


```
%% Names of students who do not have an advisor
```

```
%% Q5_RA= $\pi_{\text{name}}\text{STUDENTS}-\pi_{\text{name}}(\text{STUDENTS}\bowtie\text{ADVISING})$ 
```

```
q5(Name) :- studs(Stno,Name,_,_,_,_), \+advis(Stno,_).
```

```
%% Student and Instructor names where this student never took any course
%% with this instructor
```

```
%% Q6_RA= $\pi_{\text{s\_name}, \text{i\_name}, ($ 
```

```
%%  $(\pi_{\text{stno}}\text{STUDENTS}\times\pi_{\text{empno}}\text{INSTRUCTORS}$ 
```

```
%%  $-\pi_{\text{stno}}\text{STUDENTS}\times\pi_{\text{empno}}\text{INSTRUCTORS}\bowtie\text{GRADES})$ 
```

```
%%  $\bowtie\rho_{\text{name}\rightarrow\text{s\_name}}\text{STUENTS}$ 
```

```
%%  $\bowtie\rho_{\text{name}\rightarrow\text{i\_name}}\text{INSTRUCTORS}$ 
```

```
%%  $)$ 
```

```
q6(SName, IName) :- studs(Stno, SName, _, _, _, _),
                    insts(Empno, IName, _, _, _),
                    \+grads(Stno, Empno, _, _, _, _).
```

Problem 2.

(ii) Write a stratified Datalog program to answer the following queries, creating a file "xxx_pr2hwk2.P". You are encouraged to reuse earlier predicates when defining later ones.

Note: I will assume that we always want the origin and destination to be different.

(a) train_only(a,b): Find the pairs of stations (a,b) such that one can go from a to b by train but not by plane.

```

train_path(A,B) :- train(A,B).
train_path(A,B) :- train(A,X), train_path(X,B), A\==B.

plane_path(A,B) :- plane(A,B).
plane_path(A,B) :- plane(A,X), plane_path(X,B), A\==B.

train_only(A,B) :- train_path(A,B), \+plane_path(A,B).

```

(b) pure_plane_path(a,b): a pure plane path from a to b is a plane itinerary from a to b such that for all consecutive stops c,d along the way, one cannot go from c to d by train. Find the pairs of stations (a,b) such that there is a pure plane path from a to b.

```

pure_plane_hop(A,B) :- plane(A,B), \+train_path(A,B).

pure_plane_path(A,B) :- pure_plane_hop(A,B).
pure_plane_path(A,B) :- pure_plane_hop(A,X), pure_plane_path(X,B), A\==B.

```

(c) neither_alone(a,b): Find the pairs of stations (a,b) such that b can be reached from a by some combination of plane or train, but not by either train or plane alone.

```

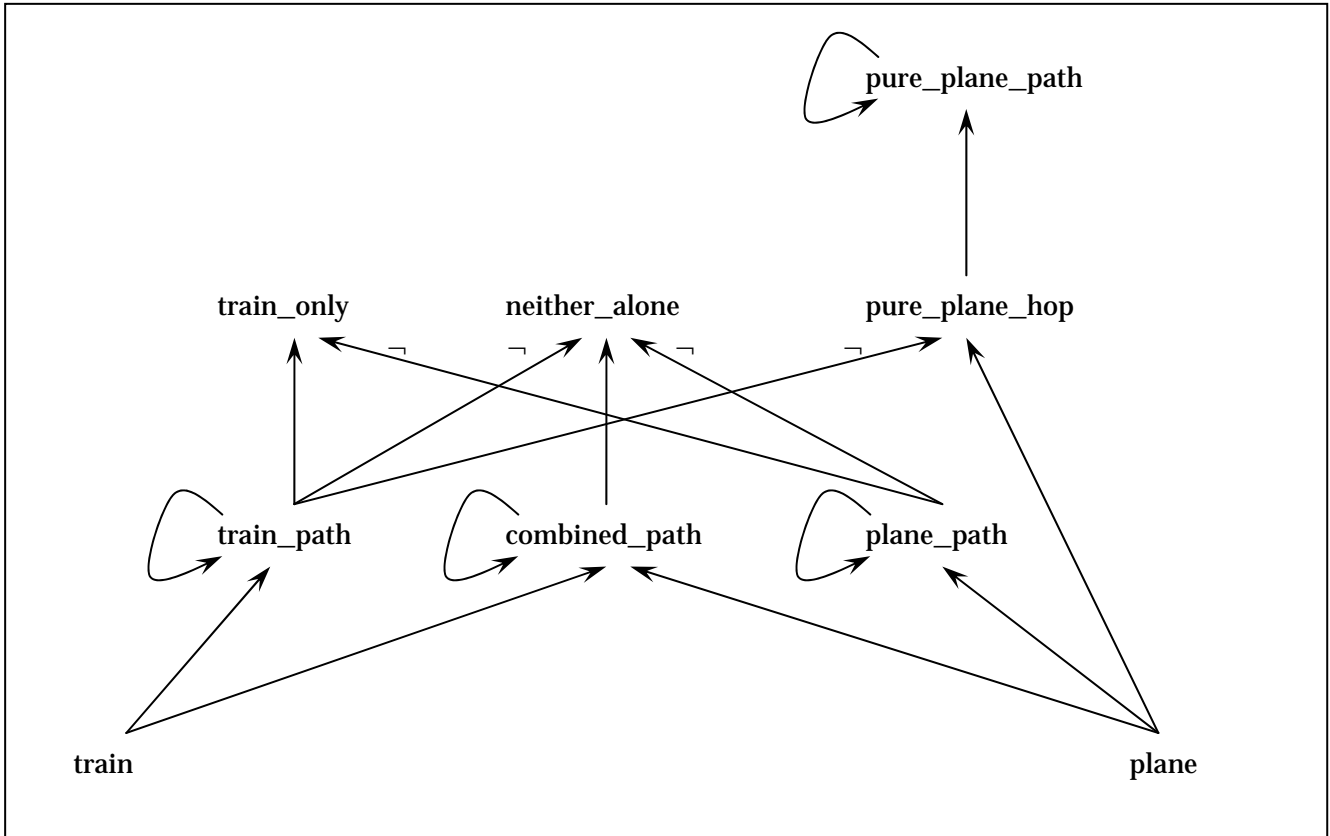
combined_hop(A,B) :- train(A,B).
combined_hop(A,B) :- plane(A,B).

combined_path(A,B) :- combined_hop(A,X).
combined_path(A,B) :- combined_hop(A,X), combined_path(X,B), A\==B.

neither_alone(A,B) :- combined_path(A,B), \+train_path(A,B),
                      \+plane_path(A,B).

```

(iii) Show the predicate dependency graph for part (ii)



Problem 3.**Redo Problem 2(ii) in Recursive SQL (as proposed in the handouts we studied)**

Note: I will assume that we always want the origin and destination to be different.

(a) train_only(a,b): Find the pairs of stations (a,b) such that one can go from a to b by train but not by plane.

```
CREATE VIEW train_path(a,b) AS
  SELECT a, b FROM train
  UNION ALL
  SELECT in.a, out.b FROM train in, train_path out
  WHERE in.b=out.a AND in.a≠out.b
```

```
CREATE VIEW plane_path(a,b) AS
  SELECT a, b FROM plane
  UNION ALL
  SELECT in.a, out.b FROM plane in, plane_path out
  WHERE in.b=out.a AND in.a≠out.b
```

```
CREATE VIEW train_only(a,b) AS
  SELECT a, b FROM train_path
  EXCEPT
  SELECT a, b FROM plane_path
```

(b) pure_plane_path(a,b): a pure plane path from a to b is a plane itinerary from a to b such that for all consecutive stops c,d along the way, one cannot go from c to d by train. Find the pairs of stations (a,b) such that there is a pure plane path from a to b.

```
CREATE VIEW pure_plane_hop(a,b) AS
  SELECT a, b FROM plane
  EXCEPT
  SELECT a, b FROM train_path
```

```
CREATE VIEW pure_plane_path(a,b) AS
  SELECT a, b FROM pure_plane_hop
  UNION ALL
  SELECT in.a, out.b FROM pure_plane_hop in, pure_plane_path out
  WHERE in.b=out.a AND in.a≠out.b
```

(c) neither_alone(a,b): Find the pairs of stations (a,b) such that b can be reached from a by some combination of plane or train, but not by either train or plane alone.

```
CREATE VIEW combined_hop(a,b) AS
  SELECT a, b FROM train
  UNION
  SELECT a, b FROM plane
```

```
CREATE VIEW combined_path(a,b) AS
  SELECT a, b FROM combined_hop
  UNION ALL
  SELECT in.a, out.b FROM combined_hop in, combined_path out
  WHERE in.b=out.a AND in.a≠out.b
```

```
CREATE VIEW neither_alone(a,b) AS
  SELECT a, b FROM combined_path
  EXCEPT
  (SELECT a, b FROM train_path UNION SELECT a, b FROM plane_path)
```

Problem 4.

Model intersection property: Let P be a positive program, and $M1$ and $M2$ be two models for P . Then $M1 \cap M2$ is also a model for P .

Proof:

Assume $M = M1 \cap M2$ is not a model for P . Then M does not satisfy some rule $r \in P$:

$$A \leftarrow A1, A2, \dots, An \quad (n \geq 0)$$

Since the above formula is false, it implies that:

$$A \notin M \text{ and } A1, A2, \dots, An \in M$$

Since $M \subseteq M1$ and $M \subseteq M2$, we have:

$$A1, A2, \dots, An \in M1 \text{ and } A1, A2, \dots, An \in M2$$

Since both $M1$ and $M2$ are models, they satisfy this rule (with $A1, A2, \dots, An$ as the body of the rule), hence there must be an A' in $M1$ and $M2$, which is the head of the rule:

$$A' \leftarrow A1, A2, \dots, An$$

Since A' exists in both $M1$ and $M2$, A' is also in M . Hence M also satisfies the rule:

$$A' \leftarrow A1, A2, \dots, An$$

This is a contradiction to the assumption that M does not satisfy this rule.

Problem 5.

This problem deals with topic covered in the guest lecture by Prof. Shvartsman, on Sep. 16. The topic deals with replicated consistent object implementation (he handed out several copies of a paper and gave web search keywords.)

In the lecture, 2-phase algorithms were given for read and write operations on atomic (aka linearizable) replicated objects. Each phase access a majority (or a quorum) of replicas.

Consider what happens if all agents have a GPS (global positioning system) device that provides global time to the agents (same time for all agents).

(a) Does this allow us to simplify the algorithm for write? If yes, how and why? If not, why not?

Yes. In the original design, the first (query) phase of write obtains timestamps from a majority of processes, and creates a new timestamp with value $V_{s+1} = \langle \max_ts+1, PID \rangle$, where \max_ts is the maximum of the timestamps it received, and PID is its own ID. In the second (propagation) phase, the process writes V_{s+1} back to the majority of replicas. This algorithm ensures that all timestamps are different, globally ordered and respect real-time order.

With the new design of GPS, all processes may simply use the $V_{s+1} = \langle \text{global time}, PID \rangle$, and avoid the first phase of **write**. The reason is because GPS itself is the real-time order, different at each step, and global.

(b) What about the algorithm for read? If yes, how and why? If not, why not?

No. The query phase of the read algorithm obtains both the value and the timestamp. The timestamp is needed from each process in order to determine whether the process' value is the latest one, before we can propagate it to the majority. We therefore cannot simplify the algorithm by avoiding the query phase.