Definitional Proof Irrelevance

Daria Walukiewicz-Chrząszcz

25 May 2021

Definitional Proof-Irrelevance without K

paper from 2019 by G.Gilbert, J.Cockx, M. Sozeau, N. Tabareau about:

- how to add a proof irrelevant sort to a dependent type theory without adding axioms,
- and preserve consistency and decidability of type checking
- criterion deciding which proof-irrelevant inductive defintions can be eliminated into types
- implementation in Coq and Agda

Recall:

- definitional equality - conversion
- propositional equality - can be proved (Lemma)
- propositional equality in conversion leads to undecidable type-checking

Definitional Proof-Irrelevance without K

paper from 2019 by G.Gilbert, J.Cockx, M. Sozeau, N. Tabareau about:

- how to add a proof irrelevant sort to a dependent type theory without adding axioms,
- and preserve consistency and decidability of type checking
- criterion deciding which proof-irrelevant inductive defintions can be eliminated into types
- implementation in Coq and Agda

Recall:

- definitional equality - conversion
- propositional equality - can be proved (Lemma)
- propositional equality in conversion leads to undecidable type-checking

Definitional Proof-Irrelevance without K

paper from 2019 by G.Gilbert, J.Cockx, M. Sozeau, N. Tabareau about:

- how to add a proof irrelevant sort to a dependent type theory without adding axioms,
- and preserve consistency and decidability of type checking
- criterion deciding which proof-irrelevant inductive defintions can be eliminated into types
- implementation in Coq and Agda

Recall:

- definitional equality - conversion
- propositional equality - can be proved (Lemma)
- propositional equality in conversion leads to undecidable type-checking

Definitional Proof-Irrelevance without K

paper from 2019 by G.Gilbert, J.Cockx, M. Sozeau, N. Tabareau about:

- how to add a proof irrelevant sort to a dependent type theory without adding axioms,
- and preserve consistency and decidability of type checking
- criterion deciding which proof-irrelevant inductive defintions can be eliminated into types
- implementation in Coq and Agda

Recall:

- definitional equality - conversion
- propositional equality - can be proved (Lemma)
- propositional equality in conversion leads to undecidable type-checking

Sort Sprop

- $SProp, Prop, Set : Type(1)$
- $SProp$ is not related by cumulativity to any other term
- $SProp$ is impredicative

$$\frac{\Gamma \vdash A : s \qquad s \texttt{ is a sort} \qquad \Gamma, x{:}A \vdash B : SProp}{\Gamma \vdash \forall x{:}A.B : SProp}$$ **Prod-SProp**

Conversion — definitional equality

$$\frac{\Gamma \vdash M : A \qquad \Gamma \vdash A =_{\beta\eta\delta\zeta\iota} B \qquad \Gamma \vdash B : s}{\Gamma \vdash M : B}$$

where $\Gamma \vdash A =_{\beta\eta\delta\zeta\iota} B$ if

- $\Gamma \vdash A \triangleright^* A'$, $\Gamma \vdash B \triangleright^* B'$ (transitive closure of beta, iota, delta and zeta reductions)
- A' and B' identical up to irrelevant subterms (of type $T : \texttt{SProp}$) , or
  $(A' = \lambda x : T.A''$ and $\Gamma, x : T \vdash B'x =_{\beta\eta\delta\zeta\iota} A'')$ or
  $(B' = \lambda x : T.B''$ and $\Gamma, x : T \vdash A'x =_{\beta\eta\delta\zeta\iota} B'')$

Conversion — definitional equality

$$\frac{\Gamma \vdash M : A \qquad \Gamma \vdash A =_{\beta\eta\delta\zeta\iota} B \qquad \Gamma \vdash B : s}{\Gamma \vdash M : B}$$

where $\Gamma \vdash A =_{\beta\eta\delta\zeta\iota} B$ if

- $\Gamma \vdash A \triangleright^* A'$, $\Gamma \vdash B \triangleright^* B'$ (transitive closure of beta, iota, delta and zeta reductions)
- A' and B' identical up to irrelevant subterms (of type $T : \texttt{SProp}$) , or
  $(A' = \lambda x : T.A''$ and $\Gamma, x : T \vdash B'x =_{\beta\eta\delta\zeta\iota} A'')$ or
  $(B' = \lambda x : T.B''$ and $\Gamma, x : T \vdash A'x =_{\beta\eta\delta\zeta\iota} B'')$

Irrelevance - example

```
Theorem irrelevance (A : SProp) (P : A -> Prop) :
        forall x : A, P x -> forall y : A, P y.
Proof.
intros x H.
exact H.
Qed.
```

See more: bounded.v boundedSProp.v

Library Coq.Logic.StrictProp

https://coq.inria.fr/stdlib/Coq.Logic.StrictProp.html

```
Record Box (A:SProp) : Prop := box  unbox : A .

Inductive Squash (A:Type) : SProp := squash : A -> Squash A.

Inductive sEmpty : SProp :=.

Inductive sUnit : SProp := stt.

Record Ssig A:Type (P:A->SProp) :=
       Sexists { Spr1 : A; Spr2 : P Spr1 }.

Lemma Spr1_inj A P a b : @Ssig A P (e : Spr1 a = Spr1 b) :
       a = b.
```

Library Coq.Logic.StrictProp

- `sEmpty` has elimination into any `Type(i)`

```
sEmpty_rect :
        forall (P : sEmpty -> Type) (s : sEmpty), P s
```

- primitive records in `SProp` are allowed when fields are strict propositions

```
Record SSsig A:SProp (P:A->SProp): SProp :=
SSexists { Spr1 : A; Spr2 : P Spr1 }.
```

- inductive types in `SProp` with elimination into `SProp`.

How to eliminate SProp into Type ?

- sEmpty_rect is given
- for some inductive types in SProp it can be done using an encoding with Fixpoint

An inductive type in SProp can be eliminated into Type if:

- every non-forced argument must be in SProp
- the return types of constructors must be pairwise orthogonal
- every recursive call must satisfy a syntactic guard condition

How to eliminate SProp into Type ?

- `sEmpty_rect` is given
- for some inductive types in `SProp` it can be done using an encoding with Fixpoint

An inductive type in `SProp` can be eliminated into `Type` if:

- every non-forced argument must be in `SProp`
- the return types of constructors must be pairwise orthogonal
- every recursive call must satisfy a syntactic guard condition

How to eliminate SProp into Type ?

- `sEmpty_rect` is given
- for some inductive types in `SProp` it can be done using an encoding with Fixpoint

An inductive type in `SProp` can be eliminated into `Type` if:

- every non-forced argument must be in `SProp`
- the return types of constructors must be pairwise orthogonal
- every recursive call must satisfy a syntactic guard condition

How to eliminate SProp into Type ?

- `sEmpty_rect` is given
- for some inductive types in `SProp` it can be done using an encoding with Fixpoint

An inductive type in `SProp` can be eliminated into `Type` if:

- every non-forced argument must be in `SProp`
- the return types of constructors must be pairwise orthogonal
- every recursive call must satisfy a syntactic guard condition

Examples

Files:

- refman.v
- LeMinus.v: definitions of `le` and `minus` in `Prop`
- LeMinusSProp.v: definitions of `le` and `minus` in `SProp`
- LeMinusDivideSProp.v: definitions of `Divide` and `prime` in `SProp`

based on Prime.v from:
https://github.com/CoqHott/sProp/blob/master/