Monad-Based Programming SS 2019

Assignment 1

Deadline for solutions: 16.05.2019

Exercise 1 Small-step v.s. Big-step (8 Points)

Consider the following rules for the small-step and big-step call-by-value semantics of untyped λ-calculus:

Small-step semantics:

$$\frac{p \to_{cbv} p'}{pq \to_{cbv} p'q}\ \textbf{(l-red)} \qquad \frac{q \to_{cbv} q' \qquad p \text{ is a value}}{pq \to_{cbv} pq'}\ \textbf{(r-red)} \qquad \frac{q \text{ is a value}}{(\lambda x.\, p)q \to_{cbv} p[q/x]}\ (\boldsymbol{\beta})$$

Big-step semantics:

$$\frac{}{\lambda x.\, p \Downarrow_{cbv} \lambda x.\, p}\ \textbf{(value)} \qquad \frac{p \Downarrow_{cbv} \lambda x.\, p' \qquad q \Downarrow_{cbv} q' \qquad p'[q'/x] \Downarrow_{cbv} v}{pq \Downarrow_{cbv} v}\ \textbf{(app)}$$

Recall that a λ-term p is a *value* if and only if it has the form $\lambda x.\, t$.

Prove that for any closed λ-term p, $p \to^{\star}_{cbv} q$ with q being a value iff $p \Downarrow_{cbv} q$. To this end, use (without a proof) the following

Well-founded Tree Induction Principle: given a set of rules S and a predicate P with the following properties:

1. $P(t)$ for any rule from S of the form
$$\frac{}{t}$$

2. whenever $P(t_1), \dots, P(t_n)$ and the rule
$$\frac{t_1 \quad \dots \quad t_n}{t}$$
belongs to S then $P(t)$.

Then $P(t)$ for any t that can be derived using S.

Hint: For one direction of the equivalence use the lemma: $p \to_{cbv} q \wedge q \Downarrow_{cbv} c \Rightarrow p \Downarrow_{cbv} c$.

Exercise 2 PCF with coproducts (6 Points)

A *disjoint union* of sets A and B is the set

$$A + B = \{\langle a, * \rangle \mid a \in A\} \cup \{\langle *, b \rangle \mid b \in B\}$$

where $* \notin A \cup B$. It comes equipped with the following structure: $\mathsf{inl} : A \to A+B$, $\mathsf{inr} : B \to A+B$ and the *copairing brackets* sending any $f : A \to C$ and $g : B \to C$ to $[f, g] : A + B \to C$ as follows:

$$\mathsf{inl}(a) = \langle a, * \rangle \qquad \mathsf{inr}(b) = \langle *, b \rangle \qquad [f, g](a, *) = f(a) \qquad [f, g](*, b) = g(b)$$

The Haskell counterpart of disjoint unions if the *sum type*

```
data Either a b = Left a | Right b
```

where the constructors `Left` and `Right` are the counterparts of inl and inr and the case-construct `case` is the counterpart of copairing.

1. Design an extension of PCF by adding a *sum type* $A + B$ and providing appropriate introduction and elimination rules for the corresponding term constructs in the Haskell style.
2. Design call-by-name small-step and big-step operational semantics for the new term constructs. Demonstrate that this semantics is consistent with the behaviour of Haskell programs using the divergence combinator `omega`.
3. Can the conditional branching operator (if) be shown to be superfluous in presence of the new term constructs? Justify your answer using the rules of the operational semantics.

Exercise 3 Pythagorean triples in Haskell (6 Points)

A triple of natural numbers $\langle a, b, c \rangle$ is called *Pythagorean* if it satisfies the angled triangle property:

$$a^2 + b^2 = c^2.$$

We say that a triple $\langle a, b, c \rangle$ is smaller than $\langle a', b', c' \rangle$ if $c < c'$.

1. Write a Haskell-program that outputs all Pythagorean triples in order.
2. Write a Haskell-program to calculate the pair of Pythagorean triples $\langle a, b, c \rangle$ and $\langle a', b', c \rangle$ of least possible c such that $\{a, b\} \neq \{a', b'\}$.

Hint: Be lazy. Use (list) comprehension.