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)

Interactive Theorem Proving and Program Verification
Lecture 2

Pablo Buiras and Karl Palmskog

Academic Year 2019/20, Period 3–4

Based on slides by Thomas Tuerk

Part V

Basic HOL4 Usage

HOL4 Technical Usage Issues

- practical issues are discussed outside of lectures
  - details on installing HOL4
  - which key-combinations to use in hol-mode for Emacs
  - detailed signatures of libraries and theories
  - all parameters and options of certain tools
  - ...
- mentioned in homeworks sometimes
  - tasks to read some documentation
  - provides examples
  - lists references where to get additional information
- if you have problems, ask lecturers (buiras@kth.se, palmskog@kth.se)
- covered only very briefly in lectures

Installing HOL4

- website: `https://hol-theorem-prover.org`
- HOL4 supports two SML implementations
  - Moscow ML (`http://mosml.org`)
  - PolyML (`http://www.polyml.org`)
- we use only PolyML 5.8 in this course
- please use emacs with
  - hol-mode
  - sml-mode
  - hol-unicode, if you want to type Unicode
- please install the Kananaskis 13 release
- documentation found on HOL4 website and with sources

General Architecture

- HOL4 is a collection of SML modules
- starting HOL4 starts a SML Read-Eval-Print-Loop (REPL) with
  - some HOL4 modules loaded
  - some default modules opened
  - an input wrapper to help parsing terms called `unquote`
- `unquote` provides special quotes for terms and types
  - implemented as input filter
  - ``` ``my-term`` ``` becomes `Parse.Term [QUOTE "my-term"]`
  - ``` ``:my-type`` ``` becomes `Parse.Type [QUOTE ":my-type"]`
- main interfaces
  - **emacs** (used in this course)
  - vim
  - bare shell

Filenames

- `*Script.sml` — HOL4 proof script file
  - script files contain definitions and proof scripts
  - executing them results in HOL4 searching and checking proofs
  - this might take very long
  - resulting theorems are stored in `*Theory.{sml|sig}` files
- `*Theory.{sml|sig}` — HOL4 theory
  - auto-generated by corresponding script file
  - load quickly, because they don't search/check proofs
  - do not edit theory files
- `*Syntax.{sml|sig}` — syntax libraries
  - contain syntax related functions
  - i. e. functions to construct and destruct terms and types
- `*Lib.{sml|sig}` — general libraries
- `*Simps.{sml|sig}` — simplifications
- `selftest.sml` — selftest for current directory

HOL4 Project Version Control Repository Guidelines

- ignore `*Theory.sml` and `*Theory.sig`
- ignore the directories `.HOLMK` and `.hollogs`
- commit all custom `*.sml` and `*.sig` files
- don't forget `*Script.sml` files and **Holmakefile**

HOL4 Release Directory Structure

- `bin` — HOL4 binaries
- `src` — HOL4 sources
- `examples` — HOL4 examples
  - interesting projects by various people
  - examples owned by their developer
  - coding style and level of maintenance differ a lot
- `help` — sources for reference manual
  - after compilation home of reference HTML page
- `Manual` — HOL4 manuals
  - Tutorial
  - Description
  - Reference (PDF version)
  - Interaction
  - Quick (cheat pages)
  - Style-guide
  - ...

Unicode

- HOL4 supports both Unicode and pure ASCII input and output
- advantages of Unicode compared to ASCII
  - easier to read (good fonts provided)
  - no need to learn special ASCII syntax
- disadvantages of Unicode compared to ASCII
  - harder to type (even with `hol-unicode.el`)
  - less portable between systems
- whether you use Unicode is highly a matter of personal taste
- HOL4's policy
  - no Unicode in HOL4's source directory `src`
  - Unicode in examples directory `examples` is fine
- we strongly recommend turning Unicode output off
  - this simplifies learning the ASCII syntax
  - no need for special fonts
  - it is easier to copy and paste terms from HOL4's output

Where to find help?

- reference manual
  - available as HTML pages, single PDF file and in-system help
- description manual
- style guide (still under development)
- HOL4 website (`https://hol-theorem-prover.org`)
- mailing-list `hol-info`
- `DB.match` and `DB.find`
- `*Theory.sig` and `selftest.sml` files
- ask the lecturers (buiras@kth.se, palmskog@kth.se)

Part VI

Forward Proofs

Kernel too detailed

- we already discussed the HOL Logic
- the kernel itself does not even contain basic logic operators
- usually one uses a much higher level of abstraction
  - many operations and datatypes are defined
  - high-level derived inference rules are used
- let's now look at this more common abstraction level

Common Terms and Types

	Unicode	ASCII
type vars	α, β, ...	`'a, 'b, ...`
type annotated term	`term:type`	`term:type`
true	`T`	`T`
false	`F`	`F`
negation	`¬b`	`~b`
conjunction	`b1 ∧ b2`	`b1 /\ b2`
disjunction	`b1 ∨ b2`	`b1 \/ b2`
implication	`b1 ⟹ b2`	`b1 ==> b2`
equivalence	`b1 ⟺ b2`	`b1 <=> b2`
inequality	`v1 ≠ v2`	`v1 <> v2`
universal quantification	`∀x. P x`	`!x. P x`
existential quantification	`∃x. P x`	`?x. P x`
Hilbert's choice	`@x. P x`	`@x. P x`

There are similar restrictions to constant and variable names as in SML.
HOL4 specific: don't start variable names with an underscore

Syntax conventions

- common function syntax
  - prefix notation, e. g. `SUC x`
  - infix notation, e. g. `x + y`
  - quantifier notation, e. g. `∀x. P x` means `(∀) (λx. P x)`
- infix and quantifier notation can be turned into prefix notation
  Example: `(+) x y` and `$+ x y` are the same as `x + y`
- quantifiers of the same type don't need to be repeated
  Example: `∀x y. P x y` is short for `∀x. ∀y. P x y`
- there is special syntax for some functions
  Example: `if c then v1 else v2` is nice syntax for `COND c v1 v2`
- associative infix operators are usually right-associative
  Example: `b1 /\ b2 /\ b3` is parsed as `b1 /\ (b2 /\ b3)`

Creating Terms

Term Parser

Use special quotation provided by `unquote`.

Operator Precedence

It is easy to misjudge the binding strength of certain operators. When in doubt, use parentheses.

Use Syntax Functions

Terms are just SML values of type `term`. You can use syntax functions (usually defined in `*Syntax.sml` files) to create them.

Creating Terms II

Parser	Syntax Funs	
``:bool``	`mk_type ("bool", [])` or `bool`	type of Booleans
``T``	`mk_const ("T", bool)` or `T`	term true
``~b``	`mk_neg (`	negation of
	`  mk_var ("b", bool))`	Boolean var b
``... /\ ...``	`mk_conj (..., ...)`	conjunction
``... \/ ...``	`mk_disj (..., ...)`	disjunction
``... ==> ...``	`mk_imp (..., ...)`	implication
``... = ...``	`mk_eq (..., ...)`	equality
``... <=> ...``	`mk_eq (..., ...)`	equivalence
``... <> ...``	`mk_neg (mk_eq (..., ...))`	negated eq.

Inference Rules for Equality

$$\frac{}{\vdash t = t}\ \text{REFL}$$

$$\frac{\Gamma \vdash s = t}{\Gamma \vdash t = s}\ \text{GSYM}$$

$$\frac{\begin{array}{c}\Gamma \vdash s = t \\ x \textit{ not free in } \Gamma\end{array}}{\Gamma \vdash \lambda x.\ s = \lambda x. t}\ \text{ABS}$$

$$\frac{\begin{array}{c}\Gamma \vdash s = t \\ \Delta \vdash t = u\end{array}}{\Gamma \cup \Delta \vdash s = u}\ \text{TRANS}$$

$$\frac{\begin{array}{c}\Gamma \vdash s = t \\ \Delta \vdash u = v \\ \textit{types fit}\end{array}}{\Gamma \cup \Delta \vdash s(u) = t(v)}\ \text{MK\_COMB}$$

$$\frac{\Gamma \vdash p \Leftrightarrow q \qquad \Delta \vdash p}{\Gamma \cup \Delta \vdash q}\ \text{EQ\_MP}$$

$$\frac{}{\vdash (\lambda x.\ t)v = t[v/x]}\ \text{BETA\_CONV}$$

Inference Rules for free Variables

$$\frac{\Gamma[x_1, \ldots, x_n] \vdash p[x_1, \ldots, x_n]}{\Gamma[t_1, \ldots, t_n] \vdash p[t_1, \ldots, t_n]} \ \mathrm{INST}$$

$$\frac{\Gamma[\alpha_1, \ldots, \alpha_n] \vdash p[\alpha_1, \ldots, \alpha_n]}{\Gamma[\gamma_1, \ldots, \gamma_n] \vdash p[\gamma_1, \ldots, \gamma_n]} \ \mathrm{INST\_TYPE}$$

Inference Rules for Implication

$$\frac{\Gamma \vdash p \Longrightarrow q \quad \Delta \vdash p}{\Gamma \cup \Delta \vdash q}\ \text{MP, MATCH\_MP}$$

$$\frac{\Gamma \vdash p}{\Gamma - \{q\} \vdash q \Longrightarrow p}\ \text{DISCH}$$

$$\frac{\Gamma \vdash p = q}{\Gamma \vdash p \Longrightarrow q \quad \Gamma \vdash q \Longrightarrow p}\ \text{EQ\_IMP\_RULE}$$

$$\frac{\Gamma \vdash q \Longrightarrow p}{\Gamma \cup \{q\} \vdash p}\ \text{UNDISCH}$$

$$\frac{\Gamma \vdash p \Longrightarrow q \quad \Delta \vdash q \Longrightarrow p}{\Gamma \cup \Delta \vdash p = q}\ \text{IMP\_ANTISYM\_RULE}$$

$$\frac{\Gamma \vdash p \Longrightarrow \mathsf{F}}{\Gamma \vdash {\sim}p}\ \text{NOT\_INTRO}$$

$$\frac{\Gamma \vdash {\sim}p}{\Gamma \vdash p \Longrightarrow \mathsf{F}}\ \text{NOT\_ELIM}$$

$$\frac{\Gamma \vdash p \Longrightarrow q \quad \Delta \vdash q \Longrightarrow r}{\Gamma \cup \Delta \vdash p \Longrightarrow r}\ \text{IMP\_TRANS}$$

Inference Rules for Conjunction / Disjunction

$$\frac{\Gamma \vdash p \qquad \Delta \vdash q}{\Gamma \cup \Delta \vdash p \ \wedge \ q} \ \mathrm{CONJ}$$

$$\frac{\Gamma \vdash p \ \wedge \ q}{\Gamma \vdash p} \ \mathrm{CONJUNCT1}$$

$$\frac{\Gamma \vdash p \ \wedge \ q}{\Gamma \vdash q} \ \mathrm{CONJUNCT2}$$

$$\frac{\Gamma \vdash p}{\Gamma \vdash p \ \vee \ q} \ \mathrm{DISJ1}$$

$$\frac{\Gamma \vdash q}{\Gamma \vdash p \ \vee \ q} \ \mathrm{DISJ2}$$

$$\frac{\begin{array}{c}\Gamma \vdash p \vee q \\ \Delta_1 \cup \{p\} \vdash r \\ \Delta_2 \cup \{q\} \vdash r\end{array}}{\Gamma \cup \Delta_1 \cup \Delta_2 \vdash r} \ \mathrm{DISJ\_CASES}$$

Inference Rules for Quantifiers

$$\frac{\Gamma \vdash p \qquad x \text{ not free in } \Gamma}{\Gamma \vdash \forall x.\ p}\ \mathrm{GEN}$$

$$\frac{\Gamma \vdash \forall x.\ p}{\Gamma \vdash p[u/x]}\ \mathrm{SPEC}$$

$$\frac{\Gamma \vdash p[u/x]}{\Gamma \vdash \exists x.\ p}\ \mathrm{EXISTS}$$

$$\frac{\begin{array}{c}\Gamma \vdash \exists x.\ p \\ \Delta \cup \{p[u/x]\} \vdash r \\ u \text{ not free in } \Gamma, \Delta, p \text{ and } r\end{array}}{\Gamma \cup \Delta \vdash r}\ \mathrm{CHOOSE}$$

Forward Proofs

- axioms and inference rules are used to derive theorems
- this method is called **forward proof**
  - one starts with basic building blocks
  - one moves step by step forward
  - finally the theorem one is interested in is derived
- one can also implement custom proof tools

Forward Proofs — Example I

Let's prove $\forall p.\ p \Longrightarrow p$.

```
val IMP_REFL_THM = let
  val tm1 = ``p:bool``;
  val thm1 = ASSUME tm1;
  val thm2 = DISCH tm1 thm1;
in
  GEN tm1 thm2

end

fun IMP_REFL t =
  SPEC t IMP_REFL_THM;
```

```
> val tm1 = ``p``: term
> val thm1 = [p] |- p: thm
> val thm2 = |- p ==> p: thm

> val IMP_REFL_THM =
    |- !p. p ==> p: thm

> val IMP_REFL =
    fn: term -> thm
```

Forward Proofs — Example II

Let's prove $\forall P\, v.\ (\exists x.\ (x = v) \wedge P\ x) \Longleftarrow P\ v$.

```
val tm_v = ``v:'a``;
val tm_P = ``P:'a -> bool``;
val tm_lhs = ``?x. (x = v) /\ P x``
val tm_rhs = mk_comb (tm_P, tm_v);

val thm1 = let
  val thm1a = ASSUME tm_rhs;
  val thm1b =
    CONJ (REFL tm_v) thm1a;
  val thm1c =
    EXISTS (tm_lhs, tm_v) thm1b
in
  DISCH tm_rhs thm1c
end
```

```
> val thm1a = [P v] |- P v: thm
> val thm1b =
    [P v] |- (v = v) /\ P v: thm
> val thm1c =
    [P v] |- ?x. (x = v) /\ P x

> val thm1 = [] |-
    P v ==> ?x. (x = v) /\ P x: thm
```

Forward Proofs — Example II cont.

```
val thm2 = let
  val thm2a =
    ASSUME ``(u:'a = v) /\ P u``
  val thm2b = AP_TERM tm_P
    (CONJUNCT1 thm2a);
  val thm2c = EQ_MP thm2b
    (CONJUNCT2 thm2a);
  val thm2d =
    CHOOSE (``u:'a``,
      ASSUME tm_lhs) thm2c
in
  DISCH tm_lhs thm2d
end
```

```
> val thm2a = [(u = v) /\ P u] |-
    (u = v) /\ P u: thm
> val thm2b = [(u = v) /\ P u] |-
    P u <=> P v
> val thm2c = [(u = v) /\ P u] |-
    P v
> val thm2d = [?x. (x = v) /\ P x] |-
    P v

> val thm2 = [] |-
    ?x. (x = v) /\ P x ==> P v
```

```
val thm3 = IMP_ANTISYM_RULE thm2 thm1
```

```
> val thm3 = [] |-
    ?x. (x = v) /\ P x <=> P v
```

```
val thm4 = GENL [tm_P, tm_v] thm3
```

```
> val thm4 = [] |- !P v.
    ?x. (x = v) /\ P x <=> P v
```

Forward Proofs — Example 3

```
val exp_term = ``!p q r. (p /\ q ==> r) <=>
                         (p ==> q ==> r)``;

val curry_thm =
    let val ab = ASSUME ``p /\ q ==> r``;

        val p = ASSUME ``p:bool``;
        val q = ASSUME ``q:bool``;
        val pq = CONJ p q;
        val r = MP ab pq;
    in
    DISCH ``p /\ q ==> r``
    (DISCH ``p:bool``
    (DISCH ``q:bool`` r))
    end;
```

```
> val ab =  [p /\ q ==> r]
            |- p /\ q ==> r: thm
> val p =  [p] |- p: thm
> val q =  [q] |- q: thm
> val pq =  [p, q] |- p /\ q: thm
> val r =  [p, q, p /\ q ==> r]
           |- r: thm

> val curry_thm =
    [] |- (p /\ q ==> r) ==>
          p ==> q ==> r: thm
```

Forward Proofs — Example 3 cont.

```
val uncurry_thm =
    let val imp = ASSUME ``p ==> q ==> r``;
        val pq = ASSUME ``p /\ q``;
        val p = CONJUNCT1 pq;
        val q = CONJUNCT2 pq;
        val r = MP (MP imp p) q;
    in
     DISCH ``p ==> q ==> r``
     (DISCH ``p /\ q`` r)
    end;

val exp_thm =
GEN_ALL (IMP_ANTISYM_RULE curry_thm
                          uncurry_thm);
```

```
> val imp =  [p ==> q ==> r]
             |- p ==> q ==> r: thm
> val pq =  [p /\ q] |- p /\ q: thm
> val p =  [p /\ q] |- p: thm
> val q =  [p /\ q] |- q: thm
> val r =  [p /\ q, p ==> q ==> r]
           |- r: thm

>  val uncurry_thm =
  [] |- (p ==> q ==> r) ==>
        p /\ q ==> r: thm
```

Forward Proofs — Example 4

```
val noncontr_term = ``!p. ~(p /\ ~p)``;

val noncontr_thm =
    let val contr = ASSUME ``p /\ ~p``;
        val p = CONJUNCT1 contr;
        val np = CONJUNCT2 contr;
        val np_imp = NOT_ELIM np;
        val f = MP np_imp p;
        val contr_imp =
            DISCH ``p /\ ~p`` f;
    in
        GEN_ALL (NOT_INTRO contr_imp)
    end
```

```
> val contr =  [p /\ ~p] |- p /\ ~p: thm
> val p =  [p /\ ~p] |- p: thm
> val np =  [p /\ ~p] |- ~p: thm
> val np_imp =  [p /\ ~p] |- p ==> F: thm
> val f =  [p /\ ~p] |- F: thm
> val contr_imp =
    [] |- p /\ ~p ==> F: thm

> val noncontr_thm =
    [] |- !p. ~(p /\ ~p): thm
```