Reasoning About Programs

Week 7 PMT - Loop Invariants and Variants
To discuss during PMT - do NOT hand in

Sophia Drossopoulou and Mark Wheelhouse*

Learning Aims:

A1: become familiar with loop invariants and variants at an informal level.

A2: use the postcondition to "guide" the specification of the loop invariant, and use the loop invariant to "guide" the development of the code.

A3: refine loop invariants and use the refined versions to develop more efficient code.

Comments: Note that in questions (1) and (3), the mid-condition describing the effect of the loop is precisely the conjunction of the loop invariant and the negation of the loop condition, whilst in questions (2) and (4) it is a *consequence* of the conjunction of the loop invariant and the negation of the loop condition.

1st Question:

a) Write some code which, given an integer `days` > 0 and a starting year of 1900, calculates the current year if the number of days elapsed is `days`. An input of `days` $= 1$ corresponds to 1/1/1900. Do not take into account leap years, i.e. assume that all years have 365 days[1]. You should use only addition, subtraction, and a loop. You should not use any library methods, `mod`, or integer division.

b) Write a midcondition M that describes what your loop achieves upon termination.

c) Write an invariant I that is strong enough to imply your mid-condition from (b), assuming the termination of your loop.

d) Write a variant V for your loop that is strong enough to prove total correctness.

A possible answer:

*Thank you to James Noble (Univ. Wellington) for suggesting the Zune 30 problem.

[1] For example, if `days` = 365 then the current year is 1900, if `days` = 366 then the current year is 1901, if `days` = 1095 then the current year is 1902, and if `days` = 1096 then the current year is 1903.

a)

```
// PRE: 1 ≤ days
int year = 1900;
// INV: I
// VAR: V
while (days > 365) {
    days = days -365;
    year = year + 1;
}
// MID: M
```

b) $M \longleftrightarrow \texttt{days}_0 = (\texttt{year} - 1900) * 365 + \texttt{days} \quad \wedge \quad 1 \leq \texttt{days} \leq 365$

c) $I \longleftrightarrow \texttt{days}_0 = (\texttt{year} - 1900) * 365 + \texttt{days} \quad \wedge \quad 1 \leq \texttt{days}$

d) $V = \texttt{days}$

2nd Question:

Consider the following code that comes from the Zune 30 program (as seen in the first Reasoning about Programs lecture):

```
// PRE: 1 ≤ days
year = 1900;
// INV: I
// VAR: V
while (days > 365) {
    if(isLeapYear(year)) {
        if(days > 366) {
            days -= 366;
            year++;
        }
    }
    else {
        days -= 365;
        year++;
    }
}
// MID: M
```

a) Write a mid-condition that describes what the loop is supposed to achieve, using a function $\texttt{nrDays} : \mathbb{N} \to \mathbb{N}$, which returns 366 if its argument is a leap year, and 365 otherwise.

b) Write an invariant for the loop that is strong enough to imply the mid-condition, assuming termination of the loop.

c) Is the loop partially correct?

d) Is the loop totally correct?

A possible answer:

a) $M \longleftrightarrow \quad \texttt{days}_0 = \sum_{\texttt{i}=1900}^{\texttt{year}-1} \texttt{nrDays}(\texttt{i}) + \texttt{days} \quad \land \quad 1 \leq \texttt{days} \leq \texttt{nrDays}(\texttt{year})$

b) $I \longleftrightarrow \quad \texttt{days}_0 = \sum_{\texttt{i}=1900}^{\texttt{year}-1} \texttt{nrDays}(\texttt{i}) + \texttt{days} \quad \land \quad 1 \leq \texttt{days}$

c) The loop is partially correct. If it terminates it satisfies the mid-condition M. Namely, loop execution when the loop condition holds preserves the invariant. Also, the conjunction of the invariant with the negation of the loop condition give:

$$M' \longleftrightarrow \texttt{days}_0 = \sum_{\texttt{i}=1900}^{\texttt{year}-1} \texttt{nrDays}(\texttt{i}) + \texttt{days} \quad \land \quad 1 \leq \texttt{days} \leq 365$$

which implies M[2].

[2]Note however, that M' is *strictly stronger* than M. This is an indication that the loop might not terminate in all cases where it is expected to.

d) The loop is not totally correct. There does not exist a term whose value decreases upon *every* loop iteration. In particular, `days` does not decrease in the case where $\texttt{days} = \texttt{366}$ and `isLeapYear(year)`.

3rd Question:

Consider the following mid-condition:

$$M \longleftrightarrow \texttt{days}_0 = \sum_{\texttt{i}=1900}^{\texttt{year}-1} \texttt{nrDays}(\texttt{i}) + \texttt{days} \quad \wedge \quad 1 \leq \texttt{days} \leq \texttt{nrDays}(\texttt{year})$$

a) Write some code which, for some integer $\texttt{days} \geq 1$, satisfies the above mid-condition upon termination. You should use the function `nrDays`, but should not use any library methods.

b) Write an invariant I for your loop that is strong enough to imply the mid-condition, assuming termination of the loop.

c) Is your loop partially correct?

d) Write a variant V for your loop.

e) Is your loop totally correct?

A possible answer:

a)
```
// PRE: 1 ≤ days
int year = 1900;
// INV: I
// VAR: V
while (days > nrDays(year)) {
    days = days - nrDays(year);
    year = year + 1;
}
// MID: days_0 = Σ_{i=1900}^{year−1} nrDays(i) + days   ∧   1 ≤ days ≤ nrDays(year)
```

b) $I \longleftrightarrow \texttt{days}_0 = \sum_{\texttt{i}=1900}^{\texttt{year}-1} \texttt{nrDays}(\texttt{i}) + \texttt{days} \quad \wedge \quad 1 \leq \texttt{days}$

c) The loop is partially correct. This is because loop execution preserves the invariant and the conjunction of the invariant with the negation of the loop condition are the same as M.

d) $V = \texttt{days}$

e) `days` is positive and decreases upon each loop iteration. Therefore the loop terminates. From (c) we know that loop is partially correct, therefore the loop is totally correct.

4th Question:

a) Write a more efficient version of the code from question 3, which does not use the function `nrDays`, but uses `isLeapYear`, and which does not call the function `isLeapYear` more than once per loop iteration.

b) Write the invariant I of your loop, using the function `nrDays`.

c) Is your loop partially correct?

d) Write a variant V for your loop.

e) Is your loop totally correct?

A possible answer:

a)

```
// PRE: 1 ≤ days
int year = 1900;
int nrDaysCurrentyear = 365;
if (isLeapYear(1900)) { nrDaysCurrentYear = 366; }
// INV: I
// VAR: V
while (days > nrDaysCurrentYear) {
    days -= nrDaysCurrentYear;
    year++;
    if (isLeapYear(year)) {
        nrDaysCurrentYear = 366;
    }
    else {
        nrDaysCurrentYear = 365;
    }
}
// MID: days_0 = Σ_{i=1900}^{year−1} nrDays(i) + days ∧ 1 ≤ days ≤ nrDays(year)
```

b) $I \;\longleftrightarrow\; \texttt{nrDaysCurrentYear} = \texttt{nrDays}(\texttt{year}) \wedge \texttt{days}_0 = \sum_{\texttt{i}=1900}^{\texttt{year}-1} \texttt{nrDays}(\texttt{i}) + \texttt{days} \wedge 1 \leq \texttt{days}$

c) The loop is partially correct. This is because loop execution preserves the invariant, and the conjunction of the invariant with the negation of the loop condition *imply* M.

d) $V \;=\; \texttt{days}$

e) `days` is positive and decreases upon each loop iteration. Therefore the loop terminates. From (c) we know that the loop is partially correct, therefore the loop is totally correct.

5th Question:

Recall the tail recursive function $\mathsf{DM} : \mathbb{Z} \times \mathbb{Z} \times \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z} \times \mathbb{Z}$ discussed in week_5:

$$\begin{array}{ll} \textbf{R32} & acc + n > m \;\rightarrow\; \mathsf{DM}(m, n, cnt, acc) = (cnt, m - acc) \\ \textbf{R33} & acc + n \leq m \;\rightarrow\; \mathsf{DM}(m, n, cnt, acc) = \mathsf{DM}(m, n, cnt + 1, acc + n) \\ \textbf{R34} & \mathsf{DivMod}(m, n) = \mathsf{DM}(m, n, 0, 0) \end{array}$$

We will now consider the counterpart of DM though a while-loop in Java:

```
class Pair {
    int first;
    int second;
}

Pair mathDivMod(int m, int n)
// PRE: m ≥ 0  ∧  n > 0
// POST: m = r.first ∗ n + r.second  ∧  r.second < n
{
    int cnt = 0;
    int acc = 0;
    // INV: I
    // VAR: V
    while ( acc + n <= m ) {
        cnt = cnt + 1;
        acc = acc + n;
    }
    // MID: M1
    Pair res = new Pair;
    res.first = cnt;
    res.second = m - acc;
    // MID M2
    return res;
}
```

a) Write a midcondition M_2 which holds just before the method's return statement and is strong enough to prove partial correctness of the `mathDivMod` method. (You do not need to prove anything.)

b) Write a midcondition M_1 which holds after the loop and is strong enough to prove partial correctness of the `mathDivMod` method. (You do not need to prove anything.)

c) Write a loop invariant I which is strong enough to prove partial correctness of the `mathDivMod` method. (You do not need to prove anything.)

d) Write a loop variant V which is strong enough to prove total correctness of the `mathDivMod` method. (You do not need to prove anything.)

Hints: In order to find appropriate M_2 we need to remember that it should satisfy:

$$M_2 \;\longrightarrow\; POST[\mathtt{r} \mapsto \mathtt{res}]$$

Then, for M_1, we need to be able to prove that:

$$M_1 \;\wedge\; \texttt{res.first} = \texttt{cnt} \;\wedge\; \texttt{res.second} = \texttt{m} - \texttt{acc} \;\longrightarrow\; M_2$$

Therefore, M_1 needs to describe properties of `cnt` and `acc`. At this point, you may also like to draw inspiration from the last coursework exercise, where we showed that the function DivMod represents integer division and modulus i.e.

Assrt_1: $\forall m, n, k1, k2 \in \mathbb{N}.[\ (k1, k2) = \mathsf{DivMod}(m, n) \longrightarrow m = k1 * n + k2 \wedge k2 < n\ ]$

In order to show this, we showed the following, stronger assertion about DM:

Assrt_2: $\forall m, n, acc, k1, k2 \in \mathbb{N}.$
$$[\ (k1, k2) = \mathsf{DM}(m, n, cnt, acc) \longrightarrow [\, cnt * n = acc \leq m \longrightarrow m = k1 * n + k2 \wedge k2 < n\ ]\ ]$$

A possible answer:

a) $M_2 \quad\longleftrightarrow\quad \texttt{m} = \texttt{res.first} * \texttt{n} + \texttt{res.second} \;\wedge\; \texttt{res.second} < \texttt{n}$

b) $M_1 \quad\longleftrightarrow\quad \texttt{m} = \texttt{cnt} * \texttt{n} + (\texttt{m} - \texttt{acc}) \;\wedge\; \texttt{m} - \texttt{acc} < \texttt{n}$

c) $I \quad\longleftrightarrow\quad \texttt{acc} = \texttt{cnt} * \texttt{n} \;\wedge\; \texttt{acc} \leq \texttt{m}$

d) $V \quad = \quad \texttt{m} - \texttt{acc}$.
This variant decreases at each loop iteration, and has a lower bound of 0.

For the very interested: You can find this program written for the interactive program verification tool Dafny at: `http://rise4fun.com/Dafny/WWSVw`.