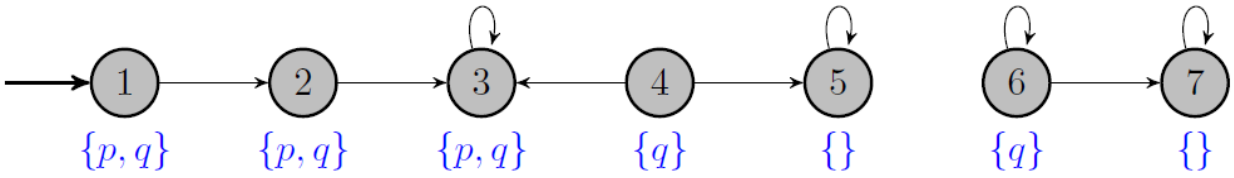
1. **[3 marks]** Recall the *ferryman* puzzle (Sect. 3.3.5, Huth & Ryan[1]) that we had solved using NuSMV.

   Your task is to model the problem using CBMC so that it can print the best solution (shortest number of trips for the ferryman) in its trace. You should output the trace into a text file, and annotate with comments (using the tags `<note>` *your comment here* `</note>`) at several places so that these comments can be followed to understand how the ferryman can transport everything to the other side of the river.

   We would also like CBMC to verify that your solution is indeed the best. Describe how we may do so (you must submit any code or command that needs to be run, and explain how we should interpret the corresponding output).

2. **[2 marks]** Consider the transition system shown below.



   We would like to model this using CBMC and prove that q is a 2-inductive invariant (recall that this works when you add the constraint that all states on any counterexample to the step case are pairwise different). Your task is to submit two files, that can be used to prove the base case and the step case of the 2-induction. You must also submit a `readme` file that contains instructions on how your code should be run, and how should the output be interpreted.

3. **[3 marks]** For the transition system in the question above, show how the IC3 algorithm will work step-by-step to prove that the bad states are unreachable. Note that for q to be an invariant, states 5 and 7 are bad. Do not generalize the CTIs in your proof.

4. **[1 marks]** Consider the program, P, shown below.

```
k = 0; j = 1;

while(k != n) {
    k = k + 1;
    j = 2 + j;
}
```

   Prove the validity of the following:
   $\{n > 0\} \quad P \quad \{j = 1 + 2.n\}$

[1] a soft copy of the book can be found online; let me know in case you cannot

5. **[3 marks]** Consider the following program over integer variables x and y.

```
L1:   x = y;
L2:   while(x < 1000) {
L3:     y = y + 1000;
L4:     x = 1000 - y;
L5:   }
L6:   assert (!((0 <= y) && (y < 1000)));
```

We wish to use predicate abstraction to determine if the above program, when executed in a state satisfying the precondition *true* can violate the assertion. Let us start off with an initial choice of predicates $\mathcal{P} = \{p_1 \equiv (x < 1000), p_2 \equiv (y < 1000), p_3 \equiv (y \geq 0)\}$.

(a) Fill in the blank below with the most precise expression (in terms of p_1, p_2, p_3, $*$) to complete the Boolean program corresponding to the above choice of predicates $\mathcal{P}$.

```
L1:    p1 = p2;
L2a:   while (*) {  // non-deterministic choice
L2b:     assume(p1);
L3:      (p2, p3) = (____________________, (!p2 || p3) ? 1 : *);
L4:      p1 = !p2 ? 1 : (!p3 ? 0 : *);
L5:    }
L2c:   assume(!p1);
L6:    assert(!(p3 && p2));
```

(b) Show that `L1, L2a, L2b, L3, L4, L5, L2a, L2c, L6` is an abstract counterexample trace, i.e. a trace of the abstract program that violates the assertion. In other words, you have to provide values of p_1, p_2, p_3 at the start of the Boolean program that causes the instructions in the above trace to be executed and the assertion (in the Boolean program) to be violated.

(c) Construct the trace formula (from the original program statements) corresponding to the above abstract counterexample trace.

(d) Is the trace formula constructed above satisfiable?

If so, solve the trace formula to obtain a value of y at the start of the original program that leads to a violation of the assertion after iterating through the while loop (in the original program) once.

Otherwise, use Craig interpolation to identify the smallest set of additional predicates that need to be added at each location of the original program to ensure that the resulting predicate abstraction excludes this counterexample trace.