Lecture Notes on Subject Reduction

15-814: Types and Programming Languages
Frank Pfenning

Lecture 4
Thursday, September 9, 2021

1 Introduction

In the last lecture we defined the notion of type and explored how to type Booleans and natural numbers laying the groundwork for a *representation theorem* on Booleans, which will prove in the next lecture.

In this lecture we first explore the limits of typing, and then discuss some properties that tie together *computation* (here in the form of reduction) and *typing*. Prove these properties requires a form of induction called *rule induction* and we will spend some time on how to perform rule inductions to prove properties of judgments defined by inference rules.

The eventual representation theorems for Booleans will say something along the following lines (perhaps to be refined):

> If $\cdot \vdash e : \alpha \to (\alpha \to \alpha)$ and e does not reduce, then $e = \mathit{true} = \lambda x.\, \lambda y.\, x$ or $e = \mathit{false} = \lambda x.\, \lambda y.\, y$.

How do we use a programming language to perform computation? We provide some definitions and then an expression e. The implementation will then normalize the expression e by reducing it until it can no longer be reduced, at least conceptually. In actuality, it may translate or compile the expression and then execute the compiled form for efficiency.

With types, we check that our expression e has the type we intended (for example, $\cdot \vdash e : \tau$) before we reduce it, because only typed expressions are seen as meaningful. Also, we usually require the context to be empty because we might view a free variable as an "undefined function". So just

as in the representation of Booleans and natural numbers, we focus on the computation with closed terms.

Now we would like to be assured that the result of the computation, that is, the normal form e' of e (with $e \longrightarrow^* e'$) is still of the same type! It would not make sense if we start an expression $e : \mathit{nat}$ and are presented with an answer of $\mathit{true} : \mathit{bool}$. To prevent this situation, we would like to prove for our programming language (so far, just the λ-calculus) that

$$\text{If } \cdot \vdash e : \tau \text{ and } e \longrightarrow^* e' \text{ and } e' \text{ does not reduce, then } \cdot \vdash e' : \tau.$$

We usually breaks this down into a *subject reduction* also called *type preservation* for the single-step reduction relation, since multi-step reduction then follows by a simple induction (see Exercise 1).

Conjecture 1 (Subject Reduction, v1) *If* $\cdot \vdash e : \tau$ *and* $e \longrightarrow e'$ *then* $\cdot \vdash e' : \tau$.

Before we investigate its proof and refine its statements, let's consider the limits of typing.

2 The Limits of Simple Types

We have proposed types as a way to classify functions, fixing their domain and their codomain, and making sure that functions are applied to arguments of the correct type. We also started to observe some patterns, such as $\mathit{true} : \alpha \to (\alpha \to \alpha)$ and $\mathit{false} : \alpha \to (\alpha \to \alpha)$, possibly using this type to characterize Booleans.

But what do we give up? Are there expressions that cannot be typed? From the historical perspective, this should definitely be the case, because types were introduced exactly to rule out certain "paradoxical" terms such as Ω, which does not have a normal form.

One term that is no longer typeable is self-application $\omega = \lambda x.\, x\, x$. As a result, we also can type neither $\Omega = \omega\, \omega$ nor Y, which can be seen as achieving a goal from the logical perspective, but it does give up computational expressiveness. How do we prove that ω cannot be typed? We begin by creating the skeleton of a typing derivation, which is unique due to the syntax-directed nature of the rules (that is, for each language construct there is exactly one typing rule). We highlight in red rules whose constraints on types have not yet been considered. When all the rules are black, we know that every solution to the accumulated constraints leads to a valid typing

derivation (and therefore a valid type in the conclusion).

$$\dfrac{\dfrac{}{x : \boxed{} \vdash x : \boxed{}}\,\textsf{tp/var} \quad \dfrac{}{x : \boxed{} \vdash x : \boxed{}}\,\textsf{tp/var}}{x : \boxed{} \vdash x\,x : \boxed{}}\,\textsf{tp/app}}{\cdot \vdash \lambda x.\, x\,x : \boxed{}}\,\textsf{tp/lam}$$

The type in the final judgment must be $?\tau_1 \to ?\tau_2$ for some types $?\tau_1$ and $?\tau_2$.

$$\dfrac{\dfrac{\dfrac{}{x : \boxed{} \vdash x : \boxed{}}\,\textsf{tp/var} \quad \dfrac{}{x : \boxed{} \vdash x : \boxed{}}\,\textsf{tp/var}}{x : \boxed{?\tau_1} \vdash x\,x : \boxed{?\tau_2}}\,\textsf{tp/app}}{\cdot \vdash \lambda x.\, x\,x : \boxed{?\tau_1 \to ?\tau_2}}\,\textsf{tp/lam}$$

Once the type of a variable is available in the context, this types is propagated upwards unchanged in a derivation, so we can fill in some more of the types.

$$\dfrac{\dfrac{\dfrac{}{x : \boxed{?\tau_1} \vdash x : \boxed{}}\,\textsf{tp/var} \quad \dfrac{}{x : \boxed{?\tau_1} \vdash x : \boxed{}}\,\textsf{tp/var}}{x : \boxed{?\tau_1} \vdash x\,x : \boxed{?\tau_2}}\,\textsf{tp/app}}{\cdot \vdash \lambda x.\, x\,x : \boxed{?\tau_1 \to ?\tau_2}}\,\textsf{tp/lam}$$

In the tp/var rules the type of variable is just looked up in the context, so we can fill in those two types as well.

$$\dfrac{\dfrac{\dfrac{}{x : \boxed{?\tau_1} \vdash x : \boxed{?\tau_1}}\,\textsf{tp/var} \quad \dfrac{}{x : \boxed{?\tau_1} \vdash x : \boxed{?\tau_1}}\,\textsf{tp/var}}{x : \boxed{?\tau_1} \vdash x\,x : \boxed{?\tau_2}}\,\textsf{tp/app}}{\cdot \vdash \lambda x.\, x\,x : \boxed{?\tau_1 \to ?\tau_2}}\,\textsf{tp/lam}$$

Finally, for the application of the tp/app rule to be correct, the type of x in the first premise must be a function type, expecting an argument of type $?\tau_1$

(the type of x in the second premise) and returning a result of type $?\tau_2$. That is:

$$\dfrac{\dfrac{}{x : \boxed{?\tau_1} \vdash x : \boxed{?\tau_1}}\,\mathsf{tp/var} \qquad \dfrac{}{x : \boxed{?\tau_1} \vdash x : \boxed{?\tau_1}}\,\mathsf{tp/var}}{\dfrac{x : \boxed{?\tau_1} \vdash x\,x : \boxed{?\tau_2}}{\cdot \vdash \lambda x.\, x\,x : \boxed{?\tau_1 \to ?\tau_2}}\,\mathsf{tp/lam}}\,\mathsf{tp/app}$$

provided $?\tau_1 = ?\tau_1 \to ?\tau_2$

Now we observe that there cannot be a solution to the required equation: there are no types τ_1 and τ_2 such that $\tau_1 = \tau_1 \to \tau_2$ since the right-hand side is always bigger (and therefore not equal) to the left-hand side.

To recover from this in full generality we would need so-called *recursive types*. In this example, we see

$$\tau_1 = F\,\tau_1$$

where $F = \lambda\alpha.\,\alpha \to \tau_2$ and we might then have a solution with $\tau_1 = Y\,F$. But such a solution is not immediately available to us. For one thing, we do not have function from types to types such as F. For another, we don't have a Y combinator at the level of types. However, it is perfectly possible to construct recursive types, and we will do so later in the course. We can also think of such recursive types as *infinite types*

$$\tau_1 = \tau_1 \to \tau_2 = (\tau_1 \to \tau_2) \to \tau_2 = ((\tau_1 \to \tau_2) \to \tau_2) \to \tau_2 = \cdots$$

Another way to recover some, but not all of the functions that can be typed in the λ-calculus is to introduce *polymorphism*, which we will also consider.

3 Reduction as a Judgment

Our characterization of normal forms so far is quite simple: they are terms that do not reduce. But this is a *negative* condition, and negative conditions can be difficult to work with in proofs. So we would like a *positive* definition normal forms. Just like typing, we tend to give such definitions in the form of inference rules. The property then holds if the judgment of interest (here, that an expression is normal) can be derived using the given rules. This is a form of *inductive definition*.

Before we get to defining normal forms by rules, we formally define β-reduction by inference rules. Previously, we just stated informally that a step of β-reduction can be "applied anywhere in an expression". Now we write this out. We refer to the last three rules as *congruence rules* because they allow the reduction of a subterm. The judgment is here $e \longrightarrow e'$ (omitting the β for brevity) expressing that e reduces to e'.

$$\frac{}{(\lambda x.\, e_1)\, e_2 \longrightarrow [e_2/x]e_1}\ \mathsf{red/beta} \qquad \frac{e \longrightarrow e'}{\lambda x.\, e \longrightarrow \lambda x.\, e'}\ \mathsf{red/lam}$$

$$\frac{e_1 \longrightarrow e_1'}{e_1\, e_2 \longrightarrow e_1'\, e_2}\ \mathsf{red/app}_1 \qquad \frac{e_2 \longrightarrow e_2'}{e_1\, e_2 \longrightarrow e_1\, e_2'}\ \mathsf{red/app}_2$$

This rules are *not* syntax-directed: there are three rules for application (red/beta, red/app$_1$, and red/app$_2$), one for λ (red/lam) and none for variables. So if we use them to construct a derivation, we may have to backtrack or presciently make a choice. When construction a derivation we usually assume e is given and we simultaneously construct a derivation and an expression e' such that $e \longrightarrow e'$. For example:

$$\frac{}{((\lambda x.\, \lambda y.\, x)\, K)\, I \longrightarrow \boxed{}}\ ?$$

We realize that this is not a redex at the top level, and we also know that I can't be reduced, so we use the red/app$_1$ rule:

$$\frac{\dfrac{}{(\lambda x.\, \lambda y.\, x)\, K \longrightarrow \boxed{}}\ ?}{((\lambda x.\, \lambda y.\, x)\, K)\, I \longrightarrow \boxed{}}\ \mathsf{red/app}_1$$

Again, a priori three rules could be applied, but only red/beta will succeed in building a derivation:

$$\frac{\dfrac{}{(\lambda x.\, \lambda y.\, x)\, K \longrightarrow \boxed{}}\ \mathsf{red/beta}}{((\lambda x.\, \lambda y.\, x)\, K)\, I \longrightarrow \boxed{}}\ \mathsf{red/app}_1$$

At this point the structure of the derivation is complete, and we just need to fill in the blanks. Starting from the top, we get $[K/x](\lambda y.\, x) = \lambda y.\, K$

$$\dfrac{\dfrac{}{(\lambda x.\, \lambda y.\, x)\, K \longrightarrow \lambda y.\, K}\ \mathsf{red/beta}}{((\lambda x.\, \lambda y.\, x)\, K)\, I \longrightarrow \boxed{}}\ \mathsf{red/app_1}$$

The final blank is the result from the previous line applied to I, and we obtain:

$$\dfrac{\dfrac{}{(\lambda x.\, \lambda y.\, x)\, K \longrightarrow \lambda y.\, K}\ \mathsf{red/beta}}{((\lambda x.\, \lambda y.\, x)\, K)\, I \longrightarrow (\lambda y.\, K)\, I}\ \mathsf{red/app_1}$$

4 Subject Reduction

Now we return to the main topic of this lecture, namely subject reduction. Recall our characterization of reduction:

$$\dfrac{e \longrightarrow e'}{\lambda x.\, e \longrightarrow \lambda x.\, e'}\ \mathsf{red/lam} \qquad \dfrac{e_1 \longrightarrow e_1'}{e_1\, e_2 \longrightarrow e_1'\, e_2}\ \mathsf{red/app_1} \qquad \dfrac{e_2 \longrightarrow e_2'}{e_1\, e_2 \longrightarrow e_1\, e_2'}\ \mathsf{red/app_2}$$

$$\dfrac{}{(\lambda x.\, e_1)\, e_2 \longrightarrow [e_2/x]e_1}\ \mathsf{beta}$$

And, for reference, here are the typing rules.

$$\dfrac{\Gamma, x_1 : \tau_1 \vdash e_2 : \tau_2}{\Gamma \vdash \lambda x_1.\, e_2 : \tau_1 \to \tau_2}\ \mathsf{lam} \qquad \dfrac{x : \tau \in \Gamma}{\Gamma \vdash x : \tau}\ \mathsf{var}$$

$$\dfrac{\Gamma \vdash e_1 : \tau_2 \to \tau_1 \quad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash e_1\, e_2 : \tau_1}\ \mathsf{app}$$

Conjecture 2 (Subject Reduction, v1)
If $\cdot \vdash e : \tau$ *and* $e \longrightarrow e'$ *then* $\cdot \vdash e' : \tau$.

Proof: (Attempt) Proving this will require some analysis of the derivations of $\cdot \vdash e : \tau$ and $e \longrightarrow e'$. We hope that in all cases, $\cdot \vdash e' : \tau$. But just a proof by cases will not work, because the judgments of typing and reduction are defined *inductively* by a collection of inference rules. That is, for example, the

judgment $e \longrightarrow e'$ holds *if and only if* there is a derivation for it. To mirror this inductive definition, we have to carry out a proof by induction. Such proofs are by induction over the structure of the derivation of a judgment. What this means is that for each rule, we get to assume our conjecture for all the premises of the rule (the *induction hypothesis*) and have to prove it for the conclusion. A rule with no premises (such as red/beta) is then a *base case* because we have no inductive hypotheses to work with. Rules with premises correspond to induction steps.

Another way to think about such a proof is: if a property of a judgment *preserved* by all rules of inference and holds for the axioms (the rules with no premises), then it must hold of all judgments.

So, let's get started. We call this form of induction *rule induction*. In this particular statement, we could use rule induction on the judgment $\cdot \vdash e : \tau$ or $e \longrightarrow e'$. In principle, we might also be able to use induction over the structure of e, e', or τ, but the derivations we have give us more information. So, generally speaking, it is unlikely that we perform induction over expressions or types when we have richer assumptions. Again, a general heuristic says that if we have a syntax-directed judgment and one that is not, it is often better to induct over the judgment that is not syntax-directed. This suggest a proof by rule induction over $e \longrightarrow e'$.

We have to distinguish the various rules for this judgment.

Case:

$$\frac{e_1 \longrightarrow e_1'}{e_1\, e_2 \longrightarrow e_1'\, e_2}\ \mathsf{red/app}_1$$

where $e = e_1\, e_2$ and $e; = e_1'\, e_2$. We start by restating what we know in this case.

$\cdot \vdash e_1\, e_2 : \tau$ Assumption

Now we perform a key step, namely *inversion*. We know that $\cdot \vdash e_1\, e_2 : \tau$ and we see that there is only inference rule whose conclusion matches this: tp/app. Since some instance of the rule must have been used, we conclude:

$\cdot \vdash e_1 : \tau_2 \to \tau$ and
$\cdot \vdash e_2 : \tau_2$ for some τ_2 By inversion

We have to be extremely careful, because it looks like we are using a rule of inference in the wrong direction. I guess that's why it is called *inversion*. So always make sure you know that a judgment is true, and there is only one (or, more generally, a finite number of) judgments that could have derived it.

At this point we have a type for e_1 and a reduction for e_1, so we can apply the induction hypothesis.

$\cdot \vdash e_1' : \tau_2 \to \tau$ By ind.hyp.

Now we can just apply the typing rule for application. Intuitively, in the typing for $e_1\, e_2$ we have replaced e_1 by e_1', which is okay since e_1' has the type of e_1.

$\cdot \vdash e_1'\, e_2 : \tau$ By rule tp/app

Case:

$$\frac{e_1 \longrightarrow e_1'}{\lambda x.\, e_1 \longrightarrow \lambda x.\, e_1'}\ \mathsf{red/lam}$$

where $e = \lambda x.\, e_1'$.

We proceed as before, applying inversion to the typing of e.

$\cdot \vdash \lambda x.\, e_1 : \tau$ Assumption
$x : \tau_2 \vdash e_1 : \tau_1$ and $\tau = \tau_2 \to \tau_1$ for some τ_1 and τ_2 By inversion

However, at this point we are stuck because we cannot apply the induction hypothesis! The problem is that in the typing for e_1 the context is not empty.

□

So it seems the property we want does not follow by rule induction! If we get stuck like that, there are usually three possibilities to consider:

(1) The conjecture is false. Maybe the failed proof attempt helps us find a counterexample so we can revise either our definitions or our conjecture.

(2) We need to generalize the induction hypothesis. Maybe the failed proof attempt helps us find the right generalization.

(3) We need to find a suitable lemma. Maybe the failed proof attempt helps us find such a lemma. In this case we can often keep the structure of the proof intact.

In this particular example, the fact that the context was nonempty in an attempted appeal to the induction hypothesis suggests we should allow arbitrary nonempty contexts in our induction hypothesis.

Theorem 3 (Subject Reduction, v2)
If $\Gamma \vdash e : \tau$ *and* $e \longrightarrow e'$ *then* $\Gamma \vdash e' : \tau$.

Proof: We try again: rule induction on the derivation of $e \longrightarrow e'$.

Case:

$$\frac{e_1 \longrightarrow e_1'}{\lambda x.\, e_1 \longrightarrow \lambda x.\, e_1'}\ \mathsf{red/lam}$$

where $e = \lambda x.\, e_1'$. In the critical step we can now appeal to the induction hypothesis using $\Gamma, x : \tau_2$ as our context.

$\Gamma \vdash \lambda x.\, e_1 : \tau$	Assumption
$\Gamma, x : \tau_2 \vdash e_1 : \tau_1$ and $\tau = \tau_2 \to \tau_1$ for some τ_1 and τ_2	By inversion
$\Gamma, x : \tau_2 \vdash e_1' : \tau_1$	By induction hypothesis
$\Gamma \vdash \lambda x.\, e_1' : \tau_2 \to \tau_1$	By rule $\mathsf{tp/lam}$

Case:

$$\frac{e_1 \longrightarrow e_1'}{e_1\, e_2 \longrightarrow e_1'\, e_2}\ \mathsf{red/app}_1$$

where $e = e_1\, e_2$. We have to reconsider this case which worked before, now for the generalized induction hypothesis. But it works the same way.

$\Gamma \vdash e_1\, e_2 : \tau$	Assumption
$\Gamma \vdash e_1 : \tau_2 \to \tau$ and	
$\Gamma \vdash e_2 : \tau_2$ for some τ_2	By inversion

At this point we have a type for e_1 and a reduction for e_1, so we can apply the induction hypothesis.

$\Gamma \vdash e_1' : \tau_2 \to \tau$ By ind.hyp.

Now we can just apply the typing rule for application. Intuitively, in the typing for $e_1\, e_2$ we have replaced e_1 by e_1', which is okay since e_1' has the type of e_1.

$\Gamma \vdash e_1'\, e_2 : \tau$ By rule tp/app

Case:

$$\frac{e_2 \longrightarrow e_2'}{e_1\, e_2 \longrightarrow e_1\, e_2'}\ \mathsf{red/app}_2$$

where $e = e_1\, e_2$. This proceeds completely analogous to the previous case.

Case:

$$\frac{}{(\lambda x.\, e_1)\, e_2 \longrightarrow [e_2/x]e_1}\ \beta$$

where $e = (\lambda x.\, e_1)\, e_2$. In this case we apply inversion twice, since the structure of e is two levels deep.

$\Gamma \vdash (\lambda x.\, e_1)\, e_2 : \tau$	Assumption
$\Gamma \vdash \lambda x.\, e_1 : \tau_2 \to \tau$	
and $\Gamma \vdash e_2 : \tau_2$ for some τ_2	By inversion
$\Gamma, x : \tau_2 \vdash e_1 : \tau$	By inversion

At this point we are once again truly stuck, because there is no obvious way to complete the proof.

To Show: $\Gamma \vdash [e_2/x]e_1 : \tau$

Fortunately, the gap that presents itself is exactly the content of the *substitution property*, stated below. The forward reference here is acceptable, since the proof of the substitution property does not depend on subject reduction.

$\Gamma \vdash [e_2/x]e_1 : \tau$ By the *substitution property* (Theorem 4)

□

The last case in the proof of subject reduction represents option (3) in the heuristic list of "ways out" if we get stuck in our induction proof.

Theorem 4 (Substitution Property)
If $\Gamma \vdash e : \tau$ *and* $\Gamma, x : \tau, \Gamma' \vdash e' : \tau'$ *then* $\Gamma, \Gamma' \vdash [e/x]e' : \tau'$

Proof sketch: By rule induction on the derivation of $\Gamma, x : \tau, \Gamma' \vdash e' : \tau'$. Intuitively, in this derivation we can use $x : \tau$ only at the leaves, and there to conclude $x : \tau$. Now we replace this leaf with the given derivation of $\Gamma \vdash e : \tau$ which concludes $e : \tau$. Luckily, $[e/x]x = e$, so this is the correct judgment.

There is only a small hiccup: we have to adjoin the additional variables declared in Γ'. This would be yet another lemma, namely, that we can always add unused variable and type to a typing derivation, a property called *weakening*. Since it is straightforward to prove, once again by induction, we will not formalize it even if we have multiple occasions to use it. □

We recommend you write out the cases of the substitution property in the style of our other proofs, just to make sure you understand the details.

The substitution property is so critical that we may elevate it to an intrinsic property of the turnstile ($\vdash$). Whenever we write $\Gamma \vdash J$ for any judgment J we imply that a substitution property for the judgments in Γ must hold. This is an example of a *hypothetical* and *generic* judgment [ML83]. We may return to this point in a future lecture, especially if the property appears to be in jeopardy at some point. It is worth remembering that, while we may not want to prove an explicit substitution property, we still need to make sure that the judgments we define *are* hypothetical/generic judgments.

Looking back at Conjecture 2, this also holds as a consequence of Theorem 3: we just use it for $\Gamma = (\cdot)$. So it was not wrong, we just couldn't prove it the way we wanted because the induction hypothesis for our rule induction wasn't strong enough.

5 Normal Forms

A *normal form* is an expression e such that there does not exists an e' such that $e \longrightarrow e'$. Basically, we have to rule out β-redices $(\lambda x.\, e_1)\, e_2$, but we would like to describe normal forms via inference rules so we can easily prove inductive theorems on them. We might start with the following incorrect

attempt:

$$\frac{}{x\ \textit{normal}}\ \textsf{norm/var} \qquad \frac{e\ \textit{normal}}{\lambda x.\, e\ \textit{normal}}\ \textsf{norm/lam}$$

$$\frac{e_1\ \textit{normal} \quad e_2\ \textit{normal}}{e_1\, e_2\ \textit{normal}}\ \textsf{norm/app}$$

It is easy to see that under such a definition *every* term would be normal. The culprit here is the rule of application, because, for example, in the application $(\lambda x.\, x)\,(\lambda y.\, y)$ both function and argument are normal, but their application is not. So we need a separate judgment for *neutral* expressions, namely those which are normal already *and* do *not* create a redex when they are applied to an argument. In particular, a λ-abstraction is *not* neutral, but a variable is. Then $e_1\, e_2$ is normal if e_1 is neutral and e_2 is normal.

$$\frac{e\ \textit{normal}}{\lambda x.\, e\ \textit{normal}}\ \textsf{norm/lam} \qquad \frac{e\ \textit{neutral}}{e\ \textit{normal}}\ \textsf{norm/neut}$$

$$\frac{}{x\ \textit{neutral}}\ \textsf{neut/var} \qquad \frac{e_1\ \textit{neutral} \quad e_2\ \textit{normal}}{e_1\, e_2\ \textit{neutral}}\ \textsf{neut/app}$$

This definition captures terms of the form

$$\lambda x_1.\ \ldots\ \lambda x_n.\, ((x\, e_1) \ldots e_k)$$

where $e_1, \ldots e_k$ are again in normal form. It is not strictly syntax-directed in the given form because, for a λ-abstraction, both rules norm/lam and norm/neut could be used. However, norm/neut will fail immediately in the next step, so we only need to "look ahead" one rule to make the construction deterministic.

As an example, to show that $\lambda x.\, x\, x$ *normal* we construct the following derivation, starting from the bottom.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{x\ \textit{neutral}}\ \textsf{neut/var} \quad \dfrac{\dfrac{}{x\ \textit{neutral}}\ \textsf{neut/var}}{x\ \textit{normal}}\ \textsf{norm/neut}}{x\, x\ \textit{neutral}}\ \textsf{neut/app}}{x\, x\ \textit{normal}}\ \textsf{norm/neut}}{\lambda x.\, x\, x\ \textit{normal}}\ \textsf{norm/lam}$$

But does this new judgment e *normal* really capture exactly the normal expressions, namely those that cannot be reduced? We will consider this question in the next lecture. Only once we are sure about this does it make sense to consider the representation theorem, now referencing the e *normal* judgment rather than saying that e cannot be reduced.

Exercises

Exercise 1 In lecture, we defined the reflexive and transitive closure ($\longrightarrow^*$) of the single-step reduction ($\longrightarrow$) with the following:

$$\frac{}{e \longrightarrow^* e}\ \mathsf{red}^*/\mathsf{refl} \qquad \frac{e_1 \longrightarrow^* e_2 \quad e_2 \longrightarrow^* e_3}{e_1 \longrightarrow^* e_3}\ \mathsf{red}^*/\mathsf{trans}$$

$$\frac{e_1 \longrightarrow e_2}{e_1 \longrightarrow^* e_2}\ \mathsf{red}^*/\mathsf{step}$$

However, it is more common to define multistep reduction with only two rules, as is done for the $\Longrightarrow$ judgment below:

$$\frac{}{e \Longrightarrow e}\ \mathsf{reds}/\mathsf{refl} \qquad \frac{e_1 \longrightarrow e_2 \quad e_2 \Longrightarrow e_3}{e_1 \Longrightarrow e_3}\ \mathsf{reds}/\mathsf{step}$$

Prove by rule induction that these two definitions are equivalent in the sense that $e \Longrightarrow e'$ iff $e \longrightarrow^* e'$.

Exercise 2 Recall the relation $\longrightarrow^*$ defined in Exercise 1. Prove by rule induction that if $\Gamma \vdash e : \tau$ and $e \longrightarrow^* e'$ then $\Gamma \vdash e' : \tau$. Here (as in general in the course), you may use theorems we have proved in the course (lecture or notes).

Exercise 3 Recall the relation $\Longrightarrow$ defined in Exercise 1. Prove by rule induction that if $\Gamma \vdash e : \tau$ and $e \Longrightarrow e'$ then $\Gamma \vdash e' : \tau$. Here (as in general in the course), you may use theorems we have proved in the course (lecture or notes).

Exercise 4 Define a new single-step relation $e \mapsto e'$ which means that e reduces to e' by *leftmost-outermost reduction*, using a collection of inference rules. Recall that I claimed this strategy is *sound* (it only performs β-reductions) and *complete for normalization* (if e has a normal form, we can reach it by performing only leftmost-outermost reductions). Prove the following statements about your reduction judgment:

(i) If $e \mapsto e'$ then $e \longrightarrow e'$.

(ii) $\mapsto$ is *small-step deterministic*, that is, if $e \mapsto e_1$ and $e \mapsto e_2$ then $e_1 = e_2$.

You should interpret $=$ as α-equality, that is, the two terms differ only in the names of their bound variables (which we always take for granted). For each of the following statements, either indicate that they are true (without proof) or provide a counterexample.

(iii) For all e, either $e \mapsto e'$ for some e' or e *normal*.

(iv) There does not exist an e such that $e \mapsto e'$ for some e' and e *normal*.

(v) If $e \longrightarrow e'$ then $e \mapsto e'$.

(vi) $\longrightarrow$ is *small-step deterministic*.

(vii) $\longrightarrow$ is *big-step deterministic*, that is, if $e \longrightarrow^* e_1$ and $e \longrightarrow^* e_2$ where e_1 *normal* and e_2 *normal*, then $e_1 = e_2$.

(viii) For arbitrary e and *normal* e', $e \longrightarrow^* e'$ iff $e \mapsto^* e'$.

References

[ML83] Per Martin-Löf. On the meanings of the logical constants and the justifications of the logical laws. Notes for three lectures given in Siena, Italy. Published in *Nordic Journal of Philosophical Logic*, 1(1):11-60, 1996, April 1983.