Technische Universität München
Fakultät für Informatik
Prof. Tobias Nipkow, Ph.D.
Simon Wimmer

WS 2019/20
09.12.2019

Semantics of Programming Languages

Exercise Sheet 08

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 *Inf_fixp*:

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

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

lemma *Inf_fixp_upperbound*: "$X \subseteq \bigcap P \Longrightarrow X \subseteq f\ X \Longrightarrow X \subseteq \mathit{Inf\_fixp}\ f\ P$"
by (*auto simp*: *Inf_fixp_def*)

lemma *Inf_fixp_least*: "$(\bigwedge u.\ u \subseteq \bigcap P \Longrightarrow u \subseteq f\ u \Longrightarrow u \subseteq X) \Longrightarrow \mathit{Inf\_fixp}\ f\ P \subseteq X$"
by (*auto simp*: *Inf_fixp_def*)

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

Hint: First prove $\mathit{Inf\_fixp}\ f\ P \subseteq f\ (\mathit{Inf\_fixp}\ f\ P)$, 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 f: "*mono f*"
assumes P: "$\bigwedge p.\ p \in P \Longrightarrow f\ p = p$"
shows "$\mathit{Inf\_fixp}\ f\ P = f\ (\mathit{Inf\_fixp}\ f\ P)$"

Now we prove that it is a lower bound:

lemma *Inf_fixp_lower*: "$\mathit{Inf\_fixp}\ f\ P \subseteq \bigcap P$"

And that it is the greatest lower bound:

lemma *Inf_fixp_greatest*:
assumes "$f\ q = q$" "$q \subseteq \bigcap P$" **shows** "$q \subseteq \mathit{Inf\_fixp}\ f\ P$"

Exercise 8.2 Denotational Semantics

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

Use the exercise template that we provide on the course web page.

Homework 8.1 Idempotence of Dead Varibale Elimination

Submission until Monday, Dec 16, 10:00am.

Dead variable elimination (*bury*) is not idempotent: multiple passes may reduce a command further and further. Give an example where $bury\ (bury\ c\ X)\ X \neq bury\ c\ X$. Hint: a sequence of two assignments.

Now define the textually identical function *bury* in the context of true liveness analysis (theory *HOL−IMP.Live_True*).

fun *bury* :: *"com ⇒ vname set ⇒ com"* **where**
"bury SKIP X = SKIP" |
"bury (x ::= a) X = (if x ∈ X then x ::= a else SKIP)" |
$\text{“}bury\ (c_1;;\ c_2)\ X = (bury\ c_1\ (L\ c_2\ X);;\ bury\ c_2\ X)\text{”}$ |
$\text{“}bury\ (IF\ b\ THEN\ c_1\ ELSE\ c_2)\ X = IF\ b\ THEN\ bury\ c_1\ X\ ELSE\ bury\ c_2\ X\text{”}$ |
"bury (WHILE b DO c) X = WHILE b DO bury c (L (WHILE b DO c) X)"

The aim of this homework is to prove that this version of *bury* is idempotent. This will involve reasoning about *lfp*. In particular we will need that *lfp* is the least pre-fixpoint. This is expressed by two lemmas from the library:

lfp_unfold: $mono\ f \Longrightarrow lfp\ f = f\ (lfp\ f)$
lfp_lowerbound: $f\ A \leq A \Longrightarrow lfp\ f \leq A$

Prove the following lemma for showing that two fixpoints are the same, where *mono_def*: $mono\ f = (\forall x\ y.\ x \leq y \longrightarrow f\ x \leq f\ y)$.

lemma *lfp_eq*: $\text{“}\llbracket\ mono\ f;\ mono\ g;\ lfp\ f \subseteq U;\ lfp\ g \subseteq U;$
$\bigwedge X.\ X \subseteq U \Longrightarrow f\ X = g\ X\ \rrbracket \Longrightarrow lfp\ f = lfp\ g\text{”}$

It says that if we have an upper bound U for the *lfp* of both f and g, and f and g behave the same below U, then they have the same *lfp*.

The following two tweaks improve proof automation:

lemmas [*simp*] = *L.simps(5)*
lemmas *L_mono2* = *L_mono*[*unfolded mono_def*]

To show that *bury* is idempotent we need a lemma:

lemma *L_bury*[*simp*]: $\text{“}X \subseteq Y \Longrightarrow L\ (bury\ c\ Y)\ X = L\ c\ X\text{”}$
proof(*induction c arbitrary*: *X Y*)

The proof is straightforward except for the case *WHILE b DO c*. The definition of L in this case means that we have to show an equality of two *lfp*s. Lemma *lfp_eq* comes to the rescue. We recommend the upper bound $lfp\ (\lambda Z.\ vars\ b \cup Y \cup L\ c\ Z)$. One of the two upper bound assumptions of lemma *lfp_eq* can be proved by showing that U is a pre-fixpoint of f or g (see lemma *lfp_lowerbound*).

Now we can prove idempotence of *bury*, again by induction on c, but this time even the *While* case should be easy.

lemma *bury_bury*: $\text{“}X \subseteq Y \Longrightarrow bury\ (bury\ c\ Y)\ X = bury\ c\ X\text{”}$

Idempotence is a corollary:

corollary *"bury (bury c X) X = bury c X"*

Homework 8.2 Denotational Semantics

Submission until Monday, Dec 16, 10:00am.

We again consider the extension of IMP with non-determinism from exercise sheet 5. However, this time, we add a construct *LOOP c* for non-deterministic looping. The idea is that *LOOP c* can non-deterministically decide to either stop iteration and do nothing or to execute the loop body *c* for one more time.

```
datatype
  com = SKIP
      | Assign vname aexp     ("_ ::= _" [1000, 61] 61)
      | Seq   com  com        ("_;;/ _"  [60, 61] 60)
      | If    bexp com com    ("(IF _/ THEN _/ ELSE _)"  [0, 0, 61] 61)
      | While  bexp com       ("(WHILE _/ DO _)"  [0, 61] 61)
      | Or com com            ("_ OR _" [57,58] 59)
      | ASSUME bexp
      | Loop com              ("(LOOP _)"  [61] 61)
```

First extend the big-step semantics with this new construct:

inductive
big_step :: *"com × state ⇒ state ⇒ bool"* (**infix** *"⇒"* 55)
where
Skip: $"(SKIP,s) \Rightarrow s"$ |
Assign: $"(x ::= a,s) \Rightarrow s(x := aval\ a\ s)"$ |
Seq: $"[\![(c_1,s_1) \Rightarrow s_2;\ (c_2,s_2) \Rightarrow s_3]\!] \Longrightarrow (c_1;;c_2,\ s_1) \Rightarrow s_3"$ |
IfTrue: $"[\![bval\ b\ s;\ (c_1,s) \Rightarrow t]\!] \Longrightarrow (IF\ b\ THEN\ c_1\ ELSE\ c_2,\ s) \Rightarrow t"$ |
IfFalse: $"[\![\neg bval\ b\ s;\ (c_2,s) \Rightarrow t]\!] \Longrightarrow (IF\ b\ THEN\ c_1\ ELSE\ c_2,\ s) \Rightarrow t"$ |
WhileFalse: $"\neg bval\ b\ s \Longrightarrow (WHILE\ b\ DO\ c,s) \Rightarrow s"$ |
WhileTrue: $"[\![bval\ b\ s_1;\ (c,s_1) \Rightarrow s_2;\ (WHILE\ b\ DO\ c,\ s_2) \Rightarrow s_3]\!] \Longrightarrow (WHILE\ b\ DO\ c,\ s_1) \Rightarrow s_3"$ |
OrLeft: $"[\![(c_1,s) \Rightarrow s']\!] \Longrightarrow (c_1\ OR\ c_2,s) \Rightarrow s'"$ |
OrRight: $"[\![(c_2,s) \Rightarrow s']\!] \Longrightarrow (c_1\ OR\ c_2,s) \Rightarrow s'"$ |
Assume: $"bval\ b\ s \Longrightarrow (ASSUME\ b,\ s) \Rightarrow s"$ |
— Your cases here:

Now, give a denotational semantics for this language:

type_synonym *com_den* = *"(state × state) set"*

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)

$= \{(s,t).\ \textit{if bval b s then}\ (s,t) \in D\ c1\ \textit{else}\ (s,t) \in D\ c2\}$" |
"$D\ (\textit{WHILE b DO c}) = \textit{lfp}\ (W\ (\textit{bval b})\ (D\ c))$" |
— Your cases here:

Then correct the proof of the equivalence theorem between big-step and denotational semantics:

theorem *denotational_is_big_step*:
"$(s,t) \in D(c) \ = \ ((c,s) \Rightarrow t)$"

Use theory *HOL−IMP.Denotational* as a template for the proof!