Types and Programming Languages (15-814)
Fall 2018
Assignment 1: Having Fun With Induction

Contact: 15-814 Course Staff

Due Tuesday, September 25, 2018

This assignment is due at the beginning of class on the above date and it must be submitted electronically as a PDF file on Canvas. As before, problems marked "WB" are subject to the whiteboard policy; all other problems must be done individually.

Normal forms

Terms in normal form should denote *values* or results of a computation. A essential condition for something to be deemed a value is that it no longer be reducible. Recall our judgments N nf and R neutral from class for normal forms and neutral expressions, respectively. They were inductively defined by the rules:

$$\frac{N \text{ nf}}{\lambda x.N \text{ nf}} \text{ (NF-LAM)} \qquad \frac{R \text{ neutral}}{R \text{ nf}} \text{ (NF-NE)}$$

$$\frac{}{x \text{ neutral}} \text{ (NE-VAR)} \qquad \frac{R \text{ neutral} \quad N \text{ nf}}{RN \text{ neutral}} \text{ (NE-APP)}$$

For your reference, recall that the β-reduction relation $\rightarrow_\beta$ is inductively defined by the rules:

$$\frac{e \rightarrow_\beta e'}{\lambda x.e \rightarrow_\beta \lambda x.e'} \text{ (B-LAM)} \qquad \frac{}{(\lambda x.e_1)e_2 \rightarrow_\beta [e_2/x]e_1} \text{ (B-RED)}$$

$$\frac{e_1 \rightarrow_\beta e_1'}{e_1 e_2 \rightarrow_\beta e_1' e_2} \text{ (B-L)} \qquad \frac{e_2 \rightarrow_\beta e_2'}{e_1 e_2 \rightarrow_\beta e_1 e_2'} \text{ (B-R)}$$

We may write $e \rightarrow_\beta$ to mean there exists some e' such that $e \rightarrow_\beta e'$. When no such e' exists, we write $e \not\rightarrow_\beta$.

Task 1 (15 pt, WB). Show that for all e, either e nf or $e \rightarrow_\beta$ (but not both).

Hint. Proceed by induction on the syntax of e. Split the case that e is an application $e_1 e_2$ into the two following subcases: e_1 is $\lambda x.e_1'$, and e_1 is x or an application.

Bidirectional type checking

Our goal is to show that type checking is decidable for normal forms[1] in the simply-typed λ-calculus. To do so, we will use a *bidirectional type checking algorithm*. We describe this algorithm using inference rules whose judgments are interpreted as having "inputs" and "outputs".

The algorithm uses the following two new judgments:

- $\Gamma \vdash N \Leftarrow \tau$ says that normal form N checks against type τ under Γ,
- $\Gamma \vdash R \Rightarrow \tau$ says that neutral expression R synthesizes type τ under Γ.

When using the synthesis judgments below, we read Γ and R as inputs and τ as an output. In the checking judgments, Γ, N, and τ should all be seen as inputs. For legibility, we have colour-coded inputs and outputs below.

Throughout, we assume our contexts are well-formed, that is, that Γ is finite list $x_1 : \tau_1, \ldots, x_n : \tau_n$ with $n \geq 0$ and $x_i \neq x_j$ for $1 \leq i \neq j \leq n$. In this case, let $\mathrm{dom}(\Gamma) = \{x_1, \ldots, x_n\}$. We assume throughout the judgments are well-formed, that is, that $\mathrm{fv}(e) \subseteq \mathrm{dom}(\Gamma)$.

The algorithm is inductively defined by the following rules:

$$\frac{x : \tau \in \Gamma}{\Gamma \vdash x \Rightarrow \tau}\ (\textsc{Var}) \qquad \frac{\Gamma, x : \tau_1 \vdash N \Leftarrow \tau_2}{\Gamma \vdash \lambda x.N \Leftarrow \tau_1 \to \tau_2}\ (\textsc{Lam})$$

$$\frac{\Gamma \vdash R \Rightarrow \tau_1 \to \tau_2 \quad \Gamma \vdash N \Leftarrow \tau_1}{\Gamma \vdash RN \Rightarrow \tau_2}\ (\textsc{App}) \qquad \frac{\Gamma \vdash R \Rightarrow \tau' \quad \tau = \tau'}{\Gamma \vdash R \Leftarrow \tau}\ (\textsc{CS})$$

To better understand these judgments, we recommend working through a few examples on your own. To help you get started, we have worked one out for you. Abbreviate $f : \alpha \to \alpha \to \alpha, y : \alpha$ by Γ. To type check $\cdot \vdash \lambda f.\lambda y.fy \Leftarrow (\alpha \to \alpha) \to \alpha \to \alpha$, the algorithm performs the following derivation, bottom-up. When it reaches the (App) rule, it applies the (Var) rule to f to synthesize the type $\alpha \to \alpha$ (an output). From this, we know the type of y must be α if we are to apply f to y, so we check y against α in Γ. We succeed in completing the derivation

[1] Showing decidability for reducible terms requires adding type annotations in key places and may be covered in class at a later date.

and conclude that the type is correct.

$$\dfrac{\dfrac{\dfrac{\dfrac{f : \alpha \to \alpha \in \Gamma}{\Gamma \vdash f \Rightarrow \alpha \to \alpha}\,(\textsc{Var}) \quad \dfrac{\dfrac{y : \alpha \in \Gamma}{\Gamma \vdash y \Rightarrow \alpha}\,(\textsc{Var}) \quad \alpha = \alpha}{\Gamma \vdash y \Leftarrow \alpha}\,(\textsc{CS})}{\Gamma \vdash fy \Leftarrow \alpha}\,(\textsc{App})}{f : \alpha \to \alpha \vdash \lambda y.fy \Leftarrow \alpha \to \alpha}\,(\textsc{Lam})}{\vdash \lambda f.\lambda y.fy \Leftarrow (\alpha \to \alpha) \to \alpha \to \alpha}\,(\textsc{Lam})$$

As a programmer, it is often useful to get an "error message" when we have a type mismatch. For this reason, we introduce the following two judgments:

- $\Gamma \vdash N \not\Leftarrow \tau$ says that N does not check with type τ under Γ,
- $\Gamma \vdash R \not\Rightarrow$ says that R does not synthesize a type under Γ.

They are inductively defined by the following rules:

$$\frac{x \notin \mathrm{dom}(\Gamma)}{\Gamma \vdash x \not\Rightarrow}\,(\#\textsc{Var})$$

$$\frac{\Gamma, x : \tau_1 \vdash N \not\Leftarrow \tau_2}{\Gamma \vdash \lambda x.N \not\Leftarrow \tau_1 \to \tau_2}\,(\#\textsc{Lam}) \qquad \frac{}{\Gamma \vdash \lambda x.N \not\Leftarrow c}\,(\#\textsc{LamC})$$

$$\frac{\Gamma \vdash R \not\Rightarrow}{\Gamma \vdash RN \not\Rightarrow}\,(\#\textsc{App}) \qquad \frac{\Gamma \vdash R \Rightarrow c}{\Gamma \vdash RN \not\Rightarrow}\,(\#\textsc{App}')$$

$$\frac{\Gamma \vdash R \Rightarrow \tau_1 \to \tau_2 \quad \Gamma \vdash N \not\Leftarrow \tau_1}{\Gamma \vdash RN \not\Rightarrow}\,(\#\textsc{App}'')$$

$$\frac{\Gamma \vdash R \Rightarrow \tau' \quad \tau \neq \tau'}{\Gamma \vdash R \not\Leftarrow \tau}\,(\#\textsc{CS}) \qquad \frac{\Gamma \vdash R \not\Rightarrow}{\Gamma \vdash R \not\Leftarrow \tau}\,(\#\textsc{CS}')$$

Task 2 (5 points)**.** If the following proposition is true, prove it, otherwise, find a counter-example:

If $\Gamma \vdash N \Leftarrow \tau$ and $\Gamma \vdash N \Leftarrow \tau'$, then $\tau = \tau'$.

Theorem 1 (Decidability of bidirectional type checking)**.**

1. Given Γ, N, and τ, either $\Gamma \vdash N \Leftarrow \tau$ or $\Gamma \vdash N \not\Leftarrow \tau$ (but not both).
2. Given Γ and R, either there exists a unique τ such that $\Gamma \vdash R \Rightarrow \tau$ or $\Gamma \vdash R \not\Rightarrow$ (but not both).

The proof of this theorem uses a mutual induction on the structures of N and R. We need a mutual induction because normal forms can contain neutral terms, e.g., $N = x$, and neutral terms can contain normal forms, e.g., $R = R'N$. This induction requires care when we reach the case of N is R and consider the rule (CS), because we apply the induction hypothesis for synthesis to the same term whose type we are checking. This does not affect the inductive character of the proof or introduce any circularity in the argument, because whenever we apply the induction hypothesis for checkable terms in synthesis cases, we only do so to strictly smaller subterms.

Proof. By mutual induction on the structure of N and R.

Case N is $\lambda x.N$ and τ is some c. There are no rules with a conclusion of the form $\Gamma \vdash \lambda x.N \Leftarrow c$. We conclude $\Gamma \vdash \lambda x.N \not\Leftarrow c$ by (#LAMC).

Case N is $\lambda x.N$ and τ is of the form $\tau_1 \to \tau_2$. By task 3.

Case N is R. The only rules forming a judgment of the form $\Gamma \vdash R \Leftarrow \tau$ or $\Gamma \vdash R \not\Leftarrow \tau$ are (CS), (#CS), and (#CS'). Applying the induction hypothesis to R, either there exists a unique τ' such that $\Gamma \vdash R \Rightarrow \tau'$ or $\Gamma \vdash R \not\Rightarrow$ (but not both). Assume first there exists a τ', then (#CS') is not applicable. If $\tau = \tau'$, then (#CS) is not applicable and we are done by (CS). If $\tau \neq \tau'$, then (CS) is not applicable and we are done by (#CS). Now assume $\Gamma \vdash R \not\Rightarrow$. Then (CS) and (#CS) are not applicable and we are done by (#CS').

Case R is x. By task 3.

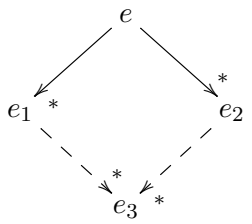
Case R is RN. The only rules with a conclusion of the form $\Gamma \vdash RN \Rightarrow \tau$ or $\Gamma \vdash RN \not\Rightarrow$ are (APP), (#APP), (#APP'), and (#APP''). Applying the induction hypothesis to R, either there exists a unique τ' such that $\Gamma \vdash R \Rightarrow \tau'$ or $\Gamma \vdash R \not\Rightarrow$ (but not both). If $\Gamma \vdash R \not\Rightarrow$, then the only applicable rule is (#APP), by which $\Gamma \vdash RN \not\Rightarrow$. Now assume there exists a τ'. If $\tau' = c$, then the only applicable rule is (#APP'), by which $\Gamma \vdash RN \not\Rightarrow$. Now assume $\tau' = \tau_1 \to \tau_2$. By the induction hypothesis on N, either $\Gamma \vdash N \Leftarrow \tau_1$ or $\Gamma \vdash N \not\Leftarrow \tau_1$ (but not both). The only applicable rules in these two cases are (APP) and (#APP''), respectively, by which $\Gamma \vdash RN \Rightarrow \tau_2$ or $\Gamma \vdash RN \not\Rightarrow$ (but not both). □

Task 3 (10 points, WB)**.** Complete the following cases in the proof of decidability of bidirectional type checking.

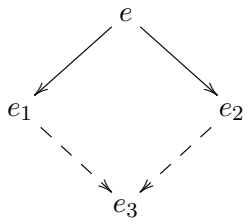
1. N is $\lambda x.N$ and τ is of the form $\tau_1 \to \tau_2$, and
2. R is x.

The Church-Rosser Theorem

The Church-Rosser theorem says that β-reduction for the untyped λ-calculus is *confluent*. A relation $\rightarrow$ is confluent if whenever $e \rightarrow^* e_1$ and $e \rightarrow^* e_2$, there exists an e_3 such that $e_1 \rightarrow^* e_3$ and $e_2 \rightarrow^* e_3$:



This is an instance of the *diamond property*. We say that a relation $\rightarrow$ satisfies the diamond property if whenever $e \rightarrow e_1$ and $e \rightarrow e_2$, then there exists an e_3 such that $e_1 \rightarrow e_3$ and $e_2 \rightarrow e_3$:



We have used the Church-Rosser theorem several times in class, and we have received questions on how to prove the theorem. In this section, we have broken up a proof by Tait and Martin-Lf into manageable pieces. The high-level overview of the proof is as follows.

1. We define a new reduction relation $\rightarrow_l$ that simultaneously reduces a function and its argument.
2. We show that $\rightarrow_l$ is confluent.
3. We show that if a reduction is confluent, then so is its reflexive, transitive closure.
4. We show that $\rightarrow_l^* = \rightarrow_\beta^*$.

We conclude that $\rightarrow_\beta^*$ is confluent, i.e., the Church-Rosser theorem holds.

Let $\rightarrow_l$ be the relation on λ-terms inductively defined by the following rules:

$$\frac{}{e \rightarrow_l e} \text{ (L-REFL)} \qquad \frac{e \rightarrow_l e'}{\lambda x.e \rightarrow_l \lambda x.e'} \text{ (L-LAM)}$$

$$\frac{e_1 \to_l e_1' \quad e_2 \to_l e_2'}{e_1 e_2 \to_l e_1' e_2'}\ (\text{L-APP}) \qquad \frac{e_1 \to_l e_1' \quad e_2 \to_l e_2'}{(\lambda x.e_1) e_2 \to_l [e_2'/x] e_1'}\ (\text{L-RED})$$

Task 4 (5 points, WB)**.** Prove that:

1. $\lambda x.e_x \to_l e'$ implies $e' \equiv \lambda x.e_x'$ with $e_x \to_l e_x'$,
2. $e_1 e_2 \to_l e'$ implies either
   - $e' \equiv e_1' e_2'$ with $e_i \to_l e_i'$ for $i = 1, 2$, or
   - $e_1 \equiv \lambda x.e_x$, $e' \equiv [e_2'/x]e_x'$, $e_x \to_l e_x'$, and $e_2 \to_l e_2'$.

Theorem 2. The relation $\to_l$ is confluent.

Proof (sketch). By task 6, it is sufficient to show that $\to_l$ satisfies the diamond property. We must show that for all e_0, e_1, and e_2 such that $e_0 \to_l e_1$ and $e_0 \to_l e_2$, there exists an e_3 such that $e_1 \to_l e_3$ and $e_2 \to_l e_3$. The proof is by induction on $e \to_l e_1$ to show that for all $e \to_l e_2$ there is an e_3. □

Transitive closure of confluent relations

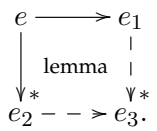
The reflexive, transitive closure $\to^*$ of a relation $\to$ is inductively defined by the rules:

$$\frac{}{e \to^* e}\ (\text{T-REFL}) \qquad \frac{e_1 \to e_2 \quad e_2 \to^* e_3}{e_1 \to^* e_3}\ (\text{T-STEP})$$

Task 5 (5 points, WB)**.** Show that $\to^*$ is actually transitive, i.e., that if $e_1 \to^* e_2$ and $e_2 \to^* e_3$, then $e_1 \to^* e_3$. Conclude that $(\to^*)^* = \to^*$.

Task 6 (10 points, WB)**.** Show that if $\to$ satisfies the diamond property, then so does its reflexive, transitive closure $\to^*$.

Hint. Prove and use the "strip lemma": if $e \to e_1$ and $e \to^* e_2$, then there exists an e_3 such that $e_1 \to^* e_3$ and $e_2 \to e_3$. Pictorially, this amounts to showing:



Then, "paste" these "strips" together to get the big confluence rectangle.

Concluding Church-Rosser

Task 7 (10 points, WB)**.** Show $\rightarrow_\beta^*$ is the reflexive, transitive closure of $\rightarrow_l$. To do so, (informally) prove that:

1. if $e \rightarrow_\beta e'$, then $e \rightarrow_l e'$,
2. if $e \rightarrow_l e'$, then $e \rightarrow_\beta^* e'$.

Then explain why it follows that $\rightarrow_l^* = \rightarrow_\beta^*$, i.e., $e \rightarrow_l^* e'$ if and only if $e \rightarrow_\beta^* e'$.

Hint. You will need to use task 5.