

On Irrelevance and Extraction in Type Theory

Andreas Abel

INRIA, Team πr^2
PPS Lab, Paris

11th Agda Meeting
Awaji, Kansai, Japan
24 March 2010

Irrelevance and Extraction in Type Theory

Definition (Irrelevance)

A type T is *irrelevant* if $\Gamma \vdash t, t' : T$ implies $\Gamma \vdash t = t' : T$.

Three motivations to consider irrelevance:

- 1 More powerful type checkers.
 - More terms type check.
 - Less proof burden for the user.
- 2 More efficient type checkers.
 - Fewer terms to compare for equality.
 - Erasure of irrelevant parts in internal representation?
- 3 More dead-code elimination in program extraction.
 - Eliminate redundant information from data structures.
 - Eliminate redundant arguments from functions.

Three Forms of Irrelevance

- 1 Irrelevance through eta-expansion.

$$\uparrow^{\text{unit}} t \longrightarrow tt$$

- 2 Irrelevance through bracket types (Awodey/Bauer 2001, Pfenning 2001).

$$\{x : A \mid P x\} = \Sigma_{x:A}. [P x]$$

- 3 Irrelevance through parametric polymorphism (Miquel 2001, Barras/Bernardo 2008).

$$\text{Vcons} : \forall (a : A)[n : \text{nat}], \text{vector } n \rightarrow \text{vector } (S n)$$

1. Eta Expansion

η – expansion

yes	more power
no?	faster
?	better extraction

- 1 Requires types in evaluation.
- 2 Extraction would rather have (careful) η -reduction, but η -expansion might reveal opportunities for erasure.

Eta-Expansion for Function Types

- Typed eta-equality for *specification* of type theory.

$$\frac{\Gamma \vdash t : \Pi x : U. T}{\Gamma \vdash t = \lambda x. (t x) : \Pi x : U. T}$$

- Add new terms $\uparrow^T t$ and $\downarrow^T t$ for *implementation*.
- New (weak head) reductions:

$$\begin{aligned} (\uparrow^{\Pi x : U. T} t) u &\longrightarrow \uparrow^{T[u]}(t \downarrow^U u) \\ \downarrow^{\Pi x : U. T} t &\longrightarrow \lambda x. \downarrow^{T[\uparrow^U x]}(t \uparrow^U x) \end{aligned}$$

- Checking equality in type inference:

$$\frac{y : V \vdash t : \Pi x : U. T \quad y : V \vdash u : U' \quad U[\uparrow^V x] \searrow \swarrow U'[\uparrow^V x]}{y : V \vdash t u : T[u]}$$

Eta-Expansion for Singletons

- The `unit` type is irrelevant: $x:\text{unit} \vdash x = \text{tt} : \text{unit}$.
- But $\text{unit_rect } P f \text{ tt} \longrightarrow f$ while $\text{unit_rect } P f x \not\longrightarrow$.
- Solution: η -expand!

$$\begin{array}{l} \uparrow^{\text{unit}} t \longrightarrow \text{tt} \\ \downarrow^{\text{unit}} t \longrightarrow \text{tt} \end{array}$$

- Now $\text{unit_rect } P f (\uparrow^{\text{unit}} x) \longrightarrow f$.

Eta-Expansion for Records

- Surjective pairing:

$$\begin{aligned} &\text{Inductive Prod}(A\ B : \text{Type}) : \text{Type} := \\ &\quad \text{pair} : \forall(\text{fst} : A)(\text{snd} : B) : \text{Prod}\ A\ B. \\ &\uparrow^{\text{Prod}\ A\ B}\ t \longrightarrow \text{pair}\ (\uparrow^A(\text{fst}\ t))\ (\uparrow^B(\text{snd}\ t)) \end{aligned}$$

- Record = non-recursive inductive types with one constructor:

$$\begin{aligned} &\text{Inductive } I(\vec{X} : \vec{U}) : s := c : \forall(\vec{d} : \vec{T}),\ I\ \vec{X}. \\ &\uparrow^I\ \vec{X}\ t \longrightarrow c\ (\uparrow^{T_1}(d_1\ t))\ \dots\ (\uparrow^{T_n}(d_n\ t)) \end{aligned}$$

- So far: proof irrelevance for $\forall, \rightarrow, \wedge, \top$ -fragment (but no atoms!).

Eta-Expansion for Pattern Inductive Families

- Generalize to non-recursive inductive *families* with at most one constructor per instance.

```
Inductive Bla : bool → Set :=
| foo : Bla true
| bar : Bla false.
```

$$\uparrow^{\text{Bla } b} t \longrightarrow \begin{cases} \text{foo} & \text{if } b \text{ matches true} \\ \text{bar} & \text{if } b \text{ matches false} \end{cases}$$

- η for recursive types like `vector` would need termination check!

$$\uparrow^{\text{vector } A (S \ n)} t \longrightarrow \text{Vcons } (\uparrow^A(\text{Vhead } t)) \ n \\ (\uparrow^{\text{vector } A \ n}(\text{Vtail } t))$$

Eta-Expansion for Empty Types

- Want $\Gamma \vdash t = t' : \text{Empty_set}$.
- `Empty_set` is made irrelevant via

$$\uparrow^{\text{Empty_set}} t \longrightarrow \epsilon$$

where ϵ is an internal dummy constant (cf. Werner 2008).

Eta-Expansion for Identity Type

- *Definitional* uniqueness of identity proofs.
- `eq` is a non-linear pattern inductive family.

$$\text{refl_equal} : \forall (A : \text{Type})(a : A), \text{eq } A \ a \ a$$

$$\begin{array}{lll} \uparrow^{\text{eq } A \ a \ b} t \longrightarrow & \text{refl_equal } A \ a & \text{if } a \searrow \swarrow b \\ \downarrow^{\text{eq } A \ a \ b} t \longrightarrow & \text{refl_equal } A \ a & \text{if } a \searrow \swarrow b \\ \downarrow^{\text{eq } A \ a \ b} t \longrightarrow_w & \epsilon & \text{if not } a \searrow \swarrow b \end{array}$$

- Only weak head reduction in last line, evaluation order matters!
- We get irrelevance $\Gamma \vdash t = t' : \text{eq } A \ a \ b$.
- “Axiom” K is definable (as identity).
- `eq_rect` still blocks on false equations `eq bool true false`.

Eta: Summary

- Irrelevance for $\forall, \rightarrow, \wedge, \top, \perp$, eq-fragment. (But no atoms!)
- Such formulas have only one canonical proof, found by eta expansion.
- Theory of η for non-linear pattern inductive family (eq) not settled (Abel, NBE 09).
- Papers on eta with $\uparrow \downarrow$ -markers: Abel, Coquand, Dybjer LICS 07; Abel, FLOPS 10.

2. Bracket Types

bracket types à la Awodey/Bauer 2001

yes	more power
yes	faster
–	better extraction

- Alternative to `Prop`: less duplication?
- Saves some equality tests. Irrelevant arguments cannot be discarded entirely since they need to block reduction?!
- Extraction similar to existing one.

Rules for Bracket Types

- The bracket type $[T]$ is the irrelevant version of T .

$$\frac{\Gamma \vdash t : T}{\Gamma \vdash t : [T]} \quad \frac{\Gamma \vdash t, t' : T}{\Gamma \vdash t = t' : [T]}$$

- Getting out of the bracket.

$$\frac{\Gamma \vdash T : s \quad \Gamma \vdash u : [U] \quad \Gamma, x:U, y:U \vdash t[x] = t[y] : T}{\Gamma \vdash \text{let } x = u \text{ in } t[x] : T}$$

- But: T cannot depend on x .
- Applications: subset type $\{x:U \mid P\} = \Sigma x:U. [P]$.

3. Parametric Polymorphism

polymorphism à la Miquel 2001

yes	more power
yes	faster
yes	better extraction

- Irrelevance not only for proofs, but arbitrary values as declared by the user.
- Saves some equality tests, but irrelevant arguments cannot be discarded entirely in the presence of η .
- User-controlled extraction.

Parametric Polymorphism in Type Theory

- Type parameters are computationally irrelevant.

$$\text{cons } A_1 a l = \text{cons } A_2 a l : \text{List } A \quad \text{for all } A_1, A_2 : \text{Type}$$

- Value parameters can also be irrelevant.

$$\text{cons } A_1 n_1 a v = \text{cons } A_2 n_2 a v : \text{Vec } A n$$

- Following Miquel (2001), distinguish Π (function type) and \forall (polymorphic type).

$$\begin{aligned} \text{Vec} & : \Pi A : \text{Type}. \Pi n : \text{Nat}. \text{Type} \\ \text{cons} & : \forall A : \text{Type}. \forall n : \text{Nat}. \Pi a : A. \Pi v : \text{Vec } A n. \text{Vec } A (n + 1) \end{aligned}$$

Example 2: Finite Enumerations

- 1 In $\text{cons } A n a v$, type A is determined by a and number n by v .
- 2 Irrelevant arguments need not always be determined by other arguments:

$$\begin{aligned} \text{Fin} & : \Pi n:\text{Nat}. \text{Type} \\ \text{fzero} & : \forall n:\text{Nat}. \text{Fin } (n + 1) \\ \text{fsuc} & : \forall n:\text{Nat}. \text{Fin } n \rightarrow \text{Fin } (n + 1) \end{aligned}$$

- 3 Now $\text{fzero } n_1 = \text{fzero } n_2 : \text{Fin } n$.
- 4 How to design a type system for polymorphism?

$$\frac{m : \text{Nat} \vdash \text{fzero} : \forall n:\text{Nat}. \text{Fin } (n + 1)}{m : \text{Nat} \vdash \text{fzero } m : \text{Fin } (m + 1)}$$

- 5 m is irrelevant in the term $\text{fzero } m$, but relevant in the type $\text{Fin } (m + 1)$.

Logical framework with polymorphism

- Syntax:

t, u, T, U	$::=$	$x \mid \lambda x t \mid t u$	terms
		$\mid \Pi x : U. T \mid \forall x : U. T$	types
		$\mid *$	sort of types
Γ	$::=$	$() \mid$	empty typing context
		$\mid \Gamma, x : T$	term variable decl.
		$\mid \Gamma, x \div U$	“type” variable decl.

- $\lambda x t$ can denote value or “type” (polymorphic) abstraction.
- Resurrection Γ^\oplus replaces all \div s by $:$ s.

Typing Rules

$$\frac{(x:T) \in \Gamma}{\Gamma \vdash x : T} \quad \text{no rule for } (x \div T) \in \Gamma$$

$$\frac{\Gamma, x:U \vdash t : T}{\Gamma \vdash \lambda x t : U \rightarrow T} \quad \frac{\Gamma \vdash t : U \rightarrow T \quad \Gamma \vdash u : U}{\Gamma \vdash t u : T}$$

$$\frac{\Gamma, x \div U \vdash t : T}{\Gamma \vdash \lambda x t : \forall x:U. T} \quad \frac{\Gamma \vdash t : \forall x:U. T \quad \Gamma^\oplus \vdash u : U}{\Gamma \vdash t u : T[u/x]}$$

$$\frac{\Gamma \vdash U : * \quad \Gamma, x:U \vdash T : *}{\Gamma \vdash \Pi x:U. T : *} \quad \frac{\Gamma, x:U \vdash T : *}{\Gamma \vdash \forall x:U. T : *}$$

Type-directed equality

- Invariant: If $\Gamma \vdash t : T$ then $\Gamma^\oplus \vdash T : s$.
- Typed equality as implemented in MiniAgda:

$$\frac{\Gamma \vdash t_1 : \forall x : U. T = t_2}{\Gamma \vdash t_1 u_1 : T[u_1/x] = t_2 u_2}$$

- Implementation with NbE:

$$\begin{array}{l} (\uparrow^{\forall x : U. T} t) u \longrightarrow \uparrow^{T[u]}(t \epsilon) \\ \downarrow^{\forall x : U. T} t \longrightarrow \lambda x. \downarrow^{T[\uparrow^U x]}(t \uparrow^U x) \end{array}$$

Type-directed extraction

$$\frac{(x:T) \in \Gamma}{\Gamma \vdash x : T \rightsquigarrow x} \quad \text{no rule for } (x \div T) \in \Gamma$$

$$\frac{\Gamma, x:U \vdash t : T \rightsquigarrow M}{\Gamma \vdash \lambda x t : U \rightarrow T \rightsquigarrow \lambda x M} \quad \frac{\Gamma \vdash t : U \rightarrow T \rightsquigarrow M \quad \Gamma \vdash u : U \rightsquigarrow N}{\Gamma \vdash t u : T \rightsquigarrow M N}$$

$$\frac{\Gamma, x \div U \vdash t : T \rightsquigarrow M}{\Gamma \vdash \lambda x t : \forall x : U. T \rightsquigarrow M} \quad \frac{\Gamma \vdash t : \forall x : U. T \rightsquigarrow M}{\Gamma \vdash t u : T[u/x] \rightsquigarrow M}$$

Conclusions

- MiniAgda has \forall .
- Next: extraction!
- Clarify semantics of polymorphism (joint work with Bruno Barras).
- Related work: Edwin Brady/Conor McBride.