Lecture Notes on Representation Theorems

15-814: Types and Programming Languages
Frank Pfenning

Lecture 4
September 13, 2018

1 Introduction

In the last lecture we considered how to formulate representation theorems for Booleans, namely that normal forms type $\alpha \to (\alpha \to \alpha)$ are either *true* $= \lambda x.\, \lambda y.\, x$ or *false* $= \lambda x.\, \lambda y.\, y$. But we did not have the tools to prove that yet, so will develop them in this lecture. The final theorem then will be the representation theorem and we will mention others.

In the last lecture we gave definition of e *nf* (e is a normal form) and $e \longrightarrow e'$ (e reduces to e') via inference rule to unlock the proof technique of rule induction. We start by completing the second property relating these two judgments. However, it turns out that our definition of normal form is awkward for the proof of representation theorems, so we give a new formulation in term of *well-typed normal terms* and *well-typed neutral terms*. These then form the basis for the representation theorem that concludes this lecture.

2 Reduction and Normal Form

Recall our definition of reduction from Lecture 3.

$$\frac{e \longrightarrow e'}{\lambda x.\, e \longrightarrow \lambda x.\, e'}\ \mathsf{red/lam} \qquad \frac{e_1 \longrightarrow e_1'}{e_1\, e_2 \longrightarrow e_1'\, e_2}\ \mathsf{red/app}_1 \qquad \frac{e_2 \longrightarrow e_2'}{e_1\, e_2 \longrightarrow e_1\, e_2'}\ \mathsf{red/app}_2$$

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

And our definition of normal forms:

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

$$\frac{e_1 \neq \lambda \_ \quad e_1\ \mathit{nf} \quad e_2\ \mathit{nf}}{e_1\, e_2\ \mathit{nf}}\ \mathsf{nf/app}$$

In Theorem L3.6 we established that normal forms do not reduce. Now we show the opposite, namely that expressions that do not reduce are normal forms.

Theorem 1 (Irreducible expressions are normal forms) *For all expressions e, if $e \not\longrightarrow$ then e nf.*

Proof: We would like to use our all-purpose tool of rule induction to prove this, but unfortunately that seems impossible since we have no derivation. All we know is that there is *no derivation* of the form $e \longrightarrow e'$.

So we turn to the related technique of *structural induction* over some (abstract) syntactic form, expressions in this case. There are three possibilities: variables x, λ-abstractions $\lambda x.\, e$, and applications $e_1\, e_2$. To prove that a property P holds for all expressions we have to show three propositions:

Case: P holds for variables x.

Case: P holds for $\lambda x.\, e$ assuming it holds for e.

Case: P holds for $e_1\, e_2$ assuming it holds for e_1 and e_2.

It turns out we do not use this technique very often, but it works well here.

Case: $e = x$. Then x *nf* by rule nf/var.

Case: $e = \lambda x.\, e_1$. We start our reasoning as follows:

> $\lambda x.\, e_1 \not\longrightarrow$ Assumption
> ...
> **To show:** $\lambda x.\, e_1$ *nf*
>
> In this situation is it easier to proceed if we develop the proof from the bottom up: $\lambda x.\, e_1$ *nf* if we could show e_1 *nf*.
>
> $\lambda x.\, e_1 \not\longrightarrow$ Assumption
> ...
> **To show:** e_1 *nf*
> $\lambda x.\, e_1$ *nf* By rule nf/lam

> *It is important to observe that this is **not** an application of inversion! In fact, we do **not** have a derivation of $\lambda x.\, e_1$ nf, but we want to* construct *one. In the completed proof (and already in this partial proof), this steps corresponds to an ordinary application of an inference rule (here* nf/lam*).*

Now e_1 *nf* would follow from the induction hypothesis if only we knew that $e_1 \not\longrightarrow$.

$\lambda x.\, e_1 \not\longrightarrow$	Assumption
...	
To show: $e_1 \not\longrightarrow$	
e_1 *nf*	By induction hypothesis on e_1
$\lambda x.\, e_1$ *nf*	By rule nf/lam

In order to show that e_1 does not reduce, we assume it reduces and derive a contradiction from it.

$\lambda x.\, e_1 \not\longrightarrow$	Assumption
$e_1 \longrightarrow e_1'$ for some e_1'	Assumption
...	
To show: Contradiction	
$e_1 \not\longrightarrow$	Since $e_1 \longrightarrow$ is contradictory
e_1 *nf*	By induction hypothesis on e_1
$\lambda x.\, e_1$ *nf*	By rule nf/lam

Now the final gap is easy to fill directly from an the inference rule red/lam.

$\lambda x.\, e_1 \not\longrightarrow$	Assumption
$e_1 \longrightarrow e_1'$ for some e_1'	Assumption
$\lambda x.\, e_1 \longrightarrow \lambda x.\, e_1'$	By rule red/lam
Contradiction	Since $\lambda x.\, e_1 \not\longrightarrow$ and $\lambda x.\, e_1 \longrightarrow$
$e_1 \not\longrightarrow$	Since $e_1 \longrightarrow$ is contradictory
e_1 *nf*	By induction hypothesis on e_1
$\lambda x.\, e_1$ *nf*	By rule nf/lam

Case: $e = e_1\, e_2$. Here we just show the completed proof, not the intermediate points.

$e_1\, e_2 \not\longrightarrow$	Assumption
$e_1 = \lambda x.\, e_3$	Assumption
$e_1\, e_2 = (\lambda x.\, e_3)\, e_2 \longrightarrow$	By rule beta
(1) $e_1 \neq \lambda \_$	By contradiction
$e_1 \longrightarrow e_1'$ for some e_1'	Assumption
$e_1\, e_2 \longrightarrow e_1'\, e_2$	By rule $\mathsf{red/app}_1$
$e_1 \not\longrightarrow$	By contradiction
(2) e_1 *nf*	By ind. hyp. on e_1
$e_2 \longrightarrow e_2'$ for some e_2'	Assumption
$e_1\, e_2 \longrightarrow e_1\, e_2'$	By rule $\mathsf{red/app}_x$
$e_2 \not\longrightarrow$	By contradiction
(3) e_2 *nf*	By ind. hyp. on e_2
$e_1\, e_2$ *nf*	By rule nf/app from lines (1), (2), (3)

□

3 Normal and Neutral Terms

The characterization of e *nf* is adequate, but it has a somewhat unpleasant condition $e \neq \lambda \_$. This makes it unnecessarily complicated if we want to use it as a basis to show the desired representation theorem for the Booleans. It is also somewhat difficult to generalize when we add more constructs to our language, so we give a slightly different characterization of the normal forms. Recall that a normal term has the form

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

which consists of a (possibly empty) sequence of λ-abstractions followed by a (possibly empty) sequence of applications where all the argument e_i are again normal. The terms of the form $((x\, e_1) \ldots e_k)$ are called *neutral* because applying them to another argument will not enable any reduction. In contrast, if we apply a normal term of the form $\lambda x.\, \_$ to an argument we can then perform a β-reduction, so it may "react" to its environment. We therefore have two judgments that mutually depend on each other, e *normal*

and e *neutral*.

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

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

You should convince yourself that it captures the intuition correctly.

$$\overbrace{\lambda x_1.\ \ldots\ \lambda x_n.\ \underbrace{((x\, e_1) \ldots e_k)}_{\text{neutral}}}^{\text{normal}}$$

You should also convince yourself (and this now requires some proof) that e *nf* if and only if e *normal* because we already proved that e *nf* coincides with $e \not\longrightarrow$. Alternatively, you could skip the intermediary and just prove that e *normal* iff $e \not\longrightarrow$.

We will forego such theorems here and leave them to Exercise **??**.

4 Typed Normal and Neutral Terms

Recall that our representation theorem expresses that $\vdash e : \alpha \to (\alpha \to \alpha)$ and e is a normal form, then $e =$ *true* or $e =$ *false*. Dealing with two judgments side by side, namely being well-typed and being normal, works out, but we can streamline our reasoning if we have a combined judgment. This combined judgment also has other applications later in the course.

As a first step, we just superimpose the typing judgment with the two judgments of e *normal* and e *neutral*. We use the following two judgments

$$\begin{array}{ll} \Gamma \vdash e \Leftarrow \tau & e \textit{ normal} \text{ and } \Gamma \vdash e : \tau \\ \Gamma \vdash e \Rightarrow \tau & e \textit{ neutral} \text{ and } \Gamma \vdash e : \tau \end{array}$$

These mutually depend on each other, because normal and neutral terms are also mutually dependent. Writing out rules blindly, by which we mean changing ":" to "$\Leftarrow$" for normal terms and "$\Rightarrow$" for neutral terms gives us the rules chk/lam, syn/var and syn/app below. In addition, we need one rule syn/chk that corresponds to the norm/neut rule expressing that every neutral

term is normal.

$$\frac{\Gamma, x_1 : \tau_1 \vdash e_2 \Leftarrow \tau_2}{\Gamma \vdash \lambda x_1.\, e_2 \Leftarrow \tau_1 \to \tau_2}\ \mathsf{chk/lam} \qquad \frac{\Gamma \vdash e \Rightarrow \tau}{\Gamma \vdash e \Leftarrow \tau}\ \mathsf{chk/syn}$$

$$\frac{x : \tau \in \Gamma}{\Gamma \vdash x \Rightarrow \tau}\ \mathsf{syn/var}$$

$$\frac{\Gamma \vdash e_1 \Rightarrow \tau_2 \to \tau_1 \quad \Gamma \vdash e_2 \Leftarrow \tau_2}{\Gamma \vdash e_1\, e_2 \Rightarrow \tau_1}\ \mathsf{syn/app}$$

Observe that these rules are no longer entirely syntax-directed, because of the chk/syn rule.

It turns that these rules are indeed a correct merge of the rules for typing and normality (see Exercise **??**). They have some other remarkable properties which we sketch in Section **??**. But first, let's use them to finally prove the representation theorem.

5 Representation on Well-Typed Normal Forms

Theorem 2 (Representation of Booleans, v4) *If* $\cdot \vdash e \Leftarrow \alpha \to (\alpha \to \alpha)$ *then* $e = \mathit{true} = \lambda x.\, \lambda y.\, x$ *or* $e = \mathit{false} = \lambda x.\, \lambda y.\, y$.

Proof: The proof proceeds by repeated inversions on the given derivation of the judgment $\cdot \vdash e \Leftarrow \alpha \to (\alpha \to \alpha)$. From this proof the need for Lemma **??** will emerge, but we present it in the order one might discover the lemma and the need for it.

$\cdot \vdash e \Leftarrow \alpha \to (\alpha \to \alpha)$ Given

Matching this judgment against the conclusion of the rules, we see that two could have been used to infer this: chk/lam and chk/syn. This is a use of *inversion* with two cases.

Case:

$$\frac{\cdot \vdash e \Rightarrow \alpha \to (\alpha \to \alpha)}{\cdot \vdash e \Leftarrow \alpha \to (\alpha \to \alpha)}\ \mathsf{chk/syn}$$

It turns out this case is impossible by Lemma **??**. Intuitively, this is because e is a neutral term so it must be $((x\, e_1) \ldots\ e_k)$, but the context is empty so there is no variable we can start with.

Case:

$$\frac{x : \alpha \vdash e_1 \Leftarrow \alpha \to \alpha}{\cdot \vdash \lambda x.\, e_1 \Leftarrow \alpha \to (\alpha \to \alpha)}\ \mathsf{chk/lam}$$

where $e = \lambda x.\, e_1$. Again we can apply inversion, with the same two rule being possible.

Subcase:

$$\frac{x : \alpha \vdash e_1 \Rightarrow \alpha \to \alpha}{x : \alpha \vdash e_1 \Leftarrow \alpha \to \alpha}\ \mathsf{chk/syn}$$

Again, this case is impossible by Lemma **??**. The intuitive reason is slightly different: the neutral term e_1 must now have the form $((x\, e_1') \ldots\, e_k')$ where the variable at the head is indeed the only available x. But that is not of function type, so it cannot be applied, but $e_1 \Rightarrow \alpha \to \alpha$ (which is a contradiction).

Subcase:

$$\frac{x : \alpha, y : \alpha \vdash e_2 \Leftarrow \alpha}{x : \alpha \vdash \lambda y.\, e_2 \Leftarrow \alpha \to \alpha}\ \mathsf{chk/lam}$$

where $e_1 = \lambda y.\, e_2$. Now we apply inversion again, and this time there is only one possibility.

$x : \alpha, y : \alpha \vdash e_2 \Rightarrow \alpha$

Once more we apply inversion, with two eligible rules.

Sub^{2}case:

$$\frac{z : \alpha \in (x : \alpha, y : \alpha)}{x : \alpha, y : \alpha \vdash z \Rightarrow \alpha}\ \mathsf{syn/var}$$

where $e_2 = z$. We see the only possibilities are $z = x$ or $z = y$. Collecting the equalities: $e = \lambda x.\, e_1 = \lambda x.\, \lambda y.\, e_2 = \lambda x.\, \lambda y.\, z$ where $z = x$ or $z = y$, which is what we wanted to show. Now we just need to know that the very last sub^{2}case is impossible.

Sub^{2}case:

$$\frac{x : \alpha, y : \alpha \vdash e_3 \Rightarrow \tau \to \alpha \quad x : \alpha, y : \alpha \vdash e_4 \Leftarrow \tau}{x : \alpha, y : \alpha \vdash e_3\, e_4 \Rightarrow \alpha}\ \mathsf{syn/app}$$

for some τ, where $e_2 = e_3\, e_4$ and for some e_3 and e_4. This is indeed impossible, rigorously by Lemma **??**, and informally because now e_3 is neutral and the variable at its head must be either x or y (both of which have type α).

□

The following lemma provides a rigorous argument about the possible forms and types of neutral terms if all variables have some variable type α_i.

Lemma 3 (Neutral Terms)
If $x_1 : \alpha_1, \ldots, x_n : \alpha_n \vdash e \Rightarrow \tau$ then $e = x_i$ and $\tau = \alpha_i$ for some $0 \leq i \leq n$.

Proof: By rule induction on $x_1 : \alpha_1, \ldots, x_n : \alpha_n \vdash e \Rightarrow \tau$. There are just two cases.

Case:

$$\frac{z : \tau \in (x_1 : \alpha_1, \ldots, x_n : \alpha_n)}{x_1 : \alpha_1, \ldots, x_n : \alpha_n \vdash z \Rightarrow \tau}\ \mathsf{syn/var}$$

where $e = z$. Then $z = x_i$ and $\tau = \alpha_i$ for some i as required.

Case:

$$\frac{x_1 : \alpha_1, \ldots, x_n : \alpha_n \vdash e_1 \Rightarrow \tau_2 \to \tau \quad x_1 : \alpha_1, \ldots, x_n : \alpha_n \vdash e_2 \Leftarrow \tau_2}{x_1 : \alpha_1, \ldots, x_n : \alpha_n \vdash e_1\, e_2 \Rightarrow \tau}\ \mathsf{syn/app}$$

where $e = e_1\, e_2$. By induction hypothesis on the first premise, we have $e_1 = x_i$ and $\tau_2 \to \tau = \alpha_i$. But that's a contraction, since α_i is a variable, not a function type.

□

For some representation theorems we need to also allow some function types in the context and we obtain other, more complex characterizations of the neutral terms and their types.

6 Algorithmic Interpretation of Rules

In the theory of programming language we have a pervasive habit to present just about anything via inference rules. The typing judgment for normal forms is an excellent example of that: It also describes an algorithm for

type-checking! This, like much in programming languages, takes some time to understand and absorb. Let's talk through it.

To execute and algorithm encapsulated in the rules means to proceed by bottom-up proof construction. We match the desired judgments to the conclusion and then fill in the premises. Then we continue by trying to derive the premises. We succeed when the whole derivation is filled in and we can read off the answer (which varies with the problem, of course). Sometimes, more than one rule applies and we have to try both in turn. Sometimes, no rule applies and we fail. In case we had earlier choices, we then backtrack to the most recent choice. If not, or if all choices have been exhausted, we just fail and conclude the judgment is not derivable.

The algorithmic interpretation of the typing judgment for normal and neutral terms is the following:

$\Gamma \vdash e \Leftarrow \tau$**:** Given Γ, e, and τ, determine whether or not the given judgment holds. Success indicates that $\Gamma \vdash e \Leftarrow \tau$ holds, failure that either e is not normal or it does not have the given type τ. We say *expression e is checked against type τ*.

$\Gamma \vdash e \Rightarrow \tau$**:** Given Γ and e, calculate a type τ such that the given judgment holds or fail. Success yields that $\Gamma \vdash e \Rightarrow \tau$ holds for the τ we calculated, failure that there is no suitable type e (either e is not neutral, or it does not have any type at all). We say *expression e synthesizes type τ*.

We illustrate the algorithm by verifying that $\overline{1} = \lambda s.\, \lambda z.\, s\, z \Leftarrow (\alpha \to \alpha) \to (\alpha \to \alpha)$. We execute it step by step and write $\boxed{?}$ wherever the algorithm still has to compute an answer. If this answer is needed elsewhere we give it a name, as in $\boxed{\tau?}$ which stands for "*some type τ*".

At the beginning, we only know the judgment, but nothing yet about the derivation (which may or may not exist, as far as we know at this point).

$$\boxed{?}$$
$$\cdot \vdash \lambda s.\, \lambda z.\, s\, z \Leftarrow (\alpha \to \alpha) \to (\alpha \to \alpha)$$

Two rules could apply (chk/syn and chk/lam), but $\lambda s.\_$ cannot synthesize a type so chk/syn fails in the next step and we focus on chk/lam.

$$\dfrac{\boxed{?}}{\cdot \vdash \lambda s.\, \lambda z.\, s\, z \Leftarrow (\alpha \to \alpha) \to (\alpha \to \alpha)}\ \mathsf{chk/lam}$$
$$s : \alpha \to \alpha \vdash \lambda z.\, s\, z \Leftarrow \alpha \to \alpha$$

As before, only one possibility remains

$$\dfrac{\dfrac{\dfrac{\boxed{?}}{s : \alpha \to \alpha, z : \alpha \vdash s\ z \Leftarrow \alpha}\ \mathsf{chk/lam}}{s : \alpha \to \alpha \vdash \lambda z.\, s\ z \Leftarrow \alpha \to \alpha}\ \mathsf{chk/lam}}{\cdot \vdash \lambda s.\, \lambda z.\, s\ z \Leftarrow (\alpha \to \alpha) \to (\alpha \to \alpha)}$$

At this point, only a single rule would have this conclusion, so we continue with our construction.

$$\dfrac{\dfrac{\dfrac{\dfrac{\boxed{?}}{s : \alpha \to \alpha, z : \alpha \vdash s\ z \Rightarrow \alpha}\ \mathsf{chk/syn}}{s : \alpha \to \alpha, z : \alpha \vdash s\ z \Leftarrow \alpha}\ \mathsf{chk/lam}}{s : \alpha \to \alpha \vdash \lambda z.\, s\ z \Leftarrow \alpha \to \alpha}\ \mathsf{chk/lam}}{\cdot \vdash \lambda s.\, \lambda z.\, s\ z \Leftarrow (\alpha \to \alpha) \to (\alpha \to \alpha)}$$

Now it becomes somewhat interesting. The only rule that could have this conclusion is syn/app. Recall that we interpret $\Gamma \vdash e \Rightarrow \tau$ as saying that Γ and e as are given and we construct the τ, if it exists.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\boxed{?}}{s : \alpha \to \alpha, z : \alpha \vdash s \Rightarrow \boxed{\tau?}} \quad \boxed{\tau? = \sigma? \to \alpha} \quad \dfrac{\boxed{?}}{s : \alpha \to \alpha, z : \alpha \vdash z \Leftarrow \boxed{\sigma?}}}{s : \alpha \to \alpha, z : \alpha \vdash s\ z \Rightarrow \alpha}\ \mathsf{syn/app}}{s : \alpha \to \alpha, z : \alpha \vdash s\ z \Leftarrow \alpha}\ \mathsf{chk/syn}}{s : \alpha \to \alpha \vdash \lambda z.\, s\ z \Leftarrow \alpha \to \alpha}\ \mathsf{chk/lam}}{\cdot \vdash \lambda s.\, \lambda z.\, s\ z \Leftarrow (\alpha \to \alpha) \to (\alpha \to \alpha)}\ \mathsf{chk/lam}$$

Now we note that we cannot yet start on the second premise of the syn/app rule because we do not yet know $\boxed{\sigma?}$. So we work on the first premise. That's possible because it will return to us the type $\boxed{\tau?}$ of s, if it exists, and from that we can determine $\boxed{\tau?}$ and then in turn $\boxed{\sigma?}$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\boxed{\tau? = \alpha \to \alpha}}{s : \alpha \to \alpha, z : \alpha \vdash s \Rightarrow \alpha \to \alpha}\ \mathsf{syn/var}} \quad \boxed{\tau? = \sigma? \to \alpha} \quad \dfrac{\boxed{?}}{s : \alpha \to \alpha, z : \alpha \vdash z \Leftarrow \boxed{\sigma?}}}{s : \alpha \to \alpha, z : \alpha \vdash s\ z \Rightarrow \alpha}\ \mathsf{syn/app}}{s : \alpha \to \alpha, z : \alpha \vdash s\ z \Leftarrow \alpha}\ \mathsf{chk/syn}}{s : \alpha \to \alpha \vdash \lambda z.\, s\ z \Leftarrow \alpha \to \alpha}\ \mathsf{chk/lam}}{\cdot \vdash \lambda s.\, \lambda z.\, s\ z \Leftarrow (\alpha \to \alpha) \to (\alpha \to \alpha)}\ \mathsf{chk/lam}$$

Now that the algorithm has determined all the unknowns in the first premise, we can solve the equations, determine that $\boxed{\sigma? = \alpha}$ and proceed with the second premise.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{s : \alpha \to \alpha, z : \alpha \vdash s \Rightarrow \alpha \to \alpha}\,\textsf{syn/var} \quad \boxed{?}\ s : \alpha \to \alpha, z : \alpha \vdash z \Leftarrow \alpha}{s : \alpha \to \alpha, z : \alpha \vdash s\, z \Rightarrow \alpha}\,\textsf{syn/app}}{s : \alpha \to \alpha, z : \alpha \vdash s\, z \Leftarrow \alpha}\,\textsf{chk/syn}}{s : \alpha \to \alpha \vdash \lambda z.\, s\, z \Leftarrow \alpha \to \alpha}\,\textsf{chk/lam}}{\cdot \vdash \lambda s.\, \lambda z.\, s\, z \Leftarrow (\alpha \to \alpha) \to (\alpha \to \alpha)}\,\textsf{chk/lam}$$

Again, only one rule applies.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{s : \alpha \to \alpha, z : \alpha \vdash s \Rightarrow \alpha \to \alpha}\,\textsf{syn/var} \quad \dfrac{\boxed{?}}{s : \alpha \to \alpha, z : \alpha \vdash z \Rightarrow \boxed{\rho?}} \quad \boxed{\rho? = \alpha}}{s : \alpha \to \alpha, z : \alpha \vdash z \Leftarrow \alpha}\,\textsf{chk/syn}}{s : \alpha \to \alpha, z : \alpha \vdash s\, z \Rightarrow \alpha}\,\textsf{syn/app}}{s : \alpha \to \alpha, z : \alpha \vdash s\, z \Leftarrow \alpha}\,\textsf{chk/syn}}{s : \alpha \to \alpha \vdash \lambda z.\, s\, z \Leftarrow \alpha \to \alpha}\,\textsf{chk/lam}}{\cdot \vdash \lambda s.\, \lambda z.\, s\, z \Leftarrow (\alpha \to \alpha) \to (\alpha \to \alpha)}\,\textsf{chk/lam}$$

Now in the last step we can synthesize the type ρ for z and compare it to α—fortunately it matches with $\rho? = \alpha$ and this run of the algorithm is successful and complete.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{s : \alpha \to \alpha, z : \alpha \vdash s \Rightarrow \alpha \to \alpha}\,\textsf{syn/var} \quad \dfrac{\dfrac{}{s : \alpha \to \alpha, z : \alpha \vdash z \Rightarrow \alpha}\,\textsf{syn/var}}{s : \alpha \to \alpha, z : \alpha \vdash z \Leftarrow \alpha}\,\textsf{chk/syn}}{s : \alpha \to \alpha, z : \alpha \vdash s\, z \Rightarrow \alpha}\,\textsf{syn/app}}{s : \alpha \to \alpha, z : \alpha \vdash s\, z \Leftarrow \alpha}\,\textsf{chk/syn}}{s : \alpha \to \alpha \vdash \lambda z.\, s\, z \Leftarrow \alpha \to \alpha}\,\textsf{chk/lam}}{\cdot \vdash \lambda s.\, \lambda z.\, s\, z \Leftarrow (\alpha \to \alpha) \to (\alpha \to \alpha)}\,\textsf{chk/lam}$$

So the rules, read as an algorithm, tell us that $\cdot \vdash \lambda s.\, \lambda z.\, s\, z \Leftarrow (\alpha \to \alpha) \to (\alpha \to \alpha)$. The derivation we construct while we run the algorithm serves as explicit evidence of this fact.

Exercises

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.

Exercise 1 If we have two judgments defined simultaneously (like e *normal* and e *neutral* we often need to prove properties about them by *simultaneous induction*. In simultaneous induction you have multiple induction hypotheses and if the premise of a rule comes from a different judgment, you may apply the appropriate induction hypothesis to it. In proving properties **??** and **??** below, make a note if you needed a simple or a simultaneous induction.

1. In each case below, give an example of an expressions e and type τ with $\cdot \vdash e : \tau$ and also the stated property, or indicate no such expression and type exist. You do not need to justify your answer further (no need for typing derivations or proofs).

   (i) $\cdot \vdash e \Leftarrow \tau$ and also $\cdot \vdash e \Rightarrow \tau$

   (ii) $\cdot \vdash e \Leftarrow \tau$ but not $\cdot \vdash e \Rightarrow \tau$

   (iii) $\cdot \vdash e \Rightarrow \tau$ but not $\cdot \vdash e \Leftarrow \tau$

(iv) Neither $\cdot \vdash e \Leftarrow \tau$ nor $\cdot \vdash e \Rightarrow \tau$

2. Prove that the bidirectional typing rules are *sound*, that is, we verify or synthesize only correct types.

   (i) If $\Gamma \vdash e \Leftarrow \tau$ then $\Gamma \vdash e : \tau$ and e *normal*.

   (ii) If $\Gamma \vdash e \Rightarrow \tau$ then $\Gamma \vdash e : \tau$ and e *neutral*.

3. Prove that the bidirectional rules are *complete*, that is, we can verify or infer any correct type.

   (i) If $\Gamma \vdash e : \tau$ and e *normal* then $\Gamma \vdash e \Leftarrow \tau$.

   (ii) If $\Gamma \vdash e : \tau$ and e *neutral* then $\Gamma \vdash e \Rightarrow \tau$.

Exercise 2 Prove the following theorems.

1. If e *nf* then e *normal*.

2. If e *normal* then e *nf*.

Because the judgment e *normal* is defined simultaneously with e *neutral*, you may have to generalize some of the statements before you can prove them by simultaneous induction (see Exercise **??**).