Paradoxes and Definitions

Thierry Coquand

Thanks to Thorsten for so many interesting discussions!

Some discussions with Thorsten

Type theory as total fragment of functional programming, 1995

Talk in Göteborg about extensional equality, November 1997

Discussion about fragment of system F, 2000

Discussion in a plane back from Nara about normalisation, 2005

Question about types not being of hlevel n, 2013

Discussions about parametricity, 2013

Inductive types in System F

$T : \mathsf{type} \to \mathsf{type} \qquad mon : \Pi_{X\ Y:\mathsf{type}}(X \to Y) \to T\ X \to T\ Y$

If $f : X \to Y$ we write $T\ f : T\ X \to T\ Y$ for $mon\ f$

Weak initial T-algebra $A = \Pi_{X:\mathsf{type}}(T\ X \to X) \to X$

If $f : T\ X \to X$ we have $\iota(f) : A \to X$

$\iota(f)\ a = a\ X\ f$

Inductive types in System F

We can define $\mathsf{intro} : T\ A \to A$

$\mathsf{intro} = \lambda_{u:T\ A}\lambda_{X:\mathsf{type}}\lambda_{f:T\ X\to X} f\ (T\ \iota(f)\ u)$

We have a (strict) map of T-algebras

$$\begin{tikzcd} T\ A \arrow[r, "T\ \iota(f)"] \arrow[d, "\mathsf{intro}"] & T\ X \arrow[d, "f"] \\ A \arrow[r, "\iota(f)"] & X \end{tikzcd}$$

Inductive types in System F

In particular we get $\mathsf{match} = \iota(T\ \mathsf{intro}) : A \to T\ A$ and commuting diagram

$$\begin{tikzcd}
T\ A \arrow[r, "T\ \mathsf{match}"] \arrow[d, "\mathsf{intro}"] & T\ (T\ A) \arrow[r, "T\ \mathsf{intro}"] \arrow[d, "T\ \mathsf{intro}"] & T\ A \arrow[d, "\mathsf{intro}"] \\
A \arrow[r, "\mathsf{match}"] & T\ A \arrow[r, "\mathsf{intro}"] & A
\end{tikzcd}$$

In general $\delta = \mathsf{intro} \circ \mathsf{match} : A \to A$ is not strictly the identity function

We need $T\ \delta = \mathsf{match} \circ \mathsf{intro}$ to be the identity, i.e. $T\ A$ as a retract of A

Inductive types in System F

Problem with predecessor in system F

One motivation for introducing inductive definitions as primitive notions

In this case, we have match (intro z) = z as computation rule

Reynolds 1984

Reynolds 1984 considers the particular case $T\ X = P^2\ X$ with $P\ X = \Omega^X$

One can use $\Omega = \mathsf{type}$ to get a new paradox with $\mathsf{type} : \mathsf{type}$ (Th. C. 1989)

If we work with PERs, we get a type A_0 isomorphic to $T\ A_0 = P^2\ A_0$

Since $P\ A_0$ is a retract of $P^2\ A_0$ it is then a retract of A_0

We can then apply Russell's paradox

Since $(A_0 \rightarrow B) \rightarrow B$ *and* A_0 *are well-known to have different cardinalities, we have a contradiction*

Variation of Reynolds/Hurkens

Hurkens used $\Pi_{X:\mathsf{type}}(T\ X \to X) \to T\ X$ instead of $\Pi_{X:\mathsf{type}}(T\ X \to X) \to X$

But his argument works as well with Reynolds $A = \Pi_{X:\mathsf{type}}(T\ X \to X) \to X$

It only uses that we have a strict weak initial T-algebra

It can be seen as a direct proof that we cannot have P^2A retract of A

Don't need to refer to Russell's paradox

Variation of Reynolds/Hurkens

Assume first that $\mathsf{match} : A \to T\ A$ is a *strict* retract map via $\mathsf{intro} : T\ A \to A$

Consider $p_0 : P\ $ and $x_0 : A = \mathsf{intro}\ \alpha_0$ and $C\ x\ p = \neg(p\ x \wedge \mathsf{match}\ x\ p)$

$$p_0\ x \ = \ \forall_{p:P A}\ C\ x\ p$$

$$\alpha_0\ p \ = \ \forall_{x:A}\ C\ x\ p \ = \ \mathsf{match}\ x_0\ p$$

We have $\forall_{x:A}\ C\ x\ p_0$ that is $\mathsf{match}\ x_0\ p_0$

But also $\forall_{p:P\ A}\ C\ x_0\ p$ that is $p_0\ x_0$

We get $p_0\ x_0 \wedge \mathsf{match}\ x_0\ p_0$ hence a contradiction

Variation of Reynolds/Hurkens

```
p0 : A -> Set
p0 x = (p : A -> Set) -> p x -> not (match x p)

X0 : T A
X0 p = (x : A) -> p x -> not (match x p)

x0 : A
x0 = intro X0

lem1 : X0 p0
lem1 x h = h p0 h

lem2 : p0 x0
lem2 p h h1 = h1 x0 h h1

loop : abs
loop = lem2 p0 lem2 lem1
```

Variation of Reynolds/Hurkens

The same argument works in general with $\mathsf{intro} \circ \mathsf{match} = \delta$ using instead

$$p_0\ x \ = \ \forall_{p:P\,A} \neg (p\ (\delta\ x) \wedge \mathsf{match}\ x\ p)$$

$$\alpha_0\ p \ = \ \forall_{x:A} \neg (p\ x \wedge \mathsf{match}\ x\ p)$$

We use stability of p_0 and α_0

$$p_0\ x \rightarrow p_0\ (\delta\ x)$$

$$\alpha_0\ p \rightarrow \alpha_0\ (p \circ \delta)$$

Variation of Reynolds/Hurkens

This does not look like Burali-Forti??

Because of δ not being the identity the proof of $\perp$ does not reduce to itself

Definitional equality

In order to reason about this paradox, one needs to use "abbreviations"

This is stressed both by Hurkens 1995 and Barendregt 1990

E.g. $A : \mathsf{type} = \Pi_{X:\mathsf{type}}(TX \to X) \to X$

$p_0 : A \to \mathsf{type} = \lambda_{x:A}\forall_{p:A\to\mathsf{type}}\neg(p\ x \wedge \mathsf{match}\ p\ x)$

This is *definitional equality*

Definitional equality

Definitional equality cannot be reduced to abstraction and application

$(\lambda_{P:\mathsf{type}\to\mathsf{type}} \ldots P \ldots P \ldots)\ (\lambda_{X:\mathsf{type}} X \to \mathsf{type})$

Geuvers and Nederpelt system λD *Type Theory and Formal Proof*

de Bruijn system $\lambda\Delta$

Importance of head *linear* reduction

This is exactly what is needed to analyse the behavior of paradoxes but more generally of any proof

Head Linear Reduction

```
p0 : Pow A = [z : A][p : Pow A]p (delta z) -> not (match z p)

X0 : T A = [p : Pow A][z : A] p z -> not (match z p)   x0 : A = intro X0

stablep0 : [z : A]p0 z -> p0 (delta z) = [z : A][hz : p0 z][p : Pow A]hz (cDelta p)

stableX0 : [p : Pow A]X0 p -> X0 (cDelta p) = [p : Pow A][hp : X0 p][z : A]hp (delta z)

lem1 : [p : Pow A]p x0 -> not (X0 p) = [p : Pow A][hp : p x0][h0 : X0 p]h0 x0 hp (stableX0 p h0)

lem2 : [z : A]p0 z -> not (match z p0) = [z : A][hz : p0 z]hz p0 (stablep0 z hz)

lem3 : [p : Pow A]p (delta x0) -> not (match x0 p) =  [p : Pow A]lem1 (cDelta p)

loop : abs = lem1 p0 lem3 lem2
```

Head Linear Reduction

```
loop

lem1 p0 lem3 lem2

lem2 x0 lem3 (stableX0 p0 lem2)

lem3 p0 (stablep0 x0 lem3) (stableX0 p0 lem2)

lem1 (cDelta p0) (stablep0 x0 lem3) (stableX0 p0 lem2)

stableX0 p0 lem2 x0 (stablep0 x0 lem3) (stableX0 (cDelta p0) (stableX0 p0 lem2))

lem2 (delta x0) (stablep0 x0 lem3) (stableX0 (cDelta p0) (stableX0 p0 lem2))
```

Head Linear Reduction

We do need head *linear* reduction

```
lem2 x0 lem3 (stableX0 p0 lem2)

lem3 p0 (stablep0 x0 lem3) (stableX0 p0 lem2)
```

Head Linear Reduction

Analysis of a general argument

We want to be able to analyse a given instantiation of this argument

We can simplify some lemmas in this special case, find some variations

We may then want to generalise this special case

How to Implement Definitions?

Should already be interesting to re-investigate even without data types

Cf. András Kovács

For executing functional programs, the standard practice is to have

-Immutable program code, which may be machine code or interpreted code.

-Runtime objects, consisting of constructors *and* closures*.*

The basic idea is to use the above setup during elaboration, with some extensions.

How to Implement Definitions?

Problem with definitions

```
[x : Bool] [y : Bool = x] [x : Nat]
```

What is the value of y at this point?

Hiding some definitions?

On going work with D. Grätzer, J. Sterling, C. Anguili, L. Birkedal

Use extension types