PFPL Supplement: Equality of T and F*

Robert Harper

October, 2020

1 Introduction

When considering languages such as Systems **T** and **F**, and their extensions with products, sums, inductive, and coinductive types, it is both natural and justifiable to use equational reasoning similar to standard mathematical practice. After all, these languages were introduced to explore the computational meaning of formal proofs in logical systems, and so ought to behave like all other mathematical objects. This supplement expounds some basic principles of equality for these languages.

2 Typed Equality

The most basic principle of equality is that it governs expressions of the same type, and its meaning is *determined by their type*. The judgment $e \doteq e' \in \tau$ is defined for *closed* expressions e and e' of type τ to mean

> There exist v and v' such that $e \longmapsto^* v$ val, $e' \longmapsto^* v'$ val, and v and v' are equal in the sense determined by their type, τ.

Thus, equality specifies that two expressions *co-behave* according to their type in that the values of expressions are all that matter in determining their equality. It is not at all obvious—but can be proved for specific languages—that typed equality is reflexive, let alone an equivalence relation or a congruence.[1] The notation $e \in \tau$ means that $e \doteq e \in \tau$, which states that e behaves in accordance with its type. Sometimes it is more convenient to write $e \doteq_\tau e'$ for equality of e and e' at type τ.

Assuming addition and multiplication are defined in the usual way (by recursion on either or both arguments), then obvious equations such as $\overline{2} + \overline{2} \doteq \overline{4} \in \mathtt{nat}$, $\overline{2} \times \overline{2} \doteq \overline{4} \in \mathtt{nat}$, $\overline{4} \doteq \overline{2} + \overline{2} \in \mathtt{nat}$, and $\overline{1} + \overline{3} \doteq \overline{2} + \overline{2} \in \mathtt{nat}$, would hold, simply by virtue of calculation, and the fact that $\overline{4} \in \mathtt{nat}$.[2]

Typed equality of *open* terms, written $\Gamma \vdash e \doteq e' \in \tau$, is defined for $\Gamma \vdash e : \tau$ and $\Gamma \vdash e' : \tau$ in terms of closed equality by substitution. For simplicity let us consider the case of one variable, $x : \sigma \vdash e \doteq e' \in \tau$; the generalization to many is straightforward. It might be thought that the definition should be

[1] One might argue that it should not be called "equality" until it is proved that this is so. There is merit to that argument, but in all cases it will turn out to be true, so it is not important to split this particular hair.

[2] This obvious fact will turn out to be true, once the values of type `nat` have been defined.

If $e_1 \in \sigma$, then $\{e_1/x\}e \doteq \{e_1/x\}e' \in \tau$.

That is, an equation between open terms holds whenever all of its substitution instances hold. However, a stronger condition is required:

If $e_1 \doteq e_1' \in \sigma$, then $\{e_1/x\}e \doteq \{e_1'/x\}e' \in \tau$.

The former condition follows from the latter, using reflexivity of equality at each type.

Surprisingly, reflexivity is not immediate, or even very obvious, for languages with higher-order functions or inductive types! Intuitively, reflexivity states that expressions behave at run-time according to their compile-time type. This ought to be true for any sensible language, but it does require proof.

Theorem 2.1 (Fundamental Theorem). *1. If $\Gamma \vdash e : \tau$, then $\Gamma \vdash e \doteq e \in \tau$.*

2. If $\Gamma \vdash e \equiv e' : \tau$, then $\Gamma \vdash e \doteq e' \in \tau$.

This theorem holds for System T and its extensions with products, sums, inductive, and coinductive types.

Equations involving variables, such as $x : \mathtt{nat} \vdash x + x \doteq \overline{2} \times x \in \mathtt{nat}$, are not merely a matter of calculation, but must be proved by induction on x. Specifically,

1. $\overline{0} + \overline{0} \doteq \overline{2} \times \overline{0} \in \mathtt{nat}$. This is a matter of calculation, no variables are involved.
2. Assuming that $\overline{n} + \overline{n} \doteq \overline{2} \times \overline{n} \in \mathtt{nat}$, show that $\overline{n+1} + \overline{n+1} \doteq \overline{2} \times \overline{n+1} \in \mathtt{nat}$. This is proved by simplification and appeal to the inductive assumption.

Typed equality of values is defined as follows:

1. $\langle\rangle \in \mathtt{unit}$.
2. There is no value v such that $v \in \mathtt{void}$.
3. $\langle e_1, e_2\rangle \doteq \langle e_1', e_2'\rangle \in \tau_1 \times \tau_2$ iff $e_1 \cdot \mathtt{l} \doteq e_1' \cdot \mathtt{l} \in \tau_1$ and $e_1 \cdot \mathtt{r} \doteq e_2 \cdot \mathtt{r} \in \tau_2$.
4. $\mathtt{l} \cdot e_1 \doteq \mathtt{l} \cdot e_1' \in \tau_1 + \tau_2$ iff $e_1 \doteq e_1' \in \tau_1$, and $\mathtt{r} \cdot e_2 \doteq \mathtt{r} \cdot e_2' \in \tau_1 + \tau_2$ iff $e_2 \doteq e_2' \in \tau_2$.
5. $\lambda(x : \tau_1)\, e_2 \doteq \lambda(x : \tau_1)\, e_2' \in \tau_1 \to \tau_2$ iff $e_1 \doteq e_1' \in \tau_1$ implies $\{e_1/x\}e_2 \doteq \{e_1'/x\}e' \in \tau_2$.
6. $\_ \doteq \_ \in \mathtt{nat}$ is the *strongest* binary relation between values such that if $e \doteq e' \in \mathtt{unit} + \mathtt{nat}$, then $\mathtt{fold}(e) \doteq \mathtt{fold}(e') \in \mathtt{nat}$.
7. $\_ \doteq \_ \in \mathtt{conat}$ is the *weakest* binary relation between values such that if $v \doteq v' \in \mathtt{conat}$, then $\mathtt{unfold}(v) \doteq \mathtt{unfold}(v') \in \mathtt{unit} + \mathtt{conat}$.

Equality of values of product and sum types are easily visualized using traditional "box and pointer" diagrams. Equality of functions is entirely about input/output behavior, and not about the details of the code. For example,

$$\lambda(x : \mathtt{nat})\, x + x \doteq \lambda(x : \mathtt{nat})\, 2 \times x \in \mathtt{nat} \to \mathtt{nat},$$

even though the code is different on either side. Equality of values of inductive type is the strongest, or least, relation closed under folding (in this case, inclusion of zero and closure under successor).

The restriction to the least relation means that two natural numbers are equal only if they are "forced" to be equal by these closure conditions. Dually, equality of values of coinductive type is weakest, or largest, relation consistent with unfolding (in this case, with every value being either zero or a successor). The requirement to be the largest relation means that two co-natural numbers are equal unless their equality can be "refuted" by unfolding. (Please see Harper (2020) for more on equality for these two types.)

3 Parametricity

Parametricity is the extension of typed equality to polymorphic types. As a first cut, one might postulate that

$$e_1 \doteq e_2 \in \forall(\, t \,.\, \tau\,) \text{ iff } e_1[\,\sigma\,] \doteq e_2[\,\sigma\,] \in \{\sigma/t\}\tau,$$

which states that two expressions of polymorphic type are equal iff all of their type instances are equal. But *such a definition is circular*, because σ could be $\forall(\, t \,.\, \tau\,)$ itself! Thus, the definition of equality at type $\forall(\, t \,.\, t\,)$ would be given in terms of equality at type $\forall(\, t \,.\, t\,)$ itself, because the latter type is among the σ's on the right. This phenomenon is a manifestation of what is called *Girard's Paradox*, which is akin to Cantor's theorem stating that there is no "set of all sets."

The way around this is to use what are called *candidates for equality*. Rather than confine attention to type expressions σ in the definition of polymorphic equality, which leads to circularity, the definition is made sensible by *enlarging* the quantification to range over *binary relations* on types. The idea is that typed equality interprets types as binary relations, so simply enrich the meaning of quantification to range over all binary relations,[3] and this, surprisingly, avoids the paradox:

$$e_1 \doteq e_2 \in \forall(\, t \,.\, \tau\,) \text{ iff for all } \sigma \text{ and } \ R_\sigma,\ e_1[\,\sigma\,] \doteq e_2[\,\sigma\,] \in \tau \text{ (rel. } [R_\sigma\ /t]).$$

The circularity is broken by defining typed equality for the type variable t to be *any* binary relation R_σ on expressions of type σ. Noting that τ can have t free, this is expressed by the parenthetical stating that t-equality is to be interpreted by R_σ. That is, the definition does not "call itself" as in the circular form above, but rather refers to a broad range of relations that serve as "pseudo-equality" at the type t, among which will be equality itself.[4]

The extension of the Fundamental Theorem to **F** is a major result,[5] with profound consequences. For example, suppose that $e : \forall(\, t \,.\, t \to t\,)$ is a closed expression of Girard's **F**. One choice for e might be $\Lambda(\, t\,)\, \lambda\,(\, x : t\,)\, x$, the polymorphic identity function, which has the property that $e[\,\sigma\,](\, e_0\,) \doteq e_0 \in \sigma$ for any type σ and any $e_0 : \sigma$, because the left-hand side evaluates to the right. A remarkable consequence of parametricity is that this property is true *no matter what e is*! Moreover, this can be proved *without even looking at e*, just by knowing that its type is $\forall(\, t \,.\, t \to t\,)$. Why? Choose σ arbitrarily, and let e_0 be any expression of type σ. Define $e_1\ R_\sigma\ e_2$ to hold iff $e_1 \doteq e_0 \in \sigma$ and $e_2 \doteq e_0 \in \sigma$. By the definition of equality at polymorphic type just given, and the definition of equality at function types,

if $e_1\ R_\sigma\ e_2$, then $e[\,\sigma\,](\, e_0\,)\ R_\sigma\ e[\,\sigma\,](\, e_2\,)$.

[3] Subject to some mild technical conditions, including respect for evaluation.

[4] It is important to see how the circularity is evaded using candidates for equality.

[5] See Chapter 48 of **PFPL**.

Now obviously $e_0 \; R_\sigma \; e_0$, so it follows that $e[\,\sigma\,](\,e_0\,)$ is related to itself by R_σ. But, by the choice of R_σ, that means that $e[\,\sigma\,](\,e_0\,) \doteq e_0 \in \sigma$, as claimed.

As an exercise, show that if $e : \forall(\,t\,.\,t \to t \to t\,)$, then for all σ, $e_0 : \sigma$, and $e_1 : \sigma$, then either $e[\,\sigma\,](\,x_0\,)(\,x_1\,) \doteq x_0 \in \sigma$ or $e[\,\sigma\,](\,x_0\,)(\,x_1\,) \doteq x_1 \in \sigma$.

Even more surprisingly, the definition of equality at polymorphic type can be broadened still further, while still maintaining the fundamental theorem.

$$e_1 \doteq e_2 \in \forall(\,t\,.\,\tau\,) \text{ iff for all } \sigma_1, \sigma_2, \text{ and } \; R_{\sigma_1,\sigma_2}, \; e_1[\,\sigma_1\,] \doteq e_2[\,\sigma_2\,] \in \tau \text{ (rel. } [\mathcal{R}_{\sigma_1,\sigma_2}/t]).$$

That is, the type quantifier may be interpreted as ranging over *heterogeneous* binary relations R_{σ_1,σ_2} on (possibly) different types on the left and right side; such relations are often called *correspondences*, or *simulations*.

See **PFPL** Chapter 17 and Harper (2019) for worked examples of using correspondences to prove the equivalence of two different implementations of an abstract type.

References

Robert Harper. *Practical Foundations for Programming Languages*. Cambridge University Press, Cambridge, England, Second edition, 2016.

Robert Harper. Dynamic dispatch as an abstract type. Supplement to Harper (2016), Fall 2019. URL `https://www.cs.cmu.edu/~rwh/pfpl/supplements/ddadt.pdf`.

Robert Harper. Natural and co-natural numbers. Supplement to Harper (2016), September 2020. URL `https://www.cs.cmu.edu/~rwh/pfpl/supplements/natconat.pdf`.