On Irrelevance and Extraction in Type Theory

Andreas Abel

INRIA, Team πr^2
PPS Lab, Paris

Coq Meeting at JFLA 2010
La Ciotat, Marseille, France
2 February 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.

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

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).

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

1. Eta Expansion

η − *expansion*	
yes	more power
no?	faster
?	better extraction

1. Simpler unification algorithm. Fixes annoyances like $\mathtt{sig}\ P \neq \{x : A \mid P\ x\}$ (since $P \neq \mathtt{fun}\ x : A \Rightarrow P\ x$).
2. Requires types in evaluation.
3. 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:

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

- Checking equality in type inference:

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

Eta-Expansion for Singletons

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

$$\begin{array}{lcl} \uparrow^{\texttt{unit}} t & \longrightarrow & \texttt{tt} \\ \downarrow^{\texttt{unit}} t & \longrightarrow & \texttt{tt} \end{array}$$

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

Eta-Expansion for Records

- Surjective pairing:

$$\begin{array}{l}\texttt{Inductive Prod}(A\ B : \texttt{Type}) : \texttt{Type} := \\ \quad \texttt{pair} : \forall(\texttt{fst} : A)(\texttt{snd} : B) : \texttt{Prod}\ A\ B.\end{array}$$

$$\uparrow^{\texttt{Prod}\ A\ B} t \longrightarrow \texttt{pair}\ (\uparrow^{A}(\texttt{fst}\ t))\ (\uparrow^{B}(\texttt{snd}\ t))$$

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

$$\texttt{Inductive}\ I(\vec{X} : \vec{U}) : s := c : \forall(\vec{d} : \vec{T}),\ I\ \vec{X}.$$

$$\uparrow^{I\ \vec{X}} t \longrightarrow c\ (\uparrow^{T_1}(d_1\, t)) \dots (\uparrow^{T_n}(d_n\, t))$$

- 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.

$$\begin{array}{l}\texttt{Inductive Bla} : \texttt{bool} \to \texttt{Set} := \\ \mid \texttt{foo} : \texttt{Bla true} \\ \mid \texttt{bar} : \texttt{Bla false}.\end{array}$$

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

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

$$\begin{array}{ll}\uparrow^{\texttt{vector}\ A\ (\texttt{S}\ n)}\, t \longrightarrow & \texttt{Vcons}\ (\uparrow^{A}(\texttt{Vhead}\ t))\ n \\ & (\uparrow^{\texttt{vector}\ A\ n}(\texttt{Vtail}\ t))\end{array}$$

Eta-Expansion for Empty Types

- Want $\Gamma \vdash t = t' : \texttt{Empty\_set}$.
- $\texttt{Empty\_set}$ is made irrelevant via

$$\uparrow^{\texttt{Empty\_set}} t \longrightarrow \epsilon$$

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.

$$\texttt{refl\_equal} : \forall (A : \texttt{Type})(a : A),\ \texttt{eq}\ A\ a\ a$$

$$\begin{array}{llll} \uparrow^{\texttt{eq}\ A\ a\ b}\ t & \longrightarrow & \texttt{refl\_equal}\ A\ a & \text{if } a \searrow\swarrow b \\ \downarrow^{\texttt{eq}\ A\ a\ b}\ t & \longrightarrow & \texttt{refl\_equal}\ A\ a & \text{if } a \searrow\swarrow b \\ \downarrow^{\texttt{eq}\ A\ a\ b}\ t & \longrightarrow_{\mathsf{w}} & \epsilon & \text{if not } a \searrow\swarrow b \end{array}$$

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

Eta: Summary

- Irrelevance for $\forall, \rightarrow, \wedge, \top, \bot, \mathtt{eq}$-fragment. (But no atoms!)
- Such formulas have only one canonical proof, found by eta expansion.
- Theory of η for non-linear pattern inductive family ($\mathtt{eq}$) not settled (Abel, NBE 09).
- Papers on eta with $\uparrow\downarrow$-markers: Abel, Coquand, Dybjer LICS 07; Abel, FLOPS 10.
- At least η for functions needs to be implemented.
- Addition of markers hopefully not such a big intrusion into Coq kernel!?

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.

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

- Getting out of the bracket.

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

- But: T cannot depend on x.
- Applications: subset type $\{x\!:\!U \mid P\} = \Sigma x\!:\!U.\,[P]$.
- Wellfounded recursion $f(x : A)(p : [\mathtt{Acc}\ x]) : B$.

Summary on Bracket Types

- Alternative to rigid `Set/Prop` distinction.
- What about impredicativity?
- Is this rule inconsistent?

$$\frac{\Gamma \vdash T : \mathtt{Type}}{\Gamma \vdash [T] : \mathtt{Prop}}$$

- More research needed!

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 irrlevant.

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

- Value parameters can also be irrelevant.

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

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

$$\begin{array}{lcl} \mathsf{Vec} & : & \Pi A{:}\mathsf{Type}.\,\Pi n{:}\mathsf{Nat}.\ \mathsf{Type} \\ \mathsf{cons} & : & \forall A{:}\mathsf{Type}.\forall n{:}\mathsf{Nat}.\Pi a{:}A.\Pi v{:}\mathsf{Vec}\,A\,n.\ \mathsf{Vec}\,A\,(n+1) \end{array}$$

Example 2: Finite Enumerations

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

$$\begin{array}{lcl} \mathsf{Fin} & : & \Pi n\!:\!\mathsf{Nat}.\ \mathsf{Type} \\ \mathsf{fzero} & : & \forall n\!:\!\mathsf{Nat}.\,\mathsf{Fin}\,(n+1) \\ \mathsf{fsuc} & : & \forall n\!:\!\mathsf{Nat}.\,\mathsf{Fin}\,n \to \mathsf{Fin}\,(n+1) \end{array}$$

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

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

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

(Ir)relevance in System F

- Simply-typed λ-calculus + type abstraction/application.

$$
\begin{array}{lcll}
t, u, T, U & ::= & x \mid \lambda x t \mid t\, u & \text{terms} \\
 & \mid & T \to U \mid \forall x{:}{*}.\, T & \text{types} \\
 & \mid & * & \text{sort of types} \\
\Gamma & ::= & () \mid & \text{empty typing context} \\
 & \mid & \Gamma, x : T & \text{term variable decl.} \\
 & \mid & \Gamma, x \div * & \text{type variable decl.}
\end{array}
$$

- $\lambda x t$ can denote term or type abstraction.
- Resurrection $\Gamma^{\oplus}$ replaces all $\div$s by $:$s.

System F Typing Rules

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

$$\frac{\Gamma, x\!:\!U \vdash t : T}{\Gamma \vdash \lambda x t : U \to T} \qquad \frac{\Gamma \vdash t : U \to T \qquad \Gamma \vdash u : U}{\Gamma \vdash t\, u : T}$$

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

$$\frac{\Gamma \vdash U : * \qquad \Gamma \vdash T : *}{\Gamma \vdash U \to T : *} \qquad \frac{\Gamma, x\!:\!* \vdash T : *}{\Gamma \vdash \forall x\!:\!*.\, T : *}$$

Type Theory with Parametric Polymorphism

- Invariant: If $\Gamma \vdash t : T$ then $\Gamma^{\oplus} \vdash T : s$.

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

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

- Last rule: u_1 and u_2 are arbitrary!
- Implementation:

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

Application: Sized Types

- In $\widehat{\texttt{Coq}}$, an internal representation of `nat` could look like:

$$\begin{array}{l}\texttt{Sized Inductive nat} : \texttt{Size} \to \texttt{Set} := \\ \quad \mid \texttt{O} : \forall [i : \texttt{Size}] \to \texttt{nat}\ (\$ i) \\ \quad \mid \texttt{S} : \forall [i : \texttt{Size}] \to \texttt{nat}\ i \to \texttt{nat}\ (\$ i)\end{array}$$

- Thus $i \div \texttt{Size} \vdash \texttt{O}\ i = \texttt{O}\ \infty : \texttt{nat}\ \infty$.
- Objects never depend on `Size`.

PTS with $\mathtt{Size}$

- Axioms: $(\mathtt{Prop}, \mathtt{Set}), (\mathtt{Set}, \mathtt{Type}), (\mathtt{Size}, \mathtt{Type})$.
- New rules: $(\mathtt{Size}, s, s)$.
- No rules $(s, \mathtt{Size}, s')$.

Conclusions

- MiniAgda has η, $\forall$ and sized types.
- Add η to Coq!
- Add type-based termination with invisible $\forall i\,{:}\,\mathtt{Size}$ to Coq!
- Clarify semantics of polymorphism and bracket types (joint work with Bruno Barras).