

# The case of the smart case

## How to implement conditional convertibility?

Thorsten Altenkirch

School of Computer Science  
University of Nottingham

Based on work with Andreas Abel, Thomas Anberre,  
Nils Anders Danielsson and Shin-Cheng Mu

- Partial core language for DTP.
- Ingredients:
  - Type : Type
  - Finite enumerations, eg **Bool** = {true, false}.
  - $\Pi$ -types
  - $\Sigma$ -types
  - Flexible mutual recursive definitions
  - Lifted types to control recursive unfolding.
  - Extended  $\alpha$  conversion for recursive definitions.
- $\Pi\Sigma$ : *Dependent Types Without the Sugar*  
T.A., Nils Anders Danielsson, Andres Löh and Nicolas Oury  
FLOPS 2010

- How to implement eliminators for datatypes?
- For the moment we consider just **Bool**

# Rule for the simply typed eliminator:

$$\frac{\Gamma \vdash t_0, t_1 : \sigma \quad \Gamma \vdash u : \mathbf{Bool}}{\Gamma \vdash \text{case } u \text{ of } \{ \text{true} \rightarrow t_0 \mid \text{false} \rightarrow t_1 \} : \sigma}$$

$\Gamma \vdash \text{case } u \text{ of } \{ \text{true} \rightarrow t_0 \mid \text{false} \rightarrow t_1 \} : \sigma$

- Pattern matching is reduced to case.
- Local case expressions.

# Dependently typed eliminator with motive

$$\begin{array}{l} \Gamma, x : \mathbf{Bool} \vdash \sigma \\ \Gamma \vdash u : \mathbf{Bool} \\ \Gamma \vdash t_0 : \sigma[x := \text{true}] \\ \Gamma \vdash t_1 : \sigma[x := \text{false}] \end{array}$$

---

$$\Gamma \vdash \text{elim}_{x.\sigma}^{\mathbf{Bool}} u \text{ of } \{ \text{true} \rightarrow t_0 \mid \text{false} \rightarrow t_1 \} : \sigma[x := u]$$

- Not syntax directed!
- We have to come up with the motive  $x.\sigma$ .
- Local case expressions?
- Can be (partially) simulated using auxiliary definitions (with).

$$\Gamma \vdash x : \mathbf{Bool}$$
$$\Gamma \vdash t_0 : \sigma[x := \text{true}]$$
$$\Gamma, \vdash t_1 : \sigma[x := \text{false}]$$

---

$$\Gamma \vdash \text{case } x \text{ of } \{ \text{true} \rightarrow t_0 \mid \text{false} \rightarrow t_1 \} : \sigma$$

- Syntax directed.
- Eliminator can be easily derived.
- No need for motives.
- Variable restriction leads to failure of subject reduction.
- Also no local case analysis.

$$\Gamma \vdash u : \mathbf{Bool}$$
$$\Gamma, u = \text{true} \vdash t_0 : \sigma$$
$$\Gamma, u = \text{false} \vdash t_1 : \sigma$$

---

$$\Gamma \vdash \text{case } u \text{ of } \{ \text{true} \rightarrow t_0 \mid \text{false} \rightarrow t_1 \} : \sigma$$

- Addresses issue with Subject Reduction
- Local case expressions ( more general than with)
- Need equational assumptions in contexts.
- Need to decide convertibility with assumptions.

# Convertibility with assumptions

We allow equational assumptions of the form  $t = b$  in the context.

We add the rule

$$\Gamma, t = b \vdash t = b$$

and weakening rules.

Here  $b$  has to be a constructor (e.g. true, false)

The remaining rules remain unchanged, e.g.

$$\text{case true of } \{ \text{true} \rightarrow t_0 \mid \text{false} \rightarrow t_1 \} = t_0$$

We do not consider (for the moment):

$$\Gamma, u = \text{true} \vdash t_0 = v$$

$$\Gamma, u = \text{false} \vdash t_1 = v$$

---

$$\Gamma \vdash \text{case } u \text{ of } \{ \text{true} \rightarrow t_0 \mid \text{false} \rightarrow t_1 \} = v$$

# Inconsistency

Equational assumptions can be inconsistent.  
E.g. the context

$$x : \mathbf{Bool}, x = \text{true}, x = \text{false}$$

is inconsistent, i.e. every equation is derivable.

$$\begin{aligned} t &= \text{case true of } \{ \text{true} \rightarrow t \mid \text{false} \rightarrow u \} \\ &= \text{case } x \text{ of } \{ \text{true} \rightarrow t \mid \text{false} \rightarrow u \} \\ &= \text{case false of } \{ \text{true} \rightarrow t \mid \text{false} \rightarrow u \} \\ &= u \end{aligned}$$

How to implement conditional  $\beta$ -equality  
(for boolean pattern equations)?

# Sketch of the implementation

We define (mutually):

Constraint sets  $\mathcal{C}$

Normalisation with constraints  $\mathcal{C} \vdash t \Downarrow v$

Convertibility with constraints  $\mathcal{C} \vdash t \sim u$

Creation of constraint sets  $\Gamma \Downarrow \mathcal{C}$

Merging of constraint sets  $\mathcal{C} \# \mathcal{D} \Downarrow \mathcal{E}$

A constraint set  $\mathcal{C}$  is either

**INCONSISTENT**

or

$$n_0 = b_0, n_1 = b_1, \dots, n_m = b_m$$

where

$$b_i \in \{\text{true}, \text{false}\}$$

$n_i$  is a neutral term

such that for all  $i$ :

$$\mathcal{C} - n_i = b_i \vdash n_i \Downarrow n_i$$

Reduction We add the rule

$$\frac{n = b \in \mathcal{C}}{\mathcal{C} \vdash n \Downarrow b}$$

Convertibility

**INCONSISTENT**  $\vdash t \sim u$

$$\frac{\mathcal{C} \vdash t \Downarrow v \quad \mathcal{C} \vdash u \Downarrow v}{\mathcal{C} \vdash t \sim u}$$

Creation of constraint sets

$$\frac{\Gamma \Downarrow \mathcal{C} \quad \mathcal{C} \vdash t \Downarrow n \quad n = b \vdash \mathcal{C} \Downarrow \mathcal{D}}{\Gamma, t = b \Downarrow \mathcal{D}}$$

# Merging Constraint sets

$$\overline{\mathcal{C} \# \epsilon \Downarrow \mathcal{C}}$$

$$\frac{\mathcal{C} \vdash n \Downarrow b \quad \mathcal{C} \# \mathcal{D} \Downarrow \mathcal{E}}{\mathcal{C} \# n = b, \mathcal{D} \Downarrow \mathcal{E}}$$

$$\frac{\mathcal{C} \vdash n \Downarrow \neg b}{\mathcal{C} \# n = b, \mathcal{D} \Downarrow \text{INCONSISTENT}}$$

$$\frac{\mathcal{C} \vdash n \Downarrow n \quad \mathcal{C}, n = b \# \mathcal{D} \Downarrow \mathcal{E}}{\mathcal{C} \# n = b, \mathcal{D} \Downarrow \mathcal{E}}$$

$$\frac{\mathcal{C} \vdash n \Downarrow n' \quad n' = b \# \mathcal{C}, \mathcal{D} \Downarrow \mathcal{E}}{\mathcal{C} \# n = b, \mathcal{D} \Downarrow \mathcal{E}}$$

# Soundness and completeness

soundness

$$\frac{\Gamma \Downarrow \mathcal{C} \quad \mathcal{C} \vdash t \sim u}{\Gamma \vdash t = u}$$

completeness

$$\frac{\Gamma \Downarrow \mathcal{C} \quad \Gamma \vdash t = u}{\mathcal{C} \vdash t \sim u}$$

relies on the key lemma:

$$\frac{n = b \# \mathcal{C} \Downarrow \mathcal{D}}{\mathcal{D} \vdash n \Downarrow b}$$

which we have been unable to prove.

**termination** Have shown termination for a simply typed variant.  
Goal: Modular termination.

- Arbitrary equations for booleans (congruence closure).  
Extensional equality for booleans.
- Extend to all first order datatypes  
All finite types and  $\Sigma$ -types.
- Conditional equality on higher order types seems undecidable.