Technische Universität München
Fakultät für Informatik
Dr. Peter Lammich
Simon Wimmer

WS 2018/19
29.01.2019

Semantics of Programming Languages

Exercise Sheet 14

Exercise 14.1 Independence analysis

In this exercise you first prove that the execution of a command only depends on its used (i.e., read or assigned) variables. Then you use this to prove commutativity of sequential composition.

Note that this development is entirely done on the small-step semantics.

First show that arithmetic and boolean expressions only depend on the variables occuring in them

lemma [*simp*]: "$s1 = s2 \ on \ X \Longrightarrow vars \ a \subseteq X \Longrightarrow aval \ a \ s1 = aval \ a \ s2$"

lemma [*simp*]: "$s1 = s2 \ on \ X \Longrightarrow vars \ b \subseteq X \Longrightarrow bval \ b \ s1 = bval \ b \ s2$"

Next, show that executing a command does not invent new variables

lemma *vars_subsetD*[*dest*]: "$(c, s) \to (c', s') \Longrightarrow vars \ c' \subseteq vars \ c$"

And that the effect of a command is confined to its variables

lemma *small_step_confinement*: "$(c, s) \to (c', s') \Longrightarrow s = s' \ on \ UNIV - vars \ c$"
lemma *small_steps_confinement*: "$(c, s) \to* (c', s') \Longrightarrow s = s' \ on \ UNIV - vars \ c$"

Hint: These proofs should go through (mostly) automatically by induction.

Now, we are ready to show that commands only depend on the variables they use:

lemma *small_step_indep*:
"$(c, s) \to (c', s') \Longrightarrow s = t \ on \ X \Longrightarrow vars \ c \subseteq X \Longrightarrow \exists t'. \ (c, t) \to (c', t') \land s' = t' \ on \ X$"
lemma *small_steps_indep*: "$[\![(c, s) \to* (c', s'); s = t \ on \ X; vars \ c \subseteq X]\!]$
$\Longrightarrow \exists t'. \ (c, t) \to* (c', t') \land s' = t' \ on \ X$"

Two lemmas that may prove useful for the next proof.

lemma *small_steps_SeqE*: "$(c1 \ ;; \ c2, s) \to* (SKIP, s')$
$\Longrightarrow \exists t. \ (c1, s) \to* (SKIP, t) \land (c2, t) \to* (SKIP, s')$"
by (*induction* "*c1 ;; c2*" *s SKIP s' arbitrary*: *c1 c2 rule*: *star_induct*)
(*blast intro*: *star.step*)

lemma *small_steps_SeqI*: "$\llbracket(c1,\ s) \rightarrow* (SKIP,\ s'); (c2,\ s') \rightarrow* (SKIP,\ t)\rrbracket$
$\Longrightarrow (c1\ ;;\ c2,\ s) \rightarrow* (SKIP,\ t)$"
by (*induction c1 s SKIP s' rule*: *star_induct*)
(*auto intro*: *star.step*)

As we operate on the small-step semantics we also need our own version of command equivalence. Two commands are equivalent iff a terminating run of one command implies a terminating run of the other command. And, of course the terminal state needs to be equal when started in the same state.

definition *equiv_com* :: "*com* $\Rightarrow$ *com* $\Rightarrow$ *bool*" (**infix** "$\sim_s$" *50*) **where**
"$c1 \sim_s c2 \longleftrightarrow (\forall s\ t.\ (c1,\ s) \rightarrow* (SKIP,\ t) \longleftrightarrow (c2,\ s) \rightarrow* (SKIP,\ t))$"

Show that we defined an equivalence relation

lemma *ec_refl*[*simp*]: "*equiv_com c c*"
lemma *ec_sym*: "*equiv_com c1 c2* $\longleftrightarrow$ *equiv_com c2 c1* "
lemma *ec_trans*[*trans*]: "*equiv_com c1 c2* $\Longrightarrow$ *equiv_com c2 c3* $\Longrightarrow$ *equiv_com c1 c3*"

Note that our small-step equivalence matches the big-step equivalence

lemma "$c1{\sim_s}c2 \longleftrightarrow c1{\sim}c2$" **unfolding** *equiv_com_def* **by** (*metis big_iff_small*)

Finally, show that commands that share no common variables can be re-ordered

theorem *Seq_equiv_Seq_reorder*:
assumes *vars*: "*vars c1* $\cap$ *vars c2* = {}"
shows "$(c1\ ;;\ c2) \sim_s (c2\ ;;\ c1)$"
proof –
{

As the statement is symmetric, we can take a shortcut by only proving one direction:

fix *c1 c2 s t*
assume *Seq*: "$(c1\ ;;\ c2,\ s) \rightarrow* (SKIP,\ t)$" **and** *vars*: "*vars c1* $\cap$ *vars c2* = {}"
have "$(c2\ ;;\ c1,\ s) \rightarrow* (SKIP,\ t)$"
} **with** *vars* **show** *?thesis* **unfolding** *equiv_com_def* **by** (*metis Int_commute*)
qed

Homework 14.1 Fixed Point Theory

Submission until Tuesday, February 5, 2019, 10:00am.

The following is the (slightly modified) text of an old exam exercise. In a real exam, we would ask you to solve the exercise on paper. For now, you should still use Isabelle. We expect you to write a detailed Isar proof, with the same level of detail you would provide in a pen&paper proof. Only trivial proof steps should be discharged by automatic methods.

Let $f :: 'a\ set \Rightarrow 'a\ set$ be a monotonic function. Moreover, let x_0 be post-fixpoint of f, i.e. $x_0 \subseteq f\, x_0$. Prove:

$$\bigcup\ \{f^i(x_0) \mid i \in \mathbb{N}\} \subseteq \bigcup\ \{f^{i+1}(x_0) \mid i \in \mathbb{N}\}$$

Hint: Set union satisfies the following properties:

upper $B \in A \Longrightarrow B \subseteq \bigcup A$

least $(\bigwedge X.\ X \in A \Longrightarrow X \subseteq C) \Longrightarrow \bigcup A \subseteq C$

In Isabelle:

theorem *postfix_step*: "$\bigcup\ \{(f\hat{}\hat{}i)(x_0) \mid i :: nat.\ True\} \subseteq \bigcup\ \{(f\hat{}\hat{}(Suc\ i))(x_0) \mid i :: nat.\ True\}$"

In Isabelle the two properties are: *Union_upper* and *Union_least*.

Homework 14.2 Kleene Fixed Point Theorem

Submission until Tuesday, February 5, 2019, 10:00am.

In this homework, we are going to prove a particular version of the Kleene fixed-point theorem. You should work at the same level of detail as in the previous exercise. Assume we are given a *continuous* function f:

context
  fixes f :: "$'a\ set \Rightarrow 'a\ set$"
  assumes *continuous*: "$\bigwedge X.\ X \neq \{\} \Longrightarrow f\ (\bigcup X) = \bigcup (f\ `\ X)$"
begin

In the end we want to prove the following: $lfp\ f = \bigcup\{(f\hat{}\hat{}i)\ \{\} \mid i.\ True\}$.

First show that continuity implies monotonicity. *Hint:* consider a 2-element set.

theorem *mono*: "$x \subseteq y \Longrightarrow f\, x \subseteq f\, y$"

From now, you can use the following lemma:

lemma *lfp_fold*: "$f\ (lfp\ f) = lfp\ f$"
  using *lfp_unfold mono* **unfolding** *mono_def* **by** *blast*

Prove the first direction of the theorem. *Hint:* Show that every f^i is below *lfp f*.

theorem *lfp_ge*: "$\bigcup\{(f\hat{}\hat{}i)\ \{\} \mid i.\ True\} \subseteq lfp\ f$"

Prove the other direction of the theorem. *Hint:* Exploit continuity.

theorem *lfp_le*: "$lfp\ f \subseteq \bigcup\{(f\hat{}\hat{}i)\ \{\} \mid i.\ True\}$" (**is** "$\_ \subseteq \bigcup\ ?S$")

corollary
  "$lfp\ f = \bigcup\{(f\hat{}\hat{}i)\ \{\} \mid i.\ True\}$"
  using *lfp_le lfp_ge* **..**