This document is based on material from the “Interactive Theorem Proving Course” by Thomas Tuerk (`https://www.thomas-tuerk.de`):
`https://github.com/thtuerk/ITP-course`

This document includes additions by:

- Pablo Buiras (`https://people.kth.se/~buiras/`)
- Karl Palmskog (`https://setoid.com`)

ITPPV Homework 2

due 23:59 CET, Tuesday February 4, 2020

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 HOL4 Documentation

Familiarise yourself with how to get help about HOL4.

- Build the various documentations in the directory `Manual`. For this, call `make` in the directory `HOL-HOME/Manual`. For building the manuals, `hol` and `Holmake` need to run. Therefore, make sure `HOL-HOME/bin` is in your PATH.
- 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 HOL4 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 HOL4, 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 the description manual[2], section 8.3.

1.4 Constructing Terms and Forward Proofs

To deepen knowledge about how to construct terms, how to program in HOL4, and how to perform forward proofs, please look at the following HOL4 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] `http://sourceforge.net/projects/hol/files/hol/kananaskis-13/kananaskis-13-description.pdf/download`

2 Terms

2.1 Free and Bound Variables

List the free variables of the following terms:

1.) `(\x. 2 + (7 * x) + y) z`

2.) `x + y + 2`

3.) `!x. x + 1 > x`

4.) `?x. x = y + 2`

2.2 Alpha Equivalence

Are the following pairs of terms alpha-equivalent? A simple “yes” or “no” is a sufficient answer. Also, indicate occurrences of free and bound variables in each term, e.g., by highlighting them with different colors. Assume that `x`, `y`, `z`, `a`, and `b` are variables.

1.)	`\x. x`	`\y. y`
2.)	`(\x. x) a`	`(\y. y) a`
3.)	`(\x. x) a`	`(\y. y) b`
4.)	`(\x. x)`	`(\x. y)`
5.)	`(\x y. x /\ y)`	`(\y x. x /\ y)`
6.)	`(\x y. x /\ y) a a`	`(\y x. x /\ y) a a`
7.)	`a /\ b`	`a /\ b`
8.)	`!x. x + 1 > x`	`!y. y + 1 > y`
9.)	`?x. x = y + 2`	`?x. x = z + 2`
10.)	`!y. ?x. x = y + 2`	`!z. ?x. x = z + 2`

2.3 Constructing Terms

Write an SML function `mk_imp_conj_term : int -> term` that constructs, for inputs n greater than 0, the term `!A1 ... An. A1 ==> ... ==> An ==> (A1 /\ ... /\ An)`. If n is 0 or less, raise a `HOL_ERR` exception (use `failwith`). You might want to read up on `boolSyntax` for this task. You can use list-make functions such as `mk_list_conj`, but also use non-list functions.

3 Basic Forward Proofs

3.1 Commutativity of Conjunction

Prove the theorem `!A B. (A /\ B) <=> (B /\ A)` using only inference rules presented in lecture 2.

3.2 Simplifying Conjunction

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

4 Writing Your 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`. For example,

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

should succeed with `|- (a /\ (b /\ a) /\ c) ==> (c /\ T /\ a)`. It should also be able to prove `(a /\ F) /\ c ==> d`. If the implication cannot be shown, the function 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 the directory `portableML`.

4.2 Equivalences between Conjunctions

Use the function `show_big_conj_imp` to 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

```
|- (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

```
|- (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. However, in order to avoid being embarrassed by announcing the wrong result, they asked you to check their reasoning using HOL4. Can you help them and show that Platon indeed came from Sparta?

5.1 Download and Compile

Get the file `philScript.sml` from the homework repository[3] . Compile it with `Holmake` to get a theory file. Read `philTheory.sig`.

5.2 Write the 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. HOL4 has such methods built 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 purposes, prove it via a low-level forward proof.

a.) Using the theorem `MONO_NOT` and the inference rules `MATCH_MP`, `SPEC`, and `IMP_TRANS`, show that `~(W p) ==> Sp p`.

b.) Similarly, show `~(B p) ==> At p`.

c.) Assume `At platon`, and using this show `[At platon] |- F` with `MP` and `MATCH_MP`. You will need many steps and many different theorems along the way.

d.) Using `DISCH`, `NOT_INTRO`, and `MATCH_MP`, show `Sp platon`.

Don’t forget to turn on printing of assumptions in HOL4, or you will have a hard time figuring out what is going on: `show_assums := true;`

You may also find it useful to turn on printing of types of terms: `show_types := true;`

[3] https://github.com/kth-step/itppv-course/blob/master/homeworks/hw2-supplementary/philScript.sml