Exercise 8.1  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.

term “s = t on X”

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

lemma [simp]: “s1 = s2 on X ⇒ vars a ⊆ X ⇒ aval a s1 = aval a s2”

lemma [simp]: “s1 = s2 on X ⇒ vars b ⊆ X ⇒ bval b s1 = bval b s2”

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

lemma vars_subsetD[dest]: “(c, s) → (c’, s’) ⇒ vars c’ ⊆ vars c”

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

lemma small_step_confinement: “(c, s) → (c’, s’) ⇒ s = s’ on UNIV − vars c”

lemma small_steps_confinement: “(c, s) →∗ (c’, s’) ⇒ 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) → (c’, s’) ⇒ s = t on X ⇒ vars c ⊆ X ⇒ ∃ t’. (c, t) → (c’, t’) ∧ s’ = t’ on X”

lemma small_steps_indep: “[[(c, s) →∗ (c’, s’); s = t on X; vars c ⊆ X]]
⇒ ∃ t’. (c, t) →∗ (c’, t’) ∧ s’ = t’ on X”

Two lemmas that may prove useful for the next proof.

lemma small_steps_SeqE: “(c1 ;; c2, s) →∗ (SKIP, s’)
⇒ ∃ t. (c1, s) →∗ (SKIP, t) ∧ (c2, t) →∗ (SKIP, s’)”

by (induction “c1 ;; c2” s SKIP s’ arbitrary: c1 c2 rule: star_induct)
(blast intro: star_step)
lemma small_steps_SeqI: "[(c1, s) \rightarrow^* (\text{SKIP}, s'); (c2, s') \rightarrow^* (\text{SKIP}, t)]
\Rightarrow (c1 ;; c2, s) \rightarrow^* (\text{SKIP}, t)"
by (induction c1 s SKIP s' rule: star_induct)
\hfill (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 :: 
  "\text{com} \Rightarrow \text{com} \Rightarrow \text{bool}" \hfill (infix "\sim_s" 50) \hfill where
  "c1 \sim_s c2 \longleftrightarrow (\forall s t. (c1, s) \rightarrow^* (\text{SKIP}, t) \longleftrightarrow (c2, s) \rightarrow^* (\text{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 \Rightarrow equiv_com c2 c3 \Rightarrow 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:
  \hfill assumes vars: "vars c1 \cap vars c2 = {}"
  \hfill 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^* (\text{SKIP}, t)" and vars: "vars c1 \cap vars c2 = {}"
  have "(c2 ;; c1, s) \rightarrow^* (\text{SKIP}, t)"
  } with vars show \hfill \text{thesis} \hfill unfolding equiv_com_def by (metis Int_commute)
qed
Homework 8.1 Idempotence of Dead Variable Elimination

Submission until Tuesday, December 17, 2013, 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 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)” |
“bury (c_1;; c_2) X = (bury c_1 (L c_2 X);; bury c_2 X)” |
“bury (IF b THEN c_1 ELSE c_2) X = IF b THEN bury c_1 X ELSE bury c_2 X” |
“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:

\[ \text{lfp_unfold: } \text{mono } ?f \implies \text{lfp } ?f = ?f (\text{lfp } ?f) \]
\[ \text{lfp_lowerbound: } ?f ?A \leq ?A \implies \text{lfp } ?f \leq ?A \]

Prove the following lemma for showing that two fixpoints are the same, where mono_def:

\[ \text{mono } ?f = (\forall x y. x \leq y \implies ?f x \leq ?f y). \]

\[ \text{lemma lfp_eq: } \[ [\text{mono } f; \text{mono } g; \text{lfp } f \subseteq ?U; \text{lfp } g \subseteq ?U; \forall X. X \subseteq ?U \implies f X = g X ] \implies \text{lfp } f = \text{lfp } g \] \]

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:

\[ \text{lemmas [simp]} = \text{L.simps}(5) \]
\[ \text{lemmas L_mono2} = \text{L_mono[unfolded mono_def]} \]

To show that bury is idempotent we need a lemma:

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

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 lfps. Lemma \[ \text{[mono } ?f; \text{mono } ?g; \text{lfp } ?f \subseteq ?U; \text{lfp } ?g \subseteq ?U; \forall X. X \subseteq ?U \implies ?f X = ?g X ] \implies \text{lfp } ?f = \text{lfp } ?g \]
comes to the rescue. We recommend the upper bound lfp (λZ. vars b ∪ Y ∪ L c Z).

One of the two upper bound assumptions of lemma \[ \text{[mono } ?f; \text{mono } ?g; \text{lfp } ?f \subseteq ?U; \text{lfp } ?g \subseteq ?U; \forall X. X \subseteq ?U \implies ?f X = ?g X ] \implies \text{lfp } ?f = \text{lfp } ?g \]
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: “$X \subseteq Y \implies bury (bury c Y) X = bury c X$”

Idempotence is a corollary:
corollary “$bury (bury c X) X = bury c X$”

Homework 8.2 Independence, in Parallel Programs

Submission until Tuesday, December 17, 2013, 10:00am. 5 bonus points.

Extend the while language with a parallel operator $\parallel$, such that $c1\parallel c2$ executes commands $c1$ and $c2$ in parallel, and define a small step semantics. Show that, for your parallel language, you have $\text{vars } c1 \cap \text{vars } c2 = \{\} \implies (c1 \parallel c2) \sim_s (c1 ;; c2)$, i.e., sequential composition can be transformed to parallel execution if it works on different variables.

To solve this exercise, use the template from the webpage, which provides a sample solution from exercise 8.1 adapted to the parallel commands.

end