Database Theory
Winter Term 2016/17
Prof. Dr. W. May

3. Unit: Well-founded and Stable Semantics

Discussion by 4./6.2.2014

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-ground) 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).})

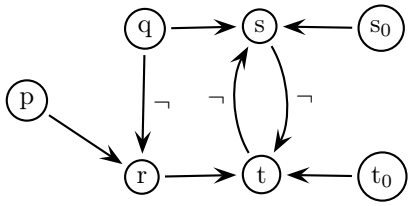
b) Consider EDB relations $p/1$, $q/1$, $s_0/1$, $t_0/1$. The program P is as follows:

```
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 $t_0(a)$).

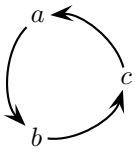
The dependency graph is



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:



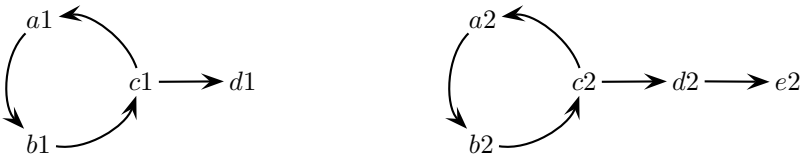

```
win(X) :-  move(X,Y), not win(Y).
lose(X) :- pos(X), not win(X).
pos(a).
pos(b).
pos(c).
move(a,b).
move(b,c).
move(c,a).
% lparse -n 0 -d none --partial winmovenontotal1.s |smodels
```

[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]

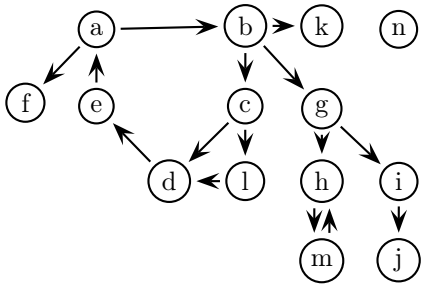
In the "1" game, the exit makes d_1 a losing position and thus c_1 is a winning position (move to d_1). Thus, b_1 is lost and a_1 is won.

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.

Note: a minimal such example ist a win-move game with only a single node and move:

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

Exercise 3 (Well-Founded Model) Consider again the win-move game from the lecture:



Consider to start the Alternating Fixpoint Computation for the rules `win(X) :- move(X,Y), not win(Y).`
`lose(X) :- pos(X), not win(X).`

with $\mathcal{H}_0$ as

- some atoms that are correct: lose(k), win(b), win(d)
- some atoms that actually are in contrast to the well-founded model of the above game: win(f), lose(c), win(m).

(it is often called "seed" when starting an iterative algorithm with some initial values)

- Note: lose is actually not used in any rule body; interpreting lose as $\neg$ win does also not contain any information for the seed, since $\neg$ win is assumed to hold by default.
  $\rightarrow$ in general, any negative seed does not have any consequence.
- So the seed is mainly an underestimate, and only the positive atoms add some flavour of overestimate.

The first reduct, $P_{\mathcal{H}_0}$ contains the following rules (all rules vor pairs x, y s.t. there is no move from x to y are omitted):

$win(a) :- move(a,f), \neg win(f).$
$win(a) :- move(a,b), \neg win(b).$
$win(b) :- move(b,c), \neg win(c).$
$win(b) :- move(b,k), \neg win(k).$
$win(c) :- move(c,d), \neg win(d).$
$win(c) :- move(c,l), \neg win(l).$
$win(l) :- move(l,d), \neg win(d).$
$win(d) :- move(d,e), \neg win(e).$
$win(e) :- move(e,a), \neg win(a).$
$win(i) :- move(i,j), \neg win(j).$
$win(g) :- move(g,i), \neg win(i).$
$win(b) :- move(b,g), \neg win(g).$
$win(g) :- move(g,h), \neg win(h).$
$win(h) :- move(h,m), \neg win(m).$
$win(m) :- move(m,h), \neg win(h).$

$lose(a) :- \neg win(a)$ *true*.
$lose(b) :- \neg win(b).$
$lose(c) :- \neg win(c)$ *true*.
$lose(d) :- \neg win(d).$
$lose(e) :- \neg win(e)$ *true*.
$lose(f) :- \neg win(f).$
$lose(g) :- \neg win(g)$ *true*.
$lose(h) :- \neg win(h)$ *true*.
$lose(i) :- \neg win(i)$ *true*.
$lose(j) :- \neg win(j)$ *true*.
$lose(k) :- \neg win(k)$ *true*.
$lose(l) :- \neg win(l)$ *true*.
$lose(m) :- \neg win(m).$
$lose(n) :- \neg win(n)$ *true*.

Thus, $\mathcal{H}_1 = T^{\omega}_{P_0}(\emptyset) = \{\ win(b), win(c), win(d), win(e), win(g), win(i), win(m),$
$lose(a), lose(c), lose(e), lose(g), lose(h), lose(i), lose(j), lose(k), lose(l), lose(n)\}$

Note: Usually, $\mathcal{H}_1$ would be an overestimate, but win(a) is (still) missing, since f was wrongly set to be won. Note that now, f is not re-derived to be won (i.e. it is already clear that is is not supported), but also lose(f) is not yet there.
win(h) is not there, since m was fixed to be won (both are drawn in the original game).

The second reduct, $P_{\mathcal{H}_1}$ contains the following rules (again, rules with no underlying move are omitted):

$win(a) :- move(a,f), \neg win(f).$
$win(a) :- move(a,b), \neg win(b).$
$win(b) :- move(b,c), \neg win(c).$
$win(b) :- move(b,k), \neg win(k).$
$win(c) :- move(c,d), \neg win(d).$
$win(c) :- move(c,l), \neg win(l).$
$win(l) :- move(l,d), \neg win(d).$
$win(d) :- move(d,e), \neg win(e).$
$win(e) :- move(e,a), \neg win(a).$
$win(i) :- move(i,j), \neg win(j).$
$win(g) :- move(g,i), \neg win(i).$
$win(b) :- move(b,g), \neg win(g).$
$win(g) :- move(g,h), \neg win(h).$
$win(h) :- move(h,m), \neg win(m).$
$win(m) :- move(m,h), \neg win(h).$

$lose(a) :- \neg win(a)$ *true*.
$lose(b) :- \neg win(b).$
$lose(c) :- \neg win(c).$
$lose(d) :- \neg win(d).$
$lose(e) :- \neg win(e).$
$lose(f) :- \neg win(f)$ *true*.
$lose(g) :- \neg win(g).$
$lose(h) :- \neg win(h)$ *true*.
$lose(i) :- \neg win(i).$
$lose(j) :- \neg win(j)$ *true*.
$lose(k) :- \neg win(k)$ *true*.
$lose(l) :- \neg win(l)$ *true*.
$lose(m) :- \neg win(m).x$
$lose(n) :- \neg win(n)$ *true*.

Thus, $\mathcal{H}_2 = T^{\omega}_{P_{\mathcal{H}_1}}(\emptyset) = \{\ win(a), win(b), win(c), win(e), win(g), win(i), win(m),$
$lose(a), lose(f), lose(h), lose(j), lose(k), lose(l), lose(n)\}$

Note: Usually, $\mathcal{H}_2$ would be an underestimate. win(a) and lose(f) are now there, thus, the f issue is already corrected. But, win(e) is there, which is a consequence of having lose(a) before ... the f issue propagates through the game now.
Note that win(d) is not there, although it was there in the seed (an is finally true), but not yet supported.
win(c) is there, lose(c) is not there, so the c issue is also corrected (if there would be a longer chain between c and b, it would propagate through it until finally dying).
lose(h) and win(m) are both there (both are drawn in the original game, but win(m) was fixed in the seed).

The third reduct, $P_{\mathcal{H}_2}$ contains the following rules (again, rules with no underlying move are omitted):

$win(a) :- move(a,f), \sout{\neg win(f)}.$
$\sout{win(a) :- move(a,b), \neg win(b).}$
$\sout{win(b) :- move(b,c), \neg win(c).}$
$win(b) :- move(b,k), \sout{\neg win(k)}.$
$win(c) :- move(c,d), \sout{\neg win(d)}.$
$win(c) :- move(c,l), \sout{\neg win(l)}.$
$win(l) :- move(l,d), \sout{\neg win(d)}.$
$\sout{win(d) :- move(d,e), \neg win(e).}$
$\sout{win(e) :- move(e,a), \neg win(a).}$
$win(i) :- move(i,j), \sout{\neg win(j)}.$
$\sout{win(g) :- move(g,j), \neg win(j).}$
$\sout{win(b) :- move(b,g), \neg win(g).}$
$win(g) :- move(g,h), \sout{\neg win(h)}.$
$\sout{win(h) :- move(h,m), \neg win(m).}$
$win(m) :- move(m,h), \sout{\neg win(h)}.$

$\sout{lose(a) :- \neg win(a).}$
$\sout{lose(b) :- \neg win(b).}$
$\sout{lose(c) :- \neg win(c).}$
$lose(d) :- \sout{\neg win(d)}\ true.$
$\sout{lose(e) :- \neg win(e).}$
$lose(f) :- \sout{\neg win(f)}\ true.$
$\sout{lose(g) :- \neg win(g).}$
$lose(h) :- \sout{\neg win(h)}\ true.$
$\sout{lose(i) :- \neg win(i).}$
$lose(j) :- \sout{\neg win(j)}\ true.$
$lose(k) :- \sout{\neg win(k)}\ true.$
$lose(l) :- \sout{\neg win(l)\ true}.$
$\sout{lose(m) :- \neg win(m).}$
$lose(n) :- \sout{\neg win(n)\ true}.$

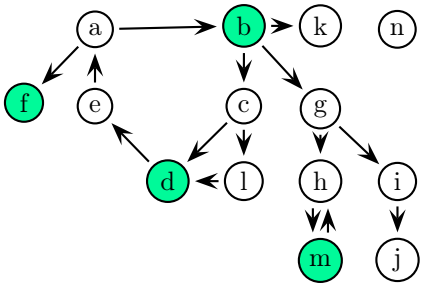
Thus, $\mathcal{H}_3 = T^{\omega}_{P_{\mathcal{H}_2}}(\emptyset) = \{\ win(a), win(b), win(c), win(g), win(i), win(l), win(m),$
$lose(d), lose(f), lose(h), lose(j), lose(k), lose(l), lose(n)\}$

$\mathcal{H}_3$ would be an overestimate, but currently contains neither win(e) nor lose(e). This is still a consequence of the f issue.
On the other hand, the sequence shows that g, h, i, j, k, l, m and n do not change any more. One can also know that a (win because of f), b (win because of k) and f (lose) will not change any more.

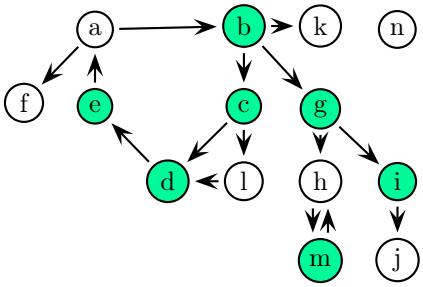
The rest follows and corrects then from the usual well-founded characteristics of the AFP computation.

The process can also be visualized by the original game (Green= win, recall that the lose predicate has been introduced only for demonstrational issues):
("win: can move to a non-win node")

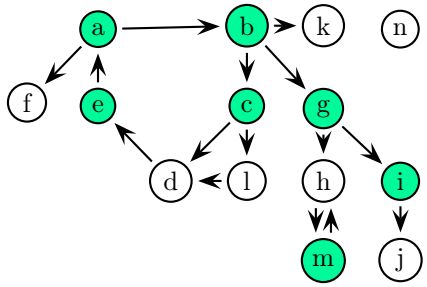
$\mathcal{H}_0$:



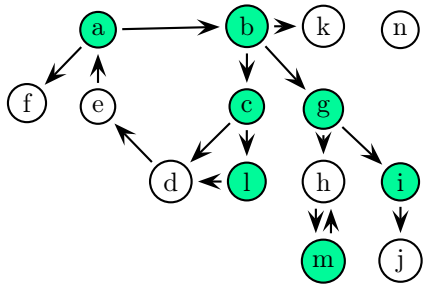
$\mathcal{H}_1$:



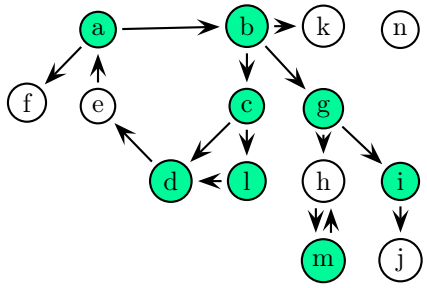
$\mathcal{H}_2$:



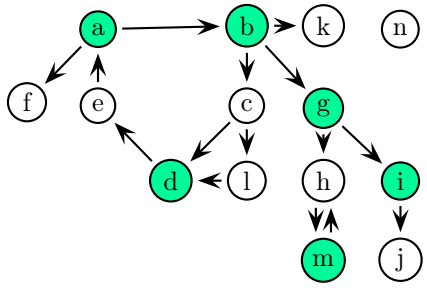
$\mathcal{H}_3$:



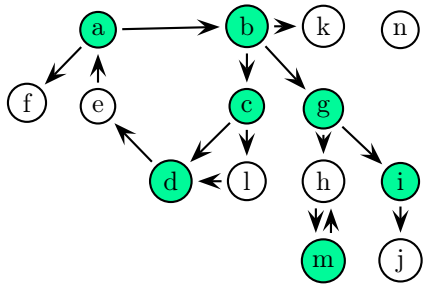
$\mathcal{H}_4$:



$\mathcal{H}_5$:



$\mathcal{H}_6$:



$\mathcal{H}_7 = \mathcal{H}_6$, is one of the two total stable models, namely the one which makes the seeded $\mathsf{win}(\mathsf{m})$ true.

Conclusions:

- "Seeding" correct positive atoms does not help much if the seed does not contain their support (recall: checking stability of the well-founded model is actually a full seed and reproduces in a single step!).

- "Seeding" wrong positive atoms will remove them since they are not supported.
- "Seeding" positive atoms where the well-founded model is undefined can result in a stable model (if the support for there stability is also contained in the seed).
- "Seeding" positive atoms in general will not result in a fixpoint (seeding into a 3-cycle, where no total stable model exists; or only one win-position into a four-cycle, which is not sufficient to reproduce).

  Note that it does in general *not* extend the interpretation to a stable model. Nevertheless, this can be used for strategies to find stable models.