Logic-Based Learning: Coursework Part Two

Mark Law

Due: 3rd March 2017

Question 1

Consider the ASP program B:

```
rock(X) :- not paper(X), not scissors(X), player(X).
paper(X) :- not rock(X), not scissors(X), player(X).
scissors(X) :- not rock(X), not paper(X), player(X).

wins(X) :- rock(X), scissors(Y).
wins(X) :- scissors(X), paper(Y).
wins(X) :- paper(X), rock(Y).

player(1).
player(2).
```

(a) Write down a program P of optimal length (which is not a constraint and does not contain the predicate *wins*) such that:

(i) $B \cup P \models_b wins(1)$ but $B \cup P \not\models_b wins(1) \wedge paper(1)$

(ii) $B \cup P \models_b wins(1)$ and $B \cup P \models_b paper(1)$ but $B \cup P \not\models_b wins(1) \wedge paper(1)$

(iii) $B \cup P \models_c wins(1)$ and $B \cup P$ is satisfiable.

(b) Write down a constraint P of optimal length (no restrictions) such that:

(i) $B \cup P \models_b wins(1)$ but $B \cup P \not\models_b wins(1) \wedge paper(1)$

(ii) $B \cup P \models_b wins(1)$ and $B \cup P \models_b paper(1)$ but $B \cup P \not\models_b wins(1) \wedge paper(1)$

(iii) $B \cup P \models_c wins(1)$ and $B \cup P$ is satisfiable.

Solution

These are some of the optimal hypotheses (in most cases there are alternatives).

(a) (i) $rock(1)$.

This forces that player 1 plays rock, and therefore never plays paper (so can't win with paper). Player 1 still wins in the case where player 2 plays scissors.

(ii) $scissors(2)$.

This forces that player 2 plays scissors, and therefore wins when player 1 plays paper (so player 1 can't win with paper). Player 1 still if they play rock, and can still play paper.

(iii) $scissors(2)$. $rock(1)$.

This forces that there is only one answer set and it contains $wins(1)$; therefore, $wins(1)$ is cautiously entailed.

(b) (i) $\leftarrow paper(1)$. This rules out all answer sets in which player 1 plays paper, so player 1 definitely can't win with paper. Player 1 can still win in some cases though.

(ii) $\leftarrow rock(2)$.

If player 2 never plays rock, then player 1 can't win with paper. But they can still play paper, and can still win if they don't play paper.

(iii) $\leftarrow$ not $wins(1)$.

This rules out all answer sets in which $wins(1)$ is false, and therefore ensures that $wins(1)$ is cautiously entailed (and the program is satisfiable as there were answer sets of the original program in which player 1 won). Note that the constraint $\leftarrow wins(2)$ would not work, as it does not rule out the answer sets in which neither player wins (when they both play the same).

```
This question was done very well. Mostly people dropped marks by giving
sub-optimal answers.
```

Question 2

Given the background knowledge $B = \{0\{p, r\}2 \leftarrow \text{not q}\}$ for each hypothesis H below, write down a learning task such that H is an optimal solution. This involves giving examples E^+ and E^- (you should assume that in all cases the hypothesis space consists of any normal rules or constraints which use the atoms p, q and r). In each question, your task may be a brave, cautious or learning from answer sets task (you must make it clear which). For example, for the hypothesis {q.} you could give the task $ILP_b(B, \{q\}, \emptyset)$.

(a) `p.`

(b) `:- r.`

(c) `p :- r.`

(d) `q :- not p.`

Solution

(a) One solution is the cautious induction task:

$ILP_c\langle B, \{p\}, \emptyset\rangle$

This says that every answer set should contain p. The empty hypothesis does not satisfy this condition and therefore the shortest hypothesis is H. Note that brave induction with the same examples does not work here as p is already contained in at least one answer set of B, so the empty hypothesis $\emptyset$ satisfies the examples.

(b) One solution is the cautious induction task:

$ILP_c\langle B, \emptyset, \{r\}\rangle$

Note again that brave induction does not work here; in fact, in general brave induction can not learn constraints.

(c) The answer sets of $B \cup H$ are a subset of the answer sets of B, therefore no matter what examples we give to learn H in brave induction, the empty hypothesis will already cover them. Also the only atom which takes the same truth value in every answer set of $B \cup H$ is q which is already cautiously entailed false by B. So there are no meaningful examples that we can give for either brave or cautious induction. We therefore need learning from answer sets.

One solution is the learning from answer sets task:

$ILP_{LAS}\langle B, \{\langle\{r\}, \emptyset\rangle, \langle\emptyset, \{p\}\rangle\}, \{\langle\{r\}, \{p\}\rangle\}\rangle$

This says that there should be at least one answer set which contains r (ruling out the constraint `:- r` or the fact `q`), at least one answer set which does not contain p (ruling out the fact `p.`) and no answer set containing r but not p (ruling out the other shorter hypotheses). So the optimal hypothesis is `p :- r`.

(d) Again, neither brave, nor cautious induction will work here and we need learning from answer sets.

One solution is:

$ILP_{LAS}\langle B, S_M, \{\langle\{q\}, \emptyset\rangle, \langle\{p\}, \emptyset\rangle\}, \emptyset\rangle$

This says that there should be at least one answer set which contains q. This means that we need to learn a rule for q as the background knowledge does not contain a rule with q in the head. The fact q would mean that there was only one answer set $\{q\}$, and so would not cover the second example. Hence as `q :- not p` is an inductive solution, it must be an optimal inductive solution as there is no shorter solution.

```
The first two parts of this question were done very well by most people. The
most common mistake in part (c) was to omit at least one positive example. This
meant, depending on which example you left out, that either ":- r." was a
solution or the fact "p." was a solution. Many people gave a brave task for
part (d) with a positive example "q". This task has the fact "q." as a
solution.
```

Question 3

A *Learning from Answer Sets cautiously* (ILP^c_{LAS}) task is a tuple $\langle B, S_M, E^+, E^-\rangle$ where B is an ASP program, S_M is a set of rules called the search space and E^+ and E^- are sets of partial interpretations called the positive and negative examples respectively. A hypothesis H is an inductive solution (written $H \in ILP^c_{LAS}\langle B, S_M, E^+, E^-\rangle$) if and only if:

1) $H \subseteq S_M$

2) $\forall e^+ \in E^+ : B \cup H \models_c e^+$

3) $\forall e^- \in E^- : B \cup H \not\models_c e^-$

a) Given an arbitrary ILP_{LAS} task $T = \langle B, S_M, E^+, E^-\rangle$, write down an ILP^c_{LAS} task T_c such that $ILP_{LAS}(T) = ILP^c_{LAS}(T_c)$

b) Given an arbitrary ILP^c_{LAS} task $T_c = \langle B, S_M, E^+, E^-\rangle$, write down an ILP_{LAS} task T such that $ILP_{LAS}(T) = ILP^c_{LAS}(T_c)$

Note that for a program P and a partial interpretation e, $P \models_c e$ is holds if and only if $\forall A \in AS(P), A$ extends e.

Hint: you will need to add to the background knowledge B; you may find it helpful to define a predicate $cov(id)$ which indicates that when $cov(id)$ appears in an Answer Set A of $B \cup H$ this Answer Set covers the example with id id. You may assume that cov appears nowhere in the original task.

Note: This question is tricky, but the answer is quite short (less than half a page), so please don't waste your time writing long answers.

Solution

For each example $e \in E^+ \cup E^-$, let e_{id} denote the unique identifier of the example e. Let B' be the program $B \cup \left\{ \begin{array}{ll} cov(e_{id}) \leftarrow & e_1^{inc}, \ldots, e_n^{inc}, \\ & \text{not } e_1^{exc}, \ldots, \text{not } e_m^{exc}. \end{array} \middle| \begin{array}{c} e \in E^+ \cup E^-, \\ e = \langle \{e_1^{inc}, \ldots, e_n^{inc}\}, \{e_1^{exc}, \ldots, e_m^{exc}\} \rangle \end{array} \right\}$

a) $ILP_{LAS}\langle B', S_M, \{\langle \emptyset, \{cov(e_{id})\}\rangle : e \in E^-\}, \{\langle \emptyset, \{cov(e_{id})\}\rangle : e \in E^+\}\rangle$

b) $ILP^c_{LAS}\langle B', S_M, \{\langle \emptyset, \{cov(e_{id})\}\rangle : e \in E^-\}, \{\langle \emptyset, \{cov(e_{id})\}\rangle : e \in E^+\}\rangle$

Note that this question was very tricky, so don't worry if you didn't get it right. Do try to understand it though. The key was to notice that there is a symmetry between the positive examples in normal LAS and the negative examples in cautious LAS, and another between the negative examples in normal LAS and the positive examples in cautious LAS.

The difficulty is that there isn't a direct translation from the one set of examples to another. This is why you need to define an extra predicate (for each example) in the background knowledge. The idea is that you, in a sense, reduce each example down to a single atom. There is then an answer set extending each example, if and only if the corresponding atom appears in at least one answer set.

Once you have reduced your examples down to single atoms, there IS a direct mapping from one set of examples to the other. It relies on the fact that the following two statements are equivalent: "Every answer set contains a" and "There is no answer set which does not contain a". Similarly, "no answer set of P contains a" is the same as "P cautiously entails not a".

Some people gave a translation that flipped exclusions and inclusions without using the background knowledge. This does not work. The reason is that A extends <{p, q}, {}> is not equivalent to A does not extend <{}, {p, q}> (A = {p} satisfies the latter but does not satisfy the former).

Others used a meta representation similar to the ILASP meta representation. This is unnecessary to solve the problem.

The other common mistake was to say that one framework was some sort of special case of the other, meaning that any task of the first framework was somehow automatically a task of the second framework. Even if one were a special case of the other (which it isn't), this would still mean that the inductive solutions of one task would only be a SUBSET of the inductive solutions of the other task. The question asks for equality.

Question 4

This question is based around an agent planning problem in which the agent must get from one corner of the map to another. The agent can see the full map to begin with, but has no information about how the objects in the map might effect the moves which it has available.

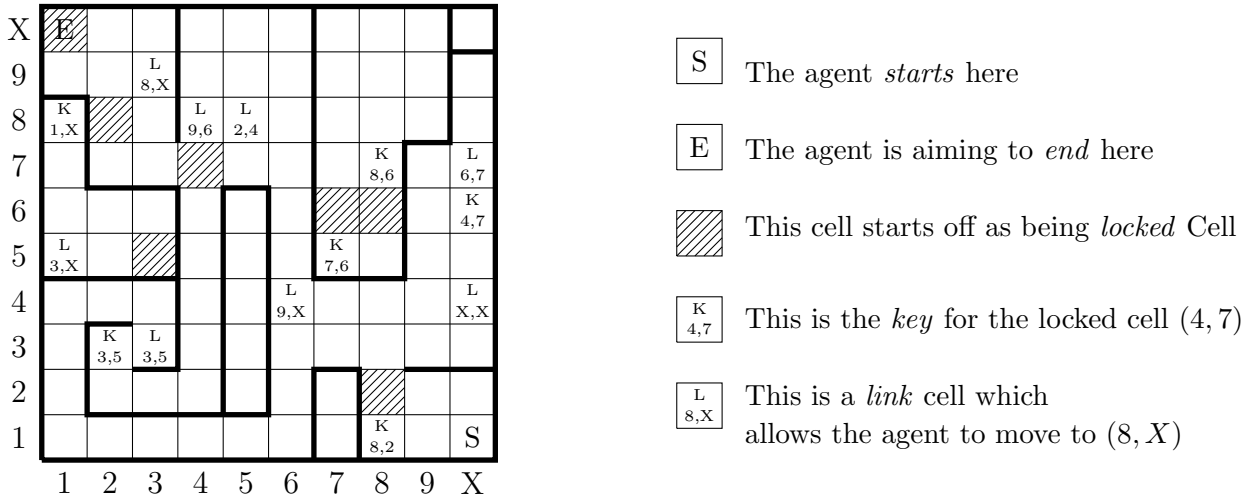

Figure 1: Cells with diagonal lines are *locked* and the agent must visit the corresponding key before it can enter these cells. *Link* cells allow the agent to jump to the indicated destination cell. The thick black lines represent walls.

At each step through the maze, the agent has a hypothesis describing the rules for which moves are *valid*. An oracle tells the agent which moves are valid at the current time point. We represent the objects in the map with the predicates: *cell*, *adjacent*, *link*, *key* and *locked*. The definition of valid move (unknown to the agent) is:

```
valid_move(C, T) :- agent_at(C2, T), not wall(C2, C),
                    adjacent(C2, C), unlocked(C, T).

valid_move(C, T) :- agent_at(C2, T), link(C2, C), unlocked(C, T).
```

If these are different, the agent updates a set of examples and uses ASPAL to learn a new hypothesis. The aim of this question is to construct the ASPAL encoding the agent uses.

(a) Given the mode declarations:

$$M = \begin{cases} modeh(1, valid\_move(+dest, +time)), \\ modeb(1, agent\_at(-cell, +time)), \\ modeb(1, \text{not } wall(+cell, +dest)), \\ modeb(1, adjacent(+cell, +dest)), \\ modeb(1, link(+cell, +dest)), \\ modeb(1, unlocked(+dest, +time)) \end{cases}$$

and $L_{max} = 4$, $V_{max} = 3$, write down a maximal set of skeleton rules S_M.

Note: we use `dest` only to keep the size of the search space smaller for you. `dest` is actually the same as `cell`.

(b) The agent's background knowledge after 16 steps through the maze is given in the file "background.lp" (downloadable from CATE).

Use this, the examples below, and your answer to part (a) to construct the complete ASPAL encoding (the meta level ASP program which is generated by ASPAL). If writing it out by hand, you may write `(background.lp)` instead of copying it out. (You can check your answer using clingo and check that the optimal answer set corresponds to a hypothesis which covers the examples).

```
E+ = { valid_move(cell(9, 1), 1), valid_move(cell(9, 10), 8) }
E- = { valid_move(cell(1, 1), 1), valid_move(cell(7, 1), 3),
       valid_move(cell(4, 7), 12), valid_move(cell(3, 5), 16) }
```

Solution

(a)
```
valid_move(D, T) :- dest(D), time(T).
valid_move(D, T) :- dest(D), time(T), cell(C), agent_at(C, T).
valid_move(D, T) :- dest(D), time(T), cell(C), agent_at(C, T), not wall(C, D).
valid_move(D, T) :- dest(D), time(T), cell(C), agent_at(C, T), not wall(C, D),
                    adjacent(C, D).
valid_move(D, T) :- dest(D), time(T), cell(C), agent_at(C, T), not wall(C, D),
                    adjacent(C, D), link(C, D).
valid_move(D, T) :- dest(D), time(T), cell(C), agent_at(C, T), not wall(C, D),
                    adjacent(C, D), unlocked(D, T).
valid_move(D, T) :- dest(D), time(T), cell(C), agent_at(C, T), not wall(C, D),
                    link(C, D).
valid_move(D, T) :- dest(D), time(T), cell(C), agent_at(C, T), not wall(C, D),
                    link(C, D), unlocked(D, T).
valid_move(D, T) :- dest(D), time(T), cell(C), agent_at(C, T), not wall(C, D),
                    unlocked(D, T).
valid_move(D, T) :- dest(D), time(T), cell(C), agent_at(C, T), link(C, D).
valid_move(D, T) :- dest(D), time(T), cell(C), agent_at(C, T), link(C, D),
                    adjacent(C, D).
valid_move(D, T) :- dest(D), time(T), cell(C), agent_at(C, T), link(C, D),
                    adjacent(C, D), unlocked(D, T).
valid_move(D, T) :- dest(D), time(T), cell(C), agent_at(C, T), link(C, D),
                    unlocked(D, T).
valid_move(D, T) :- dest(D), time(T), cell(C), agent_at(C, T), adjacent(C, D).
valid_move(D, T) :- dest(D), time(T), cell(C), agent_at(C, T), adjacent(C, D),
                    unlocked(D, T).
valid_move(D, T) :- dest(D), time(T), cell(C), agent_at(C, T), unlocked(D, T).
valid_move(D, T) :- dest(D), time(T), unlocked(D, T).
```

(b) `(background.lp)`

```
valid_move(C, T) :- rule(1), dest(C), time(T).
valid_move(C, T) :- rule(2), dest(C), time(T), cell(C2), agent_at(C2, T).
```

```
valid_move(C, T) :- rule(3), dest(C), time(T), cell(C2), agent_at(C2, T),
                    not wall(C2, C).
valid_move(C, T) :- rule(4), dest(C), time(T), cell(C2), agent_at(C2, T),
                    not wall(C2, C), adjacent(C2, C).
valid_move(C, T) :- rule(5), dest(C), time(T), cell(C2), agent_at(C2, T),
                    not wall(C2, C), adjacent(C2, C), link(C2, C).
valid_move(C, T) :- rule(6), dest(C), time(T), cell(C2), agent_at(C2, T),
                    not wall(C2, C), adjacent(C2, C), unlocked(C, T).
valid_move(C, T) :- rule(7), dest(C), time(T), cell(C2), agent_at(C2, T),
                    not wall(C2, C), link(C2, C).
valid_move(C, T) :- rule(8), dest(C), time(T), cell(C2), agent_at(C2, T),
                    not wall(C2, C), link(C2, C), unlocked(C, T).
valid_move(C, T) :- rule(9), dest(C), time(T), cell(C2), agent_at(C2, T),
                    not wall(C2, C), unlocked(C, T).
valid_move(C, T) :- rule(10), dest(C), time(T), cell(C2), agent_at(C2, T),
                    link(C2, C).
valid_move(C, T) :- rule(11), dest(C), time(T), cell(C2), agent_at(C2, T),
                    link(C2, C), adjacent(C2, C).
valid_move(C, T) :- rule(12), dest(C), time(T), cell(C2), agent_at(C2, T),
                    link(C2, C), adjacent(C2, C), unlocked(C, T).
valid_move(C, T) :- rule(13), dest(C), time(T), cell(C2), agent_at(C2, T),
                    link(C2, C), unlocked(C, T).
valid_move(C, T) :- rule(14), dest(C), time(T), cell(C2), agent_at(C2, T),
                    adjacent(C2, C).
valid_move(C, T) :- rule(15), dest(C), time(T), cell(C2), agent_at(C2, T),
                    adjacent(C2, C), unlocked(C, T).
valid_move(C, T) :- rule(16), dest(C), time(T), cell(C2), agent_at(C2, T),
                    unlocked(C, T).
valid_move(C, T) :- rule(17), dest(C), time(T), unlocked(C, T).

goal :- valid_move(cell(9, 1), 1),
        valid_move(cell(9, 10), 8),
        not valid_move(cell(1, 1), 1),
        not valid_move(cell(7, 1), 3),
        not valid_move(cell(4, 7), 12),
        not valid_move(cell(3, 5), 16).

:- not goal.

{ rule(1..17) }.

#minimise [
  rule(1) = 1, rule(2) = 2, rule(3) = 3, rule(4) = 4,
  rule(5) = 5, rule(6) = 5, rule(7) = 4, rule(8) = 5,
  rule(9) = 4, rule(10) = 3, rule(11) = 4, rule(12) = 5,
  rule(13) = 4, rule(14) = 3, rule(15) = 4, rule(16) = 3,
  rule(17) = 2
].
```

On the whole, this question was done very well. One common mistake was to ignore the input and output variables. Remember, an input variable in the body MUST occur as an input variable in the head, or as an output variable in another body literal. Some others forgot about type atoms (and sometimes

omitted rules that would be unsafe without types).

In part (b) the main mistake was to get the weights wrong. Remember that type atoms are NOT counted but heads ARE.