COMP24412
Lab 1: Reasoning in FOL
Lab exercises & submission

Francisco Lobo and Joseph Razavi*
Academic session: 2023-24

The exercises of this assignment continue from the activities set for the lab sessions. If you haven't completed those, then go back and follow the Lab 1 session activities manual. In particular, it describes the JSON specification format for Vampire proofs that you must use in your solutions, and which is simply referred to as *proof spec* below.

In this part of the lab exercise your proof specs **must** only involve the clauses which are part of the saturation process. You should disregard all formulas Vampire considers during pre-processing and clausification.

As you experiment with features of Vampire that go beyond the given-clause algorithm, the saturation process will generate inferences using rules other than resolution. When completing resolution details in your proof specs, you'll **need to** assume that these advanced rules are all applied at the same time as resolution in the saturation process, i.e. when a clause from the passive set gets selected for activation and its inferences are calculated.

Exercise 1

Consider the following problem in FOL **with** equality.

$$\left\{\begin{array}{l} \forall x.\,(\mathrm{require}(x) \to \mathrm{require}(\mathrm{depend}(x))) \\ \mathrm{depend}(\mathrm{clang}) = \mathrm{llvm} \\ \mathrm{depend}(\mathrm{llvm}) = \mathrm{libc} \\ \mathrm{require}(\mathrm{clang}) \end{array}\right\} \models \mathrm{require}(\mathrm{libc})$$

In this exercise you'll use Vampire to build different proofs for this problem. Two proofs are considered different **if and only if** each involves a resolution step which doesn't appear in the other. Remember that every resolution step consists of 2 premises, 1 mgu, and 1 conclusion.

Translate the problem to TPTP and save your code as `deps.p` then use Vampire to obtain 3 proofs which are **mutually different** and satisfy the following criteria.

- Not all of your proofs can have the same equality proxy setting.
- Not all of your proofs can have the same selection setting.
- Not all premises are ground clauses.

You must save your proofs as `deps-a.json`, `deps-b.json`, and `deps-c.json` in the proof spec format. Each one of your proofs must include the details of a resolution step and corresponding state of the saturation algorithm, which shows that the proof is different from the other two. These must be supported by a corresponding map of formulas to indices, and specifications must also include the Vampire options used to generate the proof.

*The original version of this lab was developed by Giles Reger.

Mark scheme: There are 7 marks available for this exercise which are distributed as follows.

- 1 mark for your TPTP code,
  provided that your theory is equivalent to the reference solution
- 3 marks for correct tree specifications for your choice of proof options,
  depending on how many of the above criteria are satisfied
- 3 marks for resolution steps unique to your choice of proof options,
  depending on how many of their details are correct

Exercise 2

The TPTP Problem Library is a comprehensive repository of axiom sets for various domains. One of these is called `SET001-0.ax` and included in your GitLab repo. It axiomatises membership and subset predicates for Set Theory, and can be included in your code using the TPTP directive

```
include('SET001-0.ax').
```

Once you add this line, you can translate mathematical notation to TPTP as follows.

- $a \in U$ meaning that a is an element of the set U can be written as `member(a,u)`
- $U \subset V$ meaning that U is a strict subset of V can be written as `subset(u,v)`
- $U = V$ meaning that the sets U and V are equal can be written as `equal_sets(u,v)`

In this exercise you will use the `SET001-0.ax` axiom set to produce a proof of a mathematical result, which is relevant for the Description Logic part of the course. You are **only** allowed to extend the FOL signature given below by using well-formed TPTP **`definition`** declarations.

Task A: Develop a first-order theory **without** equality that for a binary relation R encodes the following definitions.

- The *direct image* of a set A across the relation R is defined by

  $$\mathrm{dirimg}(A) = \big\{ y \mid \exists x. (x \in A \ \land \ x \ R \ y) \big\}$$

  that is the set of all y's related by R to some x in A
- The *value restriction* to a set B of the relation R is defined by

  $$\mathrm{valres}(B) = \big\{ x \mid \forall y. (x \ R \ y \ \rightarrow \ y \in B) \big\}$$

  that is the set of all x's only related by R to y's in B

Your TPTP encoding must represent the binary relation R by a binary predicate `role`, the direct image operator by a function symbol `dirimg`, and the value restriction operator by a function symbol `valres`. Your theory needs to be saved as `sets.p` and should extend the signature of the `SET001-0.ax` axiom set with the specified encoding symbols.

Task B: Translate the following mathematical statement and add it as a conjecture to your code.

$$\forall A, B. \big(\mathrm{dirimg}(A) \subseteq B \ \leftrightarrow \ A \subseteq \mathrm{valres}(B)\big)$$

Note that the statement uses the (more common) reflexive subset relation $\subseteq$ and not the irreflexive subset relation $\subset$ which is defined in the `SET001-0.ax` axiom set.

Task C: Use Vampire to produce a proof of the above mathematical statement and record it `sets.json` in the proof spec format. You do not need to give details of any resolution step, but you must provide the Vampire options you used to prove the above mathematical statement. You should ensure that Vampire clausifies the problem correctly with these options, especially if your solution extends the logical signature. But as stated above your proof spec should only include the saturation process; moreover, you need only include the clauses that are relevant for the proof.

Mark scheme: There are 5 marks available for this exercise which are distributed as follows.

- 2 marks for the first-order TPTP encoding of the definitions in Task A, depending on the equivalence of your theory with the reference solution
- 2 marks for correct clausal form of the mathematical conjecture in Task B, depending on its equivalence with the reference solution
- 1 mark for a correct proof tree for your choice of options in Task C

Exploring model building

You should complete the activity in this section before attempting the final modelling exercise. The goal is to explore and understand what happens when Vampire cannot construct a refutation.

Consider the following modified version of the problem in the Lab 1 session activities manual.

$$\left\{ \begin{array}{l} \forall x.\,(\text{happy}(x) \leftrightarrow \exists y.\,(\text{loves}(x,y))) \\ \forall x.\,(\text{rich}(x) \rightarrow \text{loves}(x,\text{money})) \\ \text{happy}(\text{giles}) \end{array} \right\} \models \text{rich}(\text{giles})$$

Translate the problem to TPTP and saved it as `rich.p` (this should just involve copying and modifying the `greed.p` file).

```
$ run_vampire rich.p --bad_option off
% Running in auto input_syntax mode. Trying TPTP
% SZS status CounterSatisfiable for rich
% # SZS output start Saturation.
cnf(u14,axiom,
    ~rich(X0) | loves(X0,money)).

cnf(u16,negated_conjecture,
    ~rich(giles)).

cnf(u17,axiom,
    loves(giles,sK0(giles))).

cnf(u13,axiom,
    ~loves(X0,X1) | happy(X0)).

cnf(u15,axiom,
    happy(giles)).

cnf(u12,axiom,
    ~happy(X0) | loves(X0,sK0(X0))).

% # SZS output end Saturation.
% ------------------------------
% Version: Vampire 4.7 (commit )
% Linked with Z3 4.8.7.0
% Termination reason: Satisfiable

% Memory used [KB]: 511
% Time elapsed: 0.001 s
% ------------------------------
% ------------------------------
```

Figure 1: Solving the `rich.p` problem

If we try to use ordered resolution on this new problem, Vampire gives the output shown in Figure 1. It reports a `CounterSatisfiable` status because a refutation was not found. Note that the command argument `--bad_option off` was added simply to disable warnings that can be safely ignored about some of the options in `run_vampire`.

The reason for the `CounterSatisfiable` status was that the saturation algorithm ended without deriving the empty clause, and so Vampire outputs the saturated set of **`cnf`** clauses. This means there are models that satisfy the axioms **and also** the negated conjecture, which is why Vampire terminates with a `Satisfiable` set of clauses.

If this is confusing then you should convince yourself that: The problem (axioms and conjecture) will be *counter-satisfiable* precisely when the set of axioms and negated conjecture is *satisfiable*.

Before we accept Vampire's output status and conclude that the negated conjecture is consistent with the axioms of `rich.p`, it's reasonable to ask if changing, say, the clause and literal selection strategy that Vampire uses would find something more about this new problem. . . Possibly even reveal that there is a refutation after all!

You can check this using unordered resolution with the command

```
$ run_vampire rich.p -s 0
```

Selecting of all literals in resolution ensures that Vampire computes all possible inferences for the problem. (You should trace the saturation process using `--show_active on --show_new on` to confirm this.)

The result of unordered resolution is a larger saturated set, but you'll find that the status for the problem continues to be `CounterSatisfiable`. This example exhibits the fact that ordered resolution is *refutationally complete*, meaning that if refutation is possible then it will be found. This is an important property that we will discuss in the lectures.

We've convinced ourselves that `rich.p` is counter-satisfiable, but to prove this we still need to construct a model of the axioms that also satisfies the negated conjecture. Vampire can help us with model building if we disable the equality proxy and select `fmb` as the saturation algorithm. This *finite model building* process works by translating the problem of finding a model of a particular size into a SAT problem and iterating this process for larger model sizes.

Try running the command

```
$ run_vampire rich.p -ep off -sa fmb
```

You'll get the output shown in Figure 2 which gives a *finite model* making the axioms true but the conjecture false.

Vampire uses typed first-order formulas to give finite models, so it outputs TPTP code in **`tff`** form. This *typed logic* is outside the scope of this course unit, but you should still be able to read and understand all the **`axiom`** declarations, while mostly ignoring the **`type`** declarations.

Let's go through the output in Figure 2. You first read that Vampire tries and succeeds in finding a model of size 1, if that wasn't possible Vampire would try to make a model of size 2, and so on. . . Since a model exists, Vampire reports a `CounterSatisfiable` status in line 5, and then the `FiniteModel` is written between lines 6 and 38.

The signature of our problem consists of

- 2 constant symbols `money` and `giles`
- 2 unary predicates `happy` and `rich`
- 1 binary predicate `loves`

for which we must give an interpretation. The domain of the model corresponds to the type `$i` and the booleans correspond to the type `$o`. The domain has a just 1 element and Vampire has identified it with the symbol `money`, lines 9 to 12 assert that everything in the domain is this element. The `giles` constant is also mapped to `money` in line 22.

```
1 $ run_vampire rich.p --bad_option off -ep off -sa fmb
2 % Running in auto input_syntax mode. Trying TPTP
3 TRYING [1]
4 Finite Model Found!
5 % SZS status CounterSatisfiable for rich
6 % SZS output start FiniteModel for rich
7 tff(declare_$i,type,$i:$tType).
8 tff(declare_$i1,type,money:$i).
9 tff(finite_domain,axiom,
10       ! [X:$i] : (
11          X = money
12       ) ).
13 
14 tff(declare_bool,type,$o:$tType).
15 tff(declare_bool1,type,fmb_bool_1:$o).
16 tff(finite_domain,axiom,
17       ! [X:$o] : (
18          X = fmb_bool_1
19       ) ).
20 
21 tff(declare_giles,type,giles:$i).
22 tff(giles_definition,axiom,giles = money).
23 tff(declare_happy,type,happy: $i > $o ).
24 tff(predicate_happy,axiom,
25           happy(money)
26 
27 ).
28 
29 tff(declare_loves,type,loves: $i * $i > $o ).
30 tff(predicate_loves,axiom,
31           loves(money,money)
32 
33 ).
34 
35 tff(declare_rich,type,rich: $i > $o ).
36 tff(predicate_rich,axiom,
37           ~rich(money)
38 
39 ).
40 
41 % SZS output end FiniteModel for rich
42 % ------------------------------
43 % Version: Vampire 4.7 (commit )
44 % Linked with Z3 4.8.7.0
45 % Termination reason: Satisfiable
46 
47 % Memory used [KB]: 4989
48 % Time elapsed: 0.0000 s
49 % ------------------------------
50 % ------------------------------
```

Figure 2: Model for the `rich.p` problem

The interpretation of the predicate symbols is given as axioms declaring what is true in the model, in particular

- `happy(money)` in lines 23 to 26
- `~rich(money)` in lines 34 to 36
- `loves(money,money)` in lines 29 to 31

In other words, `happy` is always true, `rich` is always false, and `loves` is always true. You should convince yourself that a structure with these characteristics does indeed satisfy the axioms but not the conjecture of the `rich.p` problem.

Of course, a model with a single element for the problem is not very interesting. Also, it doesn't distinguish between the constants `money` and `giles` which we instinctively want.

But if we want Vampire to make models that distinguish between constants, we must add axioms which declare all constants to be pairwise different. This can be tedious so as a shortcut for it TPTP has the keyword `$distinct`. For example, if you add

```
fof(x, axiom, $distinct(money,giles,joe,francisco)).
```

to the problem and re-run the finite model building process, Vampire will construct a model with size 4 that proves `rich.p` is counter-satisfiable. You should make sure that you can reconstruct the model from the resulting `tff` output.

Exercise 3

The TPTP Problem Library contains a large number of ready-made problems in various domains. One of these is called `PUZ031+1.p` and included in your GitLab repo. It formalises a logical puzzle known as *Schubert's Steamroller* which in the 1980s was famously too difficult to solve, due to state space explosion during the saturation process. Indeed, if you try to solve the puzzle with unordered resolution

```
$ run_vampire PUZ031+1.p -s 0
```

you'll find that Vampire times out! You should compare this with the time taken to find a refutation using ordered resolution. Read the Schubert's Steamroller puzzle in `PUZ031+1.p` and study how the puzzle was modelled in first-order logic. In this exercise you'll do the same for a different puzzle, which you'll then be able to solve with the help of Vampire.

Task A: The following puzzle is taken from the book "Arithmetical, Geometrical and Combinatorial Puzzles from Japan" by Tadao Kitazawa (page 111).

> *In the Pets' Quality Resort, rooms #1 to #6 are in a row in that order on the ground floor. Rooms #7 to #12 are directly above rooms #1 to #6 respectively. Each room can accommodate one animal, and has its own light. At night, an animal who is not nervous turns off the light and sleeps well. An animal who is nervous leaves the light on and sleeps intermittently.*
>
> *During the weekend, only the ground floor is open. Room #6 is occupied by the pet hamster of the part-time caretaker. The hamster is never nervous. Each of rooms #1 to #5 is occupied either by a dog or a cat. All five check out after a one-night stay.*
>
> ***Problem 10***
>
> *On Sunday, a dog is nervous if and only if there are other dogs in both adjacent rooms. A cat is nervous if and only there is another cat in at least one adjacent room. It is observed that only one room remains lit. How many cats are there in the P.Q.R. ?*

Develop a first-order theory that formalises Kitazawa's Problem 10, as written above[1]. Your signature must include the following symbols with the given semantics.

- 6 constant symbols `r1`, `r2`, `r3`, `r4`, `r5`, `r6`, which represent the ground floor rooms
- 1 binary predicate `next` representing the ordering of the rooms, such that `next(X,Y)` means that `Y` is the room after `X`
- 3 unary predicates `cat`, `dog`, `hamster` for the possible animals that can occupy a room, so that `cat(X)` means that room `X` is occupied by a cat, etc.
- 1 unary predicate `lit` that is only true for rooms with the light on

Do not represent the first-floor rooms of the P.Q.R. in your formalisation. Your TPTP code must be saved as `pets.p`. As in the previous exercise, you are **only** allowed to extend the FOL signature given above by using well-formed TPTP **`definition`** declarations.

1. In May 2022, a modified version of this puzzle appeared as part of Alex Bellos's Monday puzzle in The Guardian newspaper.

Task B: Using Vampire or otherwise give all the solutions of the puzzle from Task A, as finite models in **`tff`** form. You must save these in as many letter indexed files `pets-a.tff`, `pets-b.tff`, . . . , `pets-z.tff`, as you need. There are less than 26 solutions so you shouldn't have to create all of these files. Each of your files must give a unique solution. Only the **`tff`** axiom declarations in your finite models will be tested.

Mark scheme: There are 8 marks available for this exercise which are distributed as follows.

- 2 marks for the correct semantics of the TPTP signature,
  depending on your code being consistent with various implied properties;
  for example, we may test that your predicate `next` isn't transitive
- 3 marks for the first-order TPTP theory for the puzzle,
  depending on the equivalence of your theory with the reference solution
- 3 marks for all the **`tff`** finite models that solve the puzzle,
  which will be tested for consistency with the reference solution

Submission

Please follow the `README.md` instructions in your `COMP24412_2023` GitLab repo. Refresh the files of your `lab1` branch and develop your solution to the lab exercise. The solution to this lab exercise consists of the following files.

- Exercise 1: `deps.p`, `deps-a.json`, `deps-b.json`, and `deps-c.json`
- Exercise 2: `sets.p` and `sets.json`
- Exercise 3: `pets.p` and `pets-a.tff`, `pets-b.tff`, . . .

These must be submitted to your GitLab repo and tagged as `lab1_sol`. You will find a skeleton proof spec file called `template.json` in your repo, which you should adapt for your solutions to Exercises 1 and 2. The `README.md` instructions that accompany the lab files include the `git` commands necessary to commit, tag, and then push **both** the commit and the tag to your `COMP24412_2023` GitLab repo. Further instructions on coursework submission using GitLab can be found in the CS Handbook, including how to change a `git` tag after pushing it.

The deadline for submission is **09:00 on Tuesday 5th March**. The lab will be **auto-marked** offline. The auto-marker program will download your submission from GitLab and test it against reference solutions.

- The equivalence of your TPTP code with the reference solution will be tested by clausifying both and using Vampire to confirm that each clause is entailed by the other theory. Depending on how you encode the problems Vampire may **or may not** obtain these proofs within the default time limit. The consistency of your `tff` finite models will be tested in the same way.
- You proof specs will be loaded as the `spec2latex.py` script does to test their correctness.

Your TPTP code will be tested by running the same Vampire version as the one installed on the Kilburn lab machines. Vampire will be run using the options you define in your proof specs, but the auto-marker will always **reset** the `--time_limit` option to the default of 1 minute. You need to make sure that your TPTP code *solves* the exercises within this time limit.

Important: Your TPTP `definition` declarations will be tested with the same logic as Vampire's `-updr on` setting to make sure they are well-formed. If these aren't well-formed, they will be considered regular axioms (which the reference solution is unlikely to validate).

Important: Extending the logical signature of Exercises 2 and 3 can be used to focus the proof search and ensure Vampire terminates within the time limit. But note that additional predicates may also cause the proof search to diverge.

Important: Every file in your submission should only contain printable ASCII characters. If you include other Unicode characters, for example by copying and then pasting code from the PDF of the lab manuals, then the auto-marker is likely to reject your files.