Homework assignment 2 – Symbolic Systems I – UvA, June 2020

Hannah

Question 1

For a set Φ consisting of propositional logic formulas and a propositional logic formula ϕ, we can prove that ϕ is logically entailed by Φ if and only if the sentence

$$\Phi \wedge \neg\phi$$

is unsatisfiable. This is called proof by refutation or proof by contradiction. (Artificial Intelligence: A Modern Approach, Russel and Norvig, 2016. Chapter 7, p250).

To solve this the new formula needs to be converted to conjunctive normal form (CNF). This can be done according to the rules explained in section 7.5 from the book Artificial Intelligence: A Modern Approach by Russel and Norvig. So that the formula will contain conjunctions of clauses which are disjunctions of literals, which is similar to the following example:

$$\begin{aligned} formula = \quad & (x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee \neg x_1 \vee x_4) \wedge (x_3 \vee \neg x_4 \vee x_5) \wedge \\ & (x_2 \vee x_3 \vee \neg x_4) \wedge (x_3 \vee \neg x_4 \vee \neg x_5) \wedge (x_4 \vee \neg x_5 \vee x_6) \end{aligned}$$

Now formula in CNF can be given to the SAT solver, which we may assume is an efficient solver. If a solution to this problem is found, this means that $\Phi \nvDash \phi$ as there is at least one answer α that is true for Φ but not for ϕ. If we cannot find a solution to this problem, these would mean $\Phi \vDash \phi$.

Question 2

This question can be solved in a similar way as is done for question 1 by showing that $\Phi \vDash \phi$ if and only if the sentence

$$\Phi \wedge \neg\phi$$

is unsatisfiable. For answer set programming this means that if there is no answer set for this problem, it may be concluded that $\Phi \vDash \phi$ as ϕ is satisfied by every answer set of Φ. Otherwise, if an answer set is found $\Phi \nvDash \phi$.

As is done in the previous question, the formula is converted to CNF. To translate this problem to an ASP problem, the formula needs to be translated to answer set syntax. In answer syntax all clauses will be set to cardinality rules of which at least one element should be true which is similar to a disjunction. This is done in the following way for every clause:

```
1{a_1, ..., a_m, -a_m+1, ..., -a_n}.
```

If these ASP program returns an empty answer set, it can be concluded that ϕ is logically entailed by Φ. If an answer set is found, ϕ is not logically entailed by Φ.

Question 3

The PDFN default theory,

```
W = { p, q }
D = {
  q : r / r,
  p & r : -s & q / -s & q
}
```

results in the only extension,

```
E = {p,q,r,-s}
```

Converting this PDFN default theory to answer set program P results in:

```
p.
q.
r:- q, not -r.
-s :- p, r, not s, not -q.
q :- p, r, not s, not -q.
```

Let I be:

```
I = {p,q,r,-s}
```

Then P^I is

```
p.
q.
r:- q.
-s :- p, r.
q :- p, r.
```

As I is the minimal model of P^I, I is an answer set of P.

By translating the PDFN default theory (W,D) to an ASP problem and checking if there is an answer set that contains the propositional literal l, we can conclude that there is an extension of the PDFN default theory (W,D) that also contains the propositional literal l.

Question 4

In this question we try to encode the cardinality rule using a basic answer set program. So translating the following program A,

```
#const n=2.
#const m=3.
#const u=4.

n { item(1..u) } m.
```

into a basic ASP program P. This cardinality rule implies that the answer set must contain x items are between the range 1 and u, with x being at least n and at most m.

For our basic ASP program P we may only use rules of the form

```
:- b1, ..., bn, not c1, ..., not cm.
a.
```

and 'range notation' e.g.

```
item(1..u).
```

The following program P is similar to the program A

```
#const n =2.
#const m = 3.
#const u =4.
element(1..u).

lowerbound(1..n).
upperbound(1..m).

choose_LB(X, A) :- not unchoose_LB(X, A), element(A), lowerbound(X), lowerbound(Y
   ), not choose_LB(Y, A), X != Y.
unchoose_LB(X, A) :- not choose_LB(X, A), element(A), lowerbound(X), choose_LB(X,
    B), element(B), A != B.
```

```
choose_LB(X,A) :- lowerbound(X), element(A).

choose_UB(X, A) :- not unchoose_UB(X, A), element(A), upperbound(X), not
   choose_UB(X,B), element(B), A!=B.
unchoose_UB(X, A) :-  element(A), upperbound(X), choose_UB(X, B), element(B), A!=
   B.

item(A) :- choose_UB(X,A), upperbound(X).

:- item(A), unchoose_LB(X,A), lowerbound(X).
#show item/1.
```

Unfortunately this program is not yet correctly working for the lowerbound for some reason. I have also tried implementing it in another way which also was not working yet. See below for this code. This code is not finished yet, but I thought it was maybe in a better direction than the first one.

```
#const n=2.
#const m=3.
#const u=4.
element(1..u).

ctr(I, K) :- ctr(I-1, K), element(I).
ctr(I + 1, K + 1) :- ctr(I + 1, K), ctr(I, K), choose(I + 1), element(I + 1).

choose(X) :- not unchoose(X), element(X).
unchoose(X) :- not choose(X), element(X).

item(X) :- choose(X).
#show item/1.
```

Question 5

To solve the problem SAT-UNSAT for ϕ_1 being satisfied and ϕ_2 being unsatisfied we translate this to an answer set program with disjunction in the head of the rules. First we encode the CNF formulas ϕ_1 and ϕ_2 to ASP.

A CNF formula consists of conjunctions of clauses. If a CNF formula is satisfiable, there exists at least one solution for which all clauses are true. A clause consists of disjunctions. So for a clause to be true, at least one literal of the clause must be true. For every literal we can add the following rules to the ASP program.

```
literal(a).
literal(b)
literal(c)
true(L); false(L) :- literal(L).
```

For every disjunction we can select one of the following rules to the ASP program. For $a \vee b$ we select the first rule and for $a \vee \neg b$ we select the second rule.

```
true(A) ; true(B).
true(A) ; false(B).
```

, with A and B being literals which can be filled in with ground atoms, for instance,

```
true(a) ; true(b).
```

To find a solution for which ϕ_1 is satisfiable and ϕ_2 is unsatisfiable we can use the saturation technique. A problem is unsatisfiable if there is no solution, so if a literal is both true and not true. So we want to have a rule that for ϕ_1 there is a solution for which there is no contradiction and for ϕ_2 we want to have a rule for which there is at least for one literal a contradiction.