1 Equirecursive Equality

In the equirecursive view of recursive types, types are regular labeled trees, possibly infinite. However, we still represent them by finite type expressions involving the fixpoint operator μ. There can be many type expressions representing the same type; for example, $\mu\alpha.\,\mathsf{unit} \to \alpha$ and $\mu\alpha.\,\mathsf{unit} \to \mathsf{unit} \to \alpha$. This raises the question: given two finite type expressions σ and τ, how do we tell whether they represent the same type?

In the isorecursive view, the finite type expressions σ and τ themselves are the types, and there are no infinite types. In this case, the question does not arise.

One might conjecture that two type expressions are equivalent (that is, represent the same type) iff they are provably so using ordinary equational logic with the unfolding rule $\mu\alpha.\,\tau = \tau\{\mu\alpha.\,\tau/\alpha\}$ and the usual laws of equality (reflexivity, symmetry, transitivity, congruence). But this would be incorrect. To see why, let us formulate the problem more carefully.

Suppose we have type expressions $\sigma, \tau, \ldots$ over variables $\alpha, \beta, \ldots$ defined by the grammar

$$\tau \ ::= \ \mathsf{unit} \ \mid \ \sigma \to \tau \ \mid \ \alpha \ \mid \ \mu\alpha.\,\tau,$$

where the τ in $\mu\alpha.\,\tau$ is not a variable. Let $[\![\sigma]\!]$ be the type denoted by σ. This is a possibly infinite regular labeled tree obtained from σ by "unfolding" all μ-subexpressions.

Write $\vdash \sigma = \tau$ if the equality of σ and τ can be proved from the following axioms and rules:

$$\mu\alpha.\,\tau = \tau\{\mu\alpha.\,\tau/\alpha\} \qquad\qquad \tau = \tau$$

$$\frac{\sigma = \tau}{\tau = \sigma} \qquad \frac{\sigma = \tau \quad \tau = \rho}{\sigma = \rho} \qquad \frac{\sigma_1 = \sigma_2 \quad \tau_1 = \tau_2}{\sigma_1 \to \tau_1 = \sigma_2 \to \tau_2}$$

These rules generate the smallest congruence relation on type expressions satisfying the unfolding rule $\mu\alpha.\,\tau = \tau\{\mu\alpha.\,\tau/\alpha\}$. One can show inductively that if $\vdash \sigma = \tau$, then $[\![\sigma]\!] = [\![\tau]\!]$, so the rules are sound. However, they are not complete. If we define

$$\tau_0 \ \stackrel{\triangle}{=}\ \mu\alpha.\,\mathsf{unit} \to \mathsf{unit} \to \alpha \qquad\qquad \tau_{n+1} \ \stackrel{\triangle}{=}\ \mathsf{unit} \to \tau_n, \quad n \geq 0, \tag{1}$$

then $\vdash \tau_{2m} = \tau_{2n}$ and $\vdash \tau_{2m+1} = \tau_{2n+1}$ for any m and n, but not $\vdash \tau_n = \tau_{n+1}$, whereas $[\![\tau_m]\!] = [\![\tau_n]\!]$ for all m and n.

2 A Dangerous Proof System

The following proof system is sound and complete for type equivalence, but great care must be taken, because the system is fragile in a sense to be explained. Judgements are sequents of the form $E \vdash \sigma = \tau$, where E is a set of type equations.

$$E,\, \sigma = \tau \vdash \sigma = \tau \qquad\qquad E \vdash \mathsf{unit} = \mathsf{unit}$$

$$\frac{E,\, \mu\alpha.\,\sigma = \tau \vdash \sigma\{\mu\alpha.\,\sigma/\alpha\} = \tau}{E \vdash \mu\alpha.\,\sigma = \tau} \qquad \frac{E \vdash \sigma = \tau}{E \vdash \tau = \sigma} \qquad \frac{E \vdash \sigma_1 = \sigma_2 \quad E \vdash \tau_1 = \tau_2}{E \vdash \sigma_1 \to \tau_1 = \sigma_2 \to \tau_2}$$

For example, here is a proof in this system of $\vdash \tau_0 = \tau_1$ as defined in (1):

$$\dfrac{\dfrac{\dfrac{\tau_0 = \mathit{unit} \to \tau_0 \vdash \mathsf{unit} = \mathsf{unit} \quad \tau_0 = \mathsf{unit} \to \tau_0 \vdash \tau_0 = \mathsf{unit} \to \tau_0}{\tau_0 = \mathit{unit} \to \tau_0 \vdash \mathsf{unit} \to \tau_0 = \mathsf{unit} \to \mathsf{unit} \to \tau_0}}{\tau_0 = \mathit{unit} \to \tau_0 \vdash \mathsf{unit} \to \mathsf{unit} \to \tau_0 = \mathsf{unit} \to \tau_0}}{\vdash \tau_0 = \mathsf{unit} \to \tau_0}$$

The rule for unfolding is quite unusual. Note that the very equation we are trying to prove in the conclusion appears as an assumption in the premise! This makes the system fragile. In fact, it breaks if we add a transitivity rule

$$\frac{E \vdash \sigma = \tau \quad E \vdash \tau = \rho}{E \vdash \sigma = \rho}.$$

On the surface, the transitivity rule seems quite harmless, and it seems like it could not hurt to add it to our system. However, with the addition of this rule, the system becomes unsound. Here is a proof of the false statement $\vdash \mathsf{unit} = \mathsf{unit} \to \mathsf{unit}$ (writing u for unit here to make the proof fit into the page):

$$\dfrac{\dfrac{\dfrac{\mu\alpha.\,\mathsf{u} = \mathsf{u} \vdash \mathsf{u} = \mathsf{u}}{\vdash \mu\alpha.\,\mathsf{u} = \mathsf{u}}}{\vdash \mathsf{u} = \mu\alpha.\,\mathsf{u}} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\mu\alpha.\,\mathsf{u} = \mathsf{u} \to \mathsf{u},\ \mu\alpha.\,\mathsf{u} = \mathsf{u} \vdash \mathsf{u} = \mathsf{u}}{\mu\alpha.\,\mathsf{u} = \mathsf{u} \to \mathsf{u} \vdash \mu\alpha.\,\mathsf{u} = \mathsf{u}}}{\mu\alpha.\,\mathsf{u} = \mathsf{u} \to \mathsf{u} \vdash \mathsf{u} = \mu\alpha.\,\mathsf{u}} \quad \mu\alpha.\,\mathsf{u} = \mathsf{u} \to \mathsf{u} \vdash \mu\alpha.\,\mathsf{u} = \mathsf{u} \to \mathsf{u}}{\mu\alpha.\,\mathsf{u} = \mathsf{u} \to \mathsf{u} \vdash \mathsf{u} = \mathsf{u} \to \mathsf{u}}}{\vdash \mu\alpha.\,\mathsf{u} = \mathsf{u} \to \mathsf{u}}}{\vdash \mathsf{u} = \mathsf{u} \to \mathsf{u}}$$

It is also essential that we have ruled out $\mu\alpha.\,\beta$ where β is a variable. Otherwise we would have

$$\frac{\mu\alpha.\,\alpha = \tau \vdash \mu\alpha.\,\alpha = \tau}{\vdash \mu\alpha.\,\alpha = \tau}$$

for any τ.

3 Types as Labeled Trees

A more revealing view of the proof system given above is the *coinductive* view, in which we try to find witnesses to the *inequivalence* of two types. The idea is that if $[\![\sigma]\!] \neq [\![\tau]\!]$, then there is a witness to that fact in the form of a common finite path from the roots of $[\![\sigma]\!]$ and $[\![\tau]\!]$ down to some point where the labels differ. Moreover, one can calculate a bound b on the length of such a witness if it exists. The bound is quadratic in the sizes of σ and τ. This gives an algorithm for checking equivalence: unfold the trees down to depth b, and search for a witness; if none is found, then none exists.

This algorithm is still exponential in the worst case. One can do better using an automata-theoretic approach. We build deterministic automata out of σ and τ and look for an input string on which they differ. This will give an algorithm whose worst-case running time is proportional to $|\sigma| \cdot |\tau|$.

Let $\{L, R\}^*$ be the set of finite-length strings over $\{L, R\}$ (L="left", R="right"). We model (possibly infinite) types as partial functions $T : \{L, R\}^* \rightharpoonup \{\mathsf{unit}, \to\}$ such that

- the domain of T is nonempty and prefix-closed (thus the empty string ε is always in the domain of T; this is called the *root*);
- if $T(x) = \to$, then both xL and xR are in $\mathsf{dom}\,T$;
- if $T(x) = \mathsf{unit}$, then neither xL nor xR is in $\mathsf{dom}\,T$; thus x is a *leaf.*

We restrict our attention to the constructors $\to$, unit; we could add more if we wanted to, but these suffice for the purpose of illustration.

A *path* in T is a maximal subset of $\mathsf{dom}\,T$ linearly ordered by the prefix relation. Paths can be finite or infinite. A finite path ends in a leaf x, thus $T(x) = \mathsf{unit}$ and $T(y) = \to$ for all proper prefixes y of x. An infinite path has $T(x) = \to$ for all elements x along the path.

Let T be a type and $x \in \{L,R\}^*$. Define the partial function $T_x : \{L,R\}^* \rightharpoonup \{\mathsf{unit}, \to\}$ by

$$T_x(y) \ \stackrel{\triangle}{=}\ T(xy).$$

If T_x has nonempty domain, then it is a type. Intuitively, it is the subexpression of T at position x.

A type T is *finite* if its domain $\mathsf{dom}\,T$ is a finite set. By König's lemma, a type is finite iff it has no infinite paths. A type T is *regular* if $\{T_x \mid x \in \{L,R\}^*\}$ is a finite set.

4 Term Automata

Types can be represented by a special class of automata called *term automata*. These can be defined over any signature, but for our application, we consider only term automata over $\{\to, \mathsf{unit}\}$. A term automaton over this signature consists of

- a set of *states* Q;
- a *start state* $s \in Q$;
- a partial function $\delta : \{L,R\} \to Q \rightharpoonup Q$ called the *transition function*; and
- a (total) *labeling function* $\ell : Q \to \{\to, \mathsf{unit}\}$,

such that for any state $q \in Q$,

- if $\ell(q) = \to$, then both $\delta(L)(q)$ and $\delta(R)(q)$ are defined; and
- if $\ell(q) = \mathsf{unit}$, then both $\delta(L)(q)$ and $\delta(R)(q)$ are undefined.

The partial function δ extends naturally to a partial function $\widehat{\delta} : \{L,R\}^* \to Q \rightharpoonup Q$ inductively as follows:

$$\widehat{\delta}(\varepsilon)(q) \ \stackrel{\triangle}{=}\ q \qquad\qquad \widehat{\delta}(xa)(q) \ \stackrel{\triangle}{=}\ \delta(a)(\widehat{\delta}(x)(q)).$$

In other words,

$$\widehat{\delta}(\varepsilon) \ \stackrel{\triangle}{=}\ \mathsf{id}_Q \qquad\qquad \widehat{\delta}(xa) \ \stackrel{\triangle}{=}\ \delta(a) \circ \widehat{\delta}(x).$$

It follows by induction on the length of y that

$$\widehat{\delta}(xy) \ =\ \widehat{\delta}(y) \circ \widehat{\delta}(x).$$

In other words, $\widehat{\delta}$ is the unique monoid homomorphism extending δ from the free monoid $\{L,R\}^*$ on generators $\{L,R\}$ to the monoid of partial functions $Q \rightharpoonup Q$ under composition.

For any $q \in Q$, the domain of the partial function $\lambda x.\,\widehat{\delta}(x)(q)$ is nonempty (it always contains ε) and prefix-closed. Moreover, the partial function $\lambda x.\,\ell(\widehat{\delta}(x)(q))$ is a type. The type *represented by* M is the type

$$[\![M]\!] \ \stackrel{\triangle}{=}\ \lambda x.\,\ell(\widehat{\delta}(x)(s)),$$

where s is the start state.

Intuitively, $[\![M]\!](x)$ is determined by starting in the start state s and scanning the input x, following transitions of M as far as possible. If it is not possible to scan all of x because some transition along the way does not exist, then $[\![M]\!](x)$ is undefined. If on the other hand M scans the entire input x and ends up in state q, then $[\![M]\!](x) = \ell(q)$.

One can show that a type T is regular iff $T = [\![M]\!]$ for some term automaton M with finitely many states. This is also equivalent to being $[\![\tau]\!]$ for some finite type expression τ. To construct a term automaton M_τ from a closed finite type expression τ, take the set of states of M_τ to be the smallest set Q such that

- $\tau \in Q$;
- if $\sigma \to \rho \in Q$, then $\sigma \in Q$ and $\rho \in Q$; and
- if $\mu\alpha.\,\sigma \in Q$, then $\sigma\{\mu\alpha.\,\sigma/\alpha\} \in Q$.

The set Q so defined is finite. The start state is τ. The transition function is given by the following rules:

- $\delta(L)(\sigma \to \rho) \stackrel{\triangle}{=} \sigma$;
- $\delta(R)(\sigma \to \rho) \stackrel{\triangle}{=} \rho$;
- $\delta(D)(\mathsf{unit})$ is undefined, $D \in \{L, R\}$;
- $\delta(D)(\mu\alpha.\,\sigma) \stackrel{\triangle}{=} \delta(D)(\sigma\{\mu\alpha.\,\sigma/\alpha\})$, $D \in \{L, R\}$.

(The restriction that $\sigma \neq \alpha$ is crucial here.) The labeling function is given by:

- $\ell(\sigma \to \rho) \stackrel{\triangle}{=} \to$
- $\ell(\mathsf{unit}) \stackrel{\triangle}{=} \mathsf{unit}$
- $\ell(\mu\alpha.\,\sigma) \stackrel{\triangle}{=} \ell(\sigma\{\mu\alpha.\,\sigma/\alpha\})$.

Then $[\![\tau]\!] = [\![M_\tau]\!]$.

Mathematically speaking, term automata are exactly the coalgebras of signature $\{\to, \mathsf{unit}\}$ over the category of sets. The map $M \mapsto [\![M]\!]$ is the unique morphism from the coalgebra M to the final coalgebra, which consists of the finite and infinite types.

5 A Coinductive Algorithm for Type Equivalence

Now given pair σ, τ of finite type expressions, $[\![\sigma]\!] = [\![\tau]\!]$ iff for all $x \in \{L, R\}^*$, $[\![\sigma]\!](x) = [\![\tau]\!](x)$; equivalently, $[\![\sigma]\!] \neq [\![\tau]\!]$ iff there exists $x \in \mathsf{dom}\,[\![\sigma]\!] \cap \mathsf{dom}\,[\![\tau]\!]$ such that $[\![\sigma]\!](x) \neq [\![\tau]\!](x)$. Form the two term automata $M_\sigma = (Q_\sigma,\, \delta_\sigma,\, \ell_\sigma,\, s_\sigma)$ and $M_\tau = (Q_\tau,\, \delta_\tau,\, \ell_\tau,\, s_\tau)$. Then form the product automaton $M_\sigma \times M_\tau$ with states $Q_\sigma \times Q_\tau$, transition function $\lambda D.\,\lambda(p,q).\,(\delta_\sigma(D)(p), \delta_\tau(D)(q))$, start state (s_σ, s_τ), and labeling function $\lambda(p,q).\,(\ell_\sigma(p), \ell_\tau(q))$. The product automaton runs the two automata M_σ and M_τ in parallel on the same input data. Then $[\![M_\sigma]\!] \neq [\![M_\tau]\!]$ iff there exists an input string $x \in \{L, R\}^*$ that causes the product automaton to move from its start state to a state (p, q) such that $\ell_\sigma(p) \neq \ell_\tau(q)$. This can be determined by depth-first search in time linear in $|M_\sigma \times M_\tau|$, which is roughly $|M_\sigma|\cdot|M_\tau|$. This give a quadratic algorithm for testing type equivalence.

We will sketch a proof of soundness and completeness for the proof system of Section 2 next time.