Exercise 8.1 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 $\text{Inf}_{\text{fixp}}$:

**definition $\text{Inf}_{\text{fixp}}$**: 
"('a set ⇒ 'a set) ⇒ 'a set set ⇒ 'a set" where

$\text{Inf}_{\text{fixp}} f P = \bigcup \{u.\ u \subseteq \bigcap P \cap f u\}$

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

**lemma $\text{Inf}_{\text{fixp}}_{\text{upperbound}}$:** $X \subseteq \bigcap P \implies X \subseteq f X \implies X \subseteq \text{Inf}_{\text{fixp}} f P$

by (auto simp: $\text{Inf}_{\text{fixp}}$ _def)

**lemma $\text{Inf}_{\text{fixp}}_{\text{least}}$:** $(\forall u.\ u \subseteq \bigcap P \implies u \subseteq f u \implies u \subseteq X) \implies \text{Inf}_{\text{fixp}} f P \subseteq X$

by (auto simp: $\text{Inf}_{\text{fixp}}$ _def)

Now prove, that $\text{Inf}_{\text{fixp}}$ is actually a fixed point of $f$.

*Hint:* First prove $\text{Inf}_{\text{fixp}} f P \subseteq f (\text{Inf}_{\text{fixp}} f P)$, 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 $\text{Inf}_{\text{fixp}}$:**

assumes $f$: "mono $f$"

assumes $P$: "$(\forall p.\ p \in P \implies f p = p)$"

shows "$\text{Inf}_{\text{fixp}} f P = f (\text{Inf}_{\text{fixp}} f P)$"

Now we prove that it is a lower bound:

**lemma $\text{Inf}_{\text{fixp}}_{\text{lower}}$:** "$\text{Inf}_{\text{fixp}} f P \subseteq \bigcap P$"

And that it is the greatest lower bound:

**lemma $\text{Inf}_{\text{fixp}}_{\text{greatest}}$:**

assumes ""$f q = q$" "$q \subseteq \bigcap P$" shows ""$q \subseteq \text{Inf}_{\text{fixp}} f P$"
Exercise 8.2 Denotational Semantics

Define a denotational semantics for REPEAT-loops, and show its equivalence to the bigstep semantics.

```plaintext
datatype com = SKIP
    | Assign vname aexp ("_ := _" [1000, 61])
    | Seq com com ("_;/ _" [60, 61])
    | If bexp com com ("(IF _/ THEN _/ ELSE _)" [0, 0, 61])
    | While bexp com ("(WHILE _/ DO _)" [0, 61])
    | Repeat com bexp ("(REPEAT _/ UNTIL _)" [0, 61])

inductive
    big_step :: “com × state ⇒ state ⇒ bool” (infix “⇒” 55)

where
Skip: “(SKIP,s) ⇒ s” |
Assign: “(x := a,s) ⇒ s(x := aval a s)” |
Seq: “[(c1,s1) ⇒ s2; (c2,s2) ⇒ s3] ⇒ (c1;c2, s1) ⇒ s3” |
IfTrue: “[ bval b s; (c1,s) ⇒ t ] ⇒ (IF b THEN c1 ELSE c2, s) ⇒ t” |
IfFalse: “[ ¬bval b s; (c2,s) ⇒ t ] ⇒ (IF b THEN c1 ELSE c2, s) ⇒ t” |
WhileFalse: “¬bval b s ⇒ (WHILE b DO c,s) ⇒ s” |
WhileTrue: “[ bval b s1; (c1,s1) ⇒ s2; (WHILE b DO c, s2) ⇒ s3 ]
               ⇒ (WHILE b DO c, s1) ⇒ s3”
```

Proof automation:

```plaintext
lemmas [intro] = big_step.intros
lemmas big_step_induct = big_step.induct[split_format(complete)]
```

inductive_cases SkipE[elim!]: “(SKIP,s) ⇒ t”
inductive_cases AssignE[elim!]: “(x := a,s) ⇒ t”
inductive_cases SeqE[elim!]: “(c1;c2,s1) ⇒ s3”
inductive_cases IfE[elim!]: “(IF b THEN c1 ELSE c2,s) ⇒ t”
inductive_cases WhileE[elim!]: “(WHILE b DO c,s) ⇒ t”

Execution is deterministic:

```plaintext
theorem big_step_determ: “[(c,s) ⇒ t; (c,s) ⇒ u] ⇒ u = t”
    by (induction arbitrary: u rule: big_step.induct) blast+
```

type_synonym com_den = “(state × state) set”

definition W :: “(state ⇒ bool) ⇒ com_den ⇒ (com_den ⇒ com_den)” where
  “W db dc = (λd. {((s,t). if db s then (s,t) ∈ dc O dw else s=t)})”

fun D :: “com ⇒ com_den” where
  "D SKIP = Id” |
  "D (x := a) = {(s,t). t = s(x := aval a s)}” |
  "D (c1;c2) = D(c1) O D(c2)” |
  "D (IF b THEN c1 ELSE c2)“
\begin{verbatim}

lemma D_While_If:  "D(\text{WHILE } b \text{ DO } c) = D(\text{IF } b \text{ THEN } c;; \text{WHILE } b \text{ DO } c \text{ ELSE } \text{SKIP})"
proof-
  let ?w = "\text{WHILE } b \text{ DO } c"  let ?f = "W (bval b) (D c)"
  have "D ?w = lfp ?f" by simp
  also have "\ldots = ?f (lfp ?f)" by (rule lfp_unfold [OF W_mono])
  also have "\ldots = D(\text{IF } b \text{ THEN } c;; ?w \text{ ELSE } \text{SKIP})" by (simp add: W_def)
  finally show ?thesis .
qed

Equivalence of denotational and big-step semantics:

lemma D_if_big_step: "(c, s) \Rightarrow t \Rightarrow (s, t) \in D(c)"
proof (induction rule: big_step_induct)
  case WhileFalse
  thus ?case by fastforce
next
  case WhileTrue
  show ?case unfolding D_While_If using WhileTrue by auto
nextqed auto

abbreviation Big_step :: "com \Rightarrow \text{com_den}" where
  "Big_step c \equiv \{(s, t). (c, s) \Rightarrow t\}"

lemma Big_step_if_D: "(s, t) \in D(c) \Rightarrow (s, t) : Big_step c"
proof (induction c arbitrary: s t)
  case Seq thus ?case by fastforce
next
  case (While b c)
  let ?B = "Big_step (\text{WHILE } b \text{ DO } c)"  let ?f = "W (bval b) (D c)"
  have "\ldots \in \text{ifp } ?B" using While.IH by (auto simp: W_def)
  from lfp_lowerbound\[where ?f = "?f", OF this\] While.prems
  show ?case by auto
nextqed (auto split: if_splits)

theorem denotational_is_big_step:
  "(s, t) \in D(c) = ((c, s) \Rightarrow t)"
by (metis D_if_big_step Big_step_if_D[simplified])

\end{verbatim}
**Homework 8.1  Be Original!**

*Submission until Sunday, Jan 10, 23:59. In total, this exercise is worth 15 points, plus bonus points for nice submissions.*

You should now have a topic to formalize, for example:

- Prove some interesting result about algorithms/graphs/automata/formal language theory
- Formalize some results from mathematics
- Find interesting modifications of IMP material and prove interesting properties about them
- ...

Do the formalization! You can submit your work via the submission system or by email. You should set yourself a time limit before starting your project. Also incomplete/unfinished formalizations are welcome and will be graded!

Please comment your formalization well, such that we can see what it does/is intended to do.

Merry Christmas!