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

ΠΣ in a nutshell

- Partial core language for DTP.
- Ingredients:
    - Type : Type
    - Finite enumerations, eg **Bool** $= \{\mathrm{true}, \mathrm{false}\}$.
    - Π-types
    - Σ-types
    - Flexible mutual recursive definitions
    - Lifted types to control recursive unfolding.
    - Extended α conversion for recursive definitions.
- ΠΣ*: 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:

$$\frac{\begin{array}{c}\Gamma \vdash t_0, t_1 : \sigma \\ \Gamma \vdash u : \mathbf{Bool}\end{array}}{\Gamma \vdash \mathrm{case}\ u\ \mathrm{of}\ \{\ \mathrm{true} \to t_0 \mid \mathrm{false} \to t_1\ \} : \sigma}$$

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

Dependently typed eliminator with motive

$$\frac{\begin{array}{l}\Gamma, x : \textbf{Bool} \vdash \sigma \\ \Gamma \vdash u : \textbf{Bool} \\ \Gamma \vdash t_0 : \sigma[x := \mathrm{true}] \\ \Gamma \vdash t_1 : \sigma[x := \mathrm{false}]\end{array}}{\Gamma \vdash \mathrm{elim}^{\textbf{Bool}}_{x.\sigma}\, u\, \mathrm{of}\, \{\, \mathrm{true} \to t_0 \mid \mathrm{false} \to t_1 \,\} : \sigma[x := u]}$$

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

Case in ΠΣ

$$\frac{\begin{array}{l}\Gamma \vdash x : \mathbf{Bool} \\ \Gamma \vdash t_0 : \sigma[x := \mathrm{true}] \\ \Gamma, \vdash t_1 : \sigma[x := \mathrm{false}]\end{array}}{\Gamma \vdash \mathrm{case}\; x \;\mathrm{of}\,\{\,\mathrm{true} \rightarrow t_0 \mid \mathrm{false} \rightarrow t_1\,\} : \sigma}$$

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

Smart Case

$$\frac{\Gamma \vdash u : \mathbf{Bool} \qquad \Gamma, u = \mathrm{true} \vdash t_0 : \sigma \qquad \Gamma, u = \mathrm{false} \vdash t_1 : \sigma}{\Gamma \vdash \mathrm{case}\; u \;\mathrm{of}\; \{\, \mathrm{true} \to t_0 \mid \mathrm{false} \to t_1 \,\} : \sigma}$$

- 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. $\mathrm{true}, \mathrm{false}$)
The remaining rules remain unchanged, e.g.

$$\mathrm{case}\,\mathrm{true}\,\mathrm{of}\,\{\,\mathrm{true} \to t_0 \mid \mathrm{false} \to t_1\,\} = t_0$$

We do not consider (for the moment):

$$\frac{\Gamma, u = \mathrm{true} \vdash t_0 = v \qquad \Gamma, u = \mathrm{false} \vdash t_1 = v}{\Gamma \vdash \mathrm{case}\,u\,\mathrm{of}\,\{\,\mathrm{true} \to t_0 \mid \mathrm{false} \to t_1\,\} = v}$$

Inconsistency

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

$$x : \textbf{Bool}, x = \mathrm{true}, x = \mathrm{false}$$

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

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

How to implement conditional β-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 $\mathcal{C} +\!\!+ \mathcal{D} \Downarrow \mathcal{E}$

Constraint sets

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

INCONSISTENT

or

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

where

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

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

$$\frac{}{\textbf{INCONSISTENT} \vdash t \sim u}$$

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

Creation of constraint sets

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

Merging Constraint sets

$$\frac{}{\mathcal{C} \mathbin{+\!\!\!+} \epsilon \Downarrow \mathcal{C}}$$

$$\frac{\mathcal{C} \vdash n \Downarrow b \qquad \mathcal{C} \mathbin{+\!\!\!+} \mathcal{D} \Downarrow \mathcal{E}}{\mathcal{C} \mathbin{+\!\!\!+} n = b, \mathcal{D} \Downarrow \mathcal{E}}$$

$$\frac{\mathcal{C} \vdash n \Downarrow \neg b}{\mathcal{C} \mathbin{+\!\!\!+} n = b, \mathcal{D} \Downarrow \textbf{INCONSISTENT}}$$

$$\frac{\mathcal{C} \vdash n \Downarrow n \qquad \mathcal{C}, n = b \mathbin{+\!\!\!+} \mathcal{D} \Downarrow \mathcal{E}}{\mathcal{C} \mathbin{+\!\!\!+} n = b, \mathcal{D} \Downarrow \mathcal{E}}$$

$$\frac{\mathcal{C} \vdash n \Downarrow n' \qquad n' = b \mathbin{+\!\!\!+} \mathcal{C}, \mathcal{D} \Downarrow \mathcal{E}}{\mathcal{C} \mathbin{+\!\!\!+} n = b, \mathcal{D} \Downarrow \mathcal{E}}$$

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.

Extensions

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