

Homework 2: Proofs as Programs, Fixed-Points and Hypothetical Judgments

15-814: Types and Programming Languages
TA: Bernardo Toninho (btoninho@cs.cmu.edu)

Out: 09/27/11
Due: 10/11/11

This assignment is due before class on 10/11/11.

Edit 10/04: The descriptions of Task 8 and 9 have changed. The previous version didn't quite accurately describe what was intended.

Edit 10/09: There was a missing overbar in the definition of nm .

1 Combinators and Hilbert Systems

In class, we have seen the correspondence between the λ -calculus and propositional logic in natural deduction style. An alternative way of presenting logic is through what are typically called Hilbert systems. A Hilbert system consists of a (usually large) set of axiom schemas and a reduced number of inference rules (opposed to natural deduction systems where we typically have a small number of axioms and a large number of inference rules).

In this exercise, we will develop the connection of Hilbert systems and natural deduction. Recall the rules for natural deduction:

$$\frac{}{\Gamma, A \vdash A} \text{ (hyp)} \quad \frac{\Gamma, A \vdash B}{\Gamma \vdash A \supset B} \text{ } (\supset I) \quad \frac{\Gamma \vdash A \supset B \quad \Gamma \vdash A}{\Gamma \vdash B} \text{ } (\supset E)$$

The Hilbert system for the fragment of propositional logic consisting of implication is defined by the following axiom schemas (they are axiom schemas because you can instantiate the meta-variables A, B, C with any proposition) and rule:

Definition 1 (Hilbert System for implication)

$$\begin{array}{ll} A \supset A & \text{(I)} \\ A \supset (B \supset A) & \text{(K)} \\ (A \supset (B \supset C)) \supset ((A \supset B) \supset (A \supset C)) & \text{(S)} \end{array}$$
$$\frac{A \supset B \quad A}{B} \text{ (mp)}$$

You might recognize the axioms presented above from the combinator calculus exercise from the previous assignment.

We will now show the correspondence between the Hilbert system and natural deduction (and therefore the correspondence between the **SKI** combinator calculus and the λ -calculus). We need to come up with a way of translating proofs in natural deduction form to Hilbert form, and vice versa. The former is the difficult direction since natural deduction proofs (in particular the rule for implication introduction) makes use of hypotheses, which are not directly available

in Hilbert proofs. We must therefore define *hypothetical* Hilbert derivations and we will show that hypotheses can always be eliminated - this is known as the *deduction theorem*. We write $\Gamma \vdash_H A$ for the hypothetical Hilbert derivation that proves A under the assumptions in Γ . As per usual, we require a hypothesis rule that allows us to use hypothesis in a derivation:

$$\frac{}{\Gamma, A \vdash_H A} \text{ (hyp)}$$

You will now prove the theorem mentioned above (you may notice how each non-inductive case, in a way, “forces our hand” in the axioms we need in this system).

Task 1 (Deduction Theorem) Show that if $\Gamma, A \vdash_H B$ then $\Gamma \vdash_H A \supset B$.

(Hint) Proceed by induction on the given derivation. You are allowed to assume the standard structural properties of hypothetical judgments.

You are now in a position to show that we can always translate a hypothetical natural deduction proof to a hypothetical Hilbert proof.

Task 2 (Natural Deduction to Hilbert) Show that if $\Gamma \vdash A$ then $\Gamma \vdash_H A$.

(Hint) You will need to use the deduction theorem.

From which trivially follows that if $\vdash A$ then $\vdash_H A$ by considering the special case with an empty context.

Task 3 (Hilbert to Natural Deduction) Show that if $\vdash_H A$ then $\vdash A$. In the non-inductive cases, write the natural deduction proof in λ notation.

We can assign proof terms to the Hilbert system as follows.

Definition 2 (Proof Term Assignment for Hilbert System)

$$\begin{aligned} \vdash_H \mathbf{I} & : A \supset A \\ \vdash_H \mathbf{K} & : A \supset (B \supset A) \\ \vdash_H \mathbf{S} & : (A \supset (B \supset C)) \supset ((A \supset B) \supset (A \supset C)) \\ & \frac{\vdash_H E_1 : A \supset B \quad \vdash_H E_2 : A}{\vdash_H (E_1 E_2) : B} \text{ (mp)} \end{aligned}$$

As you might recall, these are exactly the forms of expression in the **SKI** combinator calculus. The proof terms explicate the operational semantics of the combinator calculus (just consider the operational semantics for the λ terms of Task 3).

Task 4 (Natural Deduction to Hilbert - Redux) In Task 2 you have shown how to translate a proof in natural deduction form to a derivation in the Hilbert system. By the *proofs as programs principle*, the proof of Task 2 must therefore also describe a way of translating a λ -calculus term to a combinator calculus term. What is this translation function?

(Hint) Since the proof of Task 2 makes use of the deduction theorem, you must first define the transformation on proofs described by Task 1. Consider what are the corresponding proof terms for the derivations in each case (think of bracket abstraction from Homework 1).

2 Fixed-Points and Computability

Mutual Recursion

We will define a λ -term that computes the Catalan numbers. A way of computing these numbers is through a recurrence (C_n denotes the n th Catalan number):

$$C_{n+1} = \sum_{i=0}^n C_i C_{n-i}$$

Task 5 Define a function `sum` that computes a generalized sum from 0 to n . This function should take as arguments the church numeral for n and a function that it applies to each operand of the sum, that is, for a given n and function f , $\text{sum } \bar{n} f = f(\bar{0}) + \dots + f(\bar{n})$. You may use $+$ under the assumption that it computes the sum of two Church numerals.

Task 6 (Computing the Catalan Numbers) Write a λ -expression that computes the Catalan numbers. Do this by defining a λ -term consisting of a pair of functions. The first projection of the pair computes $C_i C_{n-i}$ and the second component of the pair computes C_n .

(Hint) Use the `sum` function of the previous task. You may use addition, multiplication, `ifz` and numerals in abbreviated form.

Halting Problem - First Order vs Higher Order

In this exercise we will show that there is no λ -term T such that, for a given λ -term M decides whether or not M converges. We will assume the usual encodings of booleans and boolean functions in the λ -calculus: `tt` corresponds to the boolean value for truth, `ff` to the boolean for falsehood.

We shall therefore specify that the term T must obey the following specification:

$$\begin{array}{ll} T M =_{\beta} \text{tt} & \text{or } T M =_{\beta} \text{ff} \\ T M =_{\beta} \text{tt} & \text{iff } M \text{ converges} \\ T M =_{\beta} \text{ff} & \text{iff } M \text{ diverges} \end{array}$$

$T M$ is β equivalent to true or false. It will be β equivalent to true if and only if M converges, and therefore it will be equivalent to false iff M diverges. To prove that no such T is definable in the λ -calculus, we will replicate the diagonal argument used to prove the undecidability of the Halting Problem, but in a *higher-order* setting. The proof of the undecidability of the Halting Problem requires a way of encoding programs as data that can then be manipulated by other programs as a way of encoding self reference. In our higher-order setting, we have already seen that we can define self-reference. The idea behind the proof consists of assuming that such a term T is indeed definable, and then defining another λ term D that applies T to D itself, if the outcome is `tt` then D diverges, otherwise it is equal to `tt` (you're allowed to use the encoding for a conditional branch; the divergent term shouldn't be too hard to figure out since you know about self application). Since D is a λ -term, we can try and observe the behavior of applying T to D itself, from which a contradiction should arise.

Task 7 (Halting Problem in a Higher Order Setting) Show that T is not λ -definable.

Now, what if T instead of taking a λ -term M (which in a sense can only really be run) instead received the "code" for M and through some clever analysis of the program code, would determine whether or not M actually converges or not. The specification for this variant of T , call it T' , is as follows (where \overline{M} denotes the representation of the "code" for the λ -term M as a λ -term):

$$\begin{array}{ll} T' \overline{M} =_{\beta} \text{tt} & \text{or } T' \overline{M} =_{\beta} \text{ff} \\ T' \overline{M} =_{\beta} \text{tt} & \text{iff } M \text{ converges} \\ T' \overline{M} =_{\beta} \text{ff} & \text{iff } M \text{ diverges} \end{array}$$

The notion of representing the code for a λ -term may appear strange, but can be made precise using a technique called Gödel numbering, which we will not develop in detail here. In a nutshell, it is possible to assign a unique natural number to all closed λ -terms (called a Gödel number) in such a way that all α -equivalent terms are assigned the same number and that given such a number we can recover the term to which it corresponds (up to α -equivalence). Since we can encode natural numbers in the λ -calculus using Church numerals, we denote by \overline{M} the Church numeral corresponding to the Gödel number of M .

To show that no such T' exists, we must devise a way of encoding self-reference by manipulating codes for λ -terms instead of the actual terms the codes denote. As we have seen, the \mathbf{Y} combinator allows us to define self-referential expressions in the higher-order setting (i.e. in $\mathbf{Y}(\lambda x.M)$ we think of x in M as standing for the term $\mathbf{Y}(\lambda x.M)$).

We must therefore define a “variant” of the \mathbf{Y} combinator, that encodes self-reference for functions that receive codes as arguments.

Task 8 (The \mathbf{Y} “Combinator”) Show that, for any given λ -term f that expect codes as arguments, you can produce a term M such that $f(\overline{\overline{M}}) =_{\beta} M$.

(Hint) For this task you will need to use two functions that are definable in the λ -calculus (this follows from the definition of the encoding of Gödel numbers in the λ -calculus):

- $\text{ap}(\overline{\overline{N}})(\overline{\overline{M}}) =_{\beta} \overline{\overline{N(M)}}$
- $\text{nm}(\overline{\overline{N}}) =_{\beta} \overline{\overline{N}}$

The first function takes two codes for λ terms and returns the code for their application. The second takes a numeral for N and returns the code of \overline{N} . Think of the application $\mathbf{Y}(f)$ and adapt it considering the restriction that f receives codes instead of “simple” λ -terms, using the functions ap and nm mentioned above. Note that since you are given a particular λ -term f , you are allowed to refer to its encoding $\overline{\overline{f}}$ (and similarly to the encoding of any term you devise using f).

Task 9 Show that T' is not λ -definable.

(Hint) The proof is similar to the higher-order setting. You need to come up with a function f that makes use of T' similar to how D did in the higher-order proof. Then, using the previous task, you can find a term D' that is fixed point of this function. You can then reason about the behavior of applying T' to the encoding of this term.

3 Hypothetical Judgments and Structural Properties

In the previous assignments, we have made use of several structural properties such as weakening by assuming them to hold (similarly, we were allowed to be a bit fast and loose with inversion). There are situations where this might not necessarily be the case, and so, in general, we must view structural properties (and inversion properties) as theorems that need to be proved about the system we are considering. For this exercise we will consider the simply typed λ -calculus with function spaces and pairs:

$$\frac{}{\Gamma, x : A \vdash x : A} \text{hyp} \quad \frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x : A. M : A \supset B} \supset I \quad \frac{\Gamma \vdash M : A \supset B \quad \Gamma \vdash N : A}{\Gamma \vdash M N : B} \supset E$$

$$\frac{\Gamma \vdash M : A \quad \Gamma \vdash N : B}{\Gamma \vdash \langle M, N \rangle : A \times B} \times I \quad \frac{\Gamma \vdash M : A \times B}{\Gamma \vdash \pi_1(M) : A} \times E_1 \quad \frac{\Gamma \vdash M : A \times B}{\Gamma \vdash \pi_2(M) : B} \times E_2$$

You will now prove the structural properties that are inherent to the meaning of a hypothetical judgment, as well as the inversion principles that we need.

Task 10 (Weakening) If $\Gamma_1, \Gamma_2 \vdash N : B$ then $\Gamma_1, x : A, \Gamma_2 \vdash N : B$, with x not occurring in either Γ_1 or Γ_2 .

Task 11 (Substitution) Show that if $\Gamma, x : A \vdash N : C$ and $\Gamma \vdash M : A$ then $\Gamma \vdash [M/x]N : C$. Carefully state why the use of the induction hypothesis is justified in the case for the λ abstraction.

Task 12 (Inversion) Carefully state and prove inversion lemmas for the constructions of the language presented above.

(Hint) An inversion lemma, for a given construct of the language, characterizes its type and the types of its subexpressions.

Consider an extension of the language above with sum types:

$$\frac{\Gamma \vdash M : A}{\Gamma \vdash \text{inl}(M) : A + B} +I_1 \quad \frac{\Gamma \vdash M : B}{\Gamma \vdash \text{inr}(M) : A + B} +I_2$$

$$\frac{\Gamma \vdash M : A + B \quad \Gamma, x : A \vdash N_1 : C \quad \Gamma, x : B \vdash N_2 : C}{\Gamma \vdash \text{case}(M; x.N_1; x.N_2) : C} +E$$

Task 13 (Safety) State progress and preservation. Carefully prove the cases for the constructs pertaining to the sum type (remember that you also need canonical forms to prove progress). Assume the operational semantics presented in class.