Exercises, 14 January 2010

Separation logic

1. The following axiom schemata are not sound: for each, give an instance which is not valid along with a description of a state in which the instance is false.

$$p_0 * p_1 \Rightarrow p_0 \wedge p_1 \qquad (p_0 * p_1) \vee q \Rightarrow (p_0 \vee q) * (p_1 \vee q) \qquad (p_0 * q) \wedge (p_1 * q) \Rightarrow (p_0 \wedge p_1) * q$$

2. Prove that

$$(x \mapsto y * x' \mapsto y') * \mathbf{true} \;\Rightarrow\; ((x \mapsto y * \mathbf{true}) \wedge (x' \mapsto y' * \mathbf{true})) \wedge x \neq x'.$$

3. Fill in the postconditions in

$$\{(e_1 \mapsto -) * (e_2 \mapsto -)\} \;\; [e_1] := e_1'; [e_2] := e_2' \;\; \{?\}$$

$$\{(e_1 \mapsto -) \wedge (e_2 \mapsto -)\} \;\; [e_1] := e_1'; [e_2] := e_2' \;\; \{?\}$$

4. A braced list segment is a list segment with an interior pointer j to its last element; in the special case where the list segment is empty, j is **nil**. Formally,

$$\mathbf{brlseg}\ \epsilon\ (i, j, k) \;=\; \mathbf{emp} \wedge i = k \wedge j = nil$$

$$\mathbf{brlseg}\ \alpha \cdot a\ (i, j, k) \;=\; \mathbf{lseg}\ \alpha\ (i, j) * j \mapsto a, k$$

   (a) Write a procedure lookuppt that returns the final pointer of a braced list segment:

$$\{\mathbf{brlseg}\ \alpha\ (i, j, k_0)\}\ \mathsf{lookuppt}\ \{\mathbf{brlseg}\ \alpha\ (i, j, k_0) \wedge k = k_0\}$$

   lookuppt accepts i, j as arguments and returns k.

   (b) Write a procedure appright that appends an element to the right:

$$\{\mathbf{brlseg}\ \alpha\ (i, j, k_0)\}\ \mathsf{appright}\ \{\mathbf{brlseg}\ \alpha \cdot a\ (i, j, k_0)\}$$

   appright accepts i, j and a as arguments and returns i, j.

Concurrent separation logic

1. Consider the program

```
init() { c := nil }

resource buf(c);

while (true) {                 while (true) {
  with buf do {                  with buf when not(c=nil) {
     x := new(c);     ||           t := [c];
     c := x;                       dispose(t);
  }                                c := t;
}                                }
                               }
```

   (a) Describe informally the behaviour of the program.

   (b) Prove that $\{\mathbf{empty}\}$ *program* $\{\mathbf{true}\}$ (and explain the invariant you picked up for `buf`).

Owicky-Gries and rely/guarantee

1. Consider the program

```
x := x-1; x := x+1   ||   y := y+1; y := y-1
```

   Prove that $\{x = y\}$ *program* $\{x = y\}$ is a theorem (detail the non-interference proofs).

2. Reformulate your solution to 1. using rely-guarantee reasoning.

Weak-memory models

1. Peterson algorithm is a classic solution to the *mutual exclusion* problem: in all executions, the instructions of the critical sections of the two threads are not interleaved.

```
flag0 := false;
flag1 := false;

flag0 := true;                          flag1 := true;
turn := 1;                              turn := 0;
while (flag1 && turn = 1);     ||      while (flag0  && turn == 0);
// critical section                     // critical section
   ...                                    ...
// end of critical section              // end of critical section
flag0 := false;                         flag1 := false;
```

   (a) Explain informally why the two threads cannot be inside the critical section at the same time.
   (b) Does Peterson algorithm guarantee mutual exclusion if executed on a multiprocessor machine where store buffers are observable (e.g. x86)?
   (c) Implement the Peterson algorithm in your favourite language, and verify experimentally if it guarantees mutual exclusion.