# Homework 3: Identity Crisis

15-819 Homotopy Type Theory

Out: 19/Oct/13
Due: 2/Nov/13

## 1 J Walkers

### 1.1 Transitive Transportation

**Task 1.** *Implement these programs.*

1. $\mathtt{trans}'_A$ *by induction on another path argument.*
2. $\mathtt{trans}''_A$ *in terms of* $\mathtt{tr}$*.*
3. $\mathtt{ap}'_f(p)$ *in terms of* $\mathtt{tr}$*, where (in the current context)* $f : A \to B$ *and* $p : \mathtt{Id}_A(m, n)$*.*

*In addition, briefly summarize the computational rules (definitional equalities) your implementation satisfies. For example, the* $\mathtt{trans}_A$ *given in class satisfies the following definitional equality.*

$$\mathtt{trans}_A(m)(m)(n)(\mathtt{refl}_A(m))(p) \equiv p$$

**Solution:**

1. $\mathtt{trans}'_A$

$$\mathtt{trans}'_A :\equiv \lambda mnpuv.\mathsf{J}[n.p.\_.\mathtt{Id}_A(m,n) \to \mathtt{Id}_A(m,p)](v, n.\mathtt{id}_A)(u)$$
$$\mathtt{trans}'_A(m)(n)(n)(u)(\mathtt{refl}_A(n)) \equiv u$$

2. $\mathtt{trans}''_A$

$$\mathtt{trans}''_A :\equiv \lambda mnpuv.\mathtt{tr}[n.\mathtt{Id}_A(m,n)](v)(u)$$
$$\mathtt{trans}''_A(m)(n)(n)(u)(\mathtt{refl}_A(n)) \equiv u$$

3. $\mathtt{ap}'_f(p)$

$$\mathtt{ap}'_f(p) :\equiv \mathtt{tr}[n.\mathtt{Id}_B(f\,m, f\,n)](p)(\mathtt{refl}_B(f\,n))$$
$$\mathtt{ap}'_f(\mathtt{refl}_A(n)) \equiv \mathtt{refl}_B(f\,n)$$

## 1.2 Group Ticket

Every type $A$ in HoTT can be viewed as a (higher) groupoid. The purpose of this part is to verify some critical rules, but before going to the task, there is one thing that you need to understand.

In the following task, you are required to explicitly write down endpoints in the motive of J. The reason is that there are cases where, if you could plug in $\mathtt{refl}_A(m)$ directly, the theorem would be trivial; however, in order to come up with a valid motive, you need to generalize the theorem considerably. The difficulty is all in finding the right motive.

One example is the Eckmann-Hilton theorem (in HoTT)

$$\prod_{(m:A)} \prod_{(p,q:\mathtt{refl}_A(m)=\mathtt{refl}_A(m))} p \cdot q = q \cdot p,$$

which would be trivial if you could simply consider the case $p :\equiv \mathtt{refl}_{\mathtt{Id}_A(m,m)}(\mathtt{refl}_A(m))$. The problem is that you cannot easily find a suitable motive for J to put such $\mathtt{refl}$ in. The motive of J, which can be written as $m.n.p.C$, requires two endpoints ($m$ and $n$) of $p$ to be "free" in $C$. In other words, $C$ needs to be generic in $m$ and $n$ and $p$. Unfortunately, coming up with such a motive for this theorem requires some work. One can show that there is an alternative, equivalent J that only requires one endpoint to be "free", but not less. Work still needs to be done in order to "free" at least one endpoint in the above theorem.

**Bonus Task 1.** *Briefly explain what would fail if you took the theorem as the motive of* J *directly.*

**Solution:** *This question is open-ended. The following is just one possible answer.* One plausible choice is to do induction on $p$ along with its endpoints directly. Without loss of generality, assume $p$ has the type $m = n$. In order to make both $p \cdot q$ and $q \cdot p$ type check, $q$ must have the type $n = m$. However, if $p \cdot q = q \cdot p$ is well-typed then $m \equiv n$, which defeats the freeness of endpoints.

What would happen if we allowed motives that are not generic in endpoints? In fact, there is another famous rule, named K, which is similar to J but does not demand such great freeness of endpoints in J. The additional rule K essentially allows one to consider restricted motives $m.p.C$ where both endpoints of $p$ must be the same.

$$\frac{\Gamma \vdash A : \mathcal{U} \qquad \Gamma \vdash M : A \qquad \Gamma \vdash P : \mathtt{Id}_A(M, M) \qquad \Gamma, m{:}A, p{:}\mathtt{Id}_A(m, m) \vdash C : \mathcal{U} \qquad \Gamma, m{:}A \vdash Q : [m, \mathtt{refl}_A(m)/m, p]C}{\Gamma \vdash \mathsf{K}[m.p.C](P, m.Q) : [M, P/m, p]C}$$

One can show that including this rule K (in addition to J) to our type theory implies *uniqueness of identity paths* and collapse of all higher structures. Thus, we must be careful about these endpoints in HoTT. The reason why this rule or freeness of endpoints has anything to do with the collapse should be more clear in the future lectures. Later on we will omit endpoints in favor of more convenient notations, but remember that you always have to pay attention to them.

**Task 2.** *Write code with the following types. You may assume that*

$$\mathtt{trans}_A(m)(m)(n)(\mathtt{refl}_A(m))(p) \equiv p$$

*and*

$$\mathtt{sym}_A(m)(m)(\mathtt{refl}_A(m)) \equiv \mathtt{refl}_A(m).$$

*Please write down the dependency on endpoints in the motive of* J *explicitly.*

- ***Inverse-right.*** *"*$p \cdot p^{-1} = \mathtt{refl}$*".*

$$\prod_{m:A} \prod_{n:A} \prod_{p:\mathtt{Id}_A(m,n)} \mathtt{Id}_{\mathtt{Id}_A(m,m)}(\mathtt{trans}_A(m)(n)(m)(p)(\mathtt{sym}_A(m)(n)(p)), \mathtt{refl}_A(m))$$

- ***Inverse-left.*** *"$p^{-1} \cdot p = \mathtt{refl}$".*

$$\prod_{m:A} \prod_{n:A} \prod_{p:\mathtt{Id}_A(m,n)} \mathtt{Id}_{\mathtt{Id}_A(n,n)}(\mathtt{trans}_A(n)(m)(n)(\mathtt{sym}_A(m)(n)(p))(p), \mathtt{refl}_A(n))$$

- ***Unit-right.*** *"$p \cdot \mathtt{refl} = p$".*

$$\prod_{m:A} \prod_{n:A} \prod_{p:\mathtt{Id}_A(m,n)} \mathtt{Id}_{\mathtt{Id}_A(m,n)}(\mathtt{trans}_A(m)(n)(n)(p)(\mathtt{refl}_A(n)), p)$$

- ***Unit-left.*** *"$\mathtt{refl} \cdot p = p$".*

$$\prod_{m:A} \prod_{n:A} \prod_{p:\mathtt{Id}_A(m,n)} \mathtt{Id}_{\mathtt{Id}_A(m,n)}(\mathtt{trans}_A(m)(m)(n)(\mathtt{refl}_A(m))(p), p)$$

- ***Associativity.*** *"$(p \cdot q) \cdot r = p \cdot (q \cdot r)$".*

$$\prod_{l:A} \prod_{m:A} \prod_{n:A} \prod_{o:A} \prod_{p:\mathtt{Id}_A(l,m)} \prod_{q:\mathtt{Id}_A(m,n)} \prod_{r:\mathtt{Id}_A(n,o)} \mathtt{Id}_{\mathtt{Id}_A(l,o)}(\mathtt{trans}_A(l)(n)(o)(\mathtt{trans}_A(l)(m)(n)(p)(q))(r), \mathtt{trans}_A(l)(m)(o)(p)(\mathtt{trans}_A(m)(n)(o)(q)(r)))$$

> ERRATUM (2013/10/29 3PM): The computational behavior of $\mathtt{sym}_A$ is added.

> MORE HINTS (2013/10/31 7PM): I highly recommend renaming all bound variables in the motives and check well-typedness of your motives. Do not let two variables of the same name fool you!

**Solution:**

- **Inverse-right.**

$$\lambda mnp.\mathsf{J}[m.n.p.\mathtt{Id}_{\mathtt{Id}_A(m,m)}(\mathtt{trans}_A(m)(n)(m)(p)(\mathtt{sym}_A(m)(n)(p)), \mathtt{refl}_A(m)))] (p, m.\mathtt{refl}_{\mathtt{Id}_A(m,m)}(\mathtt{refl}_A(m)))$$

- **Inverse-left.**

$$\lambda mnp.\mathsf{J}[m.n.p.\mathtt{Id}_{\mathtt{Id}_A(n,n)}(\mathtt{trans}_A(n)(m)(n)(\mathtt{sym}_A(m)(n)(p))(p),\mathtt{refl}_A(n)))] \\ (p, n.\mathtt{refl}_{\mathtt{Id}_A(n,n)}(\mathtt{refl}_A(n)))$$

- **Unit-right.**

$$\lambda mnp.\mathsf{J}[m.n.p.\mathtt{Id}_{\mathtt{Id}_A(m,n)}(\mathtt{trans}_A(m)(n)(n)(p)(\mathtt{refl}_A(n)),p)](p, m.\mathtt{refl}_{\mathtt{Id}(m,n)}(\mathtt{refl}_A(m)))$$

- **Unit-left.**

$$\lambda mnp.\mathtt{refl}_{\mathtt{Id}(m,n)}(p)$$

- **Associativity.**

$$\lambda lmnopqr.\mathsf{J}[l'.m'.p'.\textstyle\prod_{q':\mathtt{Id}_A(m',n)} \mathtt{Id}_{\mathtt{Id}_A(l',o)}(\mathtt{trans}_A(l')(n)(o)(\mathtt{trans}_A(l')(m')(n)(p')(q'))(r), \\ \mathtt{trans}_A(l')(m')(o)(p')(\mathtt{trans}_A(m')(n)(o)(q')(r)))] \\ (p, m'.\lambda(q' : \mathtt{Id}_A(m', n)).\mathtt{refl}_{\mathtt{Id}_A(m',o)}(\mathtt{trans}_A(m')(n)(o)(q')(r)))(q)$$

# 2 Elementary Math

In this section we will show some interesting properties of natural numbers. In this section you may omit endpoint arguments (to $\mathtt{trans}_A$ and $\mathtt{sym}_A$) and use $\cdot$ and $^{-1}$ instead, since there are no interesting higher structures in natural numbers anyway. Feel free to use $=$, $\mathtt{ap}_f$ and $\mathtt{tr}$ in your programs as well.

## 2.1 Commutativity of Addition

The goal of this section is to show $\prod_a \prod_b a + b =_{\mathtt{nat}} b + a$. Assume that our implementation of $+$ has the following computational behaviors:

- $\mathtt{zero} + b \equiv b$.
- $\mathtt{suc}(a) + b \equiv \mathtt{suc}(a + b)$.

**Task 3.** *Write code of type* $\prod_a a + \mathtt{zero} =_{\mathtt{nat}} a$ *(and probably name it).*

**Solution:** $\mathtt{lem}_1 :\equiv \lambda a.\mathtt{rec}[a.a + \mathtt{zero} =_{\mathtt{nat}} a](a, \mathtt{refl}(\mathtt{zero}), x.y.\mathtt{ap}_{\mathtt{suc}}(y))$.

**Task 4.** *Write code of type* $\prod_a \prod_b a + \mathtt{suc}(b) =_{\mathtt{nat}} \mathtt{suc}(a + b)$ *(and name it).*

**Solution:** $\mathtt{lem}_2 :\equiv \lambda ab.\mathtt{rec}[a, a{+}\mathtt{suc}(b) =_{\mathtt{nat}} \mathtt{suc}(a{+}b)](a, \mathtt{refl}(\mathtt{suc}(b)), x.y.\mathtt{ap}_{\mathtt{suc}}(y))$.

**Task 5.** *Finally, write code of type* $\prod_a \prod_b a{+}b =_{\mathtt{nat}} b{+}a$ *which uses the previous two tasks as subroutines.*

**Solution:** $\lambda ab.\mathtt{rec}[b.a + b =_{\mathtt{nat}} b + a](b, \mathtt{lem}_1(a), b.y.\mathtt{lem}_2(a)(b) \cdot \mathtt{ap}_{\mathtt{suc}}(y))$

## 2.2 Injectivity of Successor

**Task 6.** *Prove (a.k.a. write code of type)*

$$\prod_a \prod_b (\mathtt{suc}(a) =_{\mathtt{nat}} \mathtt{suc}(b)) \to (a =_{\mathtt{nat}} b).$$

**(Hint)** *Write a function* $f$ *such that* $f(\mathtt{suc}(a)) \equiv a$.

**Solution:** Let $f :\equiv \lambda a.\mathtt{rec}[a.\mathtt{nat}](a, \mathtt{zero}, x.y.x)$. $\lambda abp.\mathtt{ap}_f(p)$ does the job.

## 2.3 Non-degeneracy of Natural Numbers

**Task 7.** *Prove* $\prod_a (\mathtt{suc}(a) =_{\mathtt{nat}} \mathtt{zero}) \to \bot$. **(Hint)** *Can you come up with a function* $f$ *and a path* $p : a =_{\mathtt{nat}} b$ *such that* $f(a) \equiv \top$ *but* $f(b) \equiv \bot$*? What can you do with such* $f$ *and* $p$*?*

**Solution:** Let $f :\equiv \lambda a.\mathtt{rec}[a.\mathcal{U}](a, \bot, x.y.\top)$. $\lambda abp.\mathtt{tr}[x.f(x)](p)(\langle\rangle)$ will work.