**Lab 2: FOL\***

This lab consists of two problems, each expected to take roughly 45 min. We assume here that you completed Lab 1 and therefore have an installation of LEGOS.

**Note: Before working on the lab, please take a moment to go through Tutorial2/Intro.py to familiarize yourself with the FOL\* python interface. You can also review the two FOL\* demos in the lecture (demo\_full\_tree.py and demo2\_max\_func.py)**

Problem 1:

In this problem we will model a puzzle in FOL\* and then solve it using FOL\* satisfiability.

**Natural number puzzle:**Suppose we are given a set of natural numbers (e.g., ) and would like to find a way to construct a number with the following rule:

A number can be constructed if:

1. or
2. where and are ***previously*** constructed numbers

**Task 1.a:** *Define an FOL\* signature to capture the concept of a constructed number.*

*Hint: you may need to keep track not only of values of numbers produced but also of the time when they are produced.*

**First declare it on paper and then encode it Problem1.py.**

***'''***

***Task 1: define the FOL\* signature to capture the concept of constructed number***

***'''***

***'''***

***Solution 1:***

***'''***

***# go here***

**Task 1.b:** *Encode an FOL\* constraint to capture the construction rules. You may assume that the set is given.*

**First write down the constraint on paper and then encode it in Problem1.py**

**'''**

**Task 2: encode FOL\* constraint to capture the construction rules. You may assume the set `Basics` is given.**

**'''**

**'''**

**Solution 2:**

**'''**

**Task 1.c:** *Suppose the set is given. Construct the following target numbers: , , , , , . Encode the target as FOL\* constraint and check FOL\* satisfiability*

*Hint: for the large target, you may add additional FOL\* rules to constrain the search space.*

**Encode the goal constraint using the FOL\* interface in Problem1.py. Finish the implementation of the function check\_target(target:int). You can run Problem1.py after implementing the function to observe the result.**

**'''**

**Task 3:Suppose Basics = {2, 3} are given, try to find ways to construct the**

**following target numbers: i = 1, i =6 , i =17, i =28, i =45, i =150, i =1501**

**Please call clear() after each call to LEGOs:**

**e.g.,**

**solve(property)**

**clear()**

**'''**

**'''**

**Solution 3:**

**'''**

**def check\_target(target\_value:int):**

**pass**

**check\_target(1)**

**check\_target(6)**

**check\_target(17)**

**check\_target(28)**

**check\_target(45)**

**check\_target(150)**

**check\_target(1501)**

**Task 1.d:** *Suppose the set is given. Can the target number be constructed? Why not?*

*Now try running LEGOS on this problem (kill the process after 30 seconds!). Indeed, it cannot conclude UNSAT! Add a new FOL\* class and rules to help conclude that the result is UNSAT.*

*Hint1: Adding three odd numbers always yields an odd number. Therefore, if contains only ODD numbers, then all constructed numbers are ODD. You can try to define two classes, ODD and EVEN, as abstractions of a number, and define their construction rules respectively.*

*An EVEN number can be constructed if:*

*or is the abstracted number*

*There were one EVEN number constructed*

*Similar rules for ODD, you may also apply transformation to the goal rule too.*

*Hint2: There is always the first or the last Number/Even/ODD being constructed.*

**Add or modify FOL\* classes and encode the goal constraint using the FOL\* interface in problem1.py. Finish the implementation of the function** *target\_transformation(target:int)* **which constructs an FOL\* formula representing the goal to be reached. Run Problem1.py after implementing the function to observe the result.**

***'''***

***Task 4: Suppose Basics = {3, 5} are given, try to find ways to construct the following target numbers:***

***i = 14. You might have noticed that the construction is impossible,***

***but the tool cannot conclude UNSAT. Try to think about adding new FOL\* class***

***and rules to help conclude UNSAT.***

***Hint1: Adding three odd numbers always yields an odd number.***

***Therefore, if Basics contains only ODD numbers, then all Numbers constructed are ODD.***

***You can try to define two classes ODD and EVEN as abstractions off Number,***

***and define their construction rules respectively.***

***'''***

*Basics = [3, 5]*

***'''***

***Solution 4:***

***'''***

***def*** *target\_transformation(target:int):*

***pass***

***def*** *check\_target\_with\_abstraction(target\_value):*

*target\_property = target\_transformation(target\_value)*

*solve(target\_property, proof\_mode=****True****)*

*check\_target\_with\_abstraction(79)*

*check\_target\_with\_abstraction(78)*

*check\_target\_with\_abstraction(129)*

*check\_target\_with\_abstraction(1100)*

*# we check the proof of UNSAT for 1100*

*UNSAT\_core, \_ = check\_and\_minimize(****"proof.txt"****,* ***"simply.txt"****)*

***for*** *r* ***in*** *UNSAT\_core:*

*print(r)*

*Task 1e. Rename Problem1.py as “Lab2P1-yourname.pt”. Email it to* [*chechik@cs.toronto.edu*](mailto:chechik@cs.toronto.edu) *with the subject line “Tutorial 2 Task 1 answer”.*

Problem 2:

We will model the Impossible Asylum problem (see <https://arxiv.org/abs/2112.02142>) using FOL\* and solve it using LEGOS.

**Description:**

*The last asylum Craig visited he found to be the most bizarre of all. This asylum was run by two doctors named Doctor Tarr and Professor Fether.*

*There were other doctors on the staff as well. Now, an inhabitant was called peculiar if he believed that he was a patient.*

*An inhabitant was called special if all patients believed he was peculiar and no doctor believed he was peculiar.*

*Inspector Craig found out that at least one inhabitant was sane and that the following condition held:*

*Condition C: Each inhabitant had a best friend in the asylum.*

*Moreover, given any two inhabitants, A and B, if A believed that B was special, then A’s best friend believed that B was a patient.*

*Shortly after this discovery, Inspector Craig had private interviews with Doctor Tarr and Professor Fether. Here is the interview with Doctor Tarr:*

*Craig: Tell me, Doctor Tarr, are all the doctors in this asylum sane?*

*Tarr: Of course they are!*

*Craig: What about the patients? Are they all insane?*

*Tarr: At least one of them is.*

*The second answer struck Craig as a surprisingly modest claim! Of course, if all the patients are insane, then it certainly is true that at least one is. But why was Doctor Tarr being so cautious? Craig then had his interview with Professor Fether, which went as follows:*

*Craig: Doctor Tarr said that at least one patient here is insane. Surely that is true, isn’t it?*

*Professor Fether: Of course it is true! All the patients in this asylum are insane! What kind of asylum do you think we are running?*

*Craig: What about the doctors? Are they all sane?*

*Professor Fether: At least one of them is.*

*Craig: What about Doctor Tarr? Is he sane?*

*Professor Fether: Of course he is! How dare you ask me such a question?*

*At this point, Craig realized the full horror of the situation! What was it?*

**The following claims are extracted from the description:**

1. Tarr is a doctor.

2. Fether is a doctor.

3. There are other doctors in the asylum.

4. By definition, an inhabitant A is peculiar if A believes that A is a patient.

5. By definition, an inhabitant A is special if all patients believe that A is peculiar and no doctor believes that A is peculiar.

6. At least one inhabitant is sane.

7. Condition C: Given any two inhabitants, A and B, if A believes that B is special, then A’s best friend believes that B is a patient.

8. Tarr believes that every doctor is sane.

9. Tarr believes that at least one patient is insane.

10. Fether believes that every patient is insane.

11. Fether believes that at least one doctor is sane.

12. Fether believes that Tarr is sane.

Let’s begin by expressing the model in FOL:

We assume that each person is represented by a unique ID of type natural number. We further assume the id of Fether is 0 and Tarr is 1:

Feather = 0

Tar = 1

the following first-order predicates:

Sane: ID -> Bool

Peculiar: ID -> Bool

Special: ID -> Bool

Doctor: ID -> Bool

Patient: ID -> Bool

and function

BestFriend: ID -> ID

We make use of the fact that a statement of the form “ *x believes P*” is true if and only if *x is sane if and only if P holds (a sane person x always has correct briefs P, and an insane one always has incorrect briefs )*. This is logically equivalent to saying “ is sane if and only if ,” which is represented by the expression Sane(x) ↔ P.

**Task 2.a:** *Model the following claims about the asylum in FOL:*

1. Tarr is a doctor.

2. Fether is a doctor.

3. There are other doctors in the asylum.

4. By definition, an inhabitant A is peculiar if A believes that A is a patient.

5. By definition, an inhabitant A is special if all patients believe that A is peculiar and no doctor believes that A is peculiar.

6. At least one inhabitant is sane.

7. Condition C: Given any two inhabitants, A and B, if A believes that B is special, then A’s best friend believes that B is a patient.

8. Tarr believes that every doctor is sane.

9. Tarr believes that at least one patient is insane.

10. Fether believes that every patient is insane.

11. Fether believes that at least one doctor is sane.

12. Fether believes that Tarr is sane.

**Task 2.b:** *Model the claims about the asylum in FOL\* (or translate them from FOL). First let’s define the signature.*

**Please first work on this part of the lab on paper. Once you are confident about your signature, please add it Problem2.py.**

**FetherID = Int(0)**

**TarID = Int(1)**

**'''**

**Task 1, create your signature here**

**'''**

**User = create\_action("User", []) # your attributes goes here**

**# more to go if needed**

Hint1: In general, predicate in FOL can be modeled as class in FOL\*. However, notice in this case, all the predicates take the same type of input (i.e., ID). Therefore, you might find it easier to create a Class: User that is unique for each ID. Predicates and functions in FOL can be modeled as attributes of a user (i.e., as if each object of User is a data entry in an SQL data schema where the id is the primary key).

**Task 2.c**  *Now lets model the claims about the asylum in FOL\* (or translate them from FOL). Please remember that the ID should be unique for each user, and that should be captured in an FOL\* constraint.*

**Please first work on this part on paper. Once you are confident about your encodings, add them to Problem2.py. Please do not run Problem2.py yet.**

**'''**

**Task 2, define your rule here**

**'''**

**def believe(x, facts):**

**pass**

***# Your encoding goes here***

solve(TRUE())

**Task 2.d:** *Solve the constraints to obtain an assignment of the Asylum or UNSAT. You can do this by running the python file Problem2.py and observing the result.*

Hint: avoid the common mistake: definitions of Peculiar and Special in Rule 3 and 4 should be interpreted as If and only if instead of just if.

**Task 2.e:** *Once Task 2.d returns UNSAT,**rerun the solver in the proof production mode by uncommenting the following lines at the end of the Problem2.py and rerunning the file.*

*solve(TRUE(), proof\_mode=****True****, unsat\_mode=****True****)*

*UNSAT\_core, \_ = check\_and\_minimize(****"proof.txt"****,* ***"simplified.txt"****)*

*print(****'\*'*** *\* 100)*

*print(****"UNSAT CORE"****)*

***for*** *i* ***in*** *UNSAT\_core:*

*print(str(i))*

*Check the proof and the UNSAT core. Add the output from the above print statement here:*

*'''*

*Task 4a: Put the UNSAT Core here.*

*'''*

*Now, let’s remove the constraint:*

Condition C: Given any two inhabitants, A and B, if A believes that B is special, then A’s best friend believes that B is a patient.

***Rerun Problem2.py with the constraint removed. Can you obtain a solution to the problem? If so, put it at the end of Problem2.py. Explain why this solution violates the removed constraint***

*'''*

*Task 4b: The satisfying solution goes here. Explain why the solution violates the removed constraint*

*'''*

*Task 2f. Rename Problem2.py as “Lab2P2-yourname.py”. Email it to* [*chechik@cs.toronto.edu*](mailto:chechik@cs.toronto.edu) *with the subject line “Tutorial 2 Task 2 answer”.*