Lecture Notes on Representation Theorems

15-814: Types and Programming Languages
Frank Pfenning

Lecture 4
September 10, 2020

1 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 *true* : $\alpha \to (\alpha \to \alpha)$ and *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).

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

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

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

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.

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

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.

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

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:

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

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. In the notation we will introduce later, we would get

$$\tau_1 = \rho\alpha.\,\alpha \to \tau_2$$

where $\rho\alpha.\,\tau$ binds the type variable α with scope τ. Such a type will be equivalent to its unfolding $[\rho\alpha.\,\tau/\alpha]\tau$.

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.

2 Characterizing the Booleans

We would now like to show that the representation of the Booleans is in fact correct. We go through a sequence of conjectures to (hopefully) arrive at the correct conclusion.

Conjecture 1 (Representation of Booleans, v1)
If $\cdot \vdash e : \alpha \to (\alpha \to \alpha)$ *then* $e =$ *true or* $e =$ *false.*

If by "=" we mean mathematical equality that this is false. For example,

$$\cdot \vdash (\lambda z.\, z)\ (\lambda x.\, \lambda y.\, x) : \alpha \to (\alpha \to \alpha)$$

but the expression $(\lambda z.\, z)\ (\lambda x.\, \lambda y.\, x)$ represents neither true nor false. But it is in fact β-convertible to *true*, so we might loosen our conjecture:

Conjecture 2 (Representation of Booleans, v2)
If $\cdot \vdash e : \alpha \to (\alpha \to \alpha)$ *then* $e =_\beta$ *true or* $e =_\beta$ *false.*

By the Church-Rosser Theorem, if $e =_\beta e'$ where e' is a normal form (that is, cannot be reduced), then $e \longrightarrow_\beta^* e'$ so we can replace this by

Conjecture 3 (Representation of Booleans, v3)
If $\cdot \vdash e : \alpha \to (\alpha \to \alpha)$ *then* $e \longrightarrow_\beta^*$ *true or* $e \longrightarrow_\beta^*$ *false.*

This is actually quite difficult to prove. In particular, it requires that every expression of the given type does have a normal form. We have already seen that the standard divergent term Ω does not have a type, and neither does the Y combinator. In fact, it will turn out (although with a difficult proof) that every simple-typed λ-expression does have a normal form! This is commonly called the *weak normalization* property. *Strong normalization* requires that every reduction sequence terminates, which, incidentally, also holds here.

Fortunately, we can prove simpler theorems that do not directly rely on normalization. The first one concerns only *normal forms*, that is, expressions that cannot be β-reduced. They play the role that *values* play in many programming languages.

Conjecture 4 (Representation of Booleans, v4)
If $\cdot \vdash e : \alpha \to (\alpha \to \alpha)$ *and* e *is a normal form, then* $e =$ *true or* $e =$ *false.*

We will later combine this with the following theorems which yields correctness of the representation of Booleans. These theorems are quite general (not just on Booleans), and we will see multiple versions of them in the remainder of the course.

Theorem 5 (Weak Normalization) *If* $\Gamma \vdash e : \tau$ *then* $e \longrightarrow_\beta^* e'$ *for a normal form* e'.

Theorem 6 (Subject reduction) *If* $\Gamma \vdash e : \tau$ *and* $e \longrightarrow_\beta e'$ *then* $\Gamma \vdash e' : \tau$.

3 Reduction Revisited

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 closely related to the notion 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'.

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

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

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:

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

$$\frac{e_1\ \mathit{normal} \quad e_2\ \mathit{normal}}{e_1\, e_2\ \mathit{normal}}\ \mathsf{norm/app}$$

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 term itself is not. So we need a separate judgment for *neutral* terms which

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.

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

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

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 $\lambda x.\, x\ x\ \textit{normal}$ we construct the following derivation, starting from the bottom.

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

4 Normal Forms and Reduction

The characterization of normal forms via inference rules is compact, but is it really the same as saying that an expression does not reduce? We would like to work as much as possible with positive characterizations, so we break this down into the following two properties

1. For all expressions e, either e reduces or e is normal.

2. For all expressions e, it is not that case that e reduces and e is normal.

The second property just states that the "either/or" in part 1 is an exclusive or. We will prove the first, and leave the second as Exercise **??**.

To make the proof just a bit easier to write, we introduce a new judgment $e \longrightarrow$ expressing that e reduces, but we do not care what to. We obtain it by erasing the right-hand sides of all the reduction rules. It is then immediate (although formally done by induction) that $e \longrightarrow e'$ for some e' iff $e \longrightarrow$.

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

$$\frac{e \longrightarrow}{\lambda x.\, e \longrightarrow}\ \mathsf{rbl/lam} \qquad \frac{e_1 \longrightarrow}{e_1\, e_2 \longrightarrow}\ \mathsf{rbl/app_1} \qquad \frac{e_2 \longrightarrow}{e_1\, e_2 \longrightarrow}\ \mathsf{rbl/app_2}$$

Theorem 7 (Reduction and normal forms, Part (i))
For every expression e, either $e \longrightarrow$ or e normal.

Proof: We are only given an expression e, so the proof is likely by induction on the structure of e. Such a proof has the following parts:

(i) We have to establish the property outright for $e = x$.

(ii) We have to establish the property for $e = \lambda x.\, e_1$, where the induction hypothesis is the property for e_1.

(iii) We have the establish the property for $e = e_1\, e_2$ where the induction hypotheses are the properties for e_1 and e_2.

If we can cover all three cases we know that the property must hold for all expressions. Let's try!

Case: $e = x$. Then

x *neutral*	By rule neut/var
x *normal*	By rule norm/neut

Case: $e = \lambda x.\, e_1$. Then

Either $e_1 \longrightarrow$ or e_1 *normal* — By ind.hyp. on e_1

- $e_1 \longrightarrow$ — First subcase
  $e = \lambda x.\, e_1 \longrightarrow$ — By rule rbl/lam
- e_1 *normal* — Second subcase
  $e = \lambda x.\, e_1$ *normal* — By rule norm/lam

Case: $e = e_1\, e_2$. Then

Either $e_1 \longrightarrow$ or e_1 *normal* — By ind.hyp. on e_1

- $e_1 \longrightarrow$ — First subcase
  $e_1\, e_2 \longrightarrow$ — By rule $\mathsf{rbl/app}_1$
- e_1 *normal* — Second subcase
  Either $e_1 = \lambda x.\, e_1'$ and e_1' *normal*
  or e_1 *neutral* — By inversion on e_1 *normal*
  - $e_1 = \lambda x.\, e_1'$ — First sub^{2}case
    $e = e_1\, e_2 = (\lambda x.\, e_1')\, e_2 \longrightarrow$ — By rule rbl/beta
  - e_1 *neutral* — Second sub^{2}case
    Either $e_2 \longrightarrow$ or e_2 *nf* — By ind.hyp. on e_2
    - $e_2 \longrightarrow$ — First sub^{3}case
      $e = e_1\, e_2 \longrightarrow$ — By rule $\mathsf{rbl/app}_2$
    - e_2 *normal* — Second sub^{3}case
      $e = e_1\, e_2$ *neutral* — By rule neut/app

□

This proof is slightly shorter from the proof we did in lecture, using an inversion step that is highlighted in red. What are we doing in this step? We know we are in the "second subcase" so we have the knowledge that e_1 *normal*. Now we examine the inference rule and we see there are only two possible rules that could be used to conclude this judgment: *norm/lam* and *norm/neut*. So we can distinguish these to cases. In each case, we also know that the premise must hold to obtain the (known) conclusion.

This step in a proof is called *inversion* because we infer, at the metalevel at which we reason about our judgments, that the premise of a rule must

hold if the conclusion does. This is only valid if we consider all the possible cases, of which there are two in this particular situation. Often, there is only one, and sometimes there is none (which means that the case were are in is actually impossible).

Now that we have characterized normal forms, we will be able to prove a representation theorem for Booleans in the next lecture.

Exercises

Exercise 1 Fill in the blanks in the following typing judgments so the resulting judgment holds, or indicate there is no way to do so. You do not need to justify your answer or supply a typing derivation, and the types do not need to be "most general" in any sense. Remember that the function type constructor associates to the right, so that $\tau \to \sigma \to \rho = \tau \to (\sigma \to \rho)$.

(i) $\boxed{} \vdash y\, x : \alpha$

(ii) $\boxed{} \vdash x\, x : \boxed{}$

(iii) $\cdot \vdash \boxed{} : (\alpha \to \alpha) \to \alpha$

(iv) $\cdot \vdash (\lambda z.\, z)\, (\lambda x.\, \lambda y.\, \lambda p.\, p\, x\, y) : \boxed{}$

(v)

$$\cdot \vdash \lambda f.\, \lambda g.\, \lambda x.\, (f\, x)\, (g\, x)$$
$$: (\alpha \to \boxed{}) \to (\alpha \to \boxed{}) \to (\alpha \to \boxed{})$$

Since this is the first time we (that is, you) are proving theorems about judgments defined by rules, we ask you to be very explicit, as we were in the lectures and lecture notes. In particular:

- Explicitly state the overall structure of your proof: whether it proceeds by rule induction, and, if so, on the derivation of which judgment, or by structural induction, or by inversion, or just directly. If you need to split out a lemma for your proof, state it clearly and prove it separately. If you need to generalize your induction hypothesis, clearly state the generalized form.
- Explicitly list all cases in an induction proof. If a case is impossible, prove that is is impossible. Often, that's just inversion, but sometimes it is more subtle.

- Explicitly note any appeals to the induction hypothesis.
- Any appeals to inversion should be noted as such, as well as the rules that could have inferred the judgment we already know. This could lead to zero cases (a contradiction—the judgment could not have been derived), one case (there is exactly one rule whose conclusion matches our knowledge), or multiple cases, in which case your proof now splits into multiple cases.
- We recommend that you follow the line-by-line style of presentation where each line is justified by a short phrase. This will help you to check your proof and us to read and verify it.

Exercise 2 Prove that there does not exist an expression e such that $e \longrightarrow$ and e *normal*. In other words, the alternatives stated in Theorem **??** are exclusive.

As a reminder, the way we prove that a proposition A is false is to assume A is true and derive a contradiction. For those who care about such things, this is a perfectly valid intuitionistic (constructive) reasoning principle, as opposed to an *indirect proof*. The rule of indirect proof (which we should avoid at all cost, since all proofs in this course should be constructive) says that we can prove A is true by assuming that A is false and then deriving a contradiction from that.