Interactive Theorem Proving in HOL4

Course 05: Goal-Directed Proofs

Dr Chun TIAN
`chun.tian@anu.edu.au`

29 August 2024

Australian National University

Acknowledgement of Country

We acknowledge and celebrate the First Australians on whose traditional lands we meet, and pay our respect to the elders past and present.

More information about Acknowledgement of Country can be found here and here

Forward vs Backward Proofs

Forward Proofs

- Starting from existing theorems (and axioms);
- Using primitive and derived inference rules to build new theorems from old ones;
- A linear sequence of intermediate theorems are built;
- Theorem proving process finishes when the goal theorem is reached.
- **C**ons: In large proofs it's hard to see what should be built next.

Backward Proofs (aka Goal-Directed Proofs)

- Starting from the final/targeting theorem (the goal);
- Reducing the goal into several subgoals (with a function for building the goal from subgoals);
- Reducing each subgoals, until no subgoal is left;
- Re-constructing the entire proof from bottom up (to build final theorem).
- **P**ros: A proof is organised as a tree of subgoals, easy to maintain.

Backward (Goal-Directed) Proofs

Tactics, Goals and Validations

- A *tactic* is an SML function that when applied to a goal reduces it to
  1. a list of (sub)goals, along with
  2. a validation function mapping a list of theorems to a theorem.
- A *goal* is an SML value whose type is isomorphic to, but distinct from, the SML abstract type `thm` of theorems.
- A *tactic* is said to solve a goal if it reduces the goal to the empty set of subgoals.

Related SML Types

```
goal       = term list * term
tactic     = goal -> goal list * validation
validation = thm list -> thm
```

The Implementation of Tactics: CONJ_TAC

```
fun CONJ_TAC (asl,w) =
    let val (l,r) = dest_conj w
    in
        ([(asl,l), (asl,r)],
         (fn [th1, th2] => CONJ th1 th2))
    end
```

$$\frac{t_1 \wedge t_2}{t_1 \qquad t_2}\ \texttt{CONJ\_TAC}$$

“run” the tactic

```
> CONJ_TAC ([],``P /\ Q``);
val it = ([([], ``P``), ([], ``Q``)], fn): goal list * validation
```

HOL Proof Scripts (Theory) Template

tempScript.sml

```
open HolKernel Parse boolLib bossLib;
val _ = new_theory "temp";

Theorem A_THEOREM :
  <goal>
Proof
  <tactics>
QED

val _ = export_theory();
```

The `Holmake` command translates `tempScript.sml` into SML structure `tempTheory` that can be opened in other theories by "`open tempTheory`."

HOL Proof Manager—The Goal Stack

Commands for Managing Goal Stacks

Operation	Emacs	SML
Starting a goalstack proof	HOL–Goalstack–New goal	`g : term quotation -> proofs`
Applying a tactic to a goal	HOL–Goalstack–Apply tactic	`e : tactic -> proof`
Undo	HOL–Goalstack–Back up	`b : unit -> proof`
Viewing the proof state	HOL–Goalstack–Print goal	`p : unit -> proof`
Drop goal	HOL–Goalstack–Drop goal	`drop : unit -> proofs`
Drop all goals	HOL–Goalstack–Drop all goals	`drop_all : unit -> proofs`
Restart goal	HOL–Goalstack–Restart goal	`restart : unit -> proof`

When starting a goalstack proof (in Emacs), put the cursor between on the theorem statements (between the colon and `Proof`) and type `M-h g` (or go to Emacs menu "HOL–Goalstack–New goal")

Built-in Tactics: Specialisation

```
GEN_TAC      : tactic
gen_tac      : tactic
X_GEN_TAC    : term -> tactic
Q.X_GEN_TAC  : tmquote -> tactic
```

$$\frac{\forall x.\, t[x]}{t[y]}\ \texttt{X\_GEN\_TAC}\ y$$

`GEN_TAC` is often the first step of a goal directed proof.

cf. `GEN`

$$\frac{\Gamma \vdash t}{\Gamma \vdash \forall x.\, t}\ \texttt{GEN}\ x\ (\Gamma \vdash t)$$

Built-in Tactics: Undischarge

```
DISCH_TAC : tactic
disch_tac : tactic
```

$$\frac{A \vdash u \Rightarrow v}{A \cup \{u\} \vdash v}\ \texttt{DISCH\_TAC}$$

Moves the antecedent of an implicative goal into the assumptions.

cf. DISCH

$$\frac{\Gamma \vdash t_2}{\Gamma - \{t_1\} \vdash t_1 \Rightarrow t_2}\ \texttt{DISCH}$$

Built-in Tactics: Combined Simple Decompositions

```
STRIP_TAC : tactic
strip_tac : tactic
```

Breaks a goal apart:

- The tactic `strip_tac` removes one outer connective from the goal, using `conj_tac`, `gen_tac`, `disch_tac` and other tactics.
- If the goal has the form $t_1 \wedge \cdots \wedge t_n \Rightarrow t$ then `strip_tac` makes each t_i into a separate assumption.
- If the goal has the form $t_1 \vee \cdots \vee t_n \Rightarrow t$ then `strip_tac` makes each t_i into a separate subgoal.

Built-in Tactics: Case Analysis

`Cases_on : term -> tactic`

$$\frac{t[q]}{t[q \mapsto C_1] \quad \cdots \quad t[q \mapsto C_n]}\ \texttt{Cases\_on}\ q$$

"`Cases_on` q", where q is a quotation denoting a term of an algebraic type (e.g., booleans, lists, natural numbers, types defined by the user with `Datatype`), does case analysis on that term.

For example:

- `` Cases_on `n :num` `` will create two subgoals corresponding to `0` and `SUC n`.
- `` Cases_on `l :'a list` `` will create two subgoals corresponding to `[]` and `(h::t)`.

Built-in Tactics: Rewriting

```
REWRITE_TAC            : term list -> tactic
ASM_REWRITE_TAC        : term list -> tactic
PURE_REWRITE_TAC       : term list -> tactic
PURE_ASM_REWRITE_TAC   : term list -> tactic
```

`REWRITE_TAC`$[th_1, \ldots, th_n]$ transforms the term part of a goal by rewriting it with the given theorems $[th_1, \ldots, th_n]$, and the set of pre-proved standard tautologies.
Difference between variants:

- `ASM_REWRITE_TAC` adds the assumptions of the goal to the list of theorems used for rewriting.
- `PURE_ASM_REWRITE_TAC` is like `ASM_REWRITE_TAC`, but it doesn't use any built-in rewrites.
- `PURE_REWRITE_TAC` uses neither the assumptions nor the built-in rewrites.

Tacticals: Combining tactics into bigger tactics

Tacticals are in general SML functions that returns a tactic (or tactics) as results.

- Sequencing

```
op THEN : tactic * tactic -> tactic
op >>   : tactic * tactic -> tactic
op \\   : tactic * tactic -> tactic
```

- Selective sequencing

```
op THENL : tactic * tactic list -> tactic
op >|    : tactic * tactic list -> tactic
op THEN1 : tactic * tactic -> tactic
```

- Repetition

```
REPEAT : tactic -> tactic
rpt    : tactic -> tactic
```

- Revert (two) subgoals: `reverse : tactic -> tactic`

Using Tacticals in Good Ways

```
Theorem SAMPLE_THM :
    !x. P /\ Q /\ R
Proof
    Q.X_GEN_TAC ‘x‘
 >> CONJ_TAC          (* principle: easier subgoals first *)
(* subgoal: P *)
 >- (tac1 \\
     ...)
 >> reverse CONJ_TAC (* when R-proof is shorter than Q-proof *)
(* subgoal: R *)
 >- (...)
(* subgoal: Q *)
 >> tac1
 >> ..
QED
```

Improving Your Proof Skills

- Learn existing proofs in HOL core library (by replaying them)
- Read documents (and existing proofs) to know more general and theory-specific tactics.
- Write verbose (more human-readable) proofs (e.g. using `X_GEN_TAC` instead of `GEN_TAC`) as much as possible.
- Use abbreviations to shorten literal terms.
- Use forward proofs to generate temporary theorems for tactics.
- ...