Negation and Datalog

In some cases, mixing recursion and negation leads to ambiguous Datalog programs.

Consider program P

```
Avail(jenny, 2pm).
Teach(I, T) :- Avail(I, T), ¬Advise(I, T).
Advise(I, T) :- Avail(I, T), ¬Teach(I, T).
```

If we start with the one fact and apply I_P, we get M =

```
Avail(jenny, 2p).
Teach(jenny, 2p).
Advise(jenny, 2p).
```


Problem

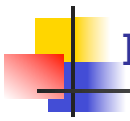
Model M is not minimal. Here are two smaller models

M1 =

```
Avail(jenny, 2p)
Teach(jenny, 2p)
```

M2 =

```
Avail(jenny, 2p).
Advise(jenny, 2p).
```



If We Have Recursion and Negation

1. Intersection of models might not be a model
   M1 ∩ M2 = M3 {`Avail(jenny, 2p).`}
2. Minimal model might not be unique
   See M1 and M2
3. I_P is not necessarily monotone ($I_P(f) \subseteq f$)
   $I_P(M)$ = M3 and $M \not\subset$ M3

Negative Cycles

The problem here is we have a recursive loop that has a negative dependency

`Avail`

`Teach` `Advise`

Not All Negation is Bad

- Recursive programs without negative cycles are relatively well behaved
  - Can define a model that is preferable to others
  - Can compute this model in a way similar to Naïve evaluation
- A program without negative cycles can be stratified.
  - Predicates are partitioned into *strata*
  - If predicate `p` depends negatively on predicate `q`, then `q` is in a lower stratum.

Example

`Flight(A1, A2, C).` There is a flight from airport `A1` to Airport `A2` with cost `C`.

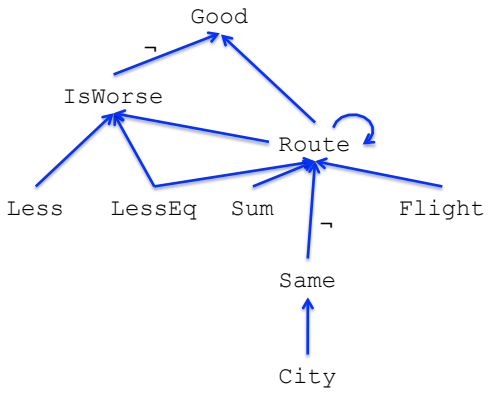
```
Route(A1, A2, C, 1) :- Flight(A1, A2, C).
Route(A1, A2, C, S) :- Flight(A1, A3, C1),
   Route(A3, A2, C2, S2),  Sum(C1, C2, C),
   Sum(S2, 1, S), ¬Same(A1, A2), LessEq(S, 5).

Same(C, C) :- City(C).

IsWorse(A1, A2, C, S) :- Route(A1, A2, C, S),
   Route(A1, A2, C1, S1), Less(C1, C), LessEq(S1, S).
IsWorse(A1, A2, C, S) :- Route(A1, A2, C, S),
   Route(A1, A2, C1, S1), LessEq(C1, C), Less(S1, S).

Good(A1, A2, C, S) :- Route(A1, A2, C, S),
   ¬IsWorse(A1, A2, C, S).
```

One Stratification for the Example



More Precisely

Given Datalog program P, a stratification of P is an assignment of an integer *stratum* to each predicate `p` in P such that

- If there is a clause
  `h(…) :- …, q(…), ….`
  then stratum(`h`) ≥ stratum(`q`)
- If there is a clause
  `h(…) :- …, ¬q(…), ….`
  then stratum(`h`) > stratum(`q`)

P is *stratifiable* if P has a stratification.

Stratification Partitions the Rules

Can think of a stratification dividing the clauses in P into

$P_1, P_2, \ldots, P_k$

where

```
h(…) :- L1, L2,…, Lm.
```

is in P_i if stratum(`h`) = i.

Stratified Evaluation

Stratified($P_1, P_2, \ldots, P_k$, g)

M = ∅

for i = 1 .. k

M = Naïve($P_i \cup$ M)

return match(M, g)

Example

Consider the following facts for our program

```
City(pdx).      Flight(sea, sfo, 500).
City(sfo).      Flight(sea, pdx, 200).
City(sea).      Flight(pdx, sfo, 180).
City(eug).      Flight(sea, eug, 280).
                Flight(eug, sfo, 250).
```


Stratum 1

Will do Naïve on `City`, `Same`.

```
City(pdx).
City(sfo).
City(sea).
City(eug).

Same(pdx, pdx).
Same(sfo, sfo).
Same(sea, sea).
Same(eug, eug).
```

Stratum 2 (additional facts)

Add `Flight`, `Route`, `IsWorse` and computable

```
Flight(sea, sfo, 500). IsWorse(sea, sfo, 530, 2).
Flight(sea, pdx, 200).
Flight(pdx, sfo, 180).
Flight(sea, eug, 280).
Flight(eug, sfo, 250).
Route(sea, sfo, 500, 1).
Route(sea, pdx, 200, 1).
Route(pdx, sfo, 180, 1).
Route(sea, eug, 280, 1).
Route(eug, sfo, 250, 1).
Route(sea, sfo, 380, 2).
Route(sea, sfo, 530, 2).
```


Stratum 3 (additional facts)

Add `Good`

```
Good(sea, sfo, 500, 1).
Good(sea, pdx, 200, 1).
Good(pdx, sfo, 180, 1).
Good(sea, eug, 280, 1).
Good(eug, sfo, 250, 1).
Good(sea, sfo, 380, 2).
```

What if We Expand Stratum 1?

```
City(pdx).
City(sfo).
City(sea).
City(eug).

Same(pdx, pdx).
Same(sfo, sfo).
Same(sea, sea).
Same(eug, eug).
Same(sea, sfo).
```


Stratum 2 Gets Smaller

```
Flight(sea, sfo, 500). ~~IsWorse(sea, sfo, 530, 2).~~
Flight(sea, pdx, 200).
Flight(pdx, sfo, 180).
Flight(sea, eug, 280).
Flight(eug, sfo, 250).
Route(sea, sfo, 500, 1).
Route(sea, pdx, 200, 1).
Route(pdx, sfo, 180, 1).
Route(sea, eug, 280, 1).
Route(eug, sfo, 250, 1).
~~Route(sea, sfo, 380, 2).~~
~~Route(sea, sfo, 530, 2).~~
```

As Does Stratum 3

Add `Good`

```
Good(sea, sfo, 500, 1).
Good(sea, pdx, 200, 1).
Good(pdx, sfo, 180, 1).
Good(sea, eug, 280, 1).
Good(eug, sfo, 250, 1).
~~Good(sea, sfo, 380, 2).~~
```

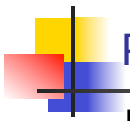
Both these models are minimal, but intuitively, we prefer the one that makes fewer things true at the lower stratum

Stratify the Model

First of these models is the *perfect* model.

Assume a model M for P = $P_1, P_2, \ldots, P_k$
is stratified the same way
$M_1, M_2, \ldots, M_k$

That is, M_i contains the facts for predicates defined in P_i.



Perfect Model

M = M_1, M_2, …, M_k is a *perfect model* for P if for any other model N = N_1, N_2, …, N_k

1. $M_1 \subseteq N_1$
2. If $M_1 = N_1$, $M_2 = N_2$, …, $M_{i-1} = N_{i-1}$ then $M_i \subseteq N_i$

M is minimal at each stratum relative to the strata below.

For a stratifable program P, there is a unique perfect model and that model is minimal.