COMP[39]151 Warm-up assignment

Aditya Keswani (z3242379)
akeswani@cse.unsw.edu.au
and
Timothy Wiley (z3109831)
timothyw@cse.unsw.edu.au

August 16, 2011

1 Question 1

1.1 Algorithms ZeroA, ZeroB and ZeroC

1.1.1 Promela implementations

ZeroA

```
#include "fdef.h"

bool found;

proctype P() {
    byte i = 0;
    found = false;

    do
        :: (found) -> break
        :: else ->
            i = i + 1;
            found = (f(i) == 0)
    od
}

proctype Q() {
    byte j = 1;
    found = false;

    do
        :: (found) -> break
```

```
        :: else ->
             j = j - 1;
             found = (f(j) == 0)
    od
}

init {
    atomic {
        run P();
        run Q()
    }
}

ltl p0 { [] ( ( <> found ) && ( found -> [] found ) ) }
```

ZeroB

```
#include "fdef.h"

bool found = false;

proctype P() {
    byte i = 0;

    do
        :: (found) -> break
        :: else ->
             i = i + 1;
             found = (f(i) == 0)
    od
}

proctype Q() {
    byte j = 1;

    do
        :: (found) -> break
        :: else ->
             j = j - 1;
             found = (f(j) == 0)
    od
}

init {
    atomic {
        run P();
        run Q()
    }
}
```

```
ltl p0 { [] ( ( <> found ) && ( found -> [] found ) ) }
```

ZeroC

```
#include "fdef.h"

bool found = false;

proctype P() {
    byte i = 0;

    do
        :: (found) -> break
        :: else ->
            i = i + 1;
            if
                :: f(i) == 0 -> found = true
            fi
    od
}

proctype Q() {
    byte j = 1;

    do
        :: (found) -> break
        :: else ->
            j = j - 1;
            if
                :: f(j) == 0 -> found = true
            fi
    od
}

init {
    atomic {
        run P();
        run Q()
    }
}

ltl p0 { [] ( ( <> found ) && ( found -> [] found ) ) }
```

1.1.2 LTL formula

The LTL formula we use to test the correctness of these algorithms is:

$$\Box((\Diamond found) \wedge (found \Rightarrow \Box found)) \tag{1}$$

$found$ is only set to true when one of the processes finds the zero. Therefore, the first part of this formula,

$$\Diamond found \tag{2}$$

ensures that one of the processes does eventually find the zero.

The second part of this formula,

$$found \Rightarrow \Box found \tag{3}$$

ensures that after one of the processes finds the zero and sets $found$ to true, it will remain true. Since $found$ is the guard on the loop in both processes, this means that both processes will exit their loops and terminate after this occurs.

Finally, the outer $\Box$ ensures that these parts of the formula are satisfied in all states. Without this, the formula would only need to be satisfied in one state for the program to pass checking.

1.1.3 Spin output

ZeroA

The verification output was:
pan:1: end state in claim reached (at depth 20)

The simulation output was:
Starting P with pid 2
2: proc 0 (:init:) zeroA.pml:31 (state 1) [(run P())]
Starting Q with pid 3
3: proc 0 (:init:) zeroA.pml:32 (state 2) [(run Q())]
5: proc 2 (Q) zeroA.pml:19 (state 1) [found = 0]
7: proc 2 (Q) zeroA.pml:23 (state 4) [else]
9: proc 2 (Q) zeroA.pml:24 (state 5) [j = (j-1)]
11: proc 2 (Q) zeroA.pml:25 (state 6) [found = (j==0)]
13: proc 2 (Q) zeroA.pml:22 (state 2) [(found)]
15: proc 2 terminates
17: proc 1 (P) zeroA.pml:7 (state 1) [found = 0]
19: proc 1 (P) zeroA.pml:11 (state 4) [else]
21: proc 1 (P) zeroA.pml:12 (state 5) [i = (i+1)]
spin: trail ends after 21 steps
#processes: 2
21: proc 1 (P) zeroA.pml:13 (state 6)
21: proc 0 (:init:) zeroA.pml:34 (state 4)

Explanation:
At 11, Q sets found to true.
At 17, P sets found to false, so the claim is violated.

ZeroB

The verification output was:
pan:1: end state in claim reached (at depth 22)

The simulation output was:
Starting P with pid 2
2: proc 0 (:init:) zeroB.pml:29 (state 1) [(run P())]
Starting Q with pid 3
3: proc 0 (:init:) zeroB.pml:30 (state 2) [(run Q())]
5: proc 2 (Q) zeroB.pml:21 (state 3) [else]
7: proc 2 (Q) zeroB.pml:22 (state 4) [j = (j-1)]
9: proc 1 (P) zeroB.pml:10 (state 3) [else]
11: proc 1 (P) zeroB.pml:11 (state 4) [i = (i+1)]
13: proc 2 (Q) zeroB.pml:23 (state 5) [found = (j==0)]
15: proc 2 (Q) zeroB.pml:20 (state 1) [(found)]
17: proc 2 terminates
19: proc 1 (P) zeroB.pml:12 (state 5) [found = (i==0)]
21: proc 1 (P) zeroB.pml:10 (state 3) [else]
spin: trail ends after 23 steps
#processes: 2
23: proc 1 (P) zeroB.pml:11 (state 4)
23: proc 0 (:init:) zeroB.pml:32 (state 4)
3 processes created
Exit-Status 0

Explanation:
At 13, Q sets found to true.
At 19, P sets found to false, so the claim is violated.

ZeroC

The verification output was:
No errors found – did you verify all claims?

1.2 Algorithms ZeroD and ZeroE

1.2.1 Promela implementations

ZeroD

```
#include ”fdef.h”

```

```
bool found = false;
byte turn = 1;

proctype P() {
    byte i = 0;

    do
        :: (found) -> break
        :: else ->
pTurnChange:
            d_step { (turn == 1); turn = 2 }
pAfterTurnChange:
            i = i + 1;
            if
                :: (f(i) == 0) -> found = true
                :: else -> skip
            fi
    od
}

proctype Q() {
    byte j = 1;

    do
        :: (found) -> break
        :: else ->
qTurnChange:
            d_step { (turn == 2); turn = 1 }
qAfterTurnChange:
            j = j - 1;
            if
                :: (f(j) == 0) -> found = true
                :: else -> skip
            fi
    od
}

init {
    atomic {
        run P();
        run Q()
    }
}

ltl p0 { [] ( ( <> found ) && ( found -> [] found ) &&
            ( P@pTurnChange -> <> P@pAfterTurnChange ) &&
            ( Q@qTurnChange -> <> Q@qAfterTurnChange ) ) }
```

ZeroE

```
#include "fdef.h"

bool found = false;
byte turn = 1;

proctype P() {
    byte i = 0;

    do
        :: (found) -> break
        :: else ->
pTurnChange:
            d_step { (turn == 1); turn = 2 }
pAfterTurnChange:
            i = i + 1;
            if
                :: (f(i) == 0) -> found = true
                :: else -> skip
            fi
    od;

    turn = 2;
}

proctype Q() {
    byte j = 1;

    do
        :: (found) -> break
        :: else ->
qTurnChange:
            d_step { (turn == 2); turn = 1 }
qAfterTurnChange:
            j = j - 1;
            if
                :: (f(j) == 0) -> found = true
                :: else -> skip
            fi
    od;

    turn = 1;
}

init {
    atomic {
        run P();
        run Q()
```

```
        }
}

ltl  p0 { []  ( (  <> found ) && ( found -> [] found ) &&
               ( P@pTurnChange -> <> P@pAfterTurnChange ) &&
               ( Q@qTurnChange -> <> Q@qAfterTurnChange ) ) }
```

1.2.2 LTL formula

The LTL formula we use to test the correctness of these algorithms is:

$$\begin{aligned} &\Box((\Diamond found) \wedge (found \Rightarrow \Box found) \\ &\wedge(P@pTurnChange \Rightarrow \Diamond P@pAfterTurnChange) \\ &\wedge(Q@qTurnChange \Rightarrow \Diamond Q@qAfterTurnChange)) \end{aligned} \tag{4}$$

The first two parts of this formula are the same as those used for algorithms ZeroA, ZeroB and ZeroC and are explained above. However, these cannot prevent a program from permanently blocking inside its await statement (and never terminating) after $found$ has been set to true. The latter two parts of the formula,

$$\begin{aligned} &(P@pTurnChange \Rightarrow \Diamond P@pAfterTurnChange) \\ &\wedge(Q@qTurnChange \Rightarrow \Diamond Q@qAfterTurnChange) \end{aligned} \tag{5}$$

prevent this by ensuring that if a process enters its await statement, it eventually exits it.

1.2.3 Spin output

ZeroD

The verification output was:
pan:1: acceptance cycle (at depth 39)

The simulation output was:
Starting P with pid 2
2: proc 0 (:init:) zeroD.pml:42 (state 1) [(run P())]
Starting Q with pid 3
3: proc 0 (:init:) zeroD.pml:43 (state 2) [(run Q())]
5: proc 2 (Q) zeroD.pml:28 (state 3) [else]
7: proc 1 (P) zeroD.pml:11 (state 3) [else]
9: proc 1 (P) zeroD.pml:13 (state 6) [((turn==1))]
9: proc 1 (P) zeroD.pml:13 (state 5) [turn = 2]
11: proc 2 (Q) zeroD.pml:30 (state 6) [((turn==2))]
11: proc 2 (Q) zeroD.pml:30 (state 5) [turn = 1]
13: proc 2 (Q) zeroD.pml:32 (state 7) [j = (j-1)]
15: proc 2 (Q) zeroD.pml:34 (state 8) [((j==0))]
17: proc 1 (P) zeroD.pml:15 (state 7) [i = (i+1)]

19: proc 1 (P) zeroD.pml:18 (state 10) [else]
21: proc 1 (P) zeroD.pml:18 (state 11) [(1)]
23: proc 1 (P) zeroD.pml:11 (state 3) [else]
25: proc 1 (P) zeroD.pml:13 (state 6) [((turn==1))]
25: proc 1 (P) zeroD.pml:13 (state 5) [turn = 2]
27: proc 1 (P) zeroD.pml:15 (state 7) [i = (i+1)]
29: proc 1 (P) zeroD.pml:18 (state 10) [else]
31: proc 1 (P) zeroD.pml:18 (state 11) [(1)]
33: proc 1 (P) zeroD.pml:11 (state 3) [else]
35: proc 2 (Q) zeroD.pml:34 (state 9) [found = 1]
37: proc 2 (Q) zeroD.pml:27 (state 1) [(found)]
39: proc 2 terminates
¡¡¡¡START OF CYCLE¿¿¿¿
spin: trail ends after 41 steps
#processes: 2
41: proc 1 (P) zeroD.pml:13 (state 6)
41: proc 0 (:init:) zeroD.pml:45 (state 4)
3 processes created
Exit-Status 0

Explanation:
At 35, Q sets found to true.
Then P becomes permanently stuck in its await because turn is never set to 1.

ZeroE

The verification output was:
No errors found – did you verify all claims?

2 Question 2

We defined the Owicki/Gries style proof of the partial correctness of `zeroE` as

$$\{f \text{ is a function over } \mathbb{Z} \text{ with at least one zero}\}\texttt{zeroE}\{x \in \mathbb{Z} \wedge f(x) = 0\} \tag{6}$$

For the proof, we constructed the transition diagram for process P (Figure 1) and the transition diagram for Q (Figure 2). Each state corresponds to the appropriate line of code in the processes and assertions at each state are given in blue.

To aid in the proof we defined two global auxiliary boolean variables fP and fQ. These variables are true if states $p6$ and $q6$ have ever been visited respectively, and are false otherwise. To achieve this, both variables are initially set to false, and are set to true when transitioning into their respective states $p6$ or $q6$.

We then introduce the global invariant

$$I: \quad found = fP \vee fQ \tag{7}$$

This invariant is maintained at every state as:

1. Initially all three variables $found$, fP and fQ are false.
2. In the states that change $found$, fP or fQ.
   a) State $p6$ sets both $found$ and fP to true, maintaining the invariant.
   b) Likewise, state $q6$ sets both $found$ and fQ to true, maintaining the invariant.

The assertions on the transition diagrams can now be established. Consider the transition diagram for P. The assertions are proven as follows:

1. At all states: $\{i \geq 0\}$ is true as initially $i := 0$ and the only transition to alter i is $p3- > p4$ which increments i.
2. At $p2$: $\{!fP\}$ is derived from the transition condition and I. There is no interference as Q does not alter fP.
3. At $p3$ and $p4$: $\{!fP\}$ is maintained as fP is not altered. There is no interference as Q does not alter fP.
4. At $p5$: $\{f(i) = 0\}$ is given from the transition condition and there is no interference as i is local to P.
5. At $p6$: $\{f(i) = 0\}$ is given as i is unchanged. $\{fP\}$ is given from the transition action. There is no interference as Q does not alter i or fP.
6. At $p1$: $\{fP \to f(i) = 0\}$ named $a1$, is given by the transitions into $p1$ from $p4$ and $p6$. From $p4$, the assertion $\{!fP\}$ makes $a1$ trivially true. From $p6$ the assertions $\{fP\}$ and $\{f(i) = 0\}$ ensure $a1$ is true.
7. At $p7$: the assertion $\{fP \to f(i) = 0\}$ is maintained as either fP or i is changed, and there is no interference as neither is modified by Q.

The assertions on the transition diagram for Q can be established in a similar way, so we do not detail them here.

We finally must establish the assertions at the terminal states of P and Q imply the overall partial correctness for `zeroE`. The set of assertions from the terminal states $p7$ and $q7$ are:

$$\begin{array}{rc} I: & found = fP \vee fQ \\ & found \\ & i \geq 0 \\ & j \leq 1 \\ & fP \to f(i) = 0 \\ & fQ \to f(j) = 0 \end{array}$$

Given $found$ and I, the implications can be collapsed to give

$$\begin{array}{c} i \geq 0 \\ j \leq 1 \\ f(i) = 0 \vee f(j) = 0 \end{array}$$

Finally, as the combined values of i and j cover $\mathbb{Z}$, we can conclude what was required, that is

$$x \in \mathbb{Z} \land f(x) = 0 \tag{8}$$

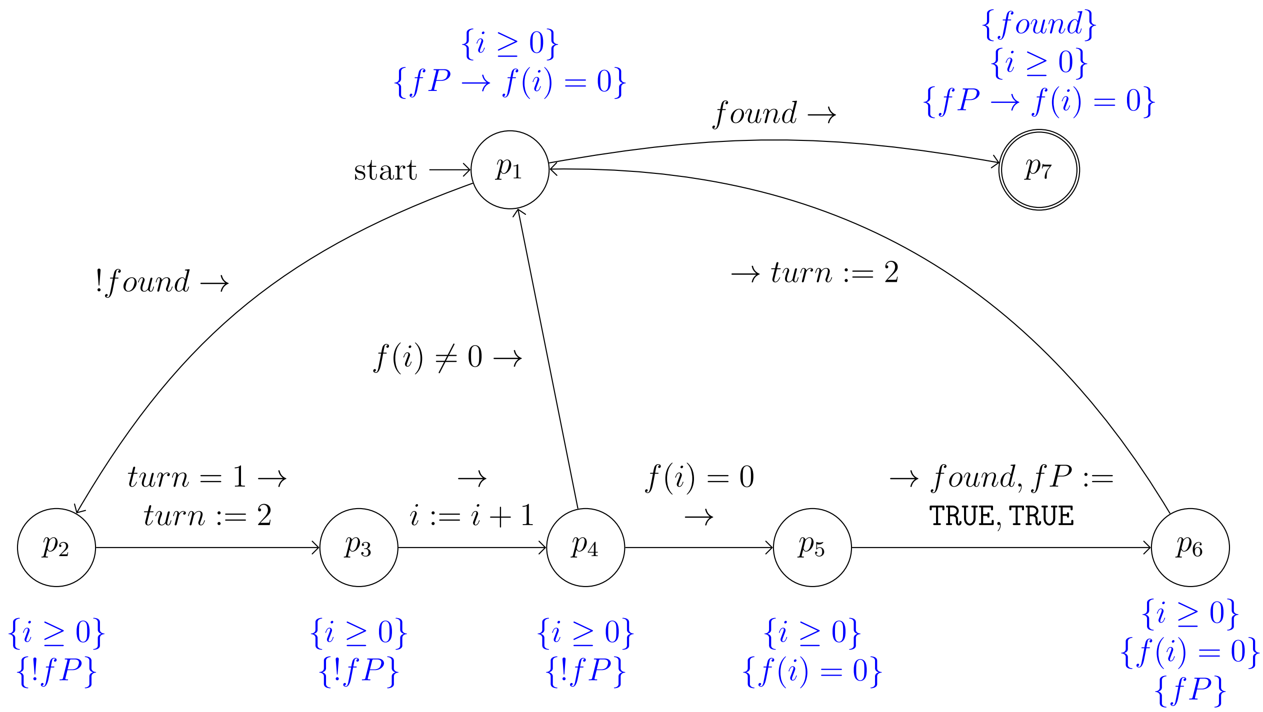

Figure 1: Transition Diagram for process P with assertions in blue

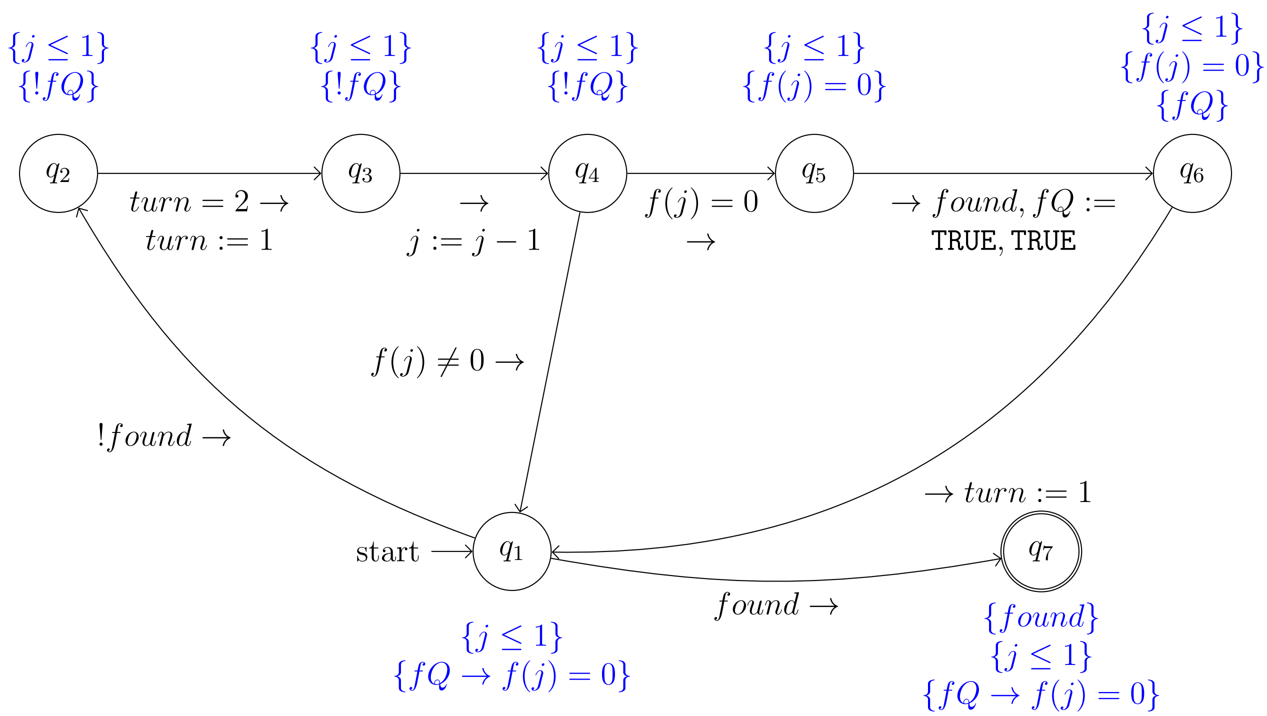

Figure 2: Transition Diagram for process Q with assertions in blue