text{imp } A (\text{imp } B A)).$

$S : \text{hil} (\text{imp} (\text{imp } A (\text{imp } B C)) (\text{imp} (\text{imp } A B) (\text{imp } A C))).$

$\text{MP} : \text{hil} (\text{imp } A B) \rightarrow \text{hil } A \rightarrow \text{hil } B.$

- **Lemma.**

If  $d :: \gamma, x:\text{hil } A \vdash \text{hil } B$

then  $d' :: \gamma \vdash \text{hil} (\text{imp } A B)$

# Bracket Abstraction

$\text{hil} : \circ \rightarrow \mathbf{type}.$

$K : \text{hil} (\text{imp } A (\text{imp } B A)).$

$S : \text{hil} (\text{imp} (\text{imp } A (\text{imp } B C)) (\text{imp} (\text{imp } A B) (\text{imp } A C))).$

$\text{MP} : \text{hil} (\text{imp } A B) \rightarrow \text{hil } A \rightarrow \text{hil } B.$

- **Lemma.** For all contexts  $\gamma$  and formulas  $A, B$ :  
If  $d :: \gamma, x:\text{hil } A \vdash \text{hil } B$   
then  $d' :: \gamma \vdash \text{hil} (\text{imp } A B)$

# Bracket Abstraction

$\text{hil} : \circ \rightarrow \mathbf{type}.$

$K : \text{hil} (\text{imp } A (\text{imp } B A)).$

$S : \text{hil} (\text{imp} (\text{imp } A (\text{imp } B C)) (\text{imp} (\text{imp } A B) (\text{imp } A C))).$

$\text{MP} : \text{hil} (\text{imp } A B) \rightarrow \text{hil } A \rightarrow \text{hil } B.$

- **Lemma.** For all contexts  $\gamma$  and formulas  $A, B$ :

If  $d :: \gamma, x:\text{hil } A \vdash \text{hil } B$

then  $d' :: \gamma \vdash \text{hil} (\text{imp } A B)$

- **Type**

$\prod \gamma:\text{hilCtx}. \prod^{\square} A::\circ. \prod^{\square} B::\circ.$

$(\text{hil } B \quad )[\gamma, x:\text{hil } A \quad ] \rightarrow (\text{hil} (\text{imp } A B))[\gamma]$

# Bracket Abstraction

$\text{hil} : \circ \rightarrow \mathbf{type}.$

$K : \text{hil} (\text{imp } A (\text{imp } B A)).$

$S : \text{hil} (\text{imp} (\text{imp } A (\text{imp } B C)) (\text{imp} (\text{imp } A B) (\text{imp } A C))).$

$\text{MP} : \text{hil} (\text{imp } A B) \rightarrow \text{hil } A \rightarrow \text{hil } B.$

- **Lemma.** For all contexts  $\gamma$  and formulas  $A, B$ :

If  $d :: \gamma, x:\text{hil } A \vdash \text{hil } B$

then  $d' :: \gamma \vdash \text{hil} (\text{imp } A B)$

- **Type**

$\prod \gamma:\text{hilCtx}. \prod^{\square} A::\circ[\gamma]. \prod^{\square} B::\circ[\gamma].$

$(\text{hil } B[\text{id}_{\gamma}])[\gamma, x:\text{hil } A[\text{id}_{\gamma}]] \rightarrow (\text{hil} (\text{imp } A[\text{id}_{\gamma}] B[\text{id}_{\gamma}])))[\gamma]$

# Context Schemas

$$\gamma = d1:\text{hil } A_1, d2:\text{hil } A_2, \dots$$

Elements have the form  $\text{hil } A$  for some formula  $A$ .

Described by **context schema**

$$\text{hilCtx} = (\text{all } A:o.\text{hil } A)^*$$

# Context Schemas

$$\gamma = d1:\text{hil } A_1, d2:\text{hil } A_2, \dots$$

Elements have the form  $\text{hil } A$  for some formula  $A$ .  
Described by **context schema**

$$\text{hilCtx} = (\text{all } A:o.\text{hil } A)^*$$

## Type

$\prod \gamma: \text{hilCtx}. \prod A::o[\gamma]. \prod B::o[\gamma].$

$$(\text{hil } B[\text{id}_\gamma])[\gamma, x:\text{hil } A[\text{id}_\gamma]] \rightarrow (\text{hil } (\text{imp } A[\text{id}_\gamma] B[\text{id}_\gamma]))[\gamma]$$

# Inductive Proof as Recursive Program

rec *ded* :  $\Pi \gamma:\text{hilCtx}. \Pi^{\square} A::\text{o}[\gamma]. \Pi^{\square} B::\text{o}[\gamma].$

(*hil* B[id<sub>γ</sub>])[γ, x:*hil* A[id<sub>γ</sub>]] → (*hil* (*imp* A[id<sub>γ</sub>] B[id<sub>γ</sub>]))[γ]

=  $\Lambda \gamma.$  fn d. case d of

box(γ, x. *K*) ⇒ box(. *MP* *K* *K*)

| box(γ, x. *S*) ⇒ box(. *MP* *K* *S*)

| box(γ, x. x) ⇒ box(. *MP* (*MP* *S* *K*) *K*)

| box(γ, x. p[id<sub>γ</sub>]) ⇒ box(. *MP* *K* p[id<sub>γ</sub>])

| box(γ, x. *MP* d1[id<sub>γ</sub>, x] d2[id<sub>γ</sub>, x]) ⇒ ...

# Outline

- Motivation and examples
- Data
- Contexts
- Computation
- Related work
- Summary

# Data Level

Dependently typed  $\lambda$ -calculus

+ contextual meta-variables  $u[\sigma]$  [Nanevski et al. 2008]

+ parameter variables  $p[\sigma]$  + context variables  $\psi$

Types  $A, B ::= a \ M_1 \dots M_n \mid \Pi x:A.B \mid \Sigma x:A.B$

Normal terms  $M, N ::= \lambda x. M \mid (M, N) \mid R$

Neutral terms  $R ::= x \mid c \mid u[\sigma] \mid p[\sigma] \mid R \ N \mid \text{proj}_k \ R$

Substitutions  $\sigma ::= \cdot \mid \sigma; M \mid \sigma, R \mid \text{id}_\psi$

Data always in canonical form:  $(\lambda x. x) c$  ill-formed  
(guaranteed through hereditary substitutions)

# Context Schemas

Contexts  $\Psi ::= \cdot \mid \psi \mid \Psi, x:A$

Example context

$x:\text{nat}, y:\text{nat}, z:\text{nat}$

$x:\text{bool}, y:\text{nat}, z:\text{bool}$

Schema  $W$

$(\text{nat})^*$

$(\text{nat} + \text{bool})^*$

# Context Schemas

Contexts  $\Psi ::= \cdot \mid \psi \mid \Psi, x:A$

Example context

$x:\text{nat}, y:\text{nat}, z:\text{nat}$

$x:\text{bool}, y:\text{nat}, z:\text{bool}$

$xs:(\text{list } (\text{Suc Zero})), ys:(\text{list } \text{Zero})$

$x:\text{nat}, d:(\text{nd } (\text{eq } x \ x))$

Schema  $W$

$(\text{nat})^*$

$(\text{nat} + \text{bool})^*$

$(\text{all } n:\text{nat}.\text{list } n)^*$

$(\text{nat} + (\text{all } y:\text{o}.\text{nd } y))^*$

# Context Schemas

Element types  $\tilde{A} ::= \Pi x:A. \tilde{A} \mid a \ M_1 \ \cdots \ M_n$

Schema elements  $F ::= \text{all } \cdots. \Sigma \cdots. \tilde{A}$

Instance of some  $F$   $\Sigma \cdots. \tilde{A}$

Schemas  $W ::= (F_1 + \cdots + F_n)^*$

# Computation Level

Computation indexed by data

Types  $\tau ::= A[\Psi] \mid \tau_1 \rightarrow \tau_2 \mid \Pi\psi:W. \tau \mid \Pi^{\square}u::A[\Psi]. \tau$

Expressions  $e ::= y \mid \text{rec } f. e \mid (e : \tau) \mid \text{fn } y. e \mid e_1 e_2$   
 $\mid \Lambda\psi. e \mid e \ulcorner \Psi \urcorner$   
 $\mid \text{box}(\hat{\Psi}. M) \mid (\text{case } e \text{ of } b_1 \mid \dots \mid b_n)$   
 $\mid \lambda^{\square}u. e \mid e \ulcorner \hat{\Psi}. M \urcorner$

Branch  $b ::= \Pi\Delta. \text{box}(\hat{\Psi}. M') : A[\Psi] \mapsto e$

## Contexts (besides $\Psi$ )

- $\Gamma = f:\tau_1, y:\tau_2$     **Computation variables**
  - $\text{rec } f. (\text{fn } y. e)$
- $\Delta = u::A[\Psi], p::A[\psi]$     **Contextual variables**
  - $u$ —meta-variable bound to data objects
  - $p$ —parameter bound to data-level variables
  - $\lambda^\square u. \text{case } \dots \Pi p::A[\psi]. \text{box}(\hat{\Psi}. p[\text{id}_\psi]) : \dots \mapsto e$
- $\Omega = \psi_1:W_1, \psi_2:W_2$     **Context variables**
  - $\psi:W$ : Context bound to  $\psi$  has schema  $W$
  - $\Lambda\psi. e$

# Computation Typing

- Bidirectional:

$$\Omega; \Delta; \Psi \vdash e \Leftarrow \tau$$

$e$  checks against  $\tau$

$$\Omega; \Delta; \Psi \vdash e \Rightarrow \tau$$

$e$  synthesizes  $\tau$

$$\Omega; \Delta; \Psi \vdash b \Leftarrow_{A[\Psi']} \tau$$

branch  $b$  analyzing  $A[\Psi']$   
checks against  $\tau$

# From Data to Computation

- Bidirectional:

$$\Omega; \Delta; \Psi \vdash e \Leftarrow \tau$$

$e$  checks against  $\tau$

$$\Omega; \Delta; \Psi \vdash e \Rightarrow \tau$$

$e$  synthesizes  $\tau$

$$\Omega; \Delta; \Psi \vdash b \Leftarrow_{A[\Psi']} \tau$$

branch  $b$  analyzing  $A[\Psi']$   
checks against  $\tau$

$$\frac{\Omega; \Delta; \Psi \vdash M \Leftarrow A}{\Omega; \Delta; \Psi \vdash \text{box}(\hat{\Psi}. M) \Leftarrow A[\Psi]} \overline{\text{box}}$$

# Context Variables

Context abstraction

$$\frac{\Omega, \psi:W; \Delta; \Gamma \vdash e \Leftarrow \tau}{\Omega; \Delta; \Gamma \vdash \Lambda\psi. e \Leftarrow \Pi\psi:W. \tau} \overline{\Pi}$$

$$\frac{\Omega; \Delta; \Gamma \vdash e \Rightarrow \Pi\psi:W. \tau \quad \Omega; \Delta \vdash \Psi \Leftarrow W}{\Omega; \Delta; \Gamma \vdash e [\Psi] \Rightarrow \llbracket \Psi/\psi \rrbracket \tau} \overline{\Pi E}$$

# $\Pi^\square$ : Computation Dependent Types

Abstraction over data-level index arguments

$$\frac{\Omega; \Delta, u::A[\Psi]; \Gamma \vdash e \Leftarrow \tau}{\Omega; \Delta; \Gamma \vdash \lambda^\square u. e \Leftarrow \Pi^\square u::A[\Psi]. \tau} \overline{\Pi^\square I}$$

$$\frac{\Omega; \Delta; \Gamma \vdash e \Rightarrow \Pi^\square u::A[\Psi]. \tau \quad \Omega; \Delta; \Psi \vdash M \Leftarrow A}{\Omega; \Delta; \Gamma \vdash e [\hat{\Psi}.M] \Rightarrow \llbracket \hat{\Psi}.M/u \rrbracket \tau} \overline{\Pi^\square E}$$

(Reconstructing  $[\dots]$ : ongoing work)

# Typing for branches

## Case expressions

$$\frac{\Omega; \Delta; \Gamma \vdash i \Rightarrow A[\Psi] \quad \text{for all } k \quad \Omega; \Delta; \Gamma \vdash b_k \Leftarrow_{A[\Psi]} \tau}{\Omega; \Delta; \Gamma \vdash \text{case } i \text{ of } b_1 \mid \dots \mid b_n \Leftarrow \tau} \overline{\text{case}}$$

$$\Omega; \Delta_k; \Psi_k \quad \vdash M_k \Leftarrow A_k$$

$$\Omega; \Delta, \Delta_k \quad \vdash \Psi \doteq \Psi_k / (\theta_1, \Delta')$$

$$\Omega; \Delta' \quad \vdash \llbracket \theta_1 \rrbracket A \doteq \llbracket \theta_1 \rrbracket A_k / (\theta_2, \Delta'')$$

$$\Omega; \Delta''; \llbracket \theta_2 \rrbracket \llbracket \theta_1 \rrbracket \Gamma \vdash \llbracket \theta_2 \rrbracket \llbracket \theta_1 \rrbracket e_k \Leftarrow \llbracket \theta_2 \rrbracket \llbracket \theta_1 \rrbracket \tau$$

$$\frac{}{\Omega; \Delta; \Gamma \vdash \prod \Delta_k. \text{box}(\hat{\Psi}. M_k) : A_k[\Psi_k] \mapsto e_k \Leftarrow_{A[\Psi]} \tau}$$

# Operational Semantics

$e \longrightarrow e'$        $e$  evaluates in one step to  $e'$

Evaluation of computation:

$$\overline{(\text{fn } y. e) v \longrightarrow [v/y]e}$$

$$\overline{(\lambda^{\square} u. e) [\hat{\Psi}.M] \longrightarrow [[\hat{\Psi}.M/u]]e}$$

$$\overline{(\Lambda\psi.e) [\Psi] \longrightarrow [[\Psi/\psi]]e}$$

# Operational Semantics of case

Branch  $b$  matches  $\text{box}(\hat{\Psi}. M)$  and steps to  $e'$

$$(\text{box}(\hat{\Psi}. M) : A[\Psi] \doteq b) \longrightarrow e'$$

$$\frac{(\text{box}(\hat{\Psi}. M) : A[\Psi]) \doteq \overbrace{(\prod \Delta. \text{box}(\hat{\Psi}. M_1) : A_1[\Psi_1])}^b / \theta}{(\text{case } (\text{box}(\hat{\Psi}. M) : A[\Psi]) \text{ of } (b \mapsto e_1) \mid bs) \longrightarrow \llbracket \theta \rrbracket e_1}$$

Evaluation of branch:

$$\begin{aligned} \Delta_k \vdash \Psi_k &\doteq \Psi / (\theta_1, \Delta_1) \\ \Delta_1; \Psi \vdash A &\doteq \llbracket \theta_1 \rrbracket A_k / (\theta_2, \Delta_2) \\ \theta &= \llbracket \theta_2 \rrbracket \theta_1 \\ \Delta_2; \Psi \vdash M &\doteq \llbracket \theta \rrbracket M_k / (\theta_3, \cdot) \end{aligned}$$

$$(\text{box}(\hat{\Psi}. M) : A[\Psi] \doteq \prod \Delta_k. \text{box}(\hat{\Psi}. M_k) : A_k[\Psi_k]) / \llbracket \theta_3 \rrbracket \theta$$

# Related Work

- $ML_\lambda$  [Miller 1990]
- Modal  $\lambda$ -calculus with primitive rec.  
[Despeyroux, Pfenning, Schürmann 1997]
  - (Closed) data and computation separated by box modality

# Related Work: Elphin and Delphin

- Elphin:
  - Explicit scope stacks
  - Global contexts
- Delphin [Poswolsky & Schürmann '08]
  - Global contexts
  - $\nabla$  extends context
  - Type for parameters
  - Dependently typed

# Related Work: Nominal Types

- FreshML [Pitts, Gabbay, Shinwell 2003]
  - $\alpha$ -renaming
  - No built-in substitution
  - Names can extrude scope
    - ... prevented by additional machinery [Pottier 2007]

# Related Work

- Dependently typed programming
  - Cayenne [Augustsson 1998]
    - Arbitrary computations in types
    - Typechecking can “time out”
  - Epigram [McBride 2004]
    - Terminating computations in types

# Summary

- Foundation for programming with higher-order data and proofs
- Dependent types
- **Explicit contexts  $\Psi$  described by schemas**
- **Contextual modal types  $\Lambda[\Psi]$**
- **Parameter variables  $p$**
- Name-safe manipulation of variables
- Progress and preservation

# Ongoing Work

- Typechecker implementation
  - Coverage checking [Dunfield/Pientka, LFMTP '08]
  - Termination checking
- Reconstruction of some types and contexts
- Context subsumption and  $\exists\psi$

# The End



[complogic.cs.mcgill.ca/beluga](http://complogic.cs.mcgill.ca/beluga)

Acknowledgments:

- The anonymous reviewers

# The End



[complogic.cs.mcgill.ca/beluga](http://complogic.cs.mcgill.ca/beluga)

Acknowledgments:

- The anonymous reviewers