A SIMPLE PROOF TECHNIQUE FOR CERTAIN PARAMETRICITY RESULTS

1. Introduction

During the course we have seen syntactic methods to prove results about type systems. We have also studied logical relations, a more "semantic" (mathematical) method to obtain advanced results in presence of polymorphism, including relational properties.

In this exam topic, you will study a research paper published in 1999 by Karl Crary[1], which offers a simple alternative to logical relations to prove a subset of logical-relation-like results:

A Simple Proof Technique for Certain Parametricity Results, Karl Crary, ICFP 1999

(We do not, of course, expect you to know about this paper or be able to read it during the exam, so we will describe it and phrase our questions in a self-contained way.)

1.1. Context

Karl Crary was working at the time on a *typed assembly language*, a low-level language representing machine instructions, but with a strong, precise type system. In his language, $\{\texttt{r1} : \tau\}$ is the type of the address of a code segment that can execute correctly if the machine register `r1` holds a value of type τ.

Assembly programs are written in a sort of continuation-passing style: when you call a function, you are expected to provide it with a *return address* in a specific register `ra`, and the function will "return" by *jumping* to this return address. For example, a function that expects an integer and returns a boolean may be represented by a code address of type $\{\texttt{r1} : \texttt{int}, \texttt{ra} : \{\texttt{r1} : \texttt{bool}\}\}$: the caller must provide a value of type `int` in register `r1`, and a return address in register `ra` that is itself an address that will be executed with a value of type `bool` in `r1`.

Now consider a function `f` of type

$$\forall \rho,\ \{\texttt{r1} : \tau_1, \texttt{sp} : \rho, \texttt{ra} : \{\texttt{r1} : \tau_2, \texttt{sp} : \rho\}\}$$

This function expects an argument of type τ_1 in `r1`, and jump to its `ra` return address with an argument of type τ_2 in `r1`. It also expects an "input value" of type ρ in the `sp` (stack pointer) register and provides an "output value" $\texttt{sp} : \rho$ to the return code at `ra`. Karl Crary remarked that:

1. The type of `f` says that the output value of `sp` is at the same type ρ as the input value of `sp`. But if `f` is really polymorphic over ρ, then the output value should in fact be **the same** as the input value: there is no other way to build a value of type ρ.
2. To formally prove this, the standard approach would be to define a logical relation for this typed assembly language, and prove the fundamental lemma. But the language is quite a bit larger than just System F: maybe there are dozens of assembly instructions, non-termination, mutable state, etc. In 1998, no one had managed to define a (step-indexed) logical relation for a language of this complexity[2].

Instead of trying to prove a parametricity result using logical relations for this scary language, Karl Crary proposed a clever trick: introduce more types in the syntax of the type system to represent some limited semantic information, and derive some parametricity results by simply instantiating polymorphic types with those extra types.

1.2. The trick

Karl Crary proposed to extend the type system with **singleton types**. A singleton type $S(v : \tau)$ is the type of all values of type τ that are equivalent to v. For example we have $3 : \texttt{int}$ and also $3 : S(3 : \texttt{int})$, but 2 does not have type $S(3 : \texttt{int})$.

The trick is to use those singleton types as intermediary steps to prove parametricity lemmas. For example, consider a polymorphic function $f : \forall \alpha.\ \alpha \to \alpha$. We want to prove that for any closed value $v : \tau$, the function application $f\ [\tau]\ v$ is equivalent to v. Informally, the proof goes as follows:

$f\ [\tau]\ v$

is the same, modulo type annotations (which do not influence computation), as

$f\ [S(v : \tau)]\ v$

which has type $S(v : \tau)$, therefore is equivalent to

v

In this exam topic, we will turn the above hand-waving argument into a precise formal argument.

[1] Karl Crary is a programming-language (PL) researcher at Carnegie Mellon University, one of the strong PL research groups in the USA.

[2] Defining step-indexed logical relations to prove parametricity for a typed assembly language has been done a while later in "Semantic foundations for typed assembly languages" by Ahmed, Appel, Richards, Swadi, Tan and Wang, 2010.

2. System F, Augmented System F

We introduce the usual rules for System F, extended with an introduction form for singleton types $S(v : \tau)$. We will write the terms of System F *without* type annotations – the type annotations can be reconstructed from the typing derivation.

$$\begin{array}{rll} \text{types} & \tau ::= \alpha \mid \tau_1 \to \tau_2 \mid \forall \alpha.\, \tau \mid S(v : \tau) & \text{evaluation contexts } E ::= \square \mid E\, u \mid t\, E \\ \text{terms} & t, u ::= x \mid \lambda x.\, t \mid t\, u & \text{typing environments } \Gamma ::= \emptyset \mid \Gamma, \alpha \mid \Gamma, x : \tau \\ \text{(open) values} & v, w ::= x \mid \lambda x.\, t & \text{type-only typing environments } \Delta ::= \emptyset \mid \Delta, \alpha \end{array}$$

$$\frac{\Gamma, x : \tau \vdash t : \tau'}{\Gamma \vdash \lambda x.\, t : \tau \to \tau'}\ \textsc{Lam} \qquad \frac{\Gamma \vdash t : \tau \to \tau' \qquad \Gamma \vdash u : \tau}{\Gamma \vdash t\, u : \tau'}\ \textsc{App} \qquad \frac{x : \tau \in \Gamma}{\Gamma \vdash x : \tau}\ \textsc{Var}$$

$$\frac{\Gamma, \alpha \vdash t : \tau}{\Gamma \vdash t : \forall \alpha.\, \tau}\ \textsc{Forall} \qquad \frac{\Gamma \vdash t : \forall \alpha.\, \tau}{\Gamma \vdash t : \tau[\alpha := \tau']}\ \textsc{Instance} \qquad \frac{\Gamma \vdash v : \tau}{\Gamma \vdash v : S(v : \tau)}\ \textsc{Singleton}$$

$$\frac{}{(\lambda x.\, t)\, v \rightsquigarrow t[x := v]} \qquad \frac{t \rightsquigarrow t'}{E[t] \rightsquigarrow E[t']}$$

Notice that the rule Instance has $t : \forall \alpha.\, \tau$ as its premise, but also $t : \tau[\alpha := \tau']$ in its conclusion, instead of $t\, [\tau'] : \tau[\alpha := \tau']$. Indeed, the terms are completely untyped. We call this presentation with untyped terms a *Curry-style* or *extrinsically-typed* system, and the other presentation with typed terms a *Church-style* or *intrisically-typed* system.[3] There is a one-to-one correspondence between typing derivations in both systems, so nothing is gained or lost.

You can still write $t\, [\tau']$ in your terms if you wish, as a type annotation. Sometimes this improves readability.

The type system described above is "System F with singleton types": we added the rule Singleton for singleton types $S(v : \tau)$, which can only be applied to the value v itself. Note that v may be a variable.

We are going to *admit* the following lemmas, which are tedious to prove. (In general, Lemmas and Theorems in this topic are admitted, you only have to work on the Questions.)

Lemma 2.1 (Value substitution in reductions).
If $t \rightsquigarrow u$ then $t[x := v] \rightsquigarrow u[x := v]$.

Lemma 2.2 (Type substitution).
If $\Gamma, \alpha \vdash t : \tau'$ then $\Gamma \vdash t : \tau'[\alpha := \tau]$.

Lemma 2.3 (Value substitution).
If $\Gamma, x : \tau \vdash t : \tau'$ and $\Gamma \vdash u : \tau$ then $\Gamma \vdash t[x := u] : \tau'[x := u]$.

Remark that singleton types $S(v : \tau)$ have terms (values) occurring in types, so in Lemma 2.3 (Value substitution) we had to substitute $[x := u]$ also in the result type τ'.

As a standard consequence of these results, we have subject reduction:

Theorem 2.4 (Subject reduction).
If $\Gamma \vdash t : \tau$ and $t \rightsquigarrow t'$ then $\Gamma \vdash t' : \tau$.

Question 1:
Write a closed derivation of the judgment $\emptyset \vdash \lambda x.\, x : \forall \alpha.\, \alpha \to \alpha$.

2.1. Reasoning about closed values

In this (sub)section we prove some results about derivations of the form $\emptyset \vdash v : \tau$, *i.e.* typing derivations for values that are closed — have no free variables.

One difficulty of Curry-style presentations is that they make reasoning by inversion on the typing derivation harder. In a Church-style presentation, if you see a derivation of the form $\emptyset \vdash v : \tau_1 \to \tau_2$, then you can easily deduce that v must be of the form $\lambda x.\, u$ – Lam is the only applicable typing rule for a closed value at this type. But this is not true in a Curry-style system, the root of a derivation can *always* be the rule Instance.

To reason rigorously about derivations of the form $\emptyset \vdash v : \tau$, we will introduce a *restricted* value judgment $\Delta \vdash_{\mathrm{r}} v : \tau$ which does not allow the rule Instance to appear immediately below the rule Forall.

$$\frac{\Delta, x : \tau \vdash t : \tau'}{\Delta \vdash_{\mathrm{r}} \lambda x.\, t : \tau \to \tau'}\ \text{R-}\textsc{Lam} \qquad \frac{\Delta \vdash v : \tau}{\Delta \vdash_{\mathrm{r}} v : S(v : \tau)}\ \text{R-}\textsc{Singleton} \qquad \frac{\Delta, \alpha \vdash_{\mathrm{r}} v : \tau}{\Delta \vdash_{\mathrm{r}} v : \forall \alpha.\, \tau}\ \text{R-}\textsc{Forall}$$

The restricted system uses a strict subset of the rules of the full system, so it is immediate that if $\Delta \vdash_{\mathrm{r}} v : \tau$ holds then $\Delta \vdash v : \tau$ also holds. Note that we use contexts of the form Δ which only contain type variables α, not term variables $x : \tau$. Indeed, our grammar of (open) values includes term variables x, so in a general context we would need a variable rule for completeness.

Here is an example of an important result that is easy to prove about (restricted) derivations of $\emptyset \vdash_{\mathrm{r}} v : \tau$, and sensibly harder to prove about (non-restricted) derivations of $\emptyset \vdash v : \tau$.

[3] Named after influential mathematicians Haskell Curry (1900-1982) and Alonzo Church (1903-1995).

Question 2 (Canonical singleton forms – restricted):
Show that, if $\emptyset \vdash_r v : S(v' : \tau)$, then $v = v'$.

Proof. By inversion: the only restricted rule applicable on a goal of the form $S(v' : \tau)$ is R-Singleton, so we have $v = v'$. □

Lemma 2.1.5 (Restricted type substitution).
If $\Delta, \alpha \vdash_r v : \tau'$, then $\Delta \vdash_r v : \tau'[\alpha := \tau]$.

Question 3 (Restricted instantiation):
Show that if $\Delta \vdash_r v : \forall\alpha.\,\tau$ is derivable, then $\Delta \vdash_r v : \tau[\alpha := \tau']$ is also derivable.

Proof. A derivation of the judgment $\Delta \vdash_r v : \forall\alpha.\,\tau$ necessarily has the following rule as its root:

$$\frac{\Delta, \alpha \vdash_r v : \tau}{\Delta \vdash_r v : \forall\alpha.\,\tau}$$

so we have $\Delta, \alpha \vdash_r v : \tau$, and so by Lemma 2.1.5 (Restricted type substitution) we have $\Delta \vdash_r v : \tau[\alpha := \tau']$. □

Question 4 (Unrestriction):
Show that if $\Delta \vdash v : \tau$ holds, then $\Delta \vdash_r v : \tau$ holds.

Proof. The proof is by induction on the typing derivation. It is immediate in the case of the rules Lam, Singleton. The rule Var cannot occur as Δ does not contain term variables, and the rule App cannot occur as we have a value v rather than an arbitrary term. The only remaining rules are Forall and Instance:

$$\frac{\Delta, \alpha \vdash v : \tau}{\Delta \vdash v : \forall\alpha.\,\tau} \qquad \frac{\Delta \vdash v : \forall\alpha.\,\tau}{\Delta \vdash v : \tau[\alpha := \tau']}$$

For Forall, we use our induction hypothesis to derive $\Delta, \alpha \vdash_r v : \tau$, and conclude by applying R-Forall. For Instance we use our induction hypothesis to derive $\Delta \vdash_r v : \forall\alpha.\,\tau$, and then Question 3 (Restricted instantiation) to derive $\Delta \vdash_r v : \tau[\alpha := \tau']$. □

Corollary 2.1.6 (Restricted inversion).
If the judgment $\Delta \vdash v : \tau$ is provable, then it admits a derivation whose root is not the rule Instance.

Question 5 (Canonical singleton forms):
Show that if $\emptyset \vdash v : S(v' : \tau)$, then $v = v'$.

Proof. There are two slightly different ways to prove this result.

1. We can use our lemma Question 2 (Canonical singleton forms – restricted). This requires a (restricted) derivation of $\emptyset \vdash_r v : S(v' : \tau)$, which we get from our hypothesis by Question 4 (Unrestriction).
2. We can do a direct proof by inversion with help from Corollary 2.1.6 (Restricted inversion), which is closer to how other results of this form will be proved in the rest of the document. The rule at the root of a judgment of the form $\emptyset \vdash v : S(v' : \tau)$ cannot be Var or App (they cannot be used on closed values), cannot be Lam or Forall (they produce incompatible types), and it cannot be Instance by Corollary 2.1.6 (Restricted inversion), therefore it must be Singleton and the result is immediate.

□

3. Your first syntactic parametricity results

Question 6 (Identity):
Show that, if $\emptyset \vdash f : \forall\alpha.\,\alpha \to \alpha$ and $\emptyset \vdash v : \tau$ and $f\ v \rightsquigarrow^* v'$, then $v' = v$.

Proof. By instantiating $f : \forall\alpha.\,\alpha \to \alpha$ at type $\alpha := \tau$ we have $f : \tau \to \tau$, and we also have $\emptyset \vdash v : \tau$ so we have $\emptyset \vdash f\ v : \tau$. By instantiating it at type $\alpha := S(v : \tau)$ we have $\emptyset \vdash f : S(v : \tau) \to S(v : \tau)$, and we also have $\emptyset \vdash v : S(v : \tau)$ so we have $\emptyset \vdash f\ v : S(v : \tau)$. By subject-reduction we have that $\emptyset \vdash v' : S(v : \tau)$. Thus, by Question 5 (Canonical singleton forms), we have $v' = v$. □

Question 7 (Swap):
For this question, let us assume that we add product types $\tau_1 \times \tau_2$ with terms (t_1, t_2), values (v_1, v_2), evaluation contexts (E, t) and (t, E), and the usual typing rule for products.

Show that, if $\emptyset \vdash f : \forall\alpha\beta.\,\alpha \times \beta \to \beta \times \alpha$ and $\emptyset \vdash v_1 : \tau_1$ and $\emptyset \vdash v_2 : \tau_2$ and $f\ (v_1, v_2) \rightsquigarrow^* v'$, then $v' = (v_2, v_1)$.

Proof.
We first remark a canonicity result for product types: if $\emptyset \vdash v : \tau_1 \times \tau_2$, then $v = v_1 \times v_2$ for some v_1, v_2 such that $\emptyset \vdash v_1 : \tau_1$ and $\emptyset \vdash v_2 : \tau_2$.

By instantiating f at types $\alpha := S(v_1 : \tau_1)$ and $\beta := S(v_2 : \tau_2)$ we have $\emptyset \vdash f\ (v_1, v_2) : S(v_2 : \tau_2) \times S(v_1 : \tau_1)$. By canonicity of product types, v' must be of the form (v_2', v_1') with $v_2' : S(v_2 : \tau_2)$ and $v_1' : S(v_1 : \tau_1)$. By Question 5 (Canonical singleton forms) we have $v_1' = v_1$ and $v_2' = v_2$, so $v' = (v_2, v_1)$. □

3.1. Multiplicate results

With singleton types alone, we cannot prove that a closed function of type $\forall\alpha.\, \alpha \to \alpha \to \alpha$ must return either its first or its second argument. Karl Crary proposes to further extend his system with **union types** $\tau_1 \vee \tau_2$:

$$\frac{\Gamma \vdash t : \tau_1}{\Gamma \vdash t : \tau_1 \vee \tau_2}\ \textsc{Union1} \qquad \frac{\Gamma \vdash t : \tau_2}{\Gamma \vdash t : \tau_1 \vee \tau_2}\ \textsc{Union2}$$

Remark: with these new typing rules, the result of Corollary 2.1.6 (Restricted inversion) still holds. Reproving it is easy, if we extend the restricted system with typing rules for union. We propose to just admit this slightly extended result.

Question 8 (Canonical union forms):
State and prove a canonicity result for union types, in the spirit of Question 5 (Canonical singleton forms).

Proof. We prove that if $\emptyset \vdash v : \tau_1 \vee \tau_2$, then for some $i \in \{1, 2\}$ we have $\emptyset \vdash v : \tau_i$.

The proof is immediate by inversion on typing derivations for values: only the introduction rules for union types can be used in an empty context, all other introduction rules are for non-value terms, non-closed terms, or produce incompatible types. □

To check that your statement is reasonable, it must let you formally prove the following.

Question 9 (Boolean):
Show that, if $\emptyset \vdash f : \forall\alpha.\, \alpha \to \alpha \to \alpha$ and $\emptyset \vdash v_1 : \tau$ and $\emptyset \vdash v_2 : \tau$ and $f\ v_1\ v_2 \rightsquigarrow^* v'$, then either $v' = v_1$ or $v' = v_2$.

Proof. We instantiate f at type $\alpha \coloneqq S(v_1 : \tau) \vee S(v_2 : \tau)$; by subject reduction we have $\emptyset \vdash v' : S(v_1 : \tau) \vee S(v_2 : \tau)$. By Question 8 (Canonical union forms) and then Question 5 (Canonical singleton forms), we conclude that $v' = v_1$ or $v' = v_2$. □

3.2. Conclusion

That's it! In the rest of his research paper, Carl Krary moves to apply the same technique to prove results on his typed assembly language. We are not going to make you work with a typed assembly language here, and instead spend the rest of the assignment on further aspects of this approach for System F.

4. Free theorems under context

The parametricity results that you have established so far are only formulated in an empty context. Let us try to establish corresponding results on open terms. (We still work with singleton and union types.)

Question 10 (Open equivalence):
Define an equivalence relation on open terms ($\Gamma \vdash t \cong u$) and prove that:

1. If v, v' are closed values (no free term or type variables), then $\emptyset \vdash v \cong v'$ is equivalent to $v = v'$.
2. If we have $\Gamma \vdash t : \forall\alpha.\, \alpha \to \alpha$ and $\Gamma \vdash v : \tau$, and $t\ v \rightsquigarrow^* v'$, then we have $\Gamma \vdash t\ v \cong v$.

Proof. Let us write γ for substitutions from type variables to closed types and variables to closed values. We write $\gamma \in [\![\Gamma]\!]$ if γ is a substitution defined on the types and term variables of Γ such that

$$\forall (x : \tau) \in \Gamma, \quad \emptyset \vdash \gamma(x) : \tau[\gamma]$$

We define $\Gamma \vdash t \cong u$ as follows:

$$\forall \gamma \in [\![\Gamma]\!], \exists v_t v_u, \quad t[\gamma] \rightsquigarrow^* v_t \wedge u[\gamma] \rightsquigarrow^* v_u \wedge v_t = v_u$$

We check that the expected properties hold:

1. If v, v' are closed, then $v[\gamma] = v$ and $v'[\gamma] = v'$ for any $\gamma \in [\![\emptyset]\!]$ (there is exactly one substitution in $[\![\emptyset]\!]$: the empty substitution). So in particular $v[\gamma] = v'[\gamma]$ if and only if $v = v'$.
2. If we have $\Gamma \vdash t : \forall\alpha.\, \alpha \to \alpha$ and $\Gamma \vdash v : \tau$, we have to prove that $\Gamma \vdash t\, v \cong v$, that is, for $\gamma \in [\![\Gamma]\!]$ we have to show that $t[\gamma]\ v[\gamma] \rightsquigarrow^* v[\gamma]$. By substitution, we have $\emptyset \vdash t[\gamma] : \forall\alpha.\, \alpha \to \alpha$ and $t[\gamma]\ v[\gamma] \rightsquigarrow^* v'[\gamma]$ and $\emptyset \vdash v[\gamma] : \tau[\gamma]$. We can apply the result of Question 6 (Identity) to conclude that $v'[\gamma] = v[\gamma]$.

□

Remark: the hypothesis that $t\ v \rightsquigarrow^* v'$ is fairly restrictive when t, v are open terms. For example, if the context is $x : \forall\alpha.\, \alpha \to \alpha$, then for $t \coloneqq x$ the result cannot be used as $x\ v$ does not reduce to a value. We will remove this restriction and obtain a strictly stronger statement (without this hypothesis) in the next section.

5. Extending the normalization proof

All our parametricity results so far come with a condition of the form $t \rightsquigarrow^* v'$: we can only prove things about terms that we *assume* will reduce to a value. In Carl Krary's work this extra condition is natural, because the real-world system he wants to apply this technique for (a typed assembly language) is **not** terminating; in a non-terminating system, $f : \forall\alpha.\, \alpha \to \alpha$ is not necessarily the identity, it may also fail to terminate, so the parametricity theorem is weaker.

But we know that System F is terminating, so it should be possible to state simply: if $\Gamma \vdash t : \forall\alpha.\, \alpha \to \alpha$ and $\Gamma \vdash v : \tau$, then $\Gamma \vdash t\, v \cong v$. Unfortunately, we don't *exactly* have System F anymore, we added singleton types $S(v : \tau)$ and union types $\tau_1 \vee \tau_2$. We need to prove that the extended system is still normalizing.

5.1. Normalization by erasure

A first approach is to erase the new type-formers and go back to standard System F types. This works well for singleton types, but not for union types. In this subsection, we will temporarily forget about union types.

Definition 5.1.7 (Erasure of types). We define an erasure operation $\lfloor\tau\rfloor$ on types and contexts as follows:

$$\begin{aligned} \lfloor\alpha\rfloor &\coloneqq \alpha \\ \lfloor\tau_1 \to \tau_2\rfloor &\coloneqq \lfloor\tau_1\rfloor \to \lfloor\tau_2\rfloor \\ \lfloor\forall\alpha.\, \tau\rfloor &\coloneqq \forall\alpha.\, \lfloor\tau\rfloor \\ \lfloor S(v:\tau)\rfloor &\coloneqq \lfloor\tau\rfloor \end{aligned} \qquad \begin{aligned} \lfloor\emptyset\rfloor &\coloneqq \emptyset \\ \lfloor\Gamma, \alpha\rfloor &\coloneqq \lfloor\Gamma\rfloor, \alpha \\ \lfloor\Gamma, x : \tau\rfloor & \lfloor\Gamma\rfloor, x : \lfloor\tau\rfloor \end{aligned}$$

Lemma 5.1.8. $\lfloor\tau[\alpha \coloneqq \tau']\rfloor = \lfloor\tau\rfloor[\alpha \coloneqq \lfloor\tau'\rfloor]$.

<u>Question 11 (Erasure of terms)</u>:
Prove that if $\Gamma \vdash t : \tau$ then we have $\lfloor\Gamma\rfloor \vdash t : \lfloor\tau\rfloor$.

Proof. The proof is by induction on the typing derivation.

$$\left\lfloor \frac{(x:\tau) \in \Gamma}{\Gamma \vdash x : \tau} \right\rfloor \coloneqq \frac{(x : \lfloor\tau\rfloor) \in \lfloor\Gamma\rfloor}{\lfloor\Gamma\rfloor \vdash x : \lfloor\tau\rfloor}$$

$$\left\lfloor \frac{\Gamma, x : \tau \vdash t : \tau'}{\Gamma \vdash \lambda x.\, t : \tau \to \tau'} \right\rfloor \coloneqq \frac{\lfloor\Gamma\rfloor, x : \lfloor\tau\rfloor \vdash t : \lfloor\tau'\rfloor}{\lfloor\Gamma\rfloor \vdash \lambda x.\, t : \lfloor\tau\rfloor \to \lfloor\tau'\rfloor}$$

$$\left\lfloor \frac{\Gamma \vdash t : \tau \to \tau' \quad \Gamma \vdash u : \tau}{\Gamma \vdash t\, u : \tau'} \right\rfloor \coloneqq \frac{\lfloor\Gamma\rfloor \vdash t : \lfloor\tau\rfloor \to \lfloor\tau'\rfloor \quad \lfloor\Gamma\rfloor \vdash u : \lfloor\tau\rfloor}{\lfloor\Gamma\rfloor \vdash t\, u : \lfloor\tau'\rfloor}$$

$$\left\lfloor \frac{\Gamma, \alpha \vdash t : \tau}{\Gamma \vdash t : \forall\alpha.\, \tau} \right\rfloor \coloneqq \frac{\lfloor\Gamma\rfloor, \alpha \vdash t : \lfloor\tau\rfloor}{\lfloor\Gamma\rfloor \vdash t : \forall\alpha.\, \lfloor\tau\rfloor}$$

$$\left\lfloor \frac{\Gamma \vdash t : \forall\alpha.\, \tau}{\Gamma \vdash t : \tau[\alpha \coloneqq \tau']} \right\rfloor \coloneqq \frac{\lfloor\Gamma\rfloor \vdash t : \forall\alpha.\, \lfloor\tau\rfloor}{\lfloor\Gamma\rfloor \vdash t : \lfloor\tau\rfloor[\alpha \coloneqq \lfloor\tau'\rfloor]}$$

$$\left\lfloor \frac{\Gamma \vdash v : \tau}{\Gamma \vdash v : S(\tau : v)} \right\rfloor \coloneqq \lfloor\Gamma\rfloor \vdash v : \lfloor\tau\rfloor$$

□

<u>Question 12 (Normalization without unions)</u>:
Deduce that System F with singleton types (but without unions) is normalizing.

Proof. If $\Gamma \vdash t : \tau$ in System F with singleton types (without union types), then $\lfloor\Gamma\rfloor \vdash t : \lfloor\tau\rfloor$ in standard System F, so t is normalizing. □

5.2. Normalization via a logical relation

We introduced singleton types and union types to get parametricity results without introducing logical relations. But if we want really nice parametricity results, without additional hypotheses, we need to prove normalization, and the easiest way to do so is via a logical relation again.

Let us recall a definition of unary logical relation that we used for System F:

$$\begin{aligned} [\![\alpha]\!]_V(\eta) &\coloneqq \eta(\alpha) \\ [\![\tau_1 \to \tau_2]\!]_V(\eta) &\coloneqq \left\{\lambda x.\, t \mid \forall v \in [\![\tau_1]\!]_V(\eta),\quad ((\lambda x.\, t)\, v) \in [\![\tau_2]\!]_E(\eta)\right\} \\ [\![\forall\alpha.\, \tau]\!]_V(\eta) &\coloneqq \{v \mid \forall\tau', \forall P \subset \text{Values}(\tau'), v \in [\![\tau]\!]_V(\eta, \alpha \mapsto P)\} \end{aligned} \qquad \begin{aligned} [\![\tau]\!]_E(\eta) &\coloneqq \{t \mid t \rightsquigarrow^* v \wedge v \in [\![\tau]\!]_V(\eta)\} \\ [\![\Gamma \vdash \tau]\!] &\coloneqq \{t \mid \forall\gamma \in [\![\Gamma]\!],\quad t[\gamma] \in [\![\tau[\gamma]]\!]_E(\emptyset)\} \end{aligned}$$

Theorem 5.2.9 (Fundamental Lemma for System F).
If $\Gamma \vdash t : \tau$ in System F (without singleton and union types) then $t \in [\![\Gamma \vdash \tau]\!]$.

The Fundamental Lemma was proved by showing that each typing rule of System F is admissible in the logical relation: if its premises are in the relation, then the conclusion is in the relation:

$$\begin{aligned} (t \in [\![\Gamma, x : \tau \vdash \tau']\!]) &\Longrightarrow (\lambda x.\, t) \in [\![\Gamma \vdash \tau \to \tau']\!] \\ t \in [\![\Gamma \vdash \tau \to \tau']\!] \wedge u \in [\![\Gamma \vdash \tau]\!] &\Longrightarrow (t\, u) \in [\![\Gamma \vdash \tau']\!] \end{aligned} \qquad \begin{aligned} t \in [\![\Gamma, \alpha \vdash \tau]\!] &\Longrightarrow t \in [\![\Gamma \vdash \forall\alpha.\, \tau]\!] \\ t \in [\![\Gamma \vdash \forall\alpha.\, \tau]\!] &\Longrightarrow t \in [\![\Gamma \vdash \tau[\alpha \coloneqq \tau']]\!] \end{aligned}$$

Interestingly, these admissibility lemmas remain correct under mild extensions of the type system; their proof works as-is in System F extended with singleton types and union types.

To establish the Fundamental Lemma with singleton and union types, and thus prove its normalization, it therefore suffices to answer the following last question:

Question 13: Propose definitions for $[\![S(v:\tau)]\!]_V(\eta)$ and $[\![\tau_1 \vee \tau_2]\!]_V(\eta)$, and prove admissibility lemmas for their typing rules:

$$v \in [\![\Gamma \vdash \tau]\!] \Longrightarrow v \in [\![\Gamma \vdash S(v:\tau)]\!]$$
$$t \in [\![\Gamma \vdash \tau_1]\!] \Longrightarrow t \in [\![\Gamma \vdash \tau_1 \vee \tau_2]\!]$$

(The $\tau_1 \vee \tau_2$ lemma above comes from Union1, there is another lemma for Union2 but – if your definition of $[\![\tau_1 \vee \tau_2]\!]$ is symmetric – we only ask to prove the τ_1 case.)

Proof.

$$[\![S(v:\tau)]\!]_V(\eta) := \{v\}$$
$$[\![\tau_1 \vee \tau_2]\!]_V(\eta) := [\![\tau_1]\!]_V(\eta) \cup [\![\tau_2]\!]_V(\eta)$$

Admissibility lemma for $S(v:\tau)$:

$$v \in [\![\Gamma \vdash \tau]\!] \Longrightarrow v \in [\![\Gamma \vdash S(v:\tau)]\!]$$

Let $\gamma \in [\![\Gamma]\!]$. We need to prove $v[\gamma] \in [\![S(v:\tau)[\gamma]]\!]_E(\emptyset)$. This is immediate from the following calculation:

$$\begin{aligned} & [\![S(v:\tau)[\gamma]]\!]_E(\emptyset) \\ = {} & [\![S(v[\gamma]:\tau[\gamma])]\!]_E(\emptyset) \\ \supseteq {} & [\![S(v[\gamma]:\tau[\gamma])]\!]_V(\emptyset) \\ = {} & \{v[\gamma]\} \end{aligned}$$

Admissibility lemma for $\tau_1 \vee \tau_2$:

$$t \in [\![\Gamma \vdash \tau_1]\!] \Longrightarrow t \in [\![\Gamma \vdash \tau_1 \vee \tau_2]\!]$$

Let $\gamma \in [\![\Gamma]\!]$. We need to prove $t[\gamma] \in [\![\tau_1[\gamma] \vee \tau_2[\gamma]]\!]_E(\emptyset)$, from the hypothesis that $t[\gamma] \in [\![\tau_1[\gamma]]\!]_E(\emptyset)$. From our hypothesis we have that $t[\gamma] \rightsquigarrow^* v \in [\![\tau_1[\gamma]]\!]_V(\emptyset)$ so in particular we have

$$\begin{aligned} & v \in [\![\tau_1[\gamma]]\!]_V(\emptyset) \cup [\![\tau_2[\gamma]]\!]_V(\emptyset) \\ & \quad = [\![\tau_1[\gamma] \vee \tau_2[\gamma]]\!]_V(\emptyset) \end{aligned}$$

which proves our goal $t[\gamma] \in [\![\tau_1[\gamma] \vee \tau_2[\gamma]]\!]_E(\emptyset)$. □

Question 14 (Normalization with unions): *Warning*: hard question.

We said that the normalization proof by erasure in Section 5.1 is hard to extend to union types $\tau_1 \vee \tau_2$. Can you actually do it, extend this approach to union types? You could proceed as follows:

1. Propose a definition of erasure $\lfloor \tau_1 \vee \tau_2 \rfloor$.
2. Propose a variant of Question 11 (Erasure of terms) and prove it.
3. Prove normalization from there.

Proof. The difficulty with union types is to decide how to erase $\tau_1 \vee \tau_2$. For example, if we decide to define $\lfloor \tau_1 \vee \tau_2 \rfloor := \lfloor \tau_1 \rfloor$, then erasure will work well in type derivations that introduce $t : \tau_1 \vee \tau_2$ from the premise $t : \tau_1$, but it will break well-typedness when the premise was $t : \tau_2$.

Instead we will erase $\tau_1 \vee \tau_2$ into a unit type, technically $\forall \alpha.\, \alpha \to \alpha$. With this translation, when $\Gamma \vdash t : \tau$ holds it is not the case that $\lfloor \Gamma \rfloor \vdash t : \lfloor \tau \rfloor$ anymore, as $t : \tau_1 \vee \tau_2$ does not imply $t : \forall \alpha.\, \alpha \to \alpha$. We have to transform t into a slightly different term t' which satisfies $\lfloor \Gamma \rfloor \vdash t' : \lfloor \tau \rfloor$. We want to prove that if t diverges, then t' diverges – to obtain our normalization result from the normalization of t' – so we have to be careful in how we define t'. In particular, if within t we replace subterms $u : \tau_1 \vee \tau_2$ by simply $\lambda x.\, x$ (at type $\lfloor \tau_1 \vee \tau_2 \rfloor$), then this translation can erase non-terminating subterms u and we cannot prove that divergence is preserved.

Defining $\lfloor \tau_1 \vee \tau_2 \rfloor$

Let us define the following constants:

$$\begin{array}{ll} \mathtt{unit} & := \forall \alpha.\, \alpha \to \alpha \\ \mathtt{tt} : \mathtt{unit} & := \lambda x.\, x \\ \mathtt{ignore} : \forall \beta.\, \beta \to \mathtt{unit} & := \lambda y.\, \mathtt{tt} \end{array}$$

We extend the definition of erasure of types $\lfloor \tau \rfloor$ from Definition 5.1.7 (Erasure of types) to union types:

$$\lfloor \tau_1 \vee \tau_2 \rfloor := \mathtt{unit}$$

Extending erasure of terms into a relation

We define an erasure *judgment* $(\Gamma \vdash t \downarrow \bar{t} : \tau)$ to capture the relation between a term $\Gamma \vdash t : \tau$ and its erasure $\lfloor \Gamma \rfloor \vdash \bar{t} : \lfloor \tau \rfloor$. (Here $\bar{t}$ is just a notation for a metavariable representing a term, just like t_1 or t'.)

$$\frac{(x : \tau) \in \Gamma}{\Gamma \vdash x \dashdownarrow x : \tau}\ \textsc{Era-Var}$$

$$\frac{\Gamma, x : \tau \vdash t \dashdownarrow \overline{t} : \tau'}{\Gamma \vdash \lambda x.\, t \dashdownarrow \lambda x.\, \overline{t} : \tau \to \tau'}\ \textsc{Era-Lam} \qquad \frac{\Gamma \vdash t \dashdownarrow \overline{t} : \tau \to \tau' \quad \Gamma \vdash u \dashdownarrow \overline{u} : \tau}{\Gamma \vdash t\, u \dashdownarrow \overline{t}\, \overline{u} : \tau'}\ \textsc{Era-App}$$

$$\frac{\Gamma, \alpha \vdash t \dashdownarrow \overline{t} : \tau}{\Gamma \vdash t \dashdownarrow \overline{t} : \forall \alpha.\, \tau}\ \textsc{Era-Forall} \qquad \frac{\Gamma \vdash t \dashdownarrow \overline{t} : \forall \alpha.\, \tau}{\Gamma \vdash t \dashdownarrow \overline{t} : \tau[\alpha := \tau']}\ \textsc{Era-Instance}$$

$$\frac{\Gamma \vdash v \dashdownarrow \overline{u} : \tau}{\Gamma \vdash v \dashdownarrow \overline{u} : S(v : \tau)}\ \textsc{Era-Singleton} \qquad \frac{\Gamma \vdash t \dashdownarrow \overline{t} : \tau_i \quad i \in \{1, 2\}}{\Gamma \vdash t \dashdownarrow \texttt{ignore}\, \overline{t} : \tau_1 \vee \tau_2}\ \textsc{Era-Union}$$

It is immediate that $\Gamma \vdash t \dashdownarrow \overline{t} : \tau$ implies $\Gamma \vdash t : \tau$, and also very easy to check in each case that it implies $\lfloor \Gamma \rfloor \vdash \overline{t} : \lfloor \tau \rfloor$. The most interesting cases of the latter implication are as follows:

$$\frac{(x : \tau) \in \Gamma}{\Gamma \vdash x \dashdownarrow x : \tau} \implies \frac{(x : \lfloor \tau \rfloor) \in \lfloor \Gamma \rfloor}{\lfloor \Gamma \rfloor \vdash x : \lfloor \tau \rfloor}$$

$$\frac{\Gamma \vdash v \dashdownarrow \overline{u} : \tau}{\Gamma \vdash v \dashdownarrow \overline{u} : S(v : \tau)} \implies \Gamma \vdash \dashdownarrow \overline{u} : \tau$$

$$\frac{\Gamma \vdash t \dashdownarrow \overline{t} : \tau_i \quad i \in \{1, 2\}}{\Gamma \vdash t \dashdownarrow \texttt{ignore}\, \overline{t} : \tau_1 \vee \tau_2} \implies \frac{\lfloor \Gamma \rfloor \vdash \dashdownarrow \overline{t} : \lfloor \tau_i \rfloor \quad i \in \{1, 2]}{\lfloor \Gamma \rfloor \vdash \dashdownarrow \texttt{ignore}\, \overline{t} : \texttt{unit} = \lfloor \tau_1 \vee \tau_2 \rfloor}$$

Note that in the case of Era-Singleton, the value v is related to a term $\overline{u}$ which is *not* necessarily a value, as the translation introduces applications of `ignore`. This does not break the desired property, as we only have to type $\overline{u}$ at type $\lfloor S(v : \tau) \rfloor$, which is $\lfloor \tau \rfloor$, and does not require that $\overline{u}$ be a value. We can however prove that there is a value $\overline{v}$ such that $\overline{u} \rightsquigarrow^* \overline{v}$. This is checked by an induction on the derivation of $\Gamma \vdash v \dashdownarrow \overline{u}$, with the only case introducing a reduction being the sum case:

$$\frac{\Gamma \vdash v \dashdownarrow \overline{u} : \tau_i \quad i \in \{1, 2\}}{\Gamma \vdash v \dashdownarrow \texttt{ignore}\, \overline{u} : \tau_1 \vee \tau_2}$$

By induction hypothesis we have $\overline{u} \rightsquigarrow^* \overline{v}$ for some value v, so we have $\texttt{ignore}\, \overline{u} \rightsquigarrow^* \texttt{ignore}\, \overline{v} \rightsquigarrow \texttt{unit}$.

Proving normalization

To conclude, we prove that if $\Gamma \vdash t_0 \dashdownarrow \overline{t}_0 : \tau_0$ and $t_0 \rightsquigarrow t_1$, then we have such $\overline{t}_1$ such that $\overline{t}_0 \rightsquigarrow^+ \overline{t}_1$ (reduction in at least one step) and $\Gamma \vdash t_1 \dashdownarrow \overline{t}_1$. This suffices because it means that if t_0 has an infinite reduction sequence, then $\overline{t}_0$ has an infinite reduction sequence, but $\overline{t}_0$ is typed in plain System F so it cannot have an infinite reduction sequence.

Reduction to the application case

Reductions in the Curry-style presentation all involve function applications. We want to reason by inversion on the application rule Era-App, but the derivation may in fact start with other rules such as Era-Forall, Era-Instance, Era-Singleton or Era-Union. We first prove by induction that if the above property holds for all derivations of $\Gamma \vdash t_0 \dashdownarrow \overline{t}_0$ that start with Era-App, then it holds in the general case. We have to check all the rules that are not themselves Era-App, and may have an application term $t\, u$ on the left-hand side.

$$\frac{\Gamma, \alpha \vdash t_0 \dashdownarrow \overline{t}_0 : \tau}{\Gamma \vdash t_0 \dashdownarrow \overline{t}_0 : \forall \alpha.\, \tau}$$

By induction hypothesis, we have $\overline{t}_1$ with $\overline{t}_0 \rightsquigarrow \overline{t}_1$ and also $\Gamma, \alpha \vdash t_1 \dashdownarrow \overline{t}_1 : \tau$. We conclude by applying Era-Forall again to get $\Gamma \vdash t_1 \dashdownarrow \overline{t}_1 : \forall \alpha.\, \tau$.

$$\frac{\Gamma \vdash t_0 \dashdownarrow \overline{t}_0 : \forall \alpha.\, \tau}{\Gamma \vdash t_0 \dashdownarrow \overline{t}_0 : \tau[\alpha := \tau']}$$

The reasoning is identical to the case Era-Forall above.

$$\frac{\Gamma \vdash v \dashdownarrow \overline{u} : \tau}{\Gamma \vdash v \dashdownarrow \overline{u} : S(v : \tau)}$$

This case cannot occur at the head of the derivation as our hypothesis $t_0 \rightsquigarrow t_1$ requires a reducible term, and v is not reducible.

$$\frac{\Gamma \vdash t_0 \dashdownarrow \overline{t}'_0 : \tau_i \quad i \in \{1, 2\}}{\Gamma \vdash t_0 \dashdownarrow \texttt{ignore}\, \overline{t}' : \tau_1 \vee \tau_2}$$

By induction hypothesis, we have $\overline{t}'_1$ with $\overline{t}'_0 \rightsquigarrow \overline{t}'_1$ and also $\Gamma \vdash t_1 \dashdownarrow \overline{t}_1 : \tau_i$. We have $\overline{t}_0 = \texttt{ignore}\, \overline{t}'_0$, and we can conclude by defining $\overline{t}_1 := \texttt{ignore}\, \overline{t}'_1$, which satisfies $\overline{t}_0 \rightsquigarrow \overline{t}_1$ and also $\Gamma \vdash t_1 \dashdownarrow \overline{t}_1 : \tau_1 \vee \tau_2$ by applying Era-Union.

Head reduction

Now that we have restricted ourselves to a derivation of $\Gamma \vdash t_0 \downarrow \overline{t}_0$ that starts with an application rule, we reason by case analysis on the derivation of $t_0 \rightsquigarrow t_1$. We first consider the case of a head reduction with $t_0 = (\lambda x.\, t)\; v$:

$$\frac{}{(\lambda x.\, t)\; v \rightsquigarrow t[x := v]}$$

By inversion, the derivation of $\Gamma \vdash t_0 \downarrow \overline{t}_0 : \tau_0$ must thus be of the following form:

$$\frac{\Gamma \vdash \lambda x.\, t \downarrow \lambda x.\, \overline{t} : \tau \to \tau' \quad \Gamma \vdash v \downarrow \overline{u} : \tau'}{\Gamma \vdash (\lambda x.\, t)\; v \downarrow (\lambda x.\, \overline{t})\; \overline{u} : \tau'}$$

Notice that the left-hand-side has a value v, but that the term $\overline{u}$ related to v is not necessarily a value. We proved previously that for any such $\overline{u}$ there is a value $\overline{v}$ with $\overline{u} \rightsquigarrow^* \overline{v}$.

Going back to the reduction $(\lambda x.\, t)\; v \rightsquigarrow t[x := v]$, we have:

$$\begin{aligned} & \lfloor \Gamma \rfloor \vdash (\lambda x.\, \overline{t})\; \overline{u} : \tau' \\ \rightsquigarrow^* \; & (\lambda x.\, \overline{t})\; \overline{v} \\ \rightsquigarrow \; & \overline{t}[x := \overline{v}] \end{aligned}$$

To conclude with the desired goal that $\Gamma \vdash t[x := v] \downarrow \overline{t}[x := \overline{v}] : \tau$, it remains to be checked that the erasure relation is stable by substitution of erased terms, which can be checked by a direct induction.

Reduction under context

This corresponds to the case $t_0 = E_0[t]$:

$$\frac{t \rightsquigarrow t'}{E_0[t] \rightsquigarrow E_0[t']}$$

We know that $E_0[t]$ is related to some $\overline{t}_0$; to conclude, it suffices to prove that this $\overline{t}_0$ must be of the form $\overline{E_0}[\overline{t}]$, where $\overline{E_0}$ is also an evaluation context and $\overline{t}$ is related to t. (This corresponds to a variant of the context decomposition lemmas seen in the course.) The proof is by induction on E_0:

- If E_0 is $\square$ this is immediate.
- If E_0 is of the form $E\; u$, then $\Gamma \vdash E[t]\; u \downarrow \overline{t}_0 : \tau$ must be (by inversion) of the form

  $$\frac{\Gamma \vdash E[t] \downarrow t' : \tau \to \tau' \quad \Gamma \vdash u \downarrow \overline{u} : \tau}{\Gamma \vdash E[t]\; u \downarrow t'\; \overline{u} : \tau'}$$

  By induction hypothesis on the first premise we have that t' must be $\overline{E}[\overline{t}]$, and so we can conclude with $E_0 := \overline{E}\; u$.
- The proof when E_0 is $t\; E$ is similar.

Once this is proved we have

$$\begin{aligned} & t \rightsquigarrow t' \quad \wedge \quad \Gamma \vdash t \downarrow \overline{t} : \tau' \\ \Longrightarrow \exists \overline{t}',\; & \overline{t} \rightsquigarrow \overline{t}' \quad \wedge \quad \Gamma \vdash t' \downarrow \overline{t}' : \tau' \end{aligned}$$

by induction hypothesis and thus we have

$$\overline{E}[\overline{t}] \rightsquigarrow \overline{E}[\overline{t}'] \quad \wedge \quad \Gamma \vdash E[t'] \downarrow E[\overline{t}'] : \tau$$

as desired.

$\square$