Assignment 3
Dynamic Duo

15-414: Bug Catching: Automated Program Verification

Due 23:59pm, Friday, Feb 16, 2024
70 pts

This assignment is due on the above date and it must be submitted electronically on Gradescope. Please carefully read the policies on collaboration and credit on the course web pages at http://www.cs.cmu.edu/~15414/assignments.html.

What To Hand In

You should hand in the following files on Gradescope:

- Submit the file `asst3.zip` to Assignment 3 (Code). You can generate this file by running `make handin`. This will include your solution `arraysum.mlw` and the proof session in `arraysum/`.
- Submit a PDF containing your answers to the written questions to Assignment 3 (Written). You may use the file `asst3-sol.tex` as a template and submit `asst3-sol.pdf`. You can generate this file by running `make sol` (assuming you have pdflatex in your system).

  Make sure your session directories and your PDF solution files are up to date before you create the handin file.

Using LaTeX

We prefer the answer to your written questions to be typeset in LaTeX, but as long as you hand in a readable PDF with your solutions it is not a requirement. We package the assignment source `asst3.tex` and a solution template `asst3-sol.tex` in the handout to get you started on this.

1 Lather, Rinse, Repeat (10 pts)

In this problem we continue to study a *repeat-until* loop as an alternative to a *while* loop in our DL language. Informally, the repeat $\alpha\, P$ loop executes α and then tests P. If P is true it exits the loop, and if P is false it repeats it.

Task 1 (5 pts). The most straightforward (but relatively difficult to use) axiom for while loops in dynamic logic is $[\mathsf{while}\, P\, \alpha]Q \leftrightarrow (P \rightarrow [\alpha][\mathsf{while}\, P\, \alpha]Q) \wedge (\neg P \rightarrow Q)$. Give a corresponding axiom for the repeat loop.

Task 2 (5 pts). Express the repeat-until loop using the constructs of nondeterministic dynamic logic where the conditional and while loop have been replaced by nondeterministic choice and repetition.

2 Looking into the Past (25 pts)

In ordinary modal logic there is a $\blacksquare P$ modality that expresses "P *has always been true*". We can extend dynamic logic with a corresponding operator $\langle\!|\alpha|\!\rangle P$ read as "*before* α P". Its semantics is defined by

$$\omega \models \langle\!|\alpha|\!\rangle P \quad \text{iff for all } \mu \text{ such that } \mu[\![\alpha]\!]\omega \text{ we have } \mu \models P$$

For each of the following parts, develop axioms for nondeterministic dynamic logic that allow you to break down proving $\langle\!|\alpha|\!\rangle P$ into properties of smaller programs or eliminate them altogether. You only need to prove one direction of one of these properties (see Task 6) but it may be helpful to convince yourself your answers are correct.

Task 3 (5 pts). $\langle\!|\alpha\,;\,\beta|\!\rangle P$

Task 4 (5 pts). $\langle\!|\alpha \cup \beta|\!\rangle P$

Task 5 (5 pts). $\langle\!|?Q|\!\rangle P$

Task 6 (10 pts). Prove one direction of one of the axioms from Tasks 3–5. For this purpose assume $\omega \models \mathsf{ONESIDE}$ and prove that $\omega \models \mathsf{OTHERSIDE}$ for an arbitrary ω. Since ω is arbitrary this means that the implication is valid. The proof regarding sequential composition in Lecture 6, Section 5 provides a good model for the format and level of detail we expect.

3 The Day of Judgment (20 pts)

Task 7 (12 pts). For each of the following judgments in dynamic logic, find a program that, when substituted for α, makes the judgment hold. Throughout these judgments, ω is an arbitrary state.

1. $\omega[x \mapsto 0, y \mapsto 42] \models [(?(x \neq y); x \leftarrow x + 1; \alpha)^*; (?(x = y))]\bot$
2. $\omega[x \mapsto 0, y \mapsto 0] \models ([\alpha](x = 42)) \wedge ([\alpha; \alpha](x = 42)) \wedge ([\alpha; \alpha; \alpha](x \neq 42))$
3. $\omega[x \mapsto 0, y \mapsto 0] \models \neg[\alpha](x \neq y \vee \langle\alpha\rangle(x = y))$
4. $\omega[x \mapsto 0, y \mapsto 0] \models (\langle\alpha\rangle(x = 42)) \wedge (\neg[\alpha](x = 42))$

Task 8 (8 pts). For each of the following judgments in dynamic logic, find a state that, when substituted for ω, makes the judgment hold. Here, skip $\triangleq$?true.

1. $\omega \models [(?(x \neq y); x \leftarrow x + 1; y \leftarrow y - 1)^*; (?(x = y))](x \neq y)$
2. $\omega \models \neg[(x \leftarrow x + 1; y \leftarrow 2x)^*](x = y)$
3. $\omega \models [\text{if } (x = 0)\ (y \leftarrow y * 0)\ (\text{skip})](x = 0 \rightarrow y = 42)$

4 Don't Go Into Debt (15 pts)

This problem introduces the concept of an exception in WhyML, which may be helpful in some of the later programming assignments. We briefly summarize the constructs relevant to this problem (for more information see the Why3 documentation and some Why3 examples).

`exception` $exn\ \tau^*$	declare exn with arguments of type τ^*
`raise` $exn\ e^*$	raise exn with arguments e^*

And the function contract

`raises` $\{exn \rightarrow P\}$

verifies the postcondition P if exn is raised inside the function and propagates to the caller.

Consider the following simple example where we show the use of exceptions. We have a function `safe_get` that accesses the index `i` of an array and returns the content of the array at position `i` or raises an exception if `i` is out-of-bounds.

```
exception OutOfBounds

let safe_get (a: array int) (i: int) =
  ensures { result = a[i] }
  ensures { 0 <= i < length a}
  raises { OutOfBounds -> i < 0 \/ i >= length a}
  if i < 0 || i >= length a then raise OutOfBounds
  else return a[i]
```

Task 9 (15 pts). Write and verify a function `sum_array (a : array int) : int` that sums the elements of the array `a` from left to right. If the partial sum ever becomes negative, the function should short-circuit by raising `Negative i`, where `i` is the index of the array at which the sum first became negative. For example, calling `sum_array [2,-1,3,-5,8]` should raise `Negative 3`, since $2 + (-1) + 3 + (-5) < 0$.

You can find a solution template in the file `arraysum.mlw` that contains the code below.

```
module SumNonNeg

  use array.Array
  use array.ArraySum
  use int.Int

  exception Negative int

  let sum_array (a : array int) : int = 0

end
```

Hint: You may find the standard library module array.ArraySum helpful.