1 5F-1 Bookkeeping

- 0 pts Correct

Peer Review ID: 72757436 — enter this when you fill out your peer evaluation via gradescope

5F-2. The backwards verification condition formula for $\mathsf{do}_{Inv}\ c\ \mathsf{while}\ b$ is given by

$$\mathrm{VC}(\mathsf{do}_{Inv}\ c\ \mathsf{while}\ b, P) = \mathrm{VC}(c, Inv) \wedge (\forall x_1 \ldots x_n\ .\ Inv \Rightarrow (b \Rightarrow \mathrm{VC}(c, Inv) \wedge \neg b \Rightarrow P)),$$

where $x_1 \ldots x_n$ are all those variables modified in c.

We derived this rule by noting that $\mathsf{do}_{Inv}\ c\ \mathsf{while}\ b$ is semantically equivalent to $c; \mathsf{while}_{Inv}\ b\ \mathsf{do}\ c$, thus

$$\begin{aligned}\mathrm{VC}(\mathsf{do}_{Inv}\ c\ \mathsf{while}\ b, P) &= \mathrm{VC}(c; \mathsf{while}_{Inv}\ b\ \mathsf{do}\ c, P) = \mathrm{VC}(c, \mathrm{VC}(\mathsf{while}_{Inv}\ b\ \mathsf{do}\ c, P)) \\ &= \mathrm{VC}(c, Inv \wedge \forall x_1 \ldots x_n\ .\ Inv \Rightarrow (b \Rightarrow \mathrm{VC}(c, Inv) \wedge \neg b \Rightarrow P)) \\ &= \mathrm{VC}(c, Inv) \wedge \mathrm{VC}(c, \forall x_1 \ldots x_n\ .\ Inv \Rightarrow (b \Rightarrow \mathrm{VC}(c, Inv) \wedge \neg b \Rightarrow P)) \\ &= \mathrm{VC}(c, Inv) \wedge (\forall x_1 \ldots x_n\ .\ Inv \Rightarrow (b \Rightarrow \mathrm{VC}(c, Inv) \wedge \neg b \Rightarrow P)),\end{aligned}$$

where the last equality is justified since executing c should not change the truth value of a statement which is univerally quanitifed over all values of all variables modified by c. □

5F-3. For fun, let us first prove that lannister is relatively complete. Since the original while rule was relatively complete, it suffices to show that the original rule is derivable from lannister. Indeed, the following derivation tree demonstrates this, using only consequence, lannister, and the fact that $X \Rightarrow (b \Rightarrow (X \wedge b) \wedge \neg b \Rightarrow (X \wedge \neg b))$ is a tautology:

$$\dfrac{\vdash X \Rightarrow (b \Rightarrow (X \wedge b) \wedge \neg b \Rightarrow (X \wedge \neg b)) \qquad \dfrac{\dfrac{\vdash \{X \wedge b\}\ c\ \{X\} \quad \vdash X \Rightarrow (b \Rightarrow (X \wedge b) \wedge \neg b \Rightarrow (X \wedge \neg b))}{\vdash \{X \wedge b\}\ c\ \{b \Rightarrow (X \wedge b) \wedge \neg b \Rightarrow (X \wedge \neg b)\}}}{\vdash \{b \Rightarrow (X \wedge b) \wedge \neg b \Rightarrow (X \wedge \neg b)\}\ \mathsf{while}\ b\ \mathsf{do}\ c\ \{X \wedge \neg b\}}}{\vdash \{X\}\ \mathsf{while}\ b\ \mathsf{do}\ c\ \{X \ \wedge\ \neg b\}}.$$

By process of elimination, the two incomplete rules are then stark and targaryen. We give concrete examples of their incompleteness below:

- 1. (name of the rule) stark
  2. (A) true
  3. (B) $x = 1$
  4. (σ) $[x := 0]$
  5. (σ') $[x := 1]$
  6. (c) while $x \neq 1$ do $x := 1$
  7. $(\langle c, \sigma\rangle \Downarrow \sigma')$ Since $\sigma(x) = 0$, the condition $x \neq 1$ holds, so the body of the while loop is entered, then 1 is assigned to x, the condition no longer holds, and the loop exits in state $\sigma' = [x := 1]$.
  8. $(\sigma \models A)$ Trivially, we have $\sigma \models$ true.
  9. $(\sigma' \models B)$ Since $\sigma'(x) = 1$, indeed $\sigma' \models x = 1$.
  10. (it is impossible to prove $\vdash \{A\}c\{B\}$) Heuristically, stark can nly prove post-conditions that are also true before the loop is run, so since B is false beforehand, it is impossible to prove $\{A\}c\{B\}$.

Peer Review ID: 72757436 — enter this when you fill out your peer evaluation via gradescope

2 5F-2 VCGen Do-While

- 0 pts Correct

Peer Review ID: 72757436 — enter this when you fill out your peer evaluation via gradescope

5F-2. The backwards verification condition formula for $\mathsf{do}_{Inv}\ c\ \mathsf{while}\ b$ is given by

$$\mathrm{VC}(\mathsf{do}_{Inv}\ c\ \mathsf{while}\ b, P) = \mathrm{VC}(c, Inv) \wedge (\forall x_1 \ldots x_n\ .\ Inv \Rightarrow (b \Rightarrow \mathrm{VC}(c, Inv) \wedge \neg b \Rightarrow P))\,,$$

where $x_1 \ldots x_n$ are all those variables modified in c.

We derived this rule by noting that $\mathsf{do}_{Inv}\ c\ \mathsf{while}\ b$ is semantically equivalent to $c; \mathsf{while}_{Inv}\ b\ \mathsf{do}\ c$, thus

$$\begin{aligned}\mathrm{VC}(\mathsf{do}_{Inv}\ c\ \mathsf{while}\ b, P) &= \mathrm{VC}(c; \mathsf{while}_{Inv}\ b\ \mathsf{do}\ c, P) = \mathrm{VC}(c, \mathrm{VC}(\mathsf{while}_{Inv}\ b\ \mathsf{do}\ c, P))\\ &= \mathrm{VC}(c, Inv \wedge \forall x_1 \ldots x_n\ .\ Inv \Rightarrow (b \Rightarrow \mathrm{VC}(c, Inv) \wedge \neg b \Rightarrow P))\\ &= \mathrm{VC}(c, Inv) \wedge \mathrm{VC}(c, \forall x_1 \ldots x_n\ .\ Inv \Rightarrow (b \Rightarrow \mathrm{VC}(c, Inv) \wedge \neg b \Rightarrow P))\\ &= \mathrm{VC}(c, Inv) \wedge (\forall x_1 \ldots x_n\ .\ Inv \Rightarrow (b \Rightarrow \mathrm{VC}(c, Inv) \wedge \neg b \Rightarrow P))\,,\end{aligned}$$

where the last equality is justified since executing c should not change the truth value of a statement which is univerally quanitifed over all values of all variables modified by c. □

5F-3. For fun, let us first prove that lannister is relatively complete. Since the original while rule was relatively complete, it suffices to show that the original rule is derivable from lannister. Indeed, the following derivation tree demonstrates this, using only consequence, lannister, and the fact that $X \Rightarrow (b \Rightarrow (X \wedge b) \wedge \neg b \Rightarrow (X \wedge \neg b))$ is a tautology:

$$\dfrac{\vdash X \Rightarrow (b \Rightarrow (X \wedge b) \wedge \neg b \Rightarrow (X \wedge \neg b)) \qquad \dfrac{\dfrac{\vdash \{X \wedge b\}\ c\ \{X\} \quad \vdash X \Rightarrow (b \Rightarrow (X \wedge b) \wedge \neg b \Rightarrow (X \wedge \neg b))}{\vdash \{X \wedge b\}\ c\ \{b \Rightarrow (X \wedge b) \wedge \neg b \Rightarrow (X \wedge \neg b)\}}}{\vdash \{b \Rightarrow (X \wedge b) \wedge \neg b \Rightarrow (X \wedge \neg b)\}\ \mathsf{while}\ b\ \mathsf{do}\ c\ \{X \wedge \neg b\}}}{\vdash \{X\}\ \mathsf{while}\ b\ \mathsf{do}\ c\ \{X\ \wedge\ \neg b\}}\,.$$

By process of elimination, the two incomplete rules are then stark and targaryen. We give concrete examples of their incompleteness below:

- 1. (name of the rule) stark
  2. (A) true
  3. (B) $x = 1$
  4. (σ) $[x := 0]$
  5. (σ') $[x := 1]$
  6. (c) $\mathsf{while}\ x \neq 1\ \mathsf{do}\ x := 1$
  7. $(\langle c, \sigma\rangle \Downarrow \sigma')$ Since $\sigma(x) = 0$, the condition $x \neq 1$ holds, so the body of the while loop is entered, then 1 is assigned to x, the condition no longer holds, and the loop exits in state $\sigma' = [x := 1]$.
  8. $(\sigma \models A)$ Trivially, we have $\sigma \models$ true.
  9. $(\sigma' \models B)$ Since $\sigma'(x) = 1$, indeed $\sigma' \models x = 1$.
  10. (it is impossible to prove $\vdash \{A\}c\{B\}$) Heuristically, stark can nly prove post-conditions that are also true before the loop is run, so since B is false beforehand, it is impossible to prove $\{A\}c\{B\}$.

Peer Review ID: 72757436 — enter this when you fill out your peer evaluation via gradescope

- 1. (name of the rule) targaryen
  2. (A) $x \geq 0$
  3. (B) $x = 0$
  4. (σ) $[x := 1]$
  5. (σ') $[x := 0]$
  6. (c) while $x > 0$ do $x := x - 1$
  7. $(\langle c, \sigma \rangle \Downarrow \sigma')$ Since initially $\sigma(x) = 1$, the condition $x > 0$ holds, so the body of the while loop is entered, then x is decremented to 0, the condition no longer holds, and the loop exits in state $\sigma' = [x := 0]$
  8. $(\sigma \models A)$ Since $\sigma(x) = 1$ and $1 \geq 0$, indeed $\sigma \models x \geq 0$.
  9. $(\sigma' \models B)$ Since $\sigma'(x) = 0$, indeed $\sigma' \models x = 0$.
  10. (it is impossible to prove $\vdash \{A\}c\{B\}$) Heuristically, targaryen requires that X holds whether the loop is run or not, so any X which is only preserved by c when b is true is not provable. Specifically, for $x \geq 0$ to hold after $x := x - 1$ is executed, we must have that $x > 0$ is true, so this is not provable. $\square$

Peer Review ID: 72757436 — enter this when you fill out your peer evaluation via gradescope

3 5F-3 VCGen Mistakes

- **- 0 pts** Correct