Alexander J. Summers

Program Verification

Exercise Solutions 7: Verification Condition Generation

Assignment 1 (Avoiding Duplication)

Consider the alternative (wrong) definition, in which we conjoin the "definition" of the fresh propositional variable; i.e. imagine that we defined *wlp*$(s_1 \; [] \; s_2, A) = (p \Leftrightarrow A) \land$ *wlp*$(s_1, A) \land$ *wlp*(s_2, A). As a precondition, this assertion is too strong for a number of reasons. Even the single conjunct *wlp*(s_1, p) would be too strong a requirement; since the propositional variable p is fresh, there is no reasonable way in which we could guarantee that *wlp*(s_1, p) holds (consider, for a simple example, the case that s_1 is simply *skip*; then we are left with p as a requirement).

Similarly, requiring $(p \Leftrightarrow A)$ to hold in the precondition is too strong, since p is fresh. Instead, this formula should be *assumed* when requiring the other two conjuncts; the addition of p is not meant to make the precondition stronger. This leads us to the correct definition: *wlp*$(s_1 \; [] \; s_2, A) = (p \Leftrightarrow A) \Rightarrow$ *wlp*$(s_1, p) \land$ *wlp*(s_2, p)

This might seem surprising compared with the Tseitin CNF conversion (where the extra formula was conjoined), but is is actually consistent: recall that we applied the Tseitin conversion directly to the formula being checked for satisfiability; the Tseitin CNF conversion preserves satisfiability. In the case of weakest-preconditions, we are generally not interested in their satisfiability, but rather the satisfiability of their *negations*: recall that when we verify a program, we check an *entailment* $A' \vDash$ *wlp*(s, A), which, when encoded to an SMT problem, will mean that we ask the SMT solver to check satisfiability of $A' \land \neg$*wlp*(s, A). In particular, the weakest-precondition will be negated in our satisfiability query. The alternative weakest-precondition definition we are considering actually performs a Tseitin-like transformation to the *negation* of the formula: note that $\neg((p \Leftrightarrow A) \Rightarrow$ *wlp*$(s_1, p) \land$ *wlp*(s_2, p) is equivalent to $(p \Leftrightarrow A) \land \neg($*wlp*$(s_1, p) \land$ *wlp*$(s_2, p))$.

Assignment 2 (Multiple Verification Conditions)

1. The following annotated version of the program may help explain the working of the algorithm (intermediate results shown in braces). Recall that if-conditions are handled as non-deterministic choices followed by assume statements (not shown explicitly, here). The

five formulas in the top-most set represent the results of applying *wlp** to the program s:

$\{(x > 0 \Rightarrow x = 2), (x > 0 \Rightarrow (x = 2 \Rightarrow x = 2)), (x \leq 0 \Rightarrow x < 0), (x \leq 0 \Rightarrow (x < 0 \Rightarrow x \neq 0)),$
$(x \leq 0 \Rightarrow (x < 0 \Rightarrow (x \neq 0 \Rightarrow x = 2)))\}$
if $(x > 0)$ $\{$
$\{x = 2, (x = 2 \Rightarrow x = 2)\}$
  assert $x = 2$
$\{x = 2 \Rightarrow x = 2\}$
  assume $x = 2$
$\{x = 2\}$
$\}$ *else* $\{$
$\{x < 0, (x < 0 \Rightarrow x \neq 0), (x < 0 \Rightarrow (x \neq 0 \Rightarrow x = 2))\}$
  assert $x < 0$;
$\{(x < 0 \Rightarrow x \neq 0), (x < 0 \Rightarrow (x \neq 0 \Rightarrow x = 2))\}$
  assume $x < 0$;
$\{x \neq 0, (x \neq 0 \Rightarrow x = 2)\}$
  assert $x \neq 0$
$\{x \neq 0 \Rightarrow x = 2\}$
  assume $x \neq 0$
$\{x = 2\}$
$\}$
$\{x = 2\}$

2. Since the entailments will all have *true* (the precondition) on the left, the verification conditions amount to showing validity of each of the five formulas. Of these, the two formulas $(x > 0 \Rightarrow (x = 2 \Rightarrow x = 2)), (x \leq 0 \Rightarrow (x < 0 \Rightarrow x \neq 0))$ are valid, and the remaining three formulas $(x > 0 \Rightarrow x = 2), (x \leq 0 \Rightarrow x < 0), (x \leq 0 \Rightarrow (x < 0 \Rightarrow (x \neq 0 \Rightarrow x = 2)))$ are not valid. The first and last of the not valid formulas correspond to postcondition failures (one for each branch), while the second corresponds to a failure for the second assertion in the program.

3. Any verification conditions to be shown *after* a conditional branch will be duplicated once per branch. This will lead to a number of verification conditions exponential in the number of branches preceding the actual potential failure point (note that the same would occur for assertions placed after if-conditionals). From an error reporting perspective, we will also obtain multiple errors for the same source location, which might be confusing unless it is clear that these come from different branches through the program.

4. We could change the case for non-deterministic choice (i.e. the case *wlp**$(s_1[]s_2, \Delta)$) as follows: for each input assertion (in Δ) to the *wlp** operator, assign it an identifier which is propagated throughout the steps of the algorithm (i.e. we track which output formula corresponds to each input formula). Say a particular input formula A was originally in Δ, and suppose that A_1 and A_2 are the corresponding output formulas for each of the recursive calls. Then, instead of putting both A_1 and A_2 into the resulting set, we replace them with the single assertion $A_1 \wedge A_2$. In this way, the number of assertions in the set will remain linear in the number of potential error sources in the program.

Assignment 3 (Labelling)

After labelling, the program might look as follows:

$$
\begin{array}{l}
\textit{if}\,(x > 0)\ \{ \\
\quad \textit{assert}\ x = 2 \vee l_0 \\
\}\ \textit{else}\ \{ \\
\quad \textit{assert}\ x < 0 \vee l_1; \\
\quad \textit{assert}\ x \neq 0 \vee l_2 \\
\}
\end{array}
$$

A possible interaction with the SMT solver might go as follows. First, we pass $\textit{true} \wedge \neg(\textit{wlp}(s,\ x = 2 \vee l_3))$ to the SMT solver (the extra label l_3 is to handle postcondition failures). We obtain a `sat` result, and a model in which l_1 is false. This indicates a failure of the second assertion. We then pass $\textit{true} \wedge l_1 = \textit{true} \wedge \neg(\textit{wlp}(s,\ x = 2 \vee l_3))$ to the SMT solver, and again obtain `sat`, and a model in which l_3 is false. This indicates a failure of the postcondition. Finally, we pass $\textit{true} \wedge l_1 = \textit{true} \wedge l_3 = \textit{true} \wedge \neg(\textit{wlp}(s,\ x = 2 \vee l_3))$ and obtain an `unsat` result, indicating that there are no further verification failures to report.