```
public abstract class LinearSearch
{
    /** The function that describes what is being sought. */
    //@ requires j >= 0;
    public abstract /*@ pure @*/ boolean f(int j);

    /** The last integer in the search space, this describes the
     * domain of f, which goes from 0 to the result.
     */
    //@ ensures 0 <= \result;
    //@ ensures (\exists int j; 0 <= j && j <= \result; f(j));
    public abstract /*@ pure @*/ int limit();

    /** Find a solution to the searching problem. */
    /*@ public normal_behavior
      @   requires (\exists int i; 0 <= i && i <= limit(); f(i));
      @   ensures \result == (\min int i; 0 <= i && f(i); i);
      @*/
    public int find() {
        int x = 0;
        //@ maintaining 0 <= x &&
        //@ (\forall int i; 0 <= i && i < x; !f(i));
        while (!f(x)) {
            /*@ assert 0 <= x && !f(x)
              @    && (\forall int i; 0 <= i && i < x; !f(i));
              @*/
            x = x + 1;
        }
        /*@ assert 0 <= x && f(x)
          @     && (\forall int i; 0 <= i && i < x; !f(i));
          @*/
        //@ hence_by (* definition of \min *);
        //@ assert x == (\min int i; 0 <= i && f(i); i);
        return x;
    }
}
```

```
var depth: int;

procedure init() {
        depth := 0;
}
procedure Acquire() {
        depth := depth + 1;
}
procedure Release() {
        depth := depth - 1;
}
procedure d_exit() {
        assert depth == 0;
}
procedure P_n()
{ call init(); call dispatchP_n(); }

procedure dispatchP_n() {
        [call Acquire()]^n;
L1: while(*)
        { call Acquire(); call Release(); }
        [call Release()]^n; call d_exit();
}
```

The predicates are:

$$\psi_i \equiv depth - old(depth)$$

$$\eta \equiv (old(depth) == 0 \Rightarrow ok)$$

Possible proof of P_n:

$$[depth == 0]@init, [\psi_1]@Acquire, [\psi_{-1}]@Release, [\eta]@d\_exit$$

$$[\psi_0]@dispatchP_n@L1, [\eta]@dispatchP_n$$

Houdini can reconstruct this proof using the annotations: $depth == 0, \psi_{-1}, \psi_1, \psi_0, \eta$.