1 [10 p]

Suppose $A = \{n \in \mathbb{N} : 1 \leq n \leq 10\}$ and a family of relations on A,

$$R_i = \{(a, b) \in A \times A : \mod (ab, a + b) = i\}$$

for any $i \in \mathbb{N}$. Here, $\mod (m, n)$ is the remainder when dividing positive integer m by positive integer n, e.g. $\mod (14, 4) = 2$, and $\mod (5, 7) = 5$.

For example, $R_3 = \{(a, b) \in A \times A : \mod (ab, a + b) = 3\}$.

Compute the following cardinalities:

1. [2 p] $\#R_5 = 9$
2. [2 p] $\#R_6 = 8$
3. [2 p] $\#R_9 = 5$
4. [4 p] $\#(R_5 \circ R_6) = 4$

R_5: #{[4 3] [5 5] [3 4] [1 5] [2 7] [5 10] [7 2] [10 5] [5 1]}
R_6: #{[6 8] [4 7] [1 6] [2 8] [8 2] [6 1] [7 4] [8 6]}
R_9: #{[9 9] [6 9] [1 9] [9 1] [9 6]}
$R_5 \circ R_6$: #{[6 5] [8 7] [7 3] [4 2]}

2 [20 p]

Given a function $f : A \longrightarrow B$ and an injection $g : B \hookrightarrow C$. The sets A, B, and C are **not empty**.

1. [5 p] Is their composition $g \circ f : A \longrightarrow C$ injective (circle answer)?

   Never Sometimes Always

   (Circle "Never" if a function f and an injective function g could never be composed in this way to result in an injective function. Circle "Sometimes", if some such compositions would be injective, others would not be, depending on f and g. Circle "Always" if all such compositions would be injective.)

2. [15 p] If you answered "Never" or "Always" to the question above, prove it. If you answered "Sometimes", construct an example where $g \circ f$ is injective and one where it is not. (Constructing an example would mean defining sets A, B, and C, as well as the two functions, according to the decription above.)

   composition injective:

   $A = B = C = \{a\}$

   $f = g = \{(a, a)\}$

   composition not injective:

   $A = B = C = \{a, b\}$

   $f = \{(a, a), (b, a)\}$

   $g = \{(a, a), (b, b)\}$

3 [30 p]

Suppose you have a graph (V, E) with vertices V and edges $E \subseteq V \times V$. Also given is a function $w : E \longrightarrow \mathbb{N}^+$, assigning each edge a positive natural number as its *edge weight*.

In this task, we represent a *path* in the graph as a finite sequence of elements in V. In other words: a path in this graph is an element of V^*, such that each pair of successive vertices in it are connected by an edge. For example, suppose $p = abdcfbe$. If $a, b, c, d, e, f \in V$ then $p \in V^*$, and if $(a, b), (b, d), (d, c), (c, f), (f, b), (b, e) \in E$, then p is a path in our graph. Its *path weight* is the sum of the edge weights of the edges in it. Note that the same vertex can appear any number of times in a path. The same is true for the edges, and if an edge occurs multiple times in a path, its edge weight is counted for each occurrence. For example, if $p = abdab$, then we add the edge weights for (a, b), (b, d), (d, a), and (a, b) again to get the path weight for p.

We want to define a function

$$P : \mathbb{N} \longrightarrow \mathcal{P}(V^*)$$

which takes a natural number and computes a set of paths in the graph. Specifically, $P(n)$ is **the set of <u>all</u> paths** p in our graph such that

- p **visits every vertex at least once**, and
- its path weight is less than or equal to n.

In order to define P, we will use a helper function

$$P' : V \times V^* \times \mathcal{P}(V) \times \mathbb{N} \longrightarrow \mathcal{P}(V^*)$$

$P'(a, p, S, n)$ is the the set of all paths in our graph of the form paq, i.e. paths consisting of a path p, followed by vertex a, followed by path q, with the following properties:

- paq is a valid path in the graph,
- q visits all the vertices in S,
- the path weight of q is less than or equal to n.

Using P', we can define P as follows:

$$P : n \mapsto \bigcup_{a \in V} P'(a, \varepsilon, V \setminus \{a\}, n)$$

1. [25 p] Define P' recursively.

$$P' : (a, p, S, n) \mapsto \begin{cases} \{pa\} \cup \bigcup_{b \in \{v \in E(a) : w(a,v) \leq n\}} P'(b, pa, \emptyset, n - w(a, b)) & \text{for } S = \emptyset \\ \bigcup_{b \in \{v \in E(a) : w(a,v) \leq n\}} P'(b, pa, S \setminus \{b\}, n - w(a, b)) & \text{otherwise} \end{cases}$$

This one was of course the most difficult task. The following parts of the problem description were frequently misunderstood or ignored:

- The function is supposed to return a *set* of sequences, in fact the set of *all* sequences meeting the two following criteria.
- The first criterion was that every vertex occur at least once in a result sequence, but it can occur multiple times.
- The second criterion was that the path weight of the remaining result sequence be less than or equal to n.

Consequently, once you found a path that met those criteria, you needed to make it part of the result, but also to keep searching for longer paths, as long as the path weight had not exceeded n. That's what the first clause above does. The second clause handles the case that the paths so far have not yet touched all vertices, so you just keep on trucking.

Some students found a very elegant way of collapsing these two cases into one (note that once S is empty, so is $S \setminus \{b\}$):

$$P' : (a, p, S, n) \mapsto \{pa : S = \emptyset\} \cup \bigcup_{b \in \{v \in E(a) : w(a,v) \leq n\}} P'(b, pa, S \setminus \{b\}, n - w(a, b))$$

A few minor errors I let slide if the overall construction worked. For example, there seems to be a lot of confusion about the difference between the empty set $\emptyset$ and the empty sequence ε. Also, many people forgot to put a path they wanted to return into braces, i.e. to return a singleton set instead of just the path.

Another error which I also let pass if the rest worked, was to not actually test that the weight of the next edge $w(a, v)$ isn't greater than n under the union sign, i.e. when constructing the set of vertices to move to next before actually calling P'. Instead, in many solutions all vertices reachable from the current one were explored and the call just did $P'(..., n - w(a, b))$. This, of course, meant that the last argument (that is, n) could become negative, which had to be sorted out by adding a case in the definition of P' testing for that possibility. Technically, that's in violation of the definition of P', which demands a natural number as its last argument, but it's a reasonable choice if one were to implement this, so I let that one go.

2. [5 p] In order to ensure that P' terminates, we require a **well-founded strict order** $\prec$ of its arguments, such that for any (a, p, S, n) that P' is called on, it will only ever call itself on $(a', p', S', n') \prec (a, p, S, n)$. Define such an order:

$$(a', p', S', n') \prec (a, p, S, n) \iff n' < n$$

The important bit here is that, unlike the Hamiltonian path search, termination of this recursion is controlled by n. The parameter S is used to determine when paths are made part of the result (viz. when it is empty, i.e. all vertices have been visited at least once), but it has no effect on termination. It is true, of course, that it becomes monotonically smaller as the recursion progresses, but not *strictly* so: while it will never grow in the recursive call, it can very well stay the same, viz. when the next node has already occurred in the path so far, and so isn't contained in S. Consequently, that set cannot be used to construct a *strict* order.

4 [20 p]

Suppose we have a tree (T, R) and a function $\lambda : T \longrightarrow \mathbb{N}$ assigning each node in the tree a natural number as a label.

1. [5 p] Define the set $T_{\mathbb{P}} \subseteq T$ consisting of all nodes that are labeled with a number in $\mathbb{P}$, i.e. a prime number (you do not have to define prime numbers, just use $\mathbb{P}$ in your definition):

   $T_{\mathbb{P}} = \{n \in T : \lambda(n) \in \mathbb{P}\}$

2. [5 p] Define the set $A_{x,y} \subseteq T$ consisting of all nodes that have label x and that have at least one child with label y (for $x, y \in \mathbb{N}$):

   $A_{x,y} = \{n \in T : \lambda(n) = x \wedge \exists m \in R(n) : \lambda(m) = y\}$

   An elegant alternative that some students found is
   $A_{x,y} = \{n \in T : \lambda(n) = x \wedge y \in \lambda(R(n))\}$

3. [5 p] Define the set $B_{x,y} \subseteq T$ consisting of all nodes that have label x and have at least one "grandchild", i.e. one child of a child, with label y (for $x, y \in \mathbb{N}$):

   $B_{x,y} = \{n \in T : \lambda(n) = x \wedge \exists m \in R(R(n)) : \lambda(m) = y\}$

4. [5 p] Define the set $C_n \subseteq T$, consisting of all nodes that are descendents (children, children of children, children of children of children and so on) of node $n \in T$:

   $C_n = R^+(n)$

There are various ways of answering this. The one above uses the transitive closure of R (which our book writes as R^*). You could also use the closure of $R[\{n\}]$, as long as you remember that n itself is an element of it, and take it out, i.e. $C_n = R[\{n\}] \setminus \{n\}$. Some folks have used the C_m of other nodes, which is quite clever, or defined a function to help them compute C_n. Assuming they are technically correct, all of those are perfectly fine answers.

5 [20 p]

1. [5 p] $((p \to q) \barwedge (q \to r)) \wedge ((r \to s) \barwedge (s \to p))$

$(\neg p \wedge q \wedge \neg r \wedge s) \vee (p \wedge \neg q \wedge r \wedge \neg s)$

```
(p q r s)
(1 1 1 1)  -->  0
(1 1 1 0)  -->  0
(1 1 0 1)  -->  0
(1 1 0 0)  -->  0
(1 0 1 1)  -->  0
(1 0 1 0)  -->  1
(1 0 0 1)  -->  0
(1 0 0 0)  -->  0
(0 1 1 1)  -->  0
(0 1 1 0)  -->  0
(0 1 0 1)  -->  1
(0 1 0 0)  -->  0
(0 0 1 1)  -->  0
(0 0 1 0)  -->  0
(0 0 0 1)  -->  0
(0 0 0 0)  -->  0
```

2. [5 p] $\neg(((p \barwedge q) \to (q \barwedge r)) \to ((r \barwedge s) \to (s \barwedge p)))$

$(p \wedge \neg r \wedge s)$

```
(p q r s)
(1 1 1 1)  -->  0
(1 1 1 0)  -->  0
(1 1 0 1)  -->  1
(1 1 0 0)  -->  0
(1 0 1 1)  -->  0
(1 0 1 0)  -->  0
(1 0 0 1)  -->  1
(1 0 0 0)  -->  0
(0 1 1 1)  -->  0
(0 1 1 0)  -->  0
(0 1 0 1)  -->  0
(0 1 0 0)  -->  0
(0 0 1 1)  -->  0
(0 0 1 0)  -->  0
(0 0 0 1)  -->  0
(0 0 0 0)  -->  0
```

3. [5 p] $(p \leftrightarrow q) \wedge ((p \rightarrow r) \barwedge (q \rightarrow s))$

$(p \wedge q \wedge \neg r) \vee (p \wedge q \wedge \neg s)$

```
(p q r s)
(1 1 1 1)  -->  0
(1 1 1 0)  -->  1
(1 1 0 1)  -->  1
(1 1 0 0)  -->  1
(1 0 1 1)  -->  0
(1 0 1 0)  -->  0
(1 0 0 1)  -->  0
(1 0 0 0)  -->  0
(0 1 1 1)  -->  0
(0 1 1 0)  -->  0
(0 1 0 1)  -->  0
(0 1 0 0)  -->  0
(0 0 1 1)  -->  0
(0 0 1 0)  -->  0
(0 0 0 1)  -->  0
(0 0 0 0)  -->  0
```

4. [5 p] $(p \rightarrow q) \barwedge ((p \leftrightarrow r) \barwedge (q \leftrightarrow s))$

$(p \wedge \neg q) \vee (p \wedge q \wedge r \wedge s) \vee (\neg p \wedge q \wedge \neg r \wedge s) \vee (\neg p \wedge \neg q \wedge \neg r \wedge \neg s)$

```
(p q r s)
(1 1 1 1)  -->  1
(1 1 1 0)  -->  0
(1 1 0 1)  -->  0
(1 1 0 0)  -->  0
(1 0 1 1)  -->  1
(1 0 1 0)  -->  1
(1 0 0 1)  -->  1
(1 0 0 0)  -->  1
(0 1 1 1)  -->  0
(0 1 1 0)  -->  0
(0 1 0 1)  -->  1
(0 1 0 0)  -->  0
(0 0 1 1)  -->  0
(0 0 1 0)  -->  0
(0 0 0 1)  -->  0
(0 0 0 0)  -->  1
```