Normalization By Evaluation

1 An Starter Example

Normalization by evaluation is a technique for studying the normal forms of rewrite systems. The idea that terms often contain too much data. Consider the "free monoid on $\{x, y, z\}$". That is, $\mathcal{M} = \{x, y, z, \epsilon\}*$ Now, we can define some rewrite rules so

1. $(e_1 \cdot e_2) \cdot e_3 \to e_1 \cdot (e_2 \cdot e_3)$
2. $e \cdot \epsilon \to e$
3. $\epsilon \cdot e \to e$

Then a term with no empties, and fully right-associated is in normal form. So insight is that all that really matters is the order of non-empty terms. All the parenthesis and ϵs are extra data. So we magic-up a semantics which preserves exactly the needed info. Turns out lists suffice! To build our semantic domain, we first need a way of translating the variables. So let's define a type $T = X \mid Y \mid Z$. So our semantic domain is $\mathcal{L} = [T]$. Our goal is to turn $\mathcal{M}$ terms into $\mathcal{L}$ terms, do the compute there, and then return to $\mathcal{M}$.

The process of turning syntactic things into semantic things is called reflection.

1. $f\,\varepsilon = []$
2. $f(e_1 \cdot e_2) = f\,e_1 \mathbin{++} f\,e_2$
3. $f\,x = X \quad f\,y = Y \quad f\,z = Z$

And the reverse if reification.

1. $g\,[] = \varepsilon$
2. $g(X : tl) = x \cdot g(tl)$
3. $g(Y : tl) = y \cdot g(tl)$
4. $g(Z : tl) = z \cdot g(tl)$

The intention is to reflect then reify to evaluate things.

$$g(f((x \cdot x) \cdot (\varepsilon \cdot y \cdot z))) = g([X, X, Y, Z]) = x \cdot x \cdot y \cdot z \cdot \varepsilon$$

Now we get a trailing ε due to a quirk of how we defined g. But that doesn't change the good news:

1. Completeness: if e_1 and e_2 are equal in the rewrite system sense (E.G. in the same component of the rewrite graph), $g(f(e_1)) = g(f(e_2))$
2. Soundness: forall e, e and $g(f(e))$ are equal in the rewrite system.

Also critically, f and g terminate. So everything has a normal form, computable in this way (with a minor patch to g to fix that trailing ε). So the system is strongly normalizing! The proofs are, unsurprisingly, by induction. Observe that f respects the rewrite relations

$$f(e \cdot \varepsilon) = f(e) + +[] = f(e) \qquad f(\varepsilon \cdot e) = [] + +f(e) = f(e)$$

and

$$f((e_1 \cdot e_2) \cdot e_3) = (f(e_1) {+\!+} f(e_2)) {+\!+} f(e_3) = f(e_1) {+\!+} (f(e_2) {+\!+} f(e_3)) = f(e_1 \cdot (e_2 \cdot e_3))$$

So completness follows by induction on the length of the path from e_1 to e_2. and soundness follows by induction on the structure of e.

2 NBE for System F

The crux of the correctness of this approach is that the rewrite rules turn into actual equalities in the semantics. That is,

$$(x \cdot y) \cdot z \to x \cdot (y \cdot z) \quad \text{but} \quad (x +\!\!+ y) +\!\!+ z = x +\!\!+ (y +\!\!+ z)$$

So of course all of the terms related by a rewrite turn into literally equal things. Can we replicate that in System F? Short answer is no. We will need to some some much more clever things. Let's rule out some ideas that definitely don't work.

The obvious induction fails somewhat catastrophically. The following is quite false

$$e_1 \,\text{and}\, e_2 \,\text{strongly normalizing} \Rightarrow e_1\, e_2 \,\text{strongly normalizing}$$

In fact, consider $e_1 = e_2 = \lambda x.x\,x$. This term is strongly normalizing, but $e_1\, e_2$ is not. So this approach is a bit hopeless. Instead our plan will be to do this in two phases. First we will interpret types into relations built up by the types of terms. That is $[\![A]\!]_\rho \subset S \times S$ where S is the strongly normalizing terms, and ρ is some interpretation of open variables. Secondly we will want to prove that for each $\Gamma \vdash t : A$, $(t, t) \in [\![A]\!]_\rho$ for some ρ that is compatible with Γ in some way.

Let's define some stuff optimistically, and see how far we can go. The plan is to interpret each type into a (nearly) equivalence relation that intuitively means

$$(e_1, e_2) \in [\![A]\!] \leftrightarrow \Gamma \vdash e_1 = e_2 : A \quad \text{roughly} \quad \Gamma \vdash e_1 : A \,\wedge\, \Gamma \vdash e_2 : A \,\wedge\, \Gamma \vdash e_1 =_{\text{ish}} e_2$$

Of course we aren't gonna define that $=_{\text{ish}}$ here. But we can handle the extra context that appears

$$[\![A]\!] : \mathcal{P}(Term \times Term) \quad \text{becomes} \quad [\![A]\!] : Cxt \to \mathcal{P}(Term \times Term)$$

But but we need a little extra something that says it respects contexts with more/lesss information. So if we a use context with less specific types (like using $\forall B.\, A \to B$ instead of the more general $A \to A$) we never break equalities.

$$\Gamma' \leq \Gamma \to [\![A]\!]_\Gamma(e_1, e_2) \to [\![A]\!]_{\Gamma'}(e_1, e_2)$$

So we will write with the notation

$$\Gamma \vdash e_1 \sim e_2 \in \mathcal{A} := (e_1, e_2) \in \mathcal{A}_\Gamma$$

3 Building these relations

Our types are defined recursively, so we'll need to define this interpretation recursively as well. Arrow types are relatively easy. Given relations $\mathcal{A}, \mathcal{B}$, two terms f, f', and a context Γ we say they send related terms to related terms when

$$\forall d\, d'\, \Gamma' \leq \Gamma,\, \Gamma' \vdash d \sim d' \in \mathcal{A} \text{ implies } \Gamma' \vdash f\, d \sim f\, d' \in \mathcal{B}$$

So we can define an arrow between relations as

$$\mathcal{A} \to_R \mathcal{B} := \Gamma \mapsto \{(f, f') \in Term \times Term \mid (f, f') \text{ send } \mathcal{A} \text{ related terms to } \mathcal{B} \text{ related terms}\}$$

3.1 Attempt 1

So we can make a very first attempt to define an interpretation. Given a ρ which interprets free type variables are relations,

1. $[\![X]\!]_\rho(X) = \rho(X)$
2. $[\![A \to B]\!]_\rho := [\![A]\!]_\rho \to_R [\![B]\!]_\rho$
3. $[\![\forall X B]\!]_\rho := \bigcap_A [\![B[X := A]]\!]_\rho$

Now, there are some things we need to show. Firstly

For any type A, if $\rho(X)$ is an equivalence relation for all X, then so is $[\![A]\!]_\rho$

The other thing we want is

$$[\![A[X := B]]\!]_\rho = [\![A]\!]_{\rho+[X:=[\![B]\!]_\rho}$$

This is going to give us significant trouble. We induct on the structure of the type, and the type variable case and the arrow case are easy. For the type abstraction case, we've got problems. The induction hypothesis just doesn't apply to $[\![B[X := A]]\!]_\rho$. This is the same problem as before, we really can't induct against these substitutions.

3.2 Attempt 2

Instead, let's define a new relation-level operator like we did for arrow, and see if that helps. Given a type $\forall X, B$, we can imagine it's built up of slices for each substitution $X := A$. So we can build a relation for each slice which says that terms are related upon type application of A

$$slice_A(\mathcal{B}) = \Gamma \mapsto \{(e_1, e_2) \in Term \times Term \mid \Gamma \vdash e_1 \cdot A \sim e_2 \cdot A \in \mathcal{B}\}$$

(Note the paper uses $(A.\mathcal{B})^{\forall X,B}$ for this). Now we can try again, updating the substitution instead.

1. $[\![X]\!]_\rho(X) = \rho(X)$
2. $[\![A \to B]\!]_\rho := [\![A]\!]_\rho \to_R [\![B]\!]_\rho$
3. $[\![\forall X B]\!]_\rho := \bigcap_A slice_A([\![B]\!]_{\rho+[X:=[\![A]\!]_\rho]})$

Ok, so we're on the right track. But now the ρ is changing, so our induction hypothesis are gonna get weird. Even for the simple attempt at the equivalence relation proof, The induction hypothesis will state

forall ρ, if forall X, $\rho(X)$ is an equivalence relation, then so is $[\![B]\!]_\rho$

Then we would like to just finish with $slice_A$ takes equivalence relations to equivalence relations, and intersections work. However, we need to show that $\rho+[X := [\![A]\!]_\rho]$ sends X to an equivalence relation for all A. But that includes $A := \forall X B$. But that's what we're trying to prove. So we're stuck!

So proving our intended goal that $[\![A]\!] \subset S \times S$ has no prayer. More-or-less, we need to get $[\![A]\!]$ out of the RHS entirely.

3.3 Attempt 3

A clever idea is intersecting over relations instead of types. But we still need the type to take the slice. But we need some way to connect the type and the relation. Some kind of "is built from this type" indiciator. We'll call it $\Vdash$.

1. $[\![X]\!]_\rho(X) = \rho(X)$
2. $[\![A \to B]\!]_\rho := [\![A]\!]_\rho \to_R [\![B]\!]_\rho$
3. $[\![\forall X B]\!]_\rho := \bigcap_{A \Vdash \mathcal{A}} slice_A([\![B]\!]_{\rho+[X:=\mathcal{A}]})$

Now good things are happening. with the right assumptions on $\Vdash$, the inductions will go through. In particular, we want it to preserve typing information, so

$$A \Vdash \mathcal{A} \ \wedge \ B \Vdash \mathcal{V} \Rightarrow A \to B \Vdash \mathcal{A} \to_R \mathcal{B}$$

and for type abstractions, if for some function F we have

$$A \Vdash \mathcal{A} \Rightarrow B[X := A] \Vdash F(\mathcal{A})$$

then

$$\forall X B \Vdash \bigcap_{A \Vdash \mathcal{A}} slice_A(F(\mathcal{A}))$$

The theory ends up with a slightly more general description of valid $\Vdash$ relations called realizability. From here we get the core lemma that when σ is a type substitution $[X1 := A_1, ..., X_n := A_n]$ and ρ sends $\rho(X_i) \Vdash [\![A_i]\!]_\rho$, then $A\sigma \Vdash [\![A]\!]_\rho$

The final step of induction to prove that

$$\Gamma \vdash e : A \Rightarrow \Gamma \vdash e \sim e \in [\![A]\!]_\rho$$

is pretty strightforward, although you'll have to work through a small lie I've told about $\mathcal{A}$ always taking values in $Term$ instead of, perhaps terms quotiented by a beta reduction. Then the normalization by evaluation technique involves two more nuances. First, using η-long β-normal forms. This is just what happens when you extend your rewrite system to include η, and see what normalization does. Second, you need to pick a more interesting domain for $\mathcal{A}$ that is, more-or-less values, types, and functions between them.

We have mostly followed https://www.cse.chalmers.se/ abela/lpar08.pdf who has a nice, detailed presentation of the construction for System F's semantics. He covers it in much more detail than I do