Database Theory
Winter Term 2013/14
Prof. Dr. W. May

5. Unit: Well-founded and Stable Semantics

Discussion by 5./7.2.2014

Exercise 1 (Well-Founded Model) a) Show that there are non-stratifiable Datalog$^\neg$ programs that have a total well-founded model (i.e., no atoms undefined).

b) Are there non-stratifiable Datalog$^\neg$ programs that have a total well-founded model for *all* EDB instances?

---

a) Take a simple win-move game that has only won and lost positions, no drawn ones:

```
pos(a). pos(b). pos(c).
move(a,b).
move(b,c).
win(X) :-  move(X,Y), not win(Y).
```

The well-founded model is

({pos(a). pos(b). pos(c). move(a,b). move(b,c). win(b)},
{move(a,a). move(a,c). move(b,a). move(b,a). move(c,a). move(c,b). move(c,c). win(a). win(c).})

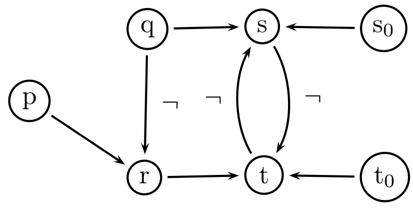
b) Consider EDB relations $p/1$, $q/1$, $s_0/1$, $t_0/1$. The program P is as follows:

```
r(x) :- p(x), not q(x).
s(x) :- s0(x).
s(x) :- q(x), not t(x).
t(x) :- t0(x).
t(x) :- r(x), not s(x).
```

Sketch: The program describes a partition that is based on splitting p into q vs. r. $p \wedge q$ is one side, $p \wedge \neg q$ the other.

Based on this, relations s vs. t are defined (which are not necessarily disjoint): By "default", elements of q belong to s, while elements of r belong to t. The membership of elements can be influenced by s_0 and t_0 that "overwrites" the above defaults, which is encoded into the $q \to s$ and $r \to t$ rules that create a negative cyclic dependency. (Note that elements a can be assigned to be both in s and t via $s_0(a)$ and $t_0(a)$).

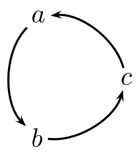
The dependency graph is



For each EDB instance that defines $I(p)$, $I(q)$, $I(s_0)$, $I(t_0)$, the well-founded model is total.

Exercise 2 (Well-Founded Model) Give an instance of the win-move game that has no total stable model.

Cycle with three positions:



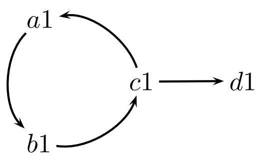

```
win(X) :-  move(X,Y), not win(Y).
lose(X) :- pos(X), not win(X).
pos(a).
pos(b).
pos(c).
move(a,b).
move(b,c).
move(c,a).
% lparse -n 0 -d none --partial winmovenontotal1.s |smodels
```

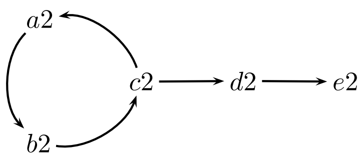
[Filename: winmovenontotal1.s]

The only stable model is M with

$$val_M(win(a)) = val_M(win(b)) = val_M(win(b)) = u,$$
$$val_M(lose(a)) = val_M(lose(b)) = val_M(lose(b)) = u.$$

In general: any cycle with an odd number of positions, and where no position is lost due to an exit from the cycle.






```
win(X) :-  move(X,Y), not win(Y).
lose(X) :- pos(X), not win(X).
pos(a1).
pos(b1).
pos(c1).
pos(d1).
move(a1,b1).
move(b1,c1).
move(c1,a1).
move(c1,d1).

pos(a2).
pos(b2).
pos(c2).
pos(d2).
pos(e2).
move(a2,b2).
```

```
move(b2,c2).
move(c2,a2).
move(c2,d2).
move(d2,e2).
% lparse -n 0 -d none --partial winmovenontotal2.s |smodels
```

[Filename: winmovenontotal2.s]

In the “1” game, the exit makes d_1 a losing position and thus c_1 is a winning position (move to d_1). Thus, b_1 is lost and a_1 is won.

In the “2” game, the exit makes e_1 lost and d_1 won, but c_1 is not lost, since he player will move to a_1 and stay in the cycle.