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 $AA :: \text{\texttt{"com \Rightarrow (vname \times aexp) set \Rightarrow (vname \times aexp) set"}}$ where

$AA \text{ SKIP } A = A$ |  
$AA (x ::= a) A = (\text{if } x \in \text{vars } a \text{ then } \emptyset \text{ else } \{(x, a)\})$  
$\cup \{(x', a'). (x', a') \in A \land x \notin \{x'\} \cup \text{vars } a'\}" |$  
$AA (c_1 ;; c_2) A = (AA c_2 \circ AA c_1) A" |$  
$AA (\text{IF } b \text{ THEN } c_1 \text{ ELSE } c_2) A = AA c_1 A \cap AA c_2 A" |$  
$AA (\text{WHILE } b \text{ DO } c) A = A \cap AA c A"$  

Show that available assignment analysis is a gen/kill analysis, i.e., define two functions $\text{gen}$ and $\text{kill}$ such that

$AA c A = (A \cup \text{gen } c) - \text{kill } c.$

Note that the above characterization differs from the one that you have seen on the slides, which is $(A - \text{kill } c) \cup \text{gen } c.$ However, the same properties (monotonicity, etc.) can be derived using expressions whose value is already available in some variable.

fun $\text{gen :: \"com \Rightarrow (vname \times aexp) set\"}$  
and $\text{\"kill :: \"com \Rightarrow (vname \times aexp) set\"}$

lemma $AA_{gen \_kill}: \text{\"AA } c \text{ A} = (A \cup \text{gen } c) - \text{kill } c\text{\"}$

Hint: Defining $\text{gen}$ and $\text{kill}$ functions for available assignments will require mutual recursion, i.e., $\text{gen}$ must make recursive calls to $\text{kill}$, and $\text{kill}$ must also make recursive calls to $\text{gen}$. The and-syntax in the function declaration allows you to define both functions
simultaneously with mutual recursion. After the \textbf{where} keyword, list all the equations
for both functions, separated by $|$ as usual.

Now show that the analysis is sound:

\textbf{theorem} \ AA\_sound:\ 
\begin{equation*}
(c, s) \Rightarrow s' \implies \forall (x, a) \in AA\ c\ \{\}.\ s' \ x = \text{aval}\ a\ s'\
\end{equation*}

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

\section*{Homework 10.1 \ Small-Step Security Typing}

\textit{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 \texttt{~/src/HOL/IMP/Sec_Typing.thy}.

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

\textbf{lemma} confinement\_steps:\ 
\begin{equation*}
\left[\begin{array}{c}
(c, s) \rightarrow^* (c', s');\ l \vdash c \\
\end{array}\right] \Rightarrow s = s' (< l)
\end{equation*}

Prove noninterference:

\textbf{theorem} \ noninterference:\ 
\begin{equation*}
\left[\begin{array}{c}
(c, s) \rightarrow (c', s');\ (c, t) \rightarrow (c', t');\ 0 \vdash c;\ s = t (\leq l) \\
\end{array}\right] \Rightarrow s' = t' (\leq l)
\end{equation*}

Does the following also hold?

\textbf{theorem} \ noninterference':\ 
\begin{equation*}
\left[\begin{array}{c}
(c, s) \rightarrow^* (c', s');\ (c, t) \rightarrow^* (c', t');\ 0 \vdash c;\ s = t (\leq l) \\
\end{array}\right] \Rightarrow s' = t' (\leq l)
\end{equation*}

\texttt{oops}