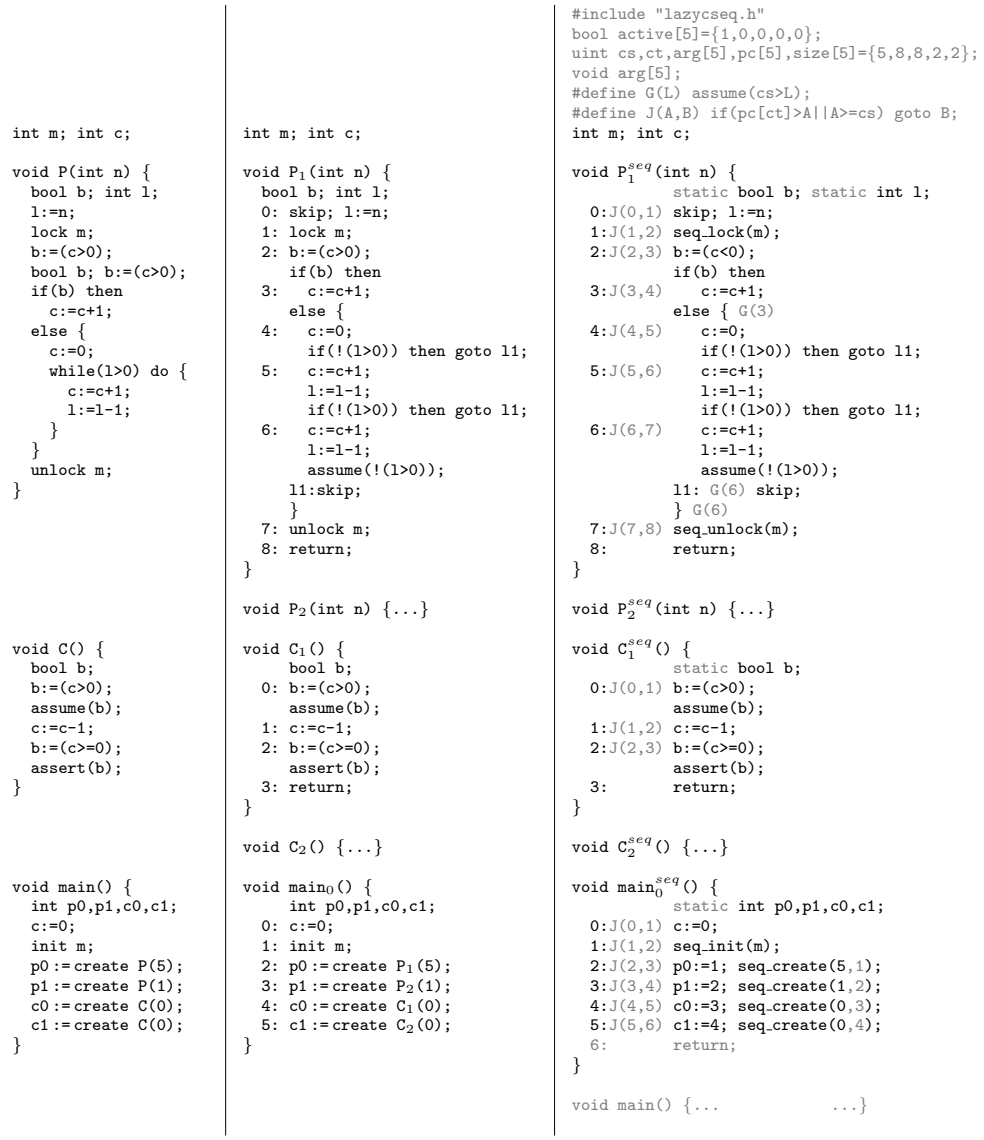

```
int m; int c;

void P(int n) {
  bool b; int l;
  l:=n;
  lock m;
  b:=(c>0);
  bool b; b:=(c>0);
  if(b) then
    c:=c+1;
  else {
    c:=0;
    while(l>0) do {
      c:=c+1;
      l:=l-1;
    }
  }
  unlock m;
}

void C() {
  bool b;
  b:=(c>0);
  assume(b);
  c:=c-1;
  b:=(c>=0);
  assert(b);
}

void main() {
  int p0,p1,c0,c1;
  c:=0;
  init m;
  p0 := create P(5);
  p1 := create P(1);
  c0 := create C(0);
  c1 := create C(0);
}
```

```
int m; int c;

void P1(int n) {
  bool b; int l;
  0: skip; l:=n;
  1: lock m;
  2: b:=(c>0);
     if(b) then
  3:   c:=c+1;
     else {
  4:   c:=0;
       if(!(l>0)) then goto l1;
  5:   c:=c+1;
       l:=l-1;
       if(!(l>0)) then goto l1;
  6:   c:=c+1;
       l:=l-1;
       assume(!(l>0));
     l1:skip;
     }
  7: unlock m;
  8: return;
}

void P2(int n) {...}

void C1() {
     bool b;
  0: b:=(c>0);
     assume(b);
  1: c:=c-1;
  2: b:=(c>=0);
     assert(b);
  3: return;
}

void C2() {...}

void main0() {
     int p0,p1,c0,c1;
  0: c:=0;
  1: init m;
  2: p0 := create P1(5);
  3: p1 := create P2(1);
  4: c0 := create C1(0);
  5: c1 := create C2(0);
}
```

```
#include "lazycseq.h"
bool active[5]={1,0,0,0,0};
uint cs,ct,arg[5],pc[5],size[5]={5,8,8,2,2};
void arg[5];
#define G(L) assume(cs>L);
#define J(A,B) if(pc[ct]>A||A>=cs) goto B;
int m; int c;

void P1^seq(int n) {
          static bool b; static int l;
  0:J(0,1) skip; l:=n;
  1:J(1,2) seq_lock(m);
  2:J(2,3) b:=(c<0);
           if(b) then
  3:J(3,4)    c:=c+1;
           else { G(3)
  4:J(4,5)    c:=0;
              if(!(l>0)) then goto l1;
  5:J(5,6)    c:=c+1;
              l:=l-1;
              if(!(l>0)) then goto l1;
  6:J(6,7)    c:=c+1;
              l:=l-1;
              assume(!(l>0));
           l1: G(6) skip;
           } G(6)
  7:J(7,8) seq_unlock(m);
  8:       return;
}

void P2^seq(int n) {...}

void C1^seq() {
          static bool b;
  0:J(0,1) b:=(c>0);
           assume(b);
  1:J(1,2) c:=c-1;
  2:J(2,3) b:=(c>=0);
           assert(b);
  3:       return;
}

void C2^seq() {...}

void main0^seq() {
          static int p0,p1,c0,c1;
  0:J(0,1) c:=0;
  1:J(1,2) seq_init(m);
  2:J(2,3) p0:=1; seq_create(5,1);
  3:J(3,4) p1:=2; seq_create(1,2);
  4:J(4,5) c0:=3; seq_create(0,3);
  5:J(5,6) c1:=4; seq_create(0,4);
  6:       return;
}

void main() {...            ...}
```

(a) original program (b) bounded program (c) sequentialized program

Fig. (a) Original multi-threaded producer-consumer program containing a reachable assertion failure. In the `main` thread, functions `P` and `C` are both used twice to spawn a thread. (b) Corresponding bounded multi-threaded program, resulting from applying standard transformations (with a loop unrolling bound of $n = 2$) to the original program. The functions $\mathtt{P}_1$ and $\mathtt{P}_2$ represent two distinct copies of the `P`-thread that was spawned twice in the original program. (c) Corresponding sequentialized program. The code injected by the source transformation is shown in gray. For succinctness, we use C-style intializers in declarations as well as macros.