# ITP Exercise 2

**due Friday 5th May**

## 1 Self-Study

### 1.1 Emacs

If you don't know Emacs well, familiarise yourself with its basic usage. Learn the key-combinations for common operations like opening a file, saving current buffer, closing buffer, switching between buffers, searching in a file, copy and paste text etc. You might consider printing the *Emacs Reference Card*[1] and putting it next to your computer.

### 1.2 HOL Documentation

Familiarise yourself with how to get help about HOL.

- Build the various documentations in directory `Manual`.
- Have a brief look at the various manuals in order to understand where which kind of information can be found.
- The lectures will cover the logic foundations of the HOL theorem prover only very briefly and lightly. If you are interested in more details, have a look at the *Logic* manual. This is purely optional.
- Familiarise yourself with the different ways to access the reference manual. As an example read up on `MATCH_MP` in the HTML reference manual, the PDF reference manual and the in-system help (type `help "MATCH_MP"`).
- Familiarise yourself with the different printing switches of HOL, in particular the ones in hol-mode's menu. Learn how to turn Unicode-output on/off, how to show assumptions of theorems and how to show type annotations.
- Use `DB.match` and `DB.find` to find theorems stating `A /\ A = A`. Use both the emacs-mode and the SML REPL. Look at the interface of `DB`.

### 1.3 Holmake

Learn about `Holmake` by reading description manual sections 7.3 - 7.3.4.

### 1.4 Constructing Terms and Forward Proofs

To deepen the knowledge about how to construct terms, how to program in HOL and how to perform forward proofs, please look at the following HOL modules: `FinalThm.sml`, `FinalTerm.sml`, `FinalType.sml`, `Drule.sig`, `Psyntax.sig`, `boolSyntax.sig`, `Lib.sig`.

[1] `https://www.gnu.org/software/emacs/refcards/pdf/refcard.pdf`

# 2 Terms

## 2.1 Free and Bound Vars

List the free variables of the following terms:

- `(\x. 2 + (7 * x) + y) z`
- `x + y + 2`
- `!x. x + 1 > x`
- `?x. x = y + 2`

## 2.2 Alpha Equivalence

Are the following pairs of terms alpha-equivalent? A simple mark on the sheet is a sufficient answer. Also take two colors and mark all occurences of free vars in one color and all occourences of bound vars in the other. Assume that `x`, `y`, `z`, `a` and `b` are variables.

| | |
|---|---|
| `\x. x` | `\y. y` |
| `(\x. x) a` | `(\y. y) a` |
| `(\x. x) a` | `(\y. y) b` |
| `(\x. x)` | `(\x. y)` |
| `(\x y. x /\ y)` | `(\y x. x /\ y)` |
| `(\x y. x /\ y) a a` | `(\y x. x /\ y) a a` |
| `a /\ b` | `a /\ b` |
| `!x. x + 1 > x` | `!y. y + 1 > y` |
| `?x. x = y + 2` | `?x. x = z + 2` |
| `!y. ?x. x = y + 2` | `!z. ?x. x = z + 2` |

## 2.3 Constructing Terms

Write a SML function `mk_imp_conj_term : int -> term` that constructs for inputs $n$ greater 1 the term `!A1 ... An. A1 ==> ... ==> An ==> (A1 /\ ... /\ An)`. If $n$ is not greater one, a `HOL_ERR` exception (use `failwith`). You might want to read up on `boolSyntax` for this exercise. You can use list-make-functions like `mk_list_conj`, but also use non-list ones.

# 3 Basic Forward Proofs

## 3.1 Commutativity of Conjunction

Prove the lemma `!A B. (A /\ B) <=> (B /\ A)` using only inferences presented in the lecture.

## 3.2 Simplifying Conjunction

Prove the lemmas `!A. (A /\ ~A) <=> F` and `!A B. (A /\ ~A) /\ B <=> F`.

# 4 Writing Own Automation

## 4.1 Implications between Conjunctions

Write a function `show_big_conj_imp : term -> term -> thm` that assumes that both terms are conjunctions and tries to prove that the first one implies the second one. It should be clever enough to handle `T` and `F`. `show_big_conj_imp ``a /\ (b /\ a) /\ c`` ``c /\ T /\ a``` for example should succeed with `|- (a /\ (b /\ a) /\ c) ==> (c /\ T /\ a)`. It should also be able to show `|- (a /\ F) /\ c ==> d`. If the implication cannot be shown, the function `show_big_conj_imp` should raise `HOL_ERR`.

For this exercise it might be useful to read up on `Term.compare` and the red-black sets and maps in directory `portableML`.

## 4.2 Equivalences between Conjunctions

Use the function `show_big_conj_imp` to now define a function `show_big_conj_eq : term -> term -> thm` that tries to shows the equivalence between the input terms. If both input terms are alpha-equivalent, it should raise an `UNCHANGED` exception. If the equivalence cannot be proved, a `HOL_ERR` exception should be raised.

## 4.3 Duplicates in Conjunctions

Use the function `show_big_conj_eq` to implement a conversion `remove_dups_in_conj_CONV` that replaces duplicate appearances of a term in a large conjunction with `T`. Given the term

```
a /\ (b /\ a) /\ c /\ b /\ a
```

it should for example return the theorem

```
|- (a /\ (b /\ a) /\ c /\ b /\ a) = (a /\ (b /\ T) /\ c /\ T /\ T).
```

. If no duplicates are found, `UNCHANGED` should be raised. If the input is not of type `bool`, a `HOL_ERR` should be raised.

## 4.4 Contradictions in Conjunctions

Use the function `show_big_conj_eq` to implement a conversion `find_contr_in_conj_CONV` that searches for terms and their negations in a large conjunction. If such contradictions are found, the term should be converted to `F`. Given the term

```
a /\ (b /\ ~a) /\ c
```

it should for example return the theorem

```
|- (a /\ (b /\ ~a) /\ c) = F.
```

. If no contradictions are found, `UNCHANGED` should be raised. If the input is not of type `bool`, a `HOL_ERR` should be raised.

# 5 Squabbling Philosophers

Recently keen historians were finally able to deduce where the less well-known ancient philosophers Platon, Diogenes and Euklid came from (see background-questionnaire). However, in order to avoid being embarrassed by announcing some wrong result, they asked you to check their reasoning using HOL. Can you help and show that Platon indeed came from Sparta?

## 5.1 Download and Compile

Get the file `philScript.sml` from the exercise repository[2]. Compile it with `Holmake` to get a theory file. Read `philTheory.sig`.

## 5.2 Proof

Open the theory `philTheory` and prove `Sp platon`. This is a simple first order logic problem. Therefore automated methods like resolution can solve it easily. HOL has such methods build in in the form of e. g. the resolution based prover `METIS`. For example,

```
METIS_PROVE [PHIL_KNOWLEDGE] ``Sp platon``
```

would already prove it. However, for learning, let us prove it via a low-level forward proof.

- Using the lemma `IMP_MONO` and the inference rules `MATCH_MP`, `SPEC` and `IMP_TRANS` show the lemma `|- ~(W p) ==> Sp p`.
- Similarly show `|- ~(B p) ==> At p`.
- Assume `At platon` and using this show the lemma `[At platon] |- F` with `MP` and `MATCH_MP`. You will need many steps and many different lemmata.
- Using `DISCH`, `NOT_INTRO` and `MATCH_MP` show `Sp platon`.

Don't forget to turn printing of assumptions on in HOL or you will have a hard time figuring out what is going on.

[2] `https://gits-15.sys.kth.se/tuerk/ITP-exercises`