On Lemma 6.10.12 in the HoTT book

Vladimir Voevodsky

December 20, 2013.

In the current version of the HoTT book there is a lemma of the following form:

Lemma Suppose $P : \mathbf{Z} \to \mathcal{U}$ is a type family and that we have

$d0 : P(0)$,

$d+ : \prod(n : \mathbf{N}) P(n) \to P(succ(n))$, and $d- : \prod(n : \mathbf{N}) P(-n) \to P(-succ(n))$

Then we have $f : \prod(z : \mathbf{Z}) P(z)$ such that $f(0) \equiv d0$ and $f(succ(n)) \equiv d + (f(n))$, and $f(-succ(n)) \equiv d-(f(-n))$ for all $n : \mathbf{N}$.

Where $\equiv$ denotes definitional equality. The following note is a summary of my attempt to prove a non-dependent version of this lemma in Coq using the definition of integers "`hz`" given in the Foundations library.

First it is unclear what the condition of the form $a(n) \equiv b(n)$ for all n is supposed to mean. Indeed there are two *different* interpretations. First that for any numeral n one has $a(n) \equiv b(n)$. Second that $a \equiv b$ in the function type. In Coq the first condition is strictly weaker than the second. In what follows we consider the second meaning.

The non-dependent form of the lemma looks as follows[1]:

"`Lemma l61012nd ( T : UU ) ( d0 : T ) ( dplus : forall n : nat , T -> T ) ( dminus : forall n : nat , T -> T ) : hz -> T .`"

The conditions are equivalent to the acceptability by Coq of the following code:

"`Lemma test1 ( T : UU ) ( d0 : T ) ( dplus : forall n : nat , T -> T ) ( dminus : forall n : nat , T -> T ) : paths ( l61012nd d0 dplus dminus ( nattohz 0 ) ) d0 . Proof . intros . apply idpath . Defined.`

`Lemma test2 ( T : UU ) ( d0 : T ) ( dplus : forall n : nat , T -> T ) ( dminus : forall n : nat , T -> T ) ( n : nat ) : paths ( l61012nd d0 dplus dminus ( nattohz ( S n ) ) ) ( dplus n ( l61012nd d0 dplus dminus ( nattohz n ) ) ) . Proof . intros . apply idpath . Defined.`

`Lemma test3 ( T : UU ) ( d0 : T ) ( dplus : forall n : nat , T -> T ) ( dminus : forall n : nat , T -> T ) ( n : nat ) : paths ( l61012nd d0 dplus dminus ( hzsign ( nattohz ( S n ) ) ) ) ( dminus n ( l61012nd d0 dplus dminus ( hzsign ( nattohz n ) ) ) ) . Proof . intros . apply idpath . Defined.`"

At the moment I am unable to find a proof of "`l61012nd`" which would make these idpath-proofs of "`test1`","`test2`" and "`test3`" to compile.

[1] All of the Coq code in the note can be copy-pasted directly into Coq. Compilation requires the file hz.v from the Foundations library.

The reason is somewhat subtle and interesting. The first surprising fact is that such a proof of "`l61012nd`" can be found if we add the additional assumption that "`T`" is an h-set. Namely, the following code does work:

"`Lemma l61012aa ( T : UU ) ( d0 : T ) ( dplus : forall n : nat , T -> T ) ( dminus : forall n : nat , T -> T ) : forall nm: dirprod nat nat , T. Proof. intros. destruct nm as [n m ] . generalize m . clear m . induction n.`

`intro m . induction m . apply d0. apply ( dminus m IHm ) . intro m . destruct m . apply ( dplus n (IHn O) ) . apply (IHn m) . Defined .`

`Lemma l61012ab ( T : hSet ) ( d0 : T ) ( dplus : forall n : nat , T -> T ) ( dminus : forall n : nat , T -> T ) ( n m : nat ) : paths ( l61012aa T d0 dplus dminus ( dirprodpair n m ) ) ( l61012aa T d0 dplus dminus ( dirprodpair ( S n ) ( S m ) ) ) . Proof. intros . apply idpath . Defined.`

`Lemma l61012a ( T : hSet ) ( d0 : T ) ( dplus : forall n : nat , T -> T ) ( dminus : forall n : nat , T -> T ) : hz -> T . Proof. intros T d0 dplus dminus. unfold hz . unfold commrigtocommrng. simpl .`

`apply ( setquotuniv (hrelabgrfrac (rigaddabmonoid natcommrig)) T ( l61012aa T d0 dplus dminus ) ). admit . Defined .`

`Lemma test1a ( T : hSet ) ( d0 : T ) ( dplus : forall n : nat , T -> T ) ( dminus : forall n : nat , T -> T ) : paths ( l61012a T d0 dplus dminus ( nattohz O ) ) d0 . Proof . intros . apply idpath . Defined.`

`Lemma test2a ( T : hSet ) ( d0 : T ) ( dplus : forall n : nat , T -> T ) ( dminus : forall n : nat , T -> T ) ( n : nat ) : paths ( l61012a T d0 dplus dminus ( nattohz ( S n ) ) ) ( dplus n ( l61012a T d0 dplus dminus ( nattohz n ) ) ) . Proof . intros . apply idpath . Defined.`

`Lemma test3a (T : hSet) (d0 : T) (dplus : forall n : nat , T -> T) (dminus : forall n : nat , T -> T) (n : nat) : paths (l61012a T d0 dplus dminus (hzsign (nattohz (S n)))) (dminus n (l61012a T d0 dplus dminus (hzsign (nattohz n)))) . Proof . intros . apply idpath . Defined.`"

The best I can do for a general "`T`" is to provide a proof of "`l61012nd`" for which the idpath-proofs of "`test1`" and "`test2`" work and a proof of "`test3`" requires a "`destruct`" *or* a proof for which idpath-proofs of "`test1`" and "`test3`" work and the proof of "`test2`" requires a detract. Here is the code for the first case:

"`(* We define the subtraction on natural numbers in two different ways. Our first definition [minus1] coincides with the one given in the standard library. It has the property that [minus1 O n] is definitionally equal to [O] but not that [minus1 n O] is definitionally equal to [n]. The other definition [minus2] has a complimentary set of properties - [minus2 n O] is definitionally equal to [n] but [minus2 O n] is not definitionally equal to [O]. *)`

`Definition minus1 ( n m : nat ) : nat . Proof. intro n . induction n .`

`intro m . apply O . intro m . destruct m . apply ( S n ) . apply (IHn m). Defined.`

`Definition minus2 ( n m : nat ) : nat . Proof. intros . generalize n . clear n . induction m .`

`intro n . apply n .`

`intro n . destruct n . apply O . apply (IHm n). Defined.`

`(* The two minus functions are used to define a section of the projection [ dirprod nat nat -> hz ] *)`

`Definition rnatnat : dirprod nat nat -> dirprod nat nat := fun nm => dirprodpair ( minus2 ( pr1 nm ) ( pr2 nm ) ) ( minus1 ( pr2 nm ) ( pr1 nm ) ) .`

`Definition rhz : hz -> dirprod nat nat . Proof . unfold hz . unfold commrigtocommrng. simpl .`

`apply ( setquotuniv (hrelabgrfrac (rigaddabmonoid natcommrig)) ( setdirprod natset natset ) rnatnat ). admit . Defined.`

`Definition l61012nd ( T : UU ) ( d0 : T ) ( dplus : forall n : nat , T -> T ) ( dminus : forall n : nat , T -> T ) ( z : hz ) : T := l61012aa T d0 dplus dminus ( rhz z ) .`

`Lemma test1 ( T : UU ) ( d0 : T ) ( dplus : forall n : nat , T -> T ) ( dminus : forall n : nat , T -> T ) : paths ( l61012nd T d0 dplus dminus ( nattohz 0 ) ) d0 . Proof . intros . apply idpath . Defined.`

`Lemma test2 ( T : UU ) ( d0 : T ) ( dplus : forall n : nat , T -> T ) ( dminus : forall n : nat , T -> T ) ( n : nat ) : paths ( l61012nd T d0 dplus dminus ( nattohz ( S n ) ) ) ( dplus n ( l61012nd T d0 dplus dminus ( nattohz n ) ) ) . Proof . intros . apply idpath . Defined.`”

The idpath-proof

“`Lemma test3 ( T : UU ) ( d0 : T ) ( dplus : forall n : nat , T -> T ) ( dminus : forall n : nat , T -> T ) ( n : nat ) : paths ( l61012nd T d0 dplus dminus ( hzsign ( nattohz ( S n ) ) ) ) ( dminus n ( l61012nd T d0 dplus dminus ( hzsign ( nattohz n ) ) ) ) .`”

does not work. The best one can get is a destruct-idpath-idpath-proof:

“`Proof . intros . destruct n . apply idpath . apply idpath . Defined.`”

which among other things implies that the idpath-proof would work if we substitute for the variable “`n:nat`” in “`test3`” any numeral (e.g. 5).

Let us consider now what are the issues which prevent us from getting a proof of “`l61012nd`” with would make the idpath-proofs of all three test lemmas work. Much of what happens can be seen in terms of the following diagram:

$$\begin{tikzcd} X \arrow[r, "r"] \arrow[d, "p"'] & X \arrow[r, "f"] \arrow[d, "p"] & T \\ X/R \arrow[ur, "r_R"] & X/R \arrow[ur, "f_R"] & \end{tikzcd}$$

where in our case $X = \mathbf{N} \times \mathbf{N}$, R is the equivalence relation such that $\mathbf{Z} = (\mathbf{N} \times \mathbf{N})/R$, r is defined in "`rnatnat`" and r_R in "`rhz`".

For the data "`d0, dplus, dminus`" we construct in "`l61012aa`" a function $f : \mathbf{N} \times \mathbf{N} \to T$ such that

$$f(0,0) = d0 \;\; f(1+n,0) = dplus(n, f(n,0))$$

$$f(0,1+m) = dminus(m, f(0,m)) \;\; f(1+n,1+m) = f(n,m)$$

To agree with the idpath-proofs of the three test lemmas we need a function f_R such that

$$f_R(p(0,0)) = d0$$

$$f_R(p(1+n,0)) = dplus(n, f_R(p(n,0))$$

$$f_R(p(0,1+m)) = dminus(m, f_R(p(0,m)))$$

The condition $f(1+n,1+m) = f(n,m)$ implies that it is compatible with R. Therefore when T is an h-set we may apply "`setquotuniv`" to obtain f_R which makes the corresponding triangle definitionally commutative and therefore satisfies all three required conditions.

When T is a general type we define f_R instead by $f_R(z) = f(r_R(z))$. In order for the conditions to be satisfied in this case we need the equations

$$f(r(0,0)) = d0$$

$$f(r(1+n,0)) = dplus(n, f(r(n,0)))$$

$$f(r(0,1+m)) = dminus(m, f(r(0,m)))$$

which would follow if we could find r such that

$$r(n,0) = (n,0)$$

$$r(0,m) = (0,m)$$

$$r(1+n,1+m) = r(n,m)$$

or equivalently we need to construct $r1 : \mathbf{N} \times \mathbf{N} \to \mathbf{N}$ such that

$$r1(n,0) = n$$

$$r1(0,m) = 0$$

$$r1(1+n,1+m) = r1(n,m)$$

The problem of this approach is that there seems to be no way in Coq to define a function $r1$ satisfying (definitionally) the first two of these conditions.

Another possibility would be to try to modify "`setquotuniv`" to obtain universality of set-quotients of *h-sets* by equivalence relations with respect to compatible functions to all types. More generally, we can ask:

Q1 Can we find a construction for set-quotients *of h-sets* by equivalence relations which would be universal with respect to compatible functions to all types?

I think this can be reduced to the case of the maximal equivalence relation leading to the following question:

Q2 Given "`(X:hSet)(T:UU)(f:X->T)(is:forall (x1 x2 : X), paths (f x1) (f x2))`" can we construct "`fis:ishinh(X)->T`" such that for all "`x:X`" one has "`fis(hinhpr x)`$\equiv$ `f x`".

A weaker version of the same question would only require a path equality between "`fis(hinhpr x)`" and "`f x`".

So far I see no way to either find a definition of "`fis`" or to show that it can not be done in general. For the later possibility we can ask the following:

Q3 Can we show that there is no proof for the following pair of lemmas

"`Lemma fis_UU (X:UU)(T:UU)(f:X->T)(is:forall (x1 x2:X), paths (f x1) (f x2)) : ishinh(X)->T.`"

"`Lemma fis_UU_paths (X:UU)(T:UU)(f:X->T)(is:forall (x1 x2:X), paths (f x1) (f x2))(x:X): paths (fis_UU X T f is (hinhpr x)) (f x).`"