Reasoning About Programs

Week 7 Coursework - Loop Invariants and Variants
Answers to be submitted to the SAO by 2pm on Monday, 27th February

Mark Wheelhouse and Sophia Drossopoulou

Learning Aims:

A1: discovering loop invariants

A2: reasoning using loop invariants and lemmas

A3: finding interesting variants

Summary

We will prove the correctness of the Java methods `isPalindrome` and `find`.

- The `isPalindrome` method, given a string `s`, determines if it is a palindrome - that is a word which is the same when reversed.
- The `find` method, given strings `s` and `t`, returns the *first* location in `s`, where `t` occurs as a sub-string.

Preliminaries

In order to write the formal specifications of our methods we must first define the predicates *Palindrome*, *PartialPalindrome* and *Match* as well as the function *Find*. These are given as follows:

$$Palindrome(\texttt{s}) \longleftrightarrow \forall k \in [0..\texttt{s.length}/2).[\ \texttt{s}[k] = \texttt{s}[\texttt{s.length} - k - 1]\ ]$$

$$PartialPalindrome(\texttt{s},\texttt{i}) \longleftrightarrow \forall k \in [0..\texttt{i}).[\ \texttt{s}[k] == \texttt{s}[\texttt{s.length} - k - 1\ ]$$

$$Match(\texttt{s},\texttt{t},m,n) \longleftrightarrow 0 \le m+n \le \texttt{s.length} \ \wedge\ 0 \le n \le \texttt{t.length} \ \wedge\ (\forall k \in [0..n).[\ \texttt{s}[m+k] = \texttt{t}[k]\ ])$$

$$Find(\texttt{s},\texttt{t}) = \begin{cases} \texttt{s.length} & \text{if } \nexists m.\ Match(\texttt{s},\texttt{t},m,\texttt{t.length}) \\ min\{\, m \mid Match(\texttt{s},\texttt{t},m,\texttt{t.length})\,\} & \text{otherwise} \end{cases}$$

Recall that we use the notation $k \in [m..n)$ to indicate that $m \le k < n$.
The following examples show the values of *Match* and *Find* for some cases:

s	t	m	n	$Match(\texttt{s},\texttt{t},m,n)$
“abab”	“abc”	0	2	TRUE
“abab”	“abc”	0	3	FALSE
“abab”	“abc”	2	2	TRUE
“abab”	“fhgh”	1	0	TRUE
“abab”	“ab”	0	2	TRUE
“abab”	“ab”	2	2	TRUE

s	t	$Find(\texttt{s},\texttt{t})$
“abab”	“ab”	0
“abCAbfCAd”	“CA”	2
“abab”	“abc”	4
“abab”	“”	0
“”	“abab”	0
“”	“”	0

You can assume that the following properties hold (you do not need to prove them), for all strings s and t, and integers m, n and n':

Lemma 0: $0 \leq n' \leq n \;\wedge\; Match(\texttt{s},\texttt{t},m,n) \;\rightarrow\; Match(\texttt{s},\texttt{t},m,n')$
Lemma 1: $0 \leq m \leq \texttt{s.length} \;\rightarrow\; Match(\texttt{s},\texttt{t},m,0)$
Lemma 2: $\texttt{t.length} = 0 \;\rightarrow\; Find(\texttt{s},\texttt{t}) = 0$
Lemma 3: $Find(\texttt{s},\texttt{t}) \geq 0$
Lemma 4: $Find(\texttt{s},\texttt{t}) = \texttt{s.length} \;\vee\; Find(\texttt{s},\texttt{t}) \leq \texttt{s.length}-\texttt{t.length}$

1st Question:

Consider the following code and specification for isPalindrome:

```
boolean isPalindrome(char[] s)
// PRE: s ≠ null  ∧  s.length > 0
// POST: s ≈ s₀  ∧  r = Palindrome(s)
{
    int i = 0;
    boolean stop = false;
    // INV: I
    // VAR: V
    while (!stop && i < s.length / 2) {
        stop = ( s[i] != s[s.length - i - 1] );
        i++;
    }
    // MID: M
    return !stop;
}
```

a) Write a midcondition M, which is strong enough to prove the partial correctness of isPalindrome.
[**Hint:** *use the postcondition and finalisation code to guide you*]

b) Write an invariant I for the loop, which is strong enough to prove the partial correctness of isPalindrome.
[**Hint:** *try to break the invariant into two parts:*

i) *describe the bounds on* i

ii) *relate* i *to the progress being made (PartialPalindrome might be helpful)*]

c) Write a variant V for the loop and justify that it decreases and is bound.

A possible answer:

a) $M \;\longleftrightarrow\; \texttt{s} \approx \texttt{s}_0 \;\wedge\; \neg\texttt{stop} \longleftrightarrow PartialPalindrome(\texttt{s},\texttt{s.length}/2)$

b) $I \;\longleftrightarrow\; \texttt{s} \approx \texttt{s}_0 \;\wedge\; 0 \leq \texttt{i} \leq \texttt{s.length}/2 \;\wedge\; \neg\texttt{stop} \longleftrightarrow PartialPalindrome(\texttt{s},\texttt{i})$

c) $V \;=\; \texttt{s.length}/2 - \texttt{i}$
This expression is bounded below by 0. We can see that in each loop iteration i increases. Therefore the expression $\texttt{s.legnth}/2 - \texttt{i}$ must decrease.

NB: the expression `s.length` − `i` is also a valid variant. It is bounded below by `s.length`/2 (0 can also be used as a lower bound even though it will never be reached in most cases) and the expression similarly decreases.

Question 1 - Marking Scheme (10 marks)

(a) Midcondition: 2 points, awarded as follows:

1 point	for a property involving `stop`
1 point	for a property involving *Palindrome* or *PartialPalindrome*

Only award both points if the midcondition is actually strong enough to prove partial correctness and includes the property that the array has not been modified.

(The points above to be awarded also for equivalent formulations of the midcondition.)

(b) Invariant: 5 points, awarded as follows:

1 point	for mentioning that `s` is not modified by the loop
1 point	for correct lower bound on `i`
1 point	for correct upper bound on `i`
1 point	for a property involving `stop`
1 point	for a property involving *PartialPalindrome*

Only award all points if the invariant is actually strong enough to prove partial correctness.

(The points above to be awarded also for equivalent formulations of the invariant.)

(c) Variant, and Loop Termination: 3 points, awarded as follows:

1 point	for a correct expression
1 point	for arguing expression has a lower bound
1 point	for arguing expression decreases on *every* loop iteration

2nd Question:

Consider the following code and specification for `find`:

```
int find( char[] s, char[] t )
// PRE: s ≠ null  ∧  t ≠ null
// POST: r = Find(s₀, t₀)
{
    int i = 0;
    int j = 0;
    boolean found = (t.length == 0);

    // INV: I
    // VAR: V
    while ( j < t.length && i < s.length && !found ) {
        if ( s[i] == t[j] ) {
            i++;
            j++;
            if ( j == t.length ) { found = true; }
        } else {
            i = i - j + 1;
            j = 0;
        }
    }
    // MID: M

    if ( found ) {
        return i - j;
    } else {
        return s.length;
    }
}
```

a) Write a midcondition M, which is strong enough to prove the partial correctness of `find`.
[**Hint:** *use the postcondition and finalisation code to guide you*]

b) Write an invariant I for the loop, which is strong enough to prove the partial correctness of `find`.
[**Hint:** *try to break the invariant into five parts, each consisting of one or more conjuncts:*

i) *describe the state of the strings* `s` *and* `t`

ii) *describe the bounds on and relationship of* `i` *and* `j`

iii) *describe what has been matched so far*

iv) *relate* `i` *and* `j` *to the progress being made*

v) *relate* `found` *to the success of the sub-string search*]

c) Write a variant V for the loop and justify that it decreases and is bound.

You can either:

(*i*) find a variant which is an integer expression, find a lower bound to this variant, and show that the variant decreases with each loop iteration, but never falls below that bound,

or:

(*ii*) find a variant which is a tuple of integers and use a lexicographic ordering to show that in each loop iteration the variant decreases in terms of this lexicographic ordering.

The next assignment

Note that there will be another assignment building on the current one. You will be asked to prove that the initialisation code (lines 4-6) establishes the invariant `INV`, that the loop body re-establishes the invariant `INV`, and that the invariant `INV` and termination of the loop imply the mid-condition `MID`. You will be allowed to use the invariant you designed for this week, or to use an invariant that will be given to you.

A possible answer:

(a) **Write a midcondition `MID`, which will be appropriate to prove the partial correctness of `find`.**

We enumerate the conjuncts of the midcondition as follows:

$$\begin{array}{ll}
\text{(M1)} & \mathtt{s} \approx \mathtt{s}_0 \quad \land \quad \mathtt{t} \approx \mathtt{t}_0 \quad \land \\
\text{(M2)} & \mathtt{found} \;\rightarrow\; \mathit{Find}(\mathtt{s},\mathtt{t}) = \mathtt{i} - \mathtt{j} \quad \land \\
\text{(M3)} & \mathtt{!found} \;\rightarrow\; \mathit{Find}(\mathtt{s},\mathtt{t}) = \mathtt{s.length}
\end{array}$$

(b) **Write an invariant `INV` for the loop, which will be appropriate to prove the partial correctness of `find`.**

We enumerate the conjuncts of the invariant as follows:

$$\begin{array}{ll}
\text{(I1)} & \mathtt{s} \approx \mathtt{s}_0 \;\land\; \mathtt{t} \approx \mathtt{t}_0 \;\land \\
\text{(I2)} & 0 \leq \mathtt{i} \leq \mathtt{s.length} \;\land\; 0 \leq \mathtt{j} \leq \mathtt{t.length} \;\land\; \mathtt{i} \geq \mathtt{j} \;\land \\
\text{(I3)} & \mathit{Match}(\mathtt{s},\mathtt{t},\mathtt{i\text{-}j},\mathtt{j}) \;\land \\
\text{(I4)} & \mathtt{i} - \mathtt{j} - 1 < \mathit{Find}(\mathtt{s},\mathtt{t}) \;\land \\
\text{(I5)} & \mathtt{found} \;\longleftrightarrow\; \mathtt{j} = \mathtt{t.length}
\end{array}$$

(c) **Write a variant `VAR` for the loop and justify that it decreases and is bound.**

1st Approach: the variant as an integer expression We choose the variant to be

$$\mathtt{t.length} * (\mathtt{s.length} - \mathtt{i} + \mathtt{j}) + \mathtt{t.length} - \mathtt{j}$$

We choose 0 as the lower bound for the variant.

We have to show that the variant is never smaller than 0, and that it decreases with every iteration.

We can prove the former, by using (I2).

In more detail[1]:

t.length*(s.length-i+j)+t.length-j

$\geq$	t.length*(s.length-i)+t.length-j	because $0 \leq$ j, by (I2)
$\geq$	t.length*0+0	by (I2), i $\leq$ s.length and j $\leq$ t.length
$\geq$	0	by arithmetic

We now need to show that the variant decreases. In other words, we need to prove that:

(∗) t.length*(s.length-i'+j')+t.length-j' < t.length*(s.length-i+j)+t.length-j

where i and j are values before execution of the loop body, and i' and j' are values after execution of the loop body.

We distinguish the two cases of the code:

1st Case: s[i] = t[j]

Then, by code, we have i' = i+1, and j' = j+1. Therefore,

t.length*(s.length-i'+j')+t.length-j'

=	t.length*(s.length-i-1+j+1)+t.length-j-1	by substit. values for i', j'
=	t.length*(s.length-i+j)+t.length-j-1	by arithmetic
<	t.length*(s.length-i+j)+t.length-j	by arithmetic

This establishes (*).

2nd Case: s[i] $\neq$ t[j]

Then, by code, we have i' = i-j+1, and j' = 0. Therefore,

t.length*(s.length-i'+j')+t.length-j'

=	t.length*(s.length-i+j-1)+t.length-0	by substit. values for i', j'
=	t.length*(s.length-i+j)-t.length+t.length-0	by arithmetic
=	t.length*(s.length-i+j)	by arithmetic
<	t.length*(s.length-i+j)+t.length-j	because, from (I2) and (I5) we have t.length-j > 0

This establishes (∗).

2nd Approach: the variant as a tuple of integers We choose the pair:

$$(\texttt{s.length} - \texttt{i} + \texttt{j}, \texttt{t.length} - \texttt{j})$$

We denote the lexicographic ordering as $\prec$.

From (I2) we obtain that $(0, 0) \prec$ (s.length-i+j, t.length-j) always holds. Therefore, there is a lower bound to the possible values of the variant.

Moreover, we need to prove that after a loop iteration the variant decreases, i.e. that:

(∗) (s.length-i'+j', t.length-j') $\prec$ (s.length-i+j, t.length-j)

where i and j are values before execution of the loop body, and i' and j' are values after execution of the loop body.

We prove (∗) by considering the two cases of the code:

[1]This detailed proof is not necessary for full marks

1st Case: `s[i]` $=$ `t[j]`

Then, by code, we have `i'` $=$ `i+1`, and `j'` $=$ `j+1`. Therefore,

$$(\texttt{s.length-i'+j'}, \texttt{t.length-j'}) = (\texttt{s.length-i+j}, \texttt{t.length-j-1})$$

The latter establishes $(*)$.

2nd Case: `s[i]` $\neq$ `t[j]`

Then, by code, we have `i'` $=$ `i-j+1`, and `j'` $=$ `0`. Therefore

$$(\texttt{s.length-i'+j'}, \texttt{t.length-j'}) = (\texttt{s.length-i+j-1}, \texttt{t.length})$$

The latter establishes $(*)$.

Notice that we did not need to use the conditions in the proof of (*), but we did use the invariants.

For the interested:

Note that there is a direct correspondence between our tuple version of the variant and our integer expression version of the variant.

In the tuple version, the value of the second element of the tuple is effectively dominated by the value of the first element of the tuple. This is analogous to how, when comparing numbers, the value of the units column is dominated by the value of the 10's column (or next higher base value to be more general). So, 31 is bigger that 26 because 3 is bigger than 2 (more precisely 3 lots of 10 is bigger than 2 lots of 10). We do not even need to consider the unit columns of 1 or 6.

Via similar thinking, it is possible to encode the dominating behaviour of our tuple variant into an integer expression variant. We do this by encoding the tuple into a base X representation, where X is determined by the maximum possible value of the second tuple element: `t.length`.

Thus, by multiplying the maximum value of the second tuple element (`t.length`) by the value of the first tuple element (`s.length-i+j`) we obtain `t.length`$*$(`s.length-i+j`) which expresses the dominating value of the tuple. We can then add to this the value of the second tuple element (`t.length-j`) to obtain the an overall integer expression:

$$\texttt{t.length} * (\texttt{s.length} - \texttt{i} + \texttt{j}) + \texttt{t.length} - \texttt{j}$$

Question 2 - Marking Scheme (20 marks)

While the UTAs have some discretion in marking the submissions, this marking scheme is indicative of how this work would be marked in exams, and the relative importance of the various parts.

(a) Midcondition: 3 points, awarded as follows

(M1)	1 point
(M2)	1 point
(M3)	1 point

The points above to be awarded also for equivalent formulations of (M1)-(M3) from above.

(b) Invariant: 9 points, awarded as follows

(I1)	1 point
(I2)	1 point for $0 \leq \texttt{i} \leq \texttt{s.length} \;\wedge\; 0 \leq \texttt{j} \leq \texttt{t.length}$
	1 point for $\texttt{i} \geq \texttt{j}$
(I3)	2 points
(I4)	2 points
(I5)	2 points

The points above to be awarded also for equivalent formulations of (I1)-(I5) from above.

(c) Variant, and Loop Termination: 8 points, awarded as follows

2 points	for a variant which, indeed, decreases
1 point	1st Approach: for finding a lower bound to the variant or 2nd Approach: for finding an appropriate well-founded ordering for this variant
5 points	for demonstrating that the variant does indeed decrease These points to be awarded as follows

1 point	for the distinction of the two cases
1 point	for expressing `i'`, `j'` correctly in terms of `i` and `j`
1 point	for expressing the value of variant before/after loop execution correctly in terms of `i`, `j` , `i'`, `j'`, `s.length` and `t.length`
2 points	for correctly showing that the expression describing the variant after loop execution is smaller than the expression describing the variant before loop execution