Assignment 6
Language Extensions

15-814: Types and Programming Languages
Frank Pfenning & David M Kahn

Due Thursday, October 21, 2021
75 points

This assignment is due on the above date at 11:59 pm and it must be submitted electronically on Gradescope. You should hand in one file

- `hw06.pdf` with your written solutions to the questions

1 Bidirectional Typechecking

Task 1 (L11.8, 35 points) Often we define two (or more) judgments simultaneously in a mutually-recursive fashion, like *neutral* and *normal* from lambda calculus, or the verification judgment $\Leftarrow$ and synthesizing judgment $\Rightarrow$ from bidirectional typechecking. When judgments are defined this way, we often need to prove properties about them by *simultaneous induction*. In simultaneous induction you have multiple induction hypotheses that mutually help to prove one another.

In this task, we will explore the relation between our usual typing judgment and our new ones for bidrectional typechecking, and may make use of simultaneous induction along the way. Put your answers in `hw06.pdf`.

We refer to an expression in the purely functional fragment $e ::= x \mid \lambda x.\, e \mid e_1\, e_2 \mid \lambda x{:}\tau.e$ as *annotation-free* if it does not contain an abstraction with an explicit type.

1. In each case below, give an example of an **annotation-free** expression e and type τ satisfying the stated property and $\cdot \vdash e : \tau$, or indicate none 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 rules are *complete* for normal forms in the simply-typed lambda calculus that is, we can verify or infer any correct type (for **annotation-free** *normal* or *neutral* expressions). Make a note stating whether you can prove each statement individually, or if you must prove them both at once with simultaneous induction. Clearly state what you are inducting over.

(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 $\Gamma \vdash e \Rightarrow \tau$.

2 Internalizing Definitions

Task 2 (L11.5, 20 points) We can *internalize* definitions as part of the core language. Specifically, we add

$$\begin{array}{lrcl} \text{Expressions} & e & ::= & \mathsf{exp}\ f : \tau = e\ \mathsf{in}\ e' \\ & & \mid & \mathsf{val}\ x = e\ \mathsf{in}\ e' \\ & & \mid & \ldots \end{array}$$

where $\mathsf{exp}\ f : \tau = e\ \mathsf{in}\ e'$ internalizes the definition of an expression variable f with scope e', and $\mathsf{val}\ x = e\ \mathsf{in}\ e'$ internalizes evaluation of e and binding x to the resulting value with scope e'.

1. Extend the rules for evaluation of expressions to account for the two new constructs.

2. Extend the rules for the typing judgment.

3. Extend the rules for the checking and synthesis judgments.

Our key language properties, namely preservation, progress, finality of values, and determinacy should continue to hold, but you do not need to prove them.

3 Corecursive Types

Task 3 (20 points) In the midterm exam, we introduced corecursive types, the lazy version of recursive types, using the rules below:

$$\frac{}{\mathsf{roll}\ e\ \mathit{value}}\ \mathsf{val/roll} \qquad \frac{}{\mathsf{unroll}\ (\mathsf{roll}\ e) \mapsto e}\ \mathsf{step/unroll/roll} \qquad \frac{e \mapsto e'}{\mathsf{unroll}\ e \mapsto \mathsf{unroll}\ e'}\ \mathsf{step/unroll}_1$$

$$\frac{\Gamma \vdash e : [\nu\alpha.\,\tau/\alpha]\tau}{\Gamma \vdash \mathsf{roll}\ e : \nu\alpha.\,\tau}\ \mathsf{tp/roll} \qquad \frac{\Gamma \vdash e : \nu\alpha.\,\tau}{\Gamma \vdash \mathsf{unroll}\ e : [\nu\alpha.\,\tau/\alpha]\tau}\ \mathsf{tp/unroll}$$

In particular, we defined streams of naturals with $\mathit{stream} = \nu\alpha.\,\mathit{nat} \times \alpha$. We will now explore how these streams stack up against other types we are more familiar with.

For each of the following types, either provide functions *forth* and *back* witnessing an isomorphism between that type and the type of streams, or conjecture that they are not isomorphic and explain why. You do not need to prove that the functions witness an isomorphism or none exists. Please provide your answers in the written `hw06.pdf`.

1. The recursive cousin of these streams, $\mu\alpha.\,\mathit{nat} \times \alpha$.

2. The function type $\mathit{nat} \to \mathit{nat}$.