Monad-Based Programming SS 2023

Assignment 4

Deadline for solutions: 22.06.2023

Exercise 1 Soundeness of (Lazy) PCF (8 Points)

Recall that the soundness property states that for every closed program p and every value v of the same type, $p \Downarrow v$ entails $[\![p]\!] = [\![v]\!]$. Using the equivalence of the big-step and the small-step semantics, this property follows from the other one, which is also called *soundness* of the individual rules: for every operational semantics rule

$$\frac{p_1 \to q_1 \quad \dots \quad p_n \to q_n}{p \to q}$$

if $[\![p_1]\!] = [\![q_1]\!], \dots, [\![p_n]\!] = [\![q_n]\!]$ then $[\![p]\!] = [\![q]\!]$.

(a) Why?

(b) Prove soundness of the following rules

$$\frac{p \to p'}{pq \to p'q} \qquad \frac{}{Y p \to p(Y p)} \qquad \frac{}{(\lambda x.\, p)q \to q[p/x]}$$

using *Substitution Lemma* from the script: Given $\Gamma \vdash q\colon A$, $\Gamma, x\colon A \vdash p\colon B$ and $\rho \in [\![\Gamma]\!]$

$$[\![\Gamma \vdash p[q/x]\colon B]\!]_\rho = [\![\Gamma, x\colon A \vdash p\colon B]\!](\rho, [\![\Gamma \vdash q\colon A]\!]_\rho)$$

(c) Prove that $[\![\Gamma \vdash f\colon A \to B]\!] = [\![\Gamma \vdash \lambda x.\, fx\colon A \to B]\!]$.

Exercise 2 Iteration from Recursion (7 Points)

Consider the following term formation rule for while:

$$\frac{\Gamma \vdash p\colon S \qquad \Gamma, x\colon S \vdash b\colon \mathit{Bool} \qquad \Gamma, x\colon S \vdash q\colon S}{\Gamma \vdash \mathsf{let}\ x \coloneqq p\ \mathsf{while}\ b\ \mathsf{do}\ q\colon S}$$

Here,

- S is regarded as a type of *states*,
- p is a program that initializes the state,
- b is a test, depending on the current state, and
- q is a loop body, which transforms a given state to a new one.

(a) Express (let $x = p$ while b do q) in PCF as a term over p, b and q in such a way that

$$[\![\Gamma \vdash \mathsf{let}\ x = p\ \mathsf{while}\ b\ \mathsf{do}\ q \colon S]\!] = [\![\Gamma \vdash \mathsf{if}\ b[p/x]\ \mathsf{then}\ (\mathsf{let}\ x = q[p/x]\ \mathsf{while}\ b\ \mathsf{do}\ q)\ \mathsf{else}\ p \colon S]\!]. \quad (*)$$

Hint: Some exploration will be needed. You should think of that, how substitution

$$(\mathsf{let}\ x = p\ \mathsf{while}\ b\ \mathsf{do}\ q)[r/y]$$

must be computed. Then note that if $x \neq y$ and y is not free either in b or in q,

$$[\![\Gamma \vdash \mathsf{let}\ x = y\ \mathsf{while}\ b\ \mathsf{do}\ q \colon S]\!] = [\![\Gamma \vdash \mathsf{if}\ b[y/x]\ \mathsf{then}\ (\lambda y.\, \mathsf{let}\ x = y\ \mathsf{while}\ b\ \mathsf{do}\ q)(q[y/x])\ \mathsf{else}\ y \colon S]\!].$$

This gives an idea how to recursively express (let $x = y$ while b do q) through itself, and thus, how to express the general case using the Y-combinator.

(b) Prove (*).

(c) The simplest implementation of Fibonacci numbers, corresponding to the following Haskell defletion

```
fib 0 = 1
fib 1 = 1
fib n = fib (n - 1) + fib (n - 2)
```

can be extremely inefficient, for every step recursively calls `fib` twice, so the number of recursive calls grows exponentially. Because of that, it makes sense to use the following variant of `fib`, which simultaneously calculates the n-th and the $(n + 1)$-th Fibonacci number:

```
fib2 0 = (1 , 1)
fib2 n = let (fn_1, fn) = fib2 (n - 1) in (fn, fn_1 + fn)
```

It can be shown that `fib` is equivalent to `fst $ fib2`.

This example demonstrates that inefficiency of unconstrained recursion cannot generally be prevented, which is sometimes considered as a drawback of functional programming. The reason why in the second program recursion is harmless is because it is essentially an iterative program: the state is a pair of numbers, which are updated in a while-loop, as long as the counter n is non-zero. Formalize this by reformulating `fib2` using the above while-operator.

(d) Use the encoding of while in PCF from (a), and run the corresponding program in your interpreter from Exercise 2, Assignment 3, to compute `fib2 3`.

Exercise 3 Dinaturality (5 Points)

The least fixpoint operator μ, figuring in the Kleene fixpoint theorem, features many properties. One such property is the so-called *dinaturality law*: given two continuous functions $f \colon A \to B$ and $g \colon B \to A$ between domains,

$$f(\mu(gf)) = \mu(fg).$$

Prove it. **Hint:** prove mutual inequality. You need not use any knowledge, except the very definition of μ – that it defines a fixpoint, and this fixpoint is the least one.