Getting Π right in Set

Thierry Coquand (thanks to discussion with Peter and Andreas)

Oct. 11, 2006

Universes

Introduced in three steps: 71 ($V \in V$), 72 (one universe) and 75 (sequence of universes)

Also 75: conversion as judgement and new method (due to Peter Hancock) for showing decidability of conversion

Analogy: computation of a term and evaluation

The normal form of a term is its *semantics*

$nf(M) = \downarrow [\![M]\!]$

Type theory with universes

It is easier to say what a PER model should be than to define the syntax of such a type theory

PER model

Untyped universe of computations (domain model, or terms with β, ι equality) with an application operation

We have a notion of *constructors*: we know when a term is of the form N or U or $\Pi\ x\ f$

Constructors are one-to-one

We can then define: a PER of types and whenever $A = B$ a PER associated to A (which is the same as the PER associated to B)

PER model

Small types: $N = N$ and $0 = 0 : N$ and $u = v : N$ implies $s\ u = s\ v : N$

If $A_1 = A_2$ small types and $u_1 = u_2 : A_1$ implies $F_1\ u_1 = F_2\ u_2$ small types then $\Pi\ A_1\ F_1 = \Pi\ A_2\ F_2$ small types

Then $v_1 = v_2 : \Pi\ A_1\ F_1$ iff $u_1 = u_2 : A_1$ implies $v_1\ u_1 = v_2\ u_2 : F_1\ u_1$

PER model

We define then the PER of all types

$U = U$ and $X_1 = X_2 : U$ iff $X_1 = X_2$ small types

$N = N$ and $0 = 0 : N$ and $u = v : N$ implies $s\ u = s\ v : N$

If $A_1 = A_2$ and $u_1 = u_2 : A_1$ implies $F_1\ u_1 = F_2\ u_2$ then $\Pi\ A_1\ F_1 = \Pi\ A_2\ F_2$

Then $v_1 = v_2 : \Pi\ A_1\ F_1$ iff $u_1 = u_2 : A_1$ implies $v_1\ u_1 = v_2\ u_2 : F_1\ u_1$

PER model

If A is a small type then A is a type

The untyped universe of computation is a combinatory algebra (model of λ-calculus)

We have K such that $K\ x\ y = x$ and we can define $A \rightarrow B$ to be $\Pi\ A\ (K\ B)$

Then $v_1 = v_2 : A \rightarrow B$ iff $u_1 = u_2 : A$ implies $v_1\ u_1 = v_2\ u_2 : B$

Extensional equality?? Almost!

PER model

$v : N \to N$ iff $v\ u : N$ if $u : N$

In general much more elements in the model than the ones that are definable

Even on definable elements, equality at type $N \to N$ is not decidable (extensional equality)

On "pure" typed lambda terms, equality is decidable (it is β, η equality)

PER model

One can add unit types or singleton types or even the types $[A]$ that have the definitions

$[A] = [B]$ iff $A = B$

$a = b : [A]$ iff $a : A$ and $b : A$

General notion of PER model

Reminiscent of Frege structure

We have first a model D of untyped λ-calculus with constructors $\Pi, N, s, 0$ and U

Let $PER(D)$ the set of PER on D. If $X \in PER(D)$ we write $|X|$ the set of elements u such that $X(u,u)$

If $X \in PER(D)$ and $F : |X| \rightarrow PER(D)$ such that $X(u_1, u_2)$ implies $F(u_1) = F(u_2)$ then $\Pi(X, F)$ is the PER defined by $\Pi(X, F)(v_1, v_2)$ iff $X(u_1, u_2) \rightarrow F(u_1)(v_1\ u_1, v_2\ u_2)$

General notion of PER model

A PER model for type theory with universe consists of an element $T \in PER(D)$ with a function $El : |T| \to PER(D)$ such that $El(u_1) = El(u_2)$ if $T(u_1, u_2)$

Furthermore if $T(A_1, A_2)$ and we have $A_1(u_1, u_2)$ implies $T(F_1\ u_1, F_2\ u_2)$ then $T(\Pi\ A_1\ F_1, \Pi\ A_2\ F_2)$ and then $El(\Pi\ A_1\ F_1) = \Pi(El(A_1), \lambda u.El(F_1\ u))$

To have a universe we require also $T(U, U)$ and $El(U)(A, B)$ implies $T(A, B)$ and the PER $El(U)$ is closed under the product operation

A special PER model

D domain of "semantical" values

T the set of "syntactical" normal terms

$T \ ::= \ \lambda x.T \mid \Pi\ x : T.T \mid s\ T \mid 0 \mid S$

$S \ ::= \ x\ \vec{T}$ we write $\nu, \nu', \ldots$ an element of S

We assume that D has a copy of S

A special PER model

We add the elements ν as small types

We add $\nu_1 = \nu_1 : \nu$

Also $\nu_1 = \nu_1 : N$

We define two functions $\uparrow_A : S \rightarrow D$ and $\downarrow_A : D \rightarrow T$ by induction on A type

A special PER model

$$\uparrow_{\Pi\ A\ F}\ \nu = \lambda u.\uparrow_{F\ u} (\nu\ (\downarrow_A u))$$

$$\downarrow_{\Pi\ A\ F}\ v = \lambda x.\downarrow_{F\ (\uparrow_A\ x)} (v\ (\uparrow_A x))$$

$$\uparrow_U \nu = \uparrow_N \nu = \uparrow_{\nu'} \nu = \nu$$

$$\downarrow_U \nu = \downarrow_N \nu = \downarrow_{\nu'} \nu = \nu$$

$$\downarrow_U = \Downarrow$$

$$\Downarrow_{\Pi\ A\ F}\ v = \Pi x : \Downarrow A. \Downarrow (F\ (\uparrow_A x))$$

$$\downarrow_N (s\ u) = s\ (\downarrow_N u), \qquad \downarrow_N 0 = 0$$

Implementation of the type-checker

The values $\uparrow_A x$ corresponds exactly to the notion of *generic* values that are used in the implementation of core agda

The method gives a nice correctness proof of the implementation and extends for sigma types, record types, singleton types ...

A special PER model

To make this definition rigourous, the simplest way seems to follow a *syntactical* approach

(I learnt this from Klaus Aehlig and Felix Joachimski and from Per Martin-Löf's talk on normalisation by evaluation)

We have untyped lambda-calculus with constants

Constructors Π, U, N and Up

We have functions defined by recursion and pattern matching

Up and Down calculus

$$\uparrow (\Pi\ A\ F)\ t = \lambda u. \uparrow (F\ u)\ (t\ (\downarrow A\ u))$$

$$\downarrow (\Pi\ A\ F)\ v = \lambda x. \downarrow (F\ (\uparrow A\ x))\ (v\ (\uparrow A\ x))$$

$$\uparrow U\ t = \uparrow N\ t = \uparrow (\mathsf{Up}\ t')\ t = \mathsf{Up}\ t$$

$$\downarrow U\ (\mathsf{Up}\ t) = \downarrow N\ (\mathsf{Up}\ t) = \downarrow (\mathsf{Up}\ t')\ (\mathsf{Up}\ t) = t$$

$$\downarrow U = \ \Downarrow$$

$$\Downarrow (\Pi\ A\ F)\ v = \Pi x : \Downarrow A. \Downarrow (F\ (\uparrow A\ x))$$

$$\downarrow N(s\ u) = s\ (\downarrow N\ u), \qquad \downarrow N\ 0 = 0$$

Defined function

If we want to represent a "syntactical" function add, defined by

$$add\ x\ 0 = x, \qquad add\ x\ (s\ y) = s\ (add\ x\ y)$$

then we should have also the clause

$$add\ x\ (\mathsf{Up}\ y) = \uparrow N\ (add\ (\downarrow N\ x)\ y)$$

We can then prove $add : N \rightarrow N \rightarrow N$

η expansion

$\downarrow_A \uparrow_A \nu$ is the η-expansion of ν at type A

(This decomposition of η expansion has been discovered by Klaus Aehlig and Felix Joachimski)

For instance, what is the η-expansion of ν at type $N \to N$: $\lambda x.\nu\ x$

At type $\Pi\ U\ (\lambda X.X \to X)$ it is $\lambda X.\lambda x.\nu\ X\ x$

In general the η expansion of ν at type A is not reducible at this type

First main result

If $A_1 = A_2$ and $u_1 = u_2 : A_1$ then $\downarrow_{A_1} u_1 = \downarrow_{A_2} u_2$ (same normal form)

If $A_1 = A_2$ then $\uparrow_{A_1} \nu = \uparrow_{A_2} \nu : A_1$

For $u_1, u_2 : A$ this gives a (decidable) necessary condition for $u_1 = u_2 : A$

It is not sufficient in general ...

but it is if u_1 and u_2 are semantics of well-typed terms!

First main result

In particular if $T = \Pi X : U.X \to X$ we have $\uparrow T\ \nu : T$

Notice that the natural η expansion of ν is *not* of type ν

This solves the problem of how to define η expansion with universes!

The free model

We want a decision procedure for equality of well-typed terms

We should have: if $\vdash A$ then $[\![A]\!]$ type and if $\vdash M_1 = M_2 : A$ then $[\![M_1]\!] = [\![M_2]\!] : [\![A]\!]$

To conclude it is enough to show:

If $\vdash M : A$ then $\vdash M =\downarrow_{[\![A]\!]} [\![M]\!] : A$

The free model

How to define the typing relations $\vdash M : A$?

A lot of possible choices: we can take most of the rules that are valid in the model

For instance if one wants, one can take an explicit substitution rule

One can also look for a minimal set of rules: usual typing rules with *conversion as judgement*

Rules for the Logical Framework

We have a special primitive constant Π of arity 2 and we write

$(x : A) \rightarrow B$ for

$\Pi\ A\ (\lambda x.B)$

We have also a special primitive constant U of arity 0, and a special primitive constant El of arity 1

Type-checking rules

rules for contexts

$$\frac{}{() \text{ correct}} \qquad \frac{\Gamma \text{ correct} \qquad \Gamma \vdash A}{\Gamma, x{:}A \text{ correct}}$$

rules for types

$$\frac{\Gamma \text{ correct}}{\Gamma \vdash \mathsf{U}} \qquad \frac{\Gamma \vdash M : \mathsf{U}}{\Gamma \vdash M} \qquad \frac{\Gamma, x{:}A \vdash B}{\Gamma \vdash (x{:}A) \to B}$$

Type-checking rules

rules for terms

$$\frac{\Gamma \text{ correct} \qquad (x{:}A) \in \Gamma}{\Gamma \vdash x{:}A}$$

$$\frac{\Gamma, x{:}A \vdash M : B}{\Gamma \vdash \lambda x.M : (x{:}A) \to B}$$

$$\frac{\Gamma \vdash N : (x{:}A) \to B \qquad \Gamma \vdash M : A}{\Gamma \vdash N\ M : B[M]}$$

Type-checking rules

type equality rule

$$\frac{\Gamma \vdash M : A \qquad \Gamma \vdash A = B}{\Gamma \vdash M : B}$$

Type-checking rules

conversion rules

$$\frac{\Gamma \vdash A}{\Gamma \vdash A = A} \qquad \frac{\Gamma \vdash A = B}{\Gamma \vdash B = A} \qquad \frac{\Gamma \vdash A = B \quad \Gamma \vdash B = C}{\Gamma \vdash A = C}$$

$$\frac{\Gamma \vdash M : A}{\Gamma \vdash M = M : A} \qquad \frac{\Gamma \vdash M = N : A}{\Gamma \vdash N = M : A} \qquad \frac{\Gamma \vdash M = N : A \quad \Gamma \vdash N = P : A}{\Gamma \vdash M = P : A}$$

$$\frac{\Gamma \vdash M = N : A \quad \Gamma \vdash A = B}{\Gamma \vdash M = N : B}$$

Soundness of the PER semantics

All these rules are valid in the PER semantics

If $\Gamma \vdash A$ and $\rho_1 = \rho_2 : \Gamma$ then $A\rho_1 = A\rho_2$

If $\Gamma \vdash A_1 = A_2$ and $\rho_1 = \rho_2 : \Gamma$ then $A_1\rho_1 = A_2\rho_2$

This is direct by induction on typing derivations

In particular if $\vdash M : A$ we know that $[\![M]\!] : [\![A]\!]$ and we can consider $\downarrow_{[\![A]\!]} [\![M]\!]$

To show that $\vdash M : A$ implies $\vdash M =\downarrow_{[\![A]\!]} [\![M]\!] : A$ one introduces a logical relation (this is the core of the method)

A logical relation

One define $R(A, X)$ for $\vdash A$ and X type and if this holds one defines $R_{A,X}(M, u)$ for $\vdash M : A$ and $u : X$

This is quite subtle and where all the checks should be done

If $\vdash C = (x : A) \rightarrow B$ and $Z = \Pi\ X\ F$ and $R(A, X)$ and $R_{A,X}(M, u)$ implies $R(B[M], F\ u)$ then we have $R(C, Z)$

If $\vdash C = N$ then $R(C, N)$ and $R_{C,N}(M, u)$ means $M =\downarrow_N u : N$

A logical relation

To make the definition works we have to add the non standard conversion rule (which is semantically valid)

$$\frac{\Gamma, x : A_1 \vdash B_1 \qquad \Gamma, x : A_2 \vdash B_2 \qquad \Gamma \vdash (x : A_1) \to B_1 = (x : A_2) \to B_2}{\Gamma \vdash A_1 = A_2 \qquad \Gamma, x : A_1 \vdash B_1 = B_2}$$

A logical relation

The rest of the argument is more standard: we introduce new constants c^A for $\vdash A$ with the unique rule $\vdash c^A : A$

We prove then that if $R(A, X)$ holds

$R_{A,X}(M, u)$ implies $\vdash M =\downarrow_A u : A$

if $\vdash \nu = \nu' : A$ then $R_{A,X}(\nu, \uparrow_A \nu')$

We can also show $R(A, [\![A]\!])$ if $\vdash A$ and $R_{A,[\![A]\!]}(M, [\![M]\!])$ if $\vdash M : A$ by induction on derivations

It follows that we have $\vdash M =\downarrow_{[\![A]\!]} [\![M]\!] : A$ if $\vdash M : A$

Forget the syntax?

One has the following result: if $\vdash M_1 : A$ and $\vdash M_2 : A$ then $[\![M_1]\!] = [\![M_2]\!] : [\![A]\!]$ is *decidable*

This suggests the following result, which would be "syntax independent" (no mention of how we build the free model in the statement)

If A in normal form then it is decidable whether or not $[\![A]\!]$ type, for *all* PER models, holds

If M, A in normal form then it is decidable whether or not $[\![M]\!] : [\![A]\!]$, for *all* PER models, holds in the model

This proof

Perfect for the Logical Framework

Can one avoid the use of the non standard conversion rule?

How did Martin-Löf 75 managed without adding the non standard conversion rules?