Technische Universität München
Institut für Informatik
Prof. Tobias Nipkow, Ph.D.
Dr. Peter Lammich

WS 2015/16
12. 01. 2016

Semantics of Programming Languages

Exercise Sheet 12

Exercise 12.1 Kleene fixed point theorem

Prove the Kleene fixed point theorem. We first introduce some auxiliary definitions:

A chain is a set such that any two elements are comparable. For the purposes of the Kleene fixed-point theorem, it is sufficient to consider only countable chains. It is easiest to formalize these as ascending sequences. (We can obtain the corresponding set using the function *range* :: ('a ⇒ 'b) ⇒ 'b set.)

definition *chain* :: "(nat ⇒ 'a::complete_lattice) ⇒ bool"
  where "chain C ⟷ (∀ n. C n ≤ C (Suc n))"

A function is continuous, if it commutes with least upper bounds of chains.

definition *continuous* :: "('a::complete_lattice ⇒ 'b::complete_lattice) ⇒ bool"
  where "continuous f ⟷ (∀ C. chain C ⟶ f (Sup (range C)) = Sup (f ` range C))"

The following lemma may be handy:

lemma *continuousD*: "⟦continuous f; chain C⟧ ⟹ f (Sup (range C)) = Sup (f ` range C)"
  unfolding *continuous_def* **by** *auto*

As warm-up, show that any continuous function is monotonic:

lemma *cont_imp_mono*:
  fixes f :: "'a::complete_lattice ⇒ 'b::complete_lattice"
  assumes "continuous f"
  shows "mono f"

Hint: The relevant lemmas are

thm *mono_def monoI monoD*

Finally show the Kleene fixed point theorem. Note that this theorem is important, as it provides a way to compute least fixed points by iteration.

theorem *kleene_lfp*:
  fixes f:: "'a::complete_lattice ⇒ 'a"
  assumes *CONT*: "continuous f"
  shows "lfp f = Sup (range (λi. (f^^i) bot))"
proof –

We propose a proof structure here, however, you may deviate from this and use your own proof structure:

```
  let ?C = "λi. (f^^i) bot"
  note MONO=cont_imp_mono[OF CONT]

  have CHAIN: "chain ?C"
  show ?thesis
  proof (rule antisym)
    show "Sup (range ?C) ≤ lfp f"
  next
    show "lfp f ≤ Sup (range ?C)"
  qed
qed
```

Hint: Some relevant lemmas are

thm *lfp_unfold lfp_lowerbound Sup_subset_mono range_eqI*

Exercise 12.2 Complete Lattice over Lists

Show that lists of the same length ordered pointwise form a partial order if the element type is partially ordered. Partial orders are predefined as the type class "order".

instantiation *list* :: (*order*) *order*

Define the infimum operation for a set of lists. The first parameter is the length of the result list.

definition *Inf_list* :: "*nat* ⇒ (*'a*::*complete_lattice*) *list set* ⇒ *'a list*"

Show that your ordering and the infimum operation indeed form a complete lattice:

interpretation
Complete_Lattice "{*xs. length xs = n*}" "*Inf_list n*" **for** *n*

Homework 12 Euclid's Algorithm

Submission until Tuesday, January 19, 2011, 10:00am.

Euclid's algorithm computes the greatest common divisor of two **positive** numbers. Its pseudocode looks as follows:

```
while a ≠ b do
  if a < b then
    b := b − a
  else
    a := a − b
```

1. Write a program *SUB a b* which computes the difference between variables a and b, without modifying them. The result should be stored in variable $''r''$. You may assume that $a \neq ''r'' \wedge b \neq ''r''$.
2. Write a program *EUCLID*, which implements Euclid's algorithm, and prove its **total** correctness.

Hints:

- In *Complex_Main*, there is a *gcd* function. It works for both, natural numbers and integers.
- You may either try to prove a rule for *SUB* (similar to the assignment rule), or unfold the definition of *SUB* during the proof of *EUCLID*.