Problem Set for Tutorial 6 — TDDD08

1. Given the program below, draw an SLDNF-forest for query $\neg member(X, [a, b])$ and for $\neg member(c, [a, b])$. What result would you expect in Prolog with the queries \+member$(X, [a, b])$ and \+member$(c, [a, b])$? If the results differ, why is that the case?

```
member(X, [X|_]).
member(X, [_|L]) :- member(X,L).
```

   SOLUTION. Discussed at the tutorial.

2. A ground propositional formula can be represented in Prolog by a term over `true`, `false`, `and/2`, `or/2`, and `not/1`. For example, `and(true, or(not(false), true))` would be an example of such a formula in this representation. Assume that we want to define a predicate `true_formula(F)` which is true if `F` is a true ground propositional formula in the specified representation. An attempt using negation as failure in Prolog might look as follows:

```
true_formula(true).
true_formula(and(X, Y)) :- true_formula(X), true_formula(Y).
true_formula(or(X, _Y)) :- true_formula(X).
true_formula(or(_X, Y)) :- true_formula(Y).
true_formula(not(X)) :- \+ true_formula(X).
```

   (a) Give an example of two queries, one ground and one non-ground, which produces unexpected or unintended answers.

   SOLUTION. For instance, query `true_formula(not(T))` produces answer `no`. This says that there is no true formula of the form `not`(t), which is wrong. Similarly, `true_formula(not(not(T)))` results in `yes`, while it should not. Another kind of queries with unintended success is `true_formula(not(a))` or `true_formula(or(true,a))`, where the argument is not a propositional formula.

   (b) Rewrite the above program without using negation. What is the result of the previous two queries to the resulting program?

   SOLUTION. We may introduce a predicate `false_formula` describing those propositional formulae that are false. In the new program, `false_formula` and `true_formula` are defined similarly as above. However `not`/1 is treated by a clause `true_formula(not(X)):-false_formula(X)`, and by a similar clause for `false_formula`/1.

   A clause like `true_formula(or(X,_Y)):-true_formula(X)` does not assure that the argument is a propositional formula. So we should impose a condition on `_Y`, and transform the clause into `true_formula(or(X,Y)):-true_formula(X),formula(Y)`. An auxiliary predicate `formula`/1 describes all the considered formulae, and can be defined by two clauses `formula(X):-true_formula(X)` and `formula(X):-false_formula(X)`. Such additional condition is needed in the clauses with heads `true_formula(or(...))` and for `false_formula(and(...))`.

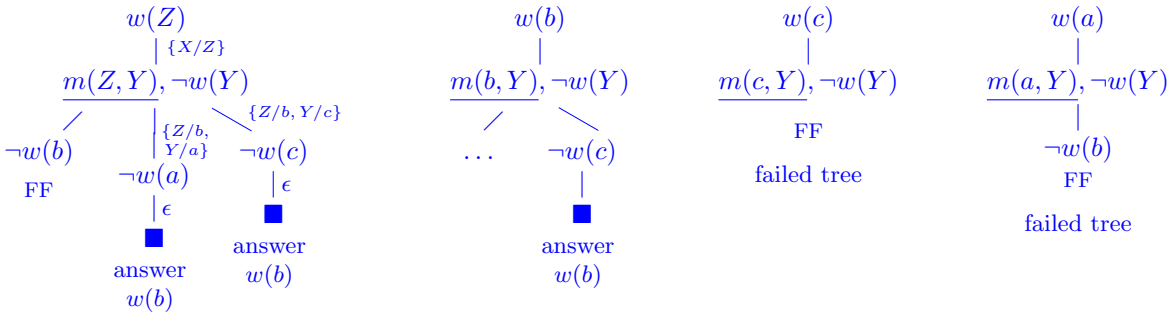
3. Consider a program WIN:

$$w(X) \leftarrow m(X,Y), \neg w(Y). \quad m(a,b). \quad m(b,a). \quad m(b,c).$$

(Predicate $m/2$ describes moves in a game; $w/1$ describes winning positions – a position X wins if you can move from X to a position Y in which your opponent cannot win.)

(a) Draw the SLDNF-forest for query $w(X)$ under the Prolog selection rule. Make it clear which branches are successful derivations and what their answers are, which leaves are floundered, and which trees are finitely failed.

SOLUTION. The forest:



We do not need to construct a whole SLDNF-tree, if the tree contains a most general answer. What matters is its successful branch. (Note that in such cases the definition of SLDNF-resolution refers to the successful SLDNF-derivation only.) So for $w(b)$ we showed here such branch only. The substitutions (mgu's) of the resolution steps are shown only where they contribute to an answer.

(b) How is the forest changed if we add $m(c,d)$ to the program?

SOLUTION. A failed tree for $w(d)$ and a successful derivation for $w(c)$ (similar to those for $w(c)$ and $w(b)$ above). A tree for $w(b)$ with a failed leaf $\neg w(c)$ and a leaf $\neg w(a)$; and a tree for $w(a)$ as in the slides (with a single leaf $\neg w(b)$). Due to the circular reference between the two trees, leaves $\neg w(a)$, $\neg w(b)$ are floundered (Prolog computation would fall into infinite recursion). The tree for $w(Z)$ of the forest consists of the 5 non-empty nodes of the tree above; however the leaves $\neg w(a)$, $\neg w(b)$ are floundered, and $\neg w(c)$ is failed,

So the trees for $w(Z)$, $w(a)$ and $w(b)$ are not failed, and do not contain any answers (thus SLDNF-resolution shows neither falsity, nor truth of these atoms).

(c) Construct the completion *comp*(WIN) of the program (except for equality axioms). Explain whether $\neg m(c,a)$ and $\neg w(c)$ are logical consequences of *comp*(WIN). The same for $w(b)$.

SOLUTION. *comp*(WIN) consists of

$$w(X) \leftrightarrow \exists Y(m(X,Y), \neg w(Y)) \tag{1}$$
$$m(X,Y) \leftrightarrow X = a, Y = b \vee X = b, Y = a \vee X = b, Y = c \tag{2}$$

and the equality axioms CET (whose main role is to imply that distinct ground terms are not equal).

Consider $\neg m(c,a)$. Replacing X, Y by c, a in (2) results (after an obvious simplification) in $m(c,a) \leftrightarrow \mathit{false}$, which is equivalent to $\neg m(c,a)$. So *comp*(WIN) $\models \neg m(c,a)$.

For future usage notice that in the same way we obtain that *comp*(WIN) $\models \neg m(c,Y)$ (equivalently, *comp*(WIN) $\models \forall Y \neg m(c,Y)$), hence *comp*(WIN) $\models \forall Y(\neg m(c,Y) \vee \alpha)$, for any α.

Consider $\neg w(c)$. From (1) we have $\neg w(c) \leftrightarrow \forall Y(\neg m(c,Y) \vee w(Y))$. As the right hand side of the equivalence is a logical consequence of *comp*(WIN), so is $\neg w(c)$.

4. (Exam exercise) Consider the following general program P:

$$p(X) \leftarrow \neg q(X). \qquad q(s(Y)) \leftarrow p(Y). \qquad q(a).$$

(a) Draw SLDNF-forests for queries $q(X)$, $p(s(b))$, and $p(s(s(b)))$. Make it clear which trees are finitely failed, which leaves are floundered, which branches are successful derivations, and what their answers are.

(b) Construct the completion $comp(P)$ of the program (except for equality axioms). Explain whether $q(a)$ is a logical consequence of $comp(P)$; the same for $q(s(b))$.

Instead of providing a solution here, we refer to a more instructive problem 2 in the note linked as "example problems with solutions" at our "Exams" web page.

5. Consider programs

$$P_1\colon\ p \leftarrow \neg p \qquad P_2\colon\ \begin{array}{l} p \leftarrow \neg q \\ q \leftarrow \neg p \end{array} \qquad P_3\colon\ \begin{array}{l} p \leftarrow \neg q \\ q \leftarrow \neg r \\ r \leftarrow \neg p \end{array}$$

(a) Is an Herbrand interpretation $\{p\}$ a model of P_2? What about P_3?

SOLUTION. (Notice that this problem is not related to negation in logic programming. Remember that an implication is true if its premise is false, or its consequent is true.)

$\{p\}$ is a model of P_2 (check this). It is not a model of P_3 (because $\neg r$ is true in $\{p\}$, but q is not true in $\{p\}$, so $\{p\}$ is not a model of the second clause of P_3).

(b) Which of the programs have stable models? Find all of them.

DETAILED SOLUTION. Let us look more closely at P_3. Assume that I is a stable model of P_3. So it is the least Herbrand model of the reduct P_3^I, i.e. $I = \mathbf{M}_{P_3^I}$.

Assume that $p \in I$. Then the reduct P_3^I does not contain any rule $r \leftarrow \ldots$. Thus $\mathbf{M}_{P_3^I}$ does not contain r. Hence the reduct contains rule q. So $\mathbf{M}_{P_3^I}$ contains q, hence the reduct does not contain any rule $p \leftarrow \ldots$, and $\mathbf{M}_{P_3^I}$ does not contain p – contradiction. Assuming that $p \notin I$ leads, similarly, to $r \in \mathbf{M}_{P_3^I}$, $q \notin \mathbf{M}_{P_3^I}$, and $p \in \mathbf{M}_{P_3^I}$, contradiction.

In the same way we obtain that P_1 does not have a stable model.

Consider P_2. Similarly as above, if a stable model contains p then it does not contain q. Analogically for q. So we have two candidates for stable models, $\{p\}$ and $\{q\}$. The reduct $P_2^{\{p\}}$ is $\{p\}$, and $P_2^{\{q\}} = \{q\}$ (and their least Herbrand models are obviously $\mathbf{M}_{P_2^{\{p\}}} = \{p\}$ and $\mathbf{M}_{P_2^{\{q\}}} = \{q\}$). Thus the stable models of P_2 are $\{p\}$ and $\{q\}$.

6. Find the stable models of programs

$$P_4\colon\ \begin{array}{l} p \leftarrow p. \\ p \leftarrow r. \\ q \leftarrow \neg p. \\ s \leftarrow \neg q, r \end{array} \qquad P_5\colon\ \begin{array}{l} p \leftarrow \neg q. \\ q \leftarrow \neg p. \\ r \leftarrow q, \neg p. \\ f \leftarrow \neg f, p \end{array}$$

Comment: Note that P_4 is stratified ("no recursion through negation"). Hence it has a single stable model, which can be constructed stepwise for consecutive strata (1. the clauses for p, 2. that for q, 3. that for s).

DETAILED SOLUTION. Consider P_4. The least Herbrand model of the first stratum is empty (p, r are false). For the second stratum, q is true (as p is false). For the last stratum we have that s is false (as r is false, or as $\neg q$ is false). So the stable model is $\{q\}$.

Consider P_5 and assume that I is its stable model. We will write "true" instead of "true in I". Similarly as above, from the first two clauses we obtain that exactly one of p, q is true; soon we see which of them. f cannot be true (as this makes the reduct not having any clause $f \leftarrow \ldots$, and thus $f \notin I$). So the body $\neg f, p$ must be false (otherwise $f \in I$, contradiction). As $\neg f$ is true, p must be false. Hence q is true. By the third rule r is true.
Thus $I = \{q, r\}$, this is the only candidate for a stable model. The reduct P_5^I is $\{q.\ \ r \leftarrow q.\ \ f \leftarrow p.\}$, and its least Herbrand model is I. So I is the only stable model for P_5.

7. Assume a small Herbrand universe (with > 1 elements) and find the stable models of

$$\begin{array}{rl} P_6\colon & p(a) \leftarrow \neg q. \\ & q \leftarrow \neg p(a). \\ & p(a) \leftarrow p(X). \\ & p(X) \leftarrow p(a). \end{array}$$

DETAILED SOLUTION. Recall that a non-ground logic program, in the context of the stable model semantics, is usually treated as a shorthand for the grounded program obtained by replacing a clause with all its instances. Thus, if we choose the Herbrand universe $\{a, b\}$ we, after removing duplicate clauses, obtain the program:

$$\begin{array}{rl} \text{ground}(P)\colon & p(a) \leftarrow \neg q. \\ & q \leftarrow \neg p(a). \\ & p(a) \leftarrow p(a). \\ & p(a) \leftarrow p(b). \\ & p(b) \leftarrow p(a). \end{array}$$

Thus, the task is then to determine which subsets of the Herbrand base $\{p(a), p(b), q\}$ that are stable models. Since trying all possibilities is time-consuming, we begin by making a few obseravtions. First if $p(b)$ is true, then $p(a)$ must be true, too, and vice versa. Thus any interpretation $I \subseteq \{p(a), p(b), q\}$ which includes $p(a)$ must also include $p(b)$, and vice versa. Second, any interpretation which does *not* include q needs to include $p(a)$, and hence also $p(b)$, due to the clause $p(a) \leftarrow \neg q$. With the help of these observations it is then not difficult to see that the only stable models are $\{q\}$ and $\{p(a), p(b)\}$.

8. Consider a program P containing a clause

$$f \leftarrow \neg f, \vec{L}$$

where $\vec{L}$ is a conjunction of literals, and f does occur elsewhere in P. Show that f and $\vec{L}$ are false in each stable model of P.

This is a usual way of forcing something to be false, a special notation $\texttt{:-}\,\vec{L}$ is introduced for such a clause.

DETAILED SOLUTION. First, assume that M is a stable model of P where $f \in M$ (thus, f is true in M). Consider the reduct of P with respect to M, and in particular the impact of the clause $f \leftarrow \neg f, \vec{L}$. Since $f \in M$, $\neg f$ is false, and the entire claused will be removed in the reduct. But since f does not appear in any other clause it follows that $M \setminus \{f\}$ satisfies P, too, meaning that M cannot be a stable model.

Second, assume that M is a stable model of P where $\vec{L}$ are true. Again, consider the reduct of P with respect to M, and in particular the impact of the clause $f \leftarrow \neg f, \vec{L}$. Since $f \notin M$ (by the argument above), and since $\vec{L}$ are true in M, it follows that the reduced clause is precisely f. But then M cannot be a model of the reduced program since it does not satisfy f.