MASTER IN ARTIFICIAL INTELLIGENCE (UDC - USC - UVigo)

REASONING AND PLANNING exam. June 17th, 2024

Surname: ______________________________

First Name: ______________________________

INSTRUCTIONS This exam covers units 1-6 and is weighted with a maximum of **42 points (pt) from a total of 100 pt** in the whole course (Unit 7 is not covered in the exam and weights 8 pt). For the test, use the original statement sheet and avoid corrections or unclear marking (ask for a new blank sheet if needed). **Completion time = 2 hours**.

— 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 the correct answers = **5pt**; only correct answers, but not all = **3pt**; leaving blank = **0pt**. A total negative score in Exercise 1 counts as 0pt in the rest of the exam.

1.1) Mark those formulas below that are equivalent to $p \vee q \to r$ in classical propositional logic:

- [x] $(p \to r) \wedge (q \to r)$
- [ ] $(p \to r) \vee (q \to r)$
- [x] $\neg r \to \neg p \wedge \neg q$
- [ ] $\neg(p \wedge q \wedge \neg r)$

1.2) The logic program P with rules `a :- not c.   a :- b   c :- b` is stratified. Mark the rules below that, if they were (individually) added to P, they would make the result a non-stratified program.

- [x] `c :- a.`
- [ ] `b :- a.`
- [x] `p :- not p.`
- [x] `b :- not a.`

1.3) Given the following logic program `p :- q.   p :- not r.   r :- p, not q`

- [ ] the reduct with respect to $\{q\}$ is the program `p :- q.   p :- not r.`
- [x] the reduct with respect to $\emptyset$ is the program `p :- q.   p.   r :- p.`
- [x] the reduct with respect to $\{r\}$ is the program `p :- q.   r :- p.`
- [ ] the reduct with respect to $\{p, q\}$ is the program ` `
- [x] the reduct with respect to $\{q\}$ is the program `p :- q.   p.`

1.4) The rule `p :- not r.` used above corresponds to the implication $\neg r \to p$ that is equivalent to $p \vee r$ in classical logic, but is strictly *stronger* in the the logic of Here-and-There (HT). Mark those HT interpretations that are HT models of $\neg r \to p$ but not of $p \vee r$.

- [ ] $H = \{p\}$, $T = \{p\}$
- [ ] $H = \emptyset$, $T = \emptyset$
- [ ] $H = \{r\}$, $T = \{p, r\}$
- [x] $H = \emptyset$, $T = \{r\}$

Explanations for the test

1.1) Let us call $\alpha := p \vee q \to r$. One way to see why the first formula is equivalent to α is the sequence of equivalences:

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

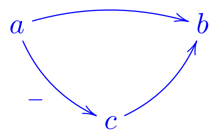
The third formula is also equivalent to α because:

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

The second formula $(p \to r) \vee (q \to r)$ is not equivalent to α. It actually amounts to $\neg p \vee r \vee \neg q \vee r \equiv \neg p \vee r \vee \neg q$, let us call it β. The interpretation $\emptyset$ (all atoms false) does not satisfy β but, in fact, is a model of α.

Finally, we can see that if we apply De Morgan in the fourth formula we get $\neg(p \wedge q \wedge \neg r) \equiv \neg p \vee \neg q \vee r \equiv \beta$ that is, it amounts to the second one, and so, it is not equivalent to α either.

1.2) The program dependence graph is:



- Adding `c :- a` creates a loop between `c` and `a` that has a negative edge, so the program becomes non-stratified.
- Adding `c :- a` creates a loop `c` and `a`, but there is no negative edge in the loop, so the program remains stratified.
- Adding `p :- not p` to any program creates a negative loop `p` to itself. Any program that contains this rule immediately becomes non-stratified.
- Adding `b :- not a` creates a loop `b` and `a`, and the new dependence is negative, so the program becomes non-stratified.

1.3) Answer number 1 is incorrect: the program reduct never contains a negation (all `not`'s are removed). On the other hand, any positive rule is preserved untouched in the reduct: this applies to the first rule `p :- q`, that must always be present in all reduct programs, so answer 4 is also incorrect. The other three answers are correct.

1.4) We look for models of $\neg r \to p$ that are countermodels of $p \vee r$. To be a model of $p \vee r$ we need that either p or r are "proved", that is, belong to H. Thus, answers 1 and 3 are models of $p \vee r$ and so, they are disregarded (we look for countermodels). Answer number 2 is not a model of $\neg r \to p$: this is because proving negation only requires checking $T \not\models r$. So $\neg r$ is proved and the implication would require p to be proved, but we have $p \notin H$. Finally, we can see that answer 4 is a model of $\neg r \to p$ simply because the antecedent $\neg r$ is neither proved or assumed ($T \models r$), and so the implication does not require anything about p. On the other hand, answer 4 is not a model of $p \vee r$, because $H = \emptyset$, there are no proved atoms.

Exercise 2 (10pt). Write an ASP program that generates all ways to place 4 bishops in a chessboard so that they do not attack each other. Use predicate `bishop(X,Y)` meaning there is a bishop at row `X` and column `Y`. (NOTE: in chess, bishops attack other pieces in the same diagonal).

```
#const n=8.
cell(1..n,1..n).
4 {bishop(X,Y): cell(X,Y)} 4.
:- bishop(X,Y), bishop(X',Y'), |X-X'|=|Y-Y'|, X!=X'.
#show bishop/2.
```

Exercise 3 (8pt). The following `telingo` program tries to move a robot in a grid from an initial position at `(0,0)` to a goal position at `(3,4)`. Complete the program to fulfil the two missing requirements: (1) move the robot to some adjacent position (up, down, left or right); (2) the robot cannot step out of the grid.

```
#program initial.
grid(0..3,0..4).
wall(0,2).  wall(2,2).   wall(3,2).   robot(0,0).   goal(3,4).

#program dynamic.
% Move the robot to some adjacent position
1 { robot(X+1,Y); robot(X-1,Y); robot(X,Y+1); robot(X,Y-1) } 1 :- 'robot(X,Y).
:- robot(X,Y), _wall(X,Y).      % Do not step into a wall
:- robot(X,Y), not _grid(X,Y).  % Do not step out of the grid

#program final.
:- robot(X,Y), not _goal(X,Y).  % Reach the goal at last state
```

Exercise 4 (4pt). Write a formula in Description Logic (DL) that describes the set of red (*Red*) cars (*Car*) that have some foreign (*Foreign*) owner (*owned_by*).

$$\mathit{Red} \sqcap \mathit{Cap} \sqcap \exists \mathit{owned\_by}.\mathit{Foreign}$$