Technische Universität München
Institut für Informatik
Prof. Tobias Nipkow, Ph.D.
Peter Lammich, Johannes Hölzl

WS 2013/14
3. 12. 2013

Semantics of Programming Languages

Exercise Sheet 7

Exercise 7.1 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 `ex07_tmpl.thy` 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 7.1 Erasing private parts

Submission until Tuesday, December 10, 2013, 10:00am.

Note: In this homework, you will do induction proofs over the big-step semantics. In these proofs, the cases *WhileFalse*, *IfTrue*, and *IfFalse* are similar to the *WhileTrue*-case.
To save you from additional (repetitive) work, you may use *sorry* for the cases *While-False*, *IfTrue*, and *IfFalse*.
However, if you cannot prove the *WhileTrue* case, try proving the other cases first, this may get you some insight and partial score.

In this homework, you should define a function that erases confidential ("private") parts of a command:

fun *erase* :: *"level* $\Rightarrow$ *com* $\Rightarrow$ *com"*

Function *erase* l should replace all assignments to variables with security level $\geq l$ by *SKIP*. It should also erase certain *IF*s and *WHILE*s, depending on the security level of the boolean condition. Now show that c and *erase* l c behave the same on the variables up to level l:

theorem
"$[\![$ $(c,s) \Rightarrow s'$; $(\mathit{erase}\ l\ c,t) \Rightarrow t'$; $0 \vdash c$; $s = t\ (< l)$ $]\!]$
$\Longrightarrow s' = t'\ (< l)$"

This lemma looks remarkably like the noninterference lemma in *Sec_Typing* (although $\leq$ has been replaced by $<$). You may want to start with that proof and modify it where needed. A lot of local modifications will be necessary, but the structure should remain the same. You may also need one or two simple additional lemmas (for example $\ldots \Longrightarrow$ $\mathit{aval}\ a\ s_1 = \mathit{aval}\ a\ s_2$), but nothing major.

In the theorem above we assumed that both $(c,\ s)$ and $(\mathit{erase}\ l\ c,\ t)$ terminate. How about the following two properties:

lemma "$[\![$ $(c,s) \Rightarrow s'$; $0 \vdash c$; $s = t\ (< l)$ $]\!]$
$\Longrightarrow \exists t'.\ (\mathit{erase}\ l\ c,t) \Rightarrow t' \wedge s' = t'\ (< l)$"
lemma "$[\![$ $(\mathit{erase}\ l\ c,s) \Rightarrow s'$; $0 \vdash c$; $s = t\ (< l)$ $]\!] \Longrightarrow \exists t'.\ (c,t) \Rightarrow t'$"

Give proofs or counterexamples.