

Semantics of Programming Languages

Exercise Sheet 9

Exercise 9.1 Knaster-Tarski Fixed Point Theorem

The Knaster-Tarski theorem tells us that for each set P of fixed points of a monotone function f we have a fixpoint of f which is a greatest lower bound of P . In this exercise, we want to prove the Knaster-Tarski theorem.

First we give a construction of the greatest lower bound of all fixed points P of the function f . This is the union of all sets u smaller than P and $f u$. Then the task is to show that this is a fixed point, and that it is the greatest lower bound of all sets in P .

Let us define Inf_fixp :

definition $Inf_fixp :: ('a set \Rightarrow 'a set) \Rightarrow 'a set set \Rightarrow 'a set$ **where**
“ $Inf_fixp f P = \bigcup \{u. u \subseteq \bigcap P \cap f u\}$ ”

To work directly with this definition is a little cumbersome, we propose to use the following two theorems:

lemma $Inf_fixp_upperbound$: “ $X \subseteq \bigcap P \Longrightarrow X \subseteq f X \Longrightarrow X \subseteq Inf_fixp f P$ ”
by (*auto simp: Inf_fixp_def*)

lemma Inf_fixp_least : “ $(\bigwedge u. u \subseteq \bigcap P \Longrightarrow u \subseteq f u \Longrightarrow u \subseteq X) \Longrightarrow Inf_fixp f P \subseteq X$ ”
by (*auto simp: Inf_fixp_def*)

Now prove, that Inf_fixp is actually a fixed point of f .

Hint: First prove $Inf_fixp f P \subseteq f (Inf_fixp f P)$, this will be used for the other direction. It may be helpful to first think about the structure of your proof using pen-and-paper and then translate it into Isar.

lemma Inf_fixp :
assumes *mono*: “*mono f*”
and P : “ $\bigwedge p. p \in P \Longrightarrow f p = p$ ”
shows “ $Inf_fixp f P = f (Inf_fixp f P)$ ”

Now we prove that it is a lower bound:

lemma Inf_fixp_lower : “ $Inf_fixp f P \subseteq \bigcap P$ ”

And that it is the greatest lower bound:

lemma *Inf_fixp_greatest*:
assumes “ $f\ q = q$ ”
and “ $q \subseteq \bigcap P$ ”
shows “ $q \subseteq \text{Inf_fixp } f\ P$ ”

Exercise 9.2 While combinator

So far, all functions that we defined were required to terminate. However, there is also a while-combinator in HOL. For instance, the IMP-program

WHILE Less (V "x") (N 3) DO "x" ::= Plus (V "x") (N 2)

could be stated in HOL as follows:

value “*while* ($\lambda x::\text{nat. } x < 3$) ($\lambda x. x + 2$) 0”

Take a look at the definition. What is surprising about it? Can you state and refute (using *nitpick*) lemmas about involved constants that should at first glance hold?

Using *while*, define an *exec* function for commands:

fun *exec* :: “*com* \Rightarrow *state* \Rightarrow *state*”

Example:

value “(*exec* (*WHILE Less (V "x") (N 3) DO "x" ::= Plus (V "x") (N 2)*) $\langle \rangle$) "x"”

Show that *exec* is correct:

lemma “(*c,s*) \Rightarrow *t* \implies *exec c s = t*”

Homework 9 Be Original!

Submission until Monday, Jan 8, 23:59pm.

Think up a nice topic to formalize yourself! It can be from any area of mathematics, computer science, etc., but should contain some interesting proof(s) – mere definitions or implementations are not interesting.

Creativity is encouraged and will be graded, but keep in mind that formalizations can often be more difficult than anticipated. Set yourself realistic goals! You are also welcome to discuss your project with us beforehand.

Comment your formalization well such that we can see what it is intended to do.

Incomplete or unfinished formalizations are welcome and will be graded (but clean them up so it is obvious what is there and what is missing).

Deadline extension - the project will run until the end winter holiday! Submit your project either via submissions system if it's a single file (it will raise an error, but that is ok) or as archive via e-mail.

In total, this exercise will be worth 15 points, plus bonus points for nice submissions.

Merry Christmas!