MASTER IN ARTIFICIAL INTELLIGENCE (UDC - USC - UVigo)
REASONING AND PLANNING exam. January 9th, 2026

Last Name: _____

First Name: _____

INSTRUCTIONS This part of the exam covers units 1-6 and is weighted with a maximum of **45 points (pt) from a total of 100 pt**. Unit 7 (15 pt) is covered in a separated sheet. For the test, use the original statement sheet and avoid corrections or unclear marking (ask for a new blank sheet if needed).

— EXAM —

Exercise 1 (20pt). Each question has at least one correct answer and its total score depends on whether you check: some incorrect answer = **-3pt**; all correct answers and nothing else = **5pt**; only correct answers, but not all = **3pt**; blank = **0pt**. A total negative score in Ex. 1 counts as 0pt in the rest of the exam.

1.1) Mark those interpretations that are *countermodels* of the formula $p \rightarrow (q \rightarrow p)$ in classical propositional logic:

- ☐ $\{p, q\}$
- ☐ $\{p\}$
- ☐ $\emptyset$
- ☑ None of the above

1.2) The logic program P `a :- b, c.  c :- a.  b :- d.` is obviously stratified since it contains no negation. Mark each rule below such that, if added to P, the result would not be stratified.

- ☐ `d :- b`
- ☑ `b :- not a`
- ☐ `c :- not d`
- ☑ `d :- not c`

1.3) Mark the stable models of the following logic program `p :- not q.  q :- p.  r :- not q.  q :- not r.`

- ☐ $\emptyset$
- ☐ $\{p\}$
- ☑ $\{q\}$
- ☐ $\{p, q\}$
- ☐ $\{r\}$
- ☐ $\{p, r\}$
- ☐ $\{q, r\}$
- ☐ $\{p, q, r\}$

1.4) The logic program P_1 `q :- not p.  q :- p.` corresponds to the formula $(\neg p \rightarrow q) \wedge (p \rightarrow q)$. Mark below the Here-and-There (HT) interpretations that are models of P_1 in HT:

- ☐ $H = \{p\}, T = \emptyset$
- ☐ $H = \emptyset, T = \{p\}$
- ☑ $H = \emptyset, T = \{p, q\}$
- ☐ $H = \emptyset, T = \{q\}$

Exercise 2 (5pt). The previous program P_1 seems to produce the same effect as program P_2 with just the fact `q.`. However, you should have found some HT-model of P_1 above that is not HT-model of q. As a consequence, P_1 and P_2 are *not strongly equivalent*. **Find a single rule** R using atoms p, q such that $P_1 \cup R$ and $P_2 \cup R$ have different behaviour (that is, they have some difference in their respective stable models). Explain the result of adding R.

Take the rule R `p :- q.`. It is easy to see that $P_2 \cup R =$ `q.  p :- q.` **has the unique stable model** $\{p, q\}$. However, $P_1 \cup R =$ `q :- not p.  q :- p.  q :- p.` **has no stable model**: if we take any interpretation I without p then the reduct $(P_1 \cup R)^I$ would be `q.  q :- p.  p :- q.` and the least model derives p. If we take I containing p then the reduct would be `q :- p.  p :- q.` and the least model $\emptyset$ does not justify p.

Explanations for the test

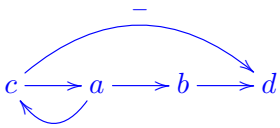
1.1) The formula is a *tautology* so it has no countermodels. We can build the truth table to check it, but it is perhaps easier to see:

$$\begin{aligned} p \to (q \to p) &\equiv \neg p \vee (\neg q \vee p) \\ &\equiv \neg p \vee \neg q \vee p \\ &\equiv (\neg p \vee p) \vee \neg q \\ &\equiv \top \vee \neg q \equiv \top \end{aligned}$$

1.2) The dependence graph of program P is shown below:

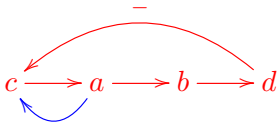
Since P is a positive program, there are no negative arcs. For the first answer, adding a positive rule like `d :- b` will not introduce negative arcs, so the program remains positive and stratified, with the new dependence graph:

If the rule added is `c :- not d` instead, then the dependence graph looks like:



but the new negative arc is not involved in any loop. For the third answer, the rule `b :- not a` forms a negative loop (in red) between a and b drawn in red below:

Finally, for the fourth answer, if we add the rule `d :- not c` then we form a loop with negation involving all the atoms as folows:



1.3) There are multiple ways to deduce the stable models of this program. The simplest one is observing that the program can be splitted into a bottom program with the loop:

```
r :- not q.     q :- not r.
```

that generates two stable models $\{q\}$ and $\{r\}$ and a top program with the other two rules

```
p :- not q.     q :- p.
```

that remove the stable model $\{r\}$ because assuming *not* q would lead to p and then to q. So the only surviving stable model is $\{q\}$.

For a more rigorous explanation, we may start observing that the classical models of the program correspond to the models of the formula $(\neg q \to p) \wedge (p \to q) \wedge (\neg q \to r) \wedge (\neg r \to q)$ in classical logic. The first two implications $(q \vee p) \wedge (\neg p \vee q) \equiv q \vee (p \wedge \neg p) \equiv q$ whereas the second two implications are equivalent to $q \vee r$ and so, subsumed by q. Thus, the formula amouts to q and the classical models are $\{q\}, \{p, q\}, \{q, r\}$ and $\{p, q, r\}$.

Since all stable models must be classical, we may only check these four combinations.

- The reduct $P^{\{q\}}$ corresponds to the program `q :- p.  q.` whose least model is $\{q\}$ again, so $\{q\}$ is stable.

- The reduct $P^{\{p,q\}}$ also corresponds to the program `q :- p.  q.` with least model $\{q\}$ but the initial assumption was $\{p, q\}$ and p is not justified, so $\{p, q\}$ is not stable,
- The reduct $P^{\{q,r\}}$ only contains the rule `q :- p.` and its least model is $\emptyset$ so none of the assumptions $\{q, r\}$ is justified and the model is not stable.
- Finally, the reduct $P^{\{p,q,r\}}$ is again `q :- p.` with least model $\emptyset$, and again none of the assumptions $\{p, q, r\}$ is justified.

1.4) The first answer $H = \{p\}, T = \emptyset$ is not a well-formed HT interpretation since H is not a subset of T.

The second answer $H = \emptyset, T = \{p, q\}$ is a model of $\neg p \rightarrow q$ because $T \models p$ and so $T \not\models \neg p$ and then, the implication $\neg p \rightarrow q$ holds both here and there (remember that negation is only checked there). It is also a model of $p \rightarrow q$ because $T \models p \rightarrow q$ (both atoms hold there) whereas $\langle H, T\rangle \not\models p$ and so, the implication $p \rightarrow q$ also holds here.

The third answer $H = \emptyset, T = \{p\}$ is not a model of $p \rightarrow q$ because $T \not\models p \rightarrow q$.

Finally, in the fourth answer $H = \emptyset, T = \{q\}$ we can check that $\langle H, T\rangle \not\models \neg p \rightarrow q$ since $\langle H, T\rangle \models \neg p$ because $p \notin T$ whereas $\langle H, T\rangle \not\models q$ because $q \notin H$.

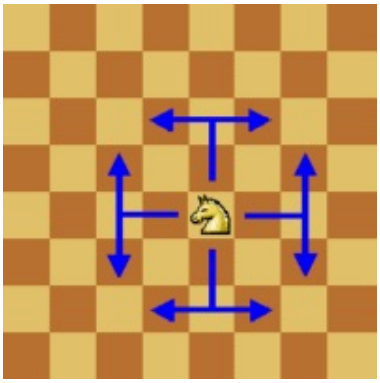
Exercise 3 (10pt). Given a chessboard of `n x n` with `n` ≥ 1 we want to place `m` > 0 knights so they do not attack each other, using predicate `horse(X,Y)` to mean that cell `X`, `Y` contains a horse. NOTE: a chess knight attacks as shown in the figure below: attacked positions correspond to the cells with arrow heads. Complete the following ASP code to solve the problem (try to write as few rules as possible).

```
#const n=3.
#const m=5.
row(1..n). col(1..n).

% Generate all possible placements of m Knights

m { horse(X,Y) : row(X), col(Y) } m.

% Forbid mutual attacks
:- horse(X,Y), horse(X+1,Y+2).            % (*)
:- horse(X,Y), horse(X-1,Y+2).
:- horse(X,Y), horse(X+2,Y+1).
:- horse(X,Y), horse(X-2,Y+1).
```



NOTE: adding these additional constraints:

```
:- horse(X,Y), horse(X+1,Y-2).
:- horse(X,Y), horse(X-1,Y-2).
:- horse(X,Y), horse(X+2,Y-1).
:- horse(X,Y), horse(X-2,Y-1).
```

is correct, but unneeded. Take, for instance, the second one of these four additional rules:

```
:- horse(X,Y), horse(X-1,Y-2).
```

This is the same as:

```
:- horse(X,Y), horse(X',Y'), X'=X-1, Y'=Y-2.
```

that, in its turn, is the same as:

```
:- horse(X,Y), horse(X',Y'), X=X'+1, Y=Y'+2.
```

that is:

```
:- horse(X'+1,Y'+2), horse(X',Y').
```

that is just another way to write the rule we marked with `(*)`. Something similar happens with the other 3 additional rules.

Exercise 4 (5pt). Explain why constraint `(*)` does not need to check the conditions `X+1<n` and `Y+2<n`. How many ground rules does rule `(*)` generate when $n = 3$? 2 (explain the answer below too)

Predicate `horse(X,Y)` is already being generated for valid positions inside the chessboard: in our proposed solution above, this is achieved by the choice rule `m {horse(X,Y) : row(X), col(Y)} m`. Therefore, if `X+1` or `Y+2` lay outside the chessboard, the fact `horse(X+1,Y+2)` will always be false and the constraint is never applied.

As for the number of ground rules, in the general case, we would get $n^2 = 9$ posible pairs for `X,Y`, that are all the possible cells. Note that the values for `X+1` and `Y+2` uniquely depend on `X,Y` so they do not generate more combinations. However, as explained before, cells `X+1,Y+2` must lay inside the chessboard, so `X` $\leq n-1$ and `Y` $\leq n-2$. This means we only have $(n-1) \times (n-2) = 2 \times 1 = 2$ posible combinations that, in our example, would amount to:

```
:- horse(1,1), horse(2,3).
:- horse(2,1), horse(3,3).
```

Exercise 5 (5pt). In the taxi routing problem of the second lab assignment, write telingo code to forbid that a taxi picks somebody that was already in a station (you may use your own predicates).

A possible formalisation could look like:

```
#program dynamic.

:- occurs(pick(T)), 'holds(taxi_location(T),L), _station(L).
```

In general, this constraint removes valid plans that are not too useful (because of moving some passenger already located at a station). But suppose we allowed a passenger to be initially at a station: could there be scenarios in which all the possible plans are ruled out by the constraint above? If so, draw a possible initial configuration where this may happen.

One possible inisital situation could be

where the taxi must mandatorily pick a who is blocking b to access any station. The constraint would remove any valid plan.