Logic-Based Learning: Answer Set Programming

Mark Law

February 5, 2015

This tutorial aims for you to practice using the stable model semantics (Unit 6) and ASP (Unit 7). Once you have completed it, you should be able to write, solve and run ASP programs involving normal rules, choice rules and constraints. These problems require the use of *clingo*. Clingo is installed on the lab machines and can be found at /vol/lab/CLASP/clingo. You should write your ASP programs and run the command `/vol/lab/CLASP/clingo -n 0 {filename}`. Clingo will then compute the Answer Sets of your program. If you wish instead to see the ground program, you should run with the option "-t". Note that this grounding is sometimes simpler than what we compute as $ground(P)$, as clingo will simplify things wherever possible.

Question 1

For the following normal logic programs P and sets X:

a) Calculate $ground(P)$

b) Calculate $(ground(P))^X$

c) Is X a stable model of P?

d) Write down all stable models of P (No proof required).

e) Check your answers using Clingo.

i) $X = \{p(1), q(2), r(1), r(2)\}$,

$$P = \begin{cases} p(X) \leftarrow \text{not } q(X), r(X). \\ q(X) \leftarrow \text{not } p(X), r(X). \\ r(1). \\ r(2). \end{cases}$$

ii) $X = \{p(1), q(2), r(1), r(2)\}$,

$$P = \begin{cases} p(X) \leftarrow \text{not } q(X), r(X). \\ q(X) \leftarrow \text{not } p(X), r(X). \\ s \leftarrow q(X), \text{not } s. \\ r(1). \\ r(2). \end{cases}$$

iii) $X = \{p\}$ and $X = \emptyset$

$$P = \begin{cases} p \leftarrow \text{not } q. \\ q \leftarrow p. \end{cases}$$

iv) $X = \{p, q\}$,

$$P = \begin{cases} p \leftarrow q. \\ q \leftarrow p. \end{cases}$$

Question 2

a) Given the following normal logic programs P and statements S, write down further normal logic programs Q, consisting of only a single normal rule which is as short as possible, such that the statement S is true.

i) $P = \begin{cases} p \leftarrow \text{not } q. \\ q \leftarrow \text{not } r. \end{cases}$

S: "$P \cup Q \models_b p$, $P \cup Q \models_b q$ and $P \cup Q \models_b r$"

ii) $P = \begin{cases} p \leftarrow \text{not } q. \\ q \leftarrow \text{not } p. \\ r \leftarrow \text{not } r, p. \\ r \leftarrow \text{not } r, q. \end{cases}$

S: "$P \cup Q \models_b p$ and $P \cup Q \not\models_b q$"

Hint: in part (i), p, q and r can all be in *different* stable models of $P \cup Q$.

b) Consider again the program Q you found in part i. Given the program P, is there any set of literals X such that $P \cup Q \models_b X$ but there is no shorter hypothesis Q' such that $P \cup Q' \models_b X$? If not, why not?

Question 3

Given a program $P = r \leftarrow p, q$, write down normal logic programs Q such that:

i) $P \cup Q \models_b p$, $P \cup Q \models_b q$ and $P \cup Q \models_c \text{not } r$

ii) $P \cup Q \models_b p \wedge q$ and $P \cup Q \not\models_c r$

Question 4

Calculate all Answer Sets of the ASP program P below. Use Clingo to verify your answers.

$$P = \begin{cases} 1\{a, b, c, d\}3 \leftarrow e. \\ e \leftarrow \text{not } f. \\ 0\{f\}1. \\ b \leftarrow d. \\ c \leftarrow \text{not } a. \end{cases}$$

Question 5

Consider the abductive task from Tutorial 1, $\langle KB, Ab, IC \rangle$, where KB, Ab and IC are defined as follows:

$$KB = \begin{bmatrix} headache(X) \leftarrow jaundice(X) \\ headache(X) \leftarrow migraine(X) \\ sickness(X) \leftarrow stomachBug(X) \end{bmatrix} \qquad Ab = \begin{bmatrix} jaundice(bob) \\ migraine(bob) \\ yellowEyes(bob) \end{bmatrix}$$

$$IC = \begin{bmatrix} \leftarrow migraine(X), jaundice(X) \\ \leftarrow jaundice(X), not\ \ yellowEyes(X) \\ \leftarrow jaundice(X), not\ \ sickness(X) \end{bmatrix}$$

i) Encode the abductive task as an ASP program P such that the Answer Sets of P can be mapped back to the abductive solutions of the task.

ii) Use clingo to find the abductive solutions of the task.

iii) Write an optimisation statement such that smaller abductive solutions are prefered.

Question 6

Consider the following abductive model (also from tutorial 1) $\langle KB, Ab, IC \rangle$, where KB, Ab and IC are defined as follows:

$$KB = \begin{bmatrix} carDoesNotStart(X) \leftarrow batteryFlat(X) \\ carDoesNotStart(X) \leftarrow hasNoFuel(X) \\ lightsGoOn(mycar) \\ fuelIndicatorEmpty(mycar) \end{bmatrix} \quad Ab = \begin{bmatrix} batteryFlat(mycar) \\ batteryFlat(yourcar) \\ hasNoFuel(mycar) \\ hasNoFuel(yourcar) \\ brokenIndicator(mycar) \\ brokenIndicator(yourcar) \end{bmatrix}$$

$$IC = \begin{bmatrix} \leftarrow batteryFlat(X), \\ \quad lightsGoOn(X) \\ \leftarrow hasNoFuel(X), \\ \quad not\ \ fuelIndicatorEmpty(X), \\ \quad not\ \ brokenIndicator(X) \end{bmatrix}$$

For each observation below, write an ASP program whose optimal Answer Sets are the smallest abductive solutions of the task.

1. $O = carDoesNotStart(mycar)$
2. $O = carDoesNotStart(yourcar)$

Question 7

For each ASP program P below:

a) Use clingo to find the Answer Sets of P (use the option `--opt-ignore` to ignore the optimisation statement)

b) Which of these Answer Sets is optimal? Why? (you can check your Answer using clingo with `--opt-all`)

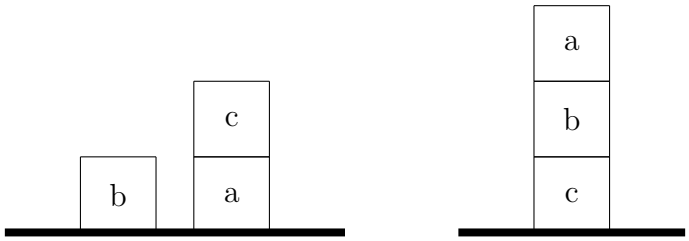
i)
```
r(1..2).
p(X) :- not q(X), r(X).
q(X) :- not p(X), r(X), in_hyp(1).
q(X) :- p(X), r(X), in_hyp(2).
q(X) :- s(X), in_hyp(3).
s(X) :- p(X), in_hyp(4).
:- not p(1).
:- not q(2).
:- p(2).
:- q(1).
0 { in_hyp(1), in_hyp(2), in_hyp(3), in_hyp(4) } 4.
#minimize[ in_hyp(1)=3, in_hyp(2)=3,
           in_hyp(3)=2, in_hyp(4)=2 ].
```

ii)
```
person(a).   person(b).   person(c).   person(d).
interested(a, logic).       interested(d, logic).
interested(b, biology).     interested(c, biology).
interested(a, sport).       interested(b, sport).
interested(c, sport).       interested(a, physics).
interested(c, music).       interested(d, music).
1 { pair(P, a), pair(P, b), pair(P, c), pair(P, d) } 1 :- person(P).
:- pair(P, P).
pair(P1, P2) :- pair(P2, P1).
shared_interest(P1, P2, I) :- interested(P1, I), interested(P2, I),
                              pair(P1, P2).
shared_disinterest(P1, P2, I) :- not interested(P1, I),
                                 not interested(P2, I),
                                 pair(P1, P2), interested(P3, I).
#maximize[ shared_interest(P1, P2, I) = 2,
           shared_disinterest(P1, P2, I) = 1 ].
```

Notice that this program has a non-ground optimisation statement. What do you think it would ground as?

Question 8

You are given some facts describing the "blocks world" problem shown below. The rules are that at each step you are allowed to move exactly one block, providing that there is no block already ontop of it. The table has enough room for all three blocks, but each block can only hold one block immediately above it.



Initial State Goal State

```
step(0..6). % This is shorthand for the facts step(0), ..., step(6).

block(a).
block(b).
block(c).

on(b, table, 0).
on(a, table, 0).
on(c, a, 0).

finished(T) :- goal(T2), step(T), T >= T2.
goal(T) :- on(c, table, T), on(b,c, T), on(a, b, T), not finished(T-1).
goalMet :- goal(T).
:- not goalMet.

clear(table, T) :- step(T).
clear(L, T) :- not blocked(L, T), block(L), step(T).
blocked(L, T) :- on(B, L, T).

0 { move(B, L, T) } 1 :- block(B), not blocked(B, T), clear(L, T), not finished(T).
:- move(B, L, T), move(B2, L2, T), B != B2.
```

1. Write some rules for $on(Block, Location, T{+}1)$ in terms of $on(Block, Location, T)$. You will need to use $step(T)$ to avoid an infinite grounding!
2. Write an optimisation statement such that the optimal Answer Sets of your program are now the ones with the fewest moves before the goal is reached.
3. Test your implementation in Clingo.

Advanced Questions

Question 9

For any ASP program P and any atom a such that $ground(P)$ does not contain a, prove that for any ground atoms $b_1, \ldots, b_m$ and $c_1, \ldots, c_n$, there are no Answer Sets A of $Q = P \cup \{a \leftarrow b_1, \ldots, b_m, \text{not } c_1, \ldots, \text{not } c_m, \text{not } a\}$ st $\{b_1, \ldots, b_m\} \subseteq A$ and $\{c_1, \ldots, c_n\} \cap A = \emptyset$.

Question 10

For any ASP program P and any atom a such that $ground(P)$ does not contain a, prove that for any ground atoms $b_1, \ldots, b_m$ and $c_1, \ldots, c_n$:

$Q = P \cup \{a \leftarrow b_1, \ldots, b_m, \text{not } c_1, \ldots, \text{not } c_n;\ \leftarrow \text{not } a\} \models_c b_1, \ldots, b_m, \text{not } c_1, \ldots, \text{not } c_n$