Assignment 3
Dynamic Duo

15-414: Bug Catching: Automated Program Verification

Due 23:59pm, Friday, Feb 14, 2025
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 solutions `partition.mlw` and the proof session in `partition/`.
- Submit a PDF containing your answers to the written questions to Assignment 3 (Written). You may use the file `asst3.tex` as a template and submit `asst3.pdf`.

  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 Looking into the Past (30 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 $(\!|\alpha|\!)P$ read as "*before α P*". Its semantics is defined by

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

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

Task 1 (5 pts). $(\!|\alpha\ ;\ \beta|\!)P$

Task 2 (5 pts). $(\!|\alpha \cup \beta|\!)P$

Task 3 (5 pts). $(\!|?Q|\!)P$

Task 4 (5 pts). $(\!|\alpha^*|\!)P$. In this task, both sides can refer to α^*.

Task 5 (10 pts). Prove one direction of othe axiom from Task 2. For this purpose, if your axiom reads $(\!|\alpha \cup \beta|\!)P \leftrightarrow Q$, then you should assume what $\omega \models Q$, and prove that $\omega \models (\!|\alpha \cup \beta|\!)P$. The proof regarding sequential composition in Lecture 6, Section 5 provides a good model for the format and level of detail we expect.

2 Day of Judgment (20 pts)

Task 6 (12 pts). For each of the following judgments in dynamic logic, find a program (substitute it for α) that makes the judgment hold. If no such program exists, explain why not. Throughout these judgments, ω is an arbitrary state.

1. $\omega[x \mapsto 1, z \mapsto 5] \ \models\ \neg\langle\alpha\rangle(z = 1) \ \wedge\ [\alpha](z = 1)$
2. $\omega[x \mapsto 1, y \mapsto 0] \ \models\ x > y \ \rightarrow\ ([\alpha](x < 0) \ \wedge\ [\alpha;\alpha](x < 0) \ \wedge\ \neg[\alpha^*](x < 0))$
3. $\omega[u \mapsto 5, v \mapsto 5] \ \models\ \neg[\alpha](u \neq v \ \vee\ \langle\alpha\rangle(u = v))$
4. $\omega[x \mapsto 0] \ \models\ (\langle\alpha\rangle(x = 7)) \ \wedge\ (\neg[\alpha](x = 7))$

Task 7 (8 pts). For each of the following judgments in dynamic logic, find a state ω (an assignment to variables) that makes the judgment hold. Recall that $\mathsf{skip} \ \triangleq\ ?\,\mathsf{true}$.

1. $\omega \ \models\ [(?(x > 0);\ x \leftarrow x + 2)^*\ ;\ ?(x \leq 0)]\,(x \neq 0)$
2. $\omega \ \models\ [(?(x > y);\ x \leftarrow x - 1;\ y \leftarrow y + 1)^*\ ;\ ?(x \leq y)]\,(x > 0)$.
3. $\omega \ \models\ [\mathsf{if}\ (x > 10)\ (y \leftarrow y + x)\ (\mathsf{skip})]\,(y > 2 \ \rightarrow\ x > 10)$.

3 Don't Go Into Debt (20 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

$$\texttt{raises}\,\{exn \rightarrow P\}$$

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 8 (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.