ETH zürich

Alexander J. Summers

Program Verification

Exercise Solutions 9: Heap Reasoning and Permissions

Assignment 1 (Pure Assertions)

1. By (structural) induction on A.

   (Case A is e for some expression e:) Then we have:

   $$\begin{array}{rcl} H, P, \sigma \vDash A & \Leftrightarrow & \lceil e \rfloor = \mathit{true} \\ & \Leftrightarrow & H, \varnothing, \sigma \vDash A \end{array}$$

   (Case A is $A_1 * A_2$ for some A_1 and A_2:) Then we have:

   $$\begin{array}{rcl} H, P, \sigma \vDash A & \Leftrightarrow & \exists P_1, P_2.\ P = P_1 \uplus P_2 \textit{ and } H, P_1, \sigma \vDash A_1 \textit{ and } H, P_2, \sigma \vDash A_2 \\ & \Rightarrow & H, \varnothing, \sigma \vDash A_1 \textit{ and } H, \varnothing, \sigma \vDash A_2 \textit{ (by induction hypothesis, twice)} \\ & \Rightarrow & H, \varnothing, \sigma \vDash A_1 * A_2 \end{array}$$

   All other cases follow analogously, by a straightforward induction argument.

2. To prove equivalence, we need to show, for all such A' and A that: $\forall H, P, \sigma.(H, P, \sigma \vDash A * A' \Leftrightarrow H, P, \sigma \vDash A \wedge A')$. We show the $\Rightarrow$ and $\Leftarrow$ directions of this property, for arbitrary such A' and (pure) A, as follows:

   ($\Rightarrow$:) To show this direction of the result, we need an additional lemma, effectively stating that increasing the permissions held in a state will never make assertions false (this result was discussed in the lecture). If we use $P_1 \sqsubseteq P_2$ to mean that P_2 has at least as much permission as P_1 for all locations, then lemma can be stated as follows:

   $$\forall A, H, P_1, P_2, \sigma.(\textit{if } H, P_1, \sigma \vDash A \textit{ and } P_1 \sqsubseteq P_2 \textit{ then } H, P_2, \sigma \vDash A)$$

   This lemma can be proved by straightforward induction on A. Using the lemma, we can now show the intended result:
   Let H,P,σ be arbitrary, and assume $H, P, \sigma \vDash A * A'$. Then, by definition, there are some P_1 and P_2 such that: $P = P_1 \uplus P_2$ *and* $H, P_1, \sigma \vDash A$ *and* $H, P_2, \sigma \vDash A'$. Note that, $P_1 \sqsubseteq P$ and $P_2 \sqsubseteq P$. Therefore, by the lemma above, we have $H, P, \sigma \vDash A$ *and* $H, P, \sigma \vDash A'$, and thus, $H, P, \sigma \vDash A \wedge A'$, as required.

   ($\Rightarrow$:) Let H,P,σ be arbitrary, and assume $H, P, \sigma \vDash A \wedge A'$. By definition, $H, P, \sigma \vDash A$ and $H, P, \sigma \vDash A'$. By part (1), we have $H, \varnothing, \sigma \vDash A$. Therefore, since $\varnothing \uplus P = P$, we have $H, P, \sigma \vDash A * A'$, as required.

Assignment 2 (Old Expressions and Procedure Calls)

1.
```
procedure assign(x,y)
  requires acc(x.val) * acc(y.val, 1/2)
  ensures acc(x.val) * acc(y.val, 1/2) * x.val==old(y.val)
{
  x.val := y.val;
}
```

   Note that the fractional amount chosen for `y.val` could equally be any fraction strictly between 0 and 1.

2. The only global state in this case is the heap. Modifications to the heap are handled via permission accounting: the heap locations whose values can be framed across a procedure call are exacmply those to which (at least some fractional) permission is retained across the call (i.e. not all of the caller's permission is required by the precondition of the procedure). Using a fractional permission allows for the possibility that a called can retain some permission to `y.val` (the exact choice of fractional amount could happen to be problematic, if a caller happens not to have more than the chosen amount, but this is impossible to avoid with only concrete fractional permissions, as covered in this course).

3. Here is an idea for a rule, inspired by that for global variables/modifies clauses on slide 201 (for a procedure p with precondition *pre*, postcondition *post*, formal in- and out-parameters $\vec{x}$ and $\vec{y}$, respectively):

$$\frac{\{\vec{e'}\} = \{e' \mid \texttt{old}(e') \in \mathit{post}[\vec{e}/\vec{x}][\vec{z}/\vec{y}]\} \quad \{\vec{z}\} \cap (FV(\vec{e}) \cup \{\vec{o}\}) = \varnothing}{\{\mathit{pre}[\vec{e}/\vec{x}] \wedge \vec{o} = \vec{e'}\}\ \texttt{call}\ \vec{z}\ \texttt{:=}\ \texttt{p}(\vec{e})\ \{\mathit{post}[\vec{e}/\vec{x}][\vec{z}/\vec{y}][\vec{o}/\texttt{old}(\vec{e'})]\}}$$

   The idea here is that we can use additional variables $\vec{o}$ to equate with the values of each expression e' used in an old-expression in the postcondition of the procedure.

   This rule works fine for old-expressions which do not occur under conditionals (such as the simple examples discussed above). In general, however, this rule has the problem that for old-expressions under conditionals in the postcondition, requiring us to nonetheless be able to evaluate them in the state before the call might be overly-restrictive (as a trivial case, suppose that `false ==> old(x.f)` occurred in the postcondition: then it would be fine to call the procedure in a state in which no permission to `x.f` was held, but this rule would not allow it). We could instead refine the rule to filter $\vec{e'}$ to include only expressions which can evaluated in the pre-state (i.e. we hold sufficient permissions). We won't formalise the general rule, here (in practice, the general case can be handled in tools such as Viper by allowing explicit evaluation of expressions in earlier states, via labelled-old expressions, which avoids the need to "save" their values up-front).