Technische Universität München
Institut für Informatik
Prof. Tobias Nipkow, Ph.D.
Andrei Popescu, Peter Lammich

WS 2012/13
4. 12. 2012

Semantics of Programming Languages

Exercise Sheet 8

Exercise 8.1 Definite Initialization Analysis

In the lecture, you have seen a definite initialization analysis that was based on the big-step semantics. Definite initialization analysis can also be based on a small-step semantics. Furthermore, the ternary predicate D from the lecture can be split into two parts: a function $AA :: com \Rightarrow name\ set$ ("assigned after") which collects the names of all variables assigned by a command and a binary predicate $D :: name\ set \Rightarrow com \Rightarrow bool$ which checks that a command accesses only previously assigned variables. Conceptually, the ternary predicate from the lecture (call it D_{lec}) and the two-step approach should relate by the equivalence $D\ V\ c \longleftrightarrow D_{lec}\ V\ c\ (V \cup AA\ c)$

1. Download the theory `ex08_template` and study the already defined small-step semantics for definite analysis.
2. Define the function AA which computes the set of variables assigned after execution of a command. Furthermore, define the predicate D which checks if a command accesses only assigned variables, assuming the variables in the argument set are already assigned.
3. Prove progress and preservation of D with respect to the small-step semantics, and conclude soundness of D. You may use (and then need to prove) the lemmas *D_incr* and *D_mono*.

Homework 8.1 Independence analysis

Submission until Tuesday, December 11, 2012, 10:00am.

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 for commands with disjoint used variables. You shall employ the big-step semantics. A template will be provided for this homework.

Start with defining the (used) variables of a command, i.e., all the variables appearing in the command. For notation convenience, you should proceed similarly to what we did for expressions in the theory *Vars*, namely, register the type of commands as an instance of the class *vars*—then you can use the name *vars* for the newly defined operation on commands. We have started the definition, you need to add the remaining clauses.

instantiation *com* :: *vars*
fun *vars_com* :: *"com ⇒ vname set"* **where**
"vars_com SKIP = {}"

A first thing you need to prove is that the effect of executing a command is confined to its variables, in that the part of the state not involving these variables does not change. (Recall the *eq_on* abbreviation from theory *Vars*.)

lemma *confinement*: *"(c,s) ⇒ s' ⟹ s = s' on (UNIV − vars c)"*

Hint: The proof should go through automatically by induction.

Now you should prove that the part of the initial state not involving the variables of a command is irrelevant for its execution. We have started the proof for you.

lemma *irrelevance*:
"⟦(c,s1) ⇒ s1'; s1 = s2 on X; vars c ⊆ X⟧ ⟹ ∃ s2'. (c,s2) ⇒ s2' ∧ s1' = s2' on X"
proof (*induction arbitrary*: *s2 rule*: *big_step_induct*)

Finally, you need to prove the commutativity of sequential composition for two commands having mutually disjoint variables: first a helper lemma *independence_aux*, then the desired fact *independence*. Note that in the statement of the latter we use the big-step equivalence relation defined in theory *Big_Step*.

lemma *independence_aux*:
assumes *v*: *"vars c1 ∩ vars c2 = {}"* **and** *c12*: *"(c1 ; c2, s) ⇒ s12"*
shows *"(c2 ; c1, s) ⇒ s12"*

Hint for the proof of lemma *independence_aux*: Let *X1* consist of all the variables not used in *c1*, namely, *UNIV − vars c1*. Similarly, let *X2* be *UNIV − vars c2*. From the hypotheses, obtain *s1* such that (*c1*, *s*) ⇒ *s1* and (*c2*, *s1*) ⇒ *s12*. Then take the other route (first executing *c2* and then *c1*), namely, obtain *s2* and *s21* such that (*c2*,*s*) ⇒ *s2* and (*c1*, *s2*) ⇒ *s21*, also making sure to carry relevant information about *X1* and *X2*. (Draw a picture!) For *s12* and *s21*, show that they are equal both on *X1* and on *X2*, which ensures that they are equal.

In the above process, you do *not* need induction, but need to apply the lemmas *confinement* and *irrelevance* several times.

lemma *independence*: *"vars c1 ∩ vars c2 = {} ⟹ c1 ; c2 ∼ c2 ; c1"*

We also include an extra-credit task, for 5 additional points: Currently, in lemma *independence* we assume that the used variables of *c1* and *c2* are disjoint. However, intuitively, one only needs to assume the used variables of *c1* disjoint from the assigned (written) variables of *c2* and vice-versa. (Thus, e.g., *c1* and *c2* should be allowed to read the same variable x provided neither of them modifies x.)

Your task to state and prove an improved version of lemma *independence*.

Homework 8.2 Fixed point reasoning

Submission until Tuesday, December 11, 2012, 10:00am.

In the lecture, you have seen the Knaster-Tarski least fixed point theorem. The relevant constant is *lfp* :: $('a \Rightarrow 'a) \Rightarrow 'a$, which assumes a complete lattice order $\leq$ on $'a$ and returns, for each monotonic operator $f :: 'a \Rightarrow 'a$, its least fixed point *lfp f*.

In the lectures as well as in this exercise, one only deals with the case where $'a$ is $'b$ *set* (the type of sets over an arbitrary type $'b$) and $\leq$ is $\subseteq$ (set inclusion). You need to prove the following fact concerning function image and set complement:

lemma *decomposition*: *"∃ X. X = − (g ‘ (− (f ‘ X)))"*

Hint: Look up and use the theorems *lfp_unfold* and *monoI*. First try to do a pen-and-paper proof. Note that:

- If $A :: 'b$ *set*, then $-A$ denotes the complement of A, that is, the set of all elements (of type $'b$) that are not in A;
- h ‘ A denotes the image of A through h, that is, the set of all elements of the form $h\ a$ with $a \in A$.

The automatic methods (auto, blast, etc.) are well customized to handle image and complement, and therefore you will not need to explicitly invoke any lemma about these operators.