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

WS 2016/17
10. 01. 2017

Semantics of Programming Languages

Exercise Sheet 10

Announcement: The deadline for the Christmas homework is extended to the same deadline as for homework 10.1.

Exercise 10.1 Available Expressions

Regard the following function AA, which computes the *available assignments* of a command. An available assignment is a pair of a variable and an expression such that the variable holds the value of the expression in the current state. The function $AA\ c\ A$ computes the available assignments after executing command c, assuming that A is the set of available assignments for the initial state.

Note that available assignments can be used for program optimization, by avoiding recomputation of expressions whose value is already available in some variable.

```
fun AA :: "com ⇒ (vname × aexp) set ⇒ (vname × aexp) set" where
  "AA SKIP A = A" |
  "AA (x ::= a) A = (if x ∈ vars a then {} else {(x, a)})
   ∪ {(x', a'). (x', a') ∈ A ∧ x ∉ {x'} ∪ vars a'}" |
  "AA (c1;; c2) A = (AA c2 ∘ AA c1) A" |
  "AA (IF b THEN c1 ELSE c2) A = AA c1 A ∩ AA c2 A" |
  "AA (WHILE b DO c) A = A ∩ AA c A"
```

Show that available assignment analysis is a gen/kill analysis, i.e., define two functions *gen* and *kill* such that
$AA\ c\ A = (A \cup gen\ c) - kill\ c$.

Note that the above characterization differs from the one that you have seen on the slides, which is $(A - kill\ c) \cup gen\ c$. However, the same properties (monotonicity, etc.) can be derived using either version.

```
fun gen :: "com ⇒ (vname × aexp) set"
and "kill" :: "com ⇒ (vname × aexp) set"

lemma AA_gen_kill: "AA c A = (A ∪ gen c) − kill c"
```

Hint: Defining *gen* and *kill* functions for available assignments will require *mutual recursion*, i.e., *gen* must make recursive calls to *kill*, and *kill* must also make recursive calls to *gen*. The **and**-syntax in the function declaration allows you to define both functions

simultaneously with mutual recursion. After the **where** keyword, list all the equations for both functions, separated by | as usual.

Now show that the analysis is sound:

theorem *AA_sound*:
"$(c, s) \Rightarrow s' \Longrightarrow \forall (x, a) \in AA\ c\ \{\}.\ s'\ x = aval\ a\ s'$"

Hint: You will have to generalize the theorem for the induction to go through.

Homework 10.1 Small-Step Security Typing

Submission until Tuesday, January 17, 2017, 10:00am. In this exercise we will consider security typing for the small-step semantics. You should start with a copy of `~~/src/HOL/IMP/Sec_Typing.thy`.

Prove confinement. *Hint:* In addition to the obvious auxiliary lemma for a single step, you may need another one.

lemma *confinement_steps*: "$[\![(c, s) \rightarrow* (c', s');\ l \vdash c]\!] \Longrightarrow s = s' (< l)$"

Prove noninterference:

theorem *noninterference*:
"$[\![(c, s) \rightarrow (c', s'); (c, t) \rightarrow (c', t');\ 0 \vdash c;\ s = t (\leq l)]\!]$
$\Longrightarrow s' = t' (\leq l)$"

Does the following also hold?

theorem *noninterference'*:
"$[\![(c, s) \rightarrow* (c', s'); (c, t) \rightarrow* (c', t');\ 0 \vdash c;\ s = t (\leq l)]\!]$
$\Longrightarrow s' = t' (\leq l)$"
oops