

Department of Computer Science

COMP4100-2007-02

Lab Session in Week 4

Backwards Proof – Tactics and Tacticals

The main objective of this session is to clarify your notions about tactics and tacticals through some exercises using the goal package.

## 1 A Worked Example

### 1.1 Proof Discovery

`set_goal`, `expand` and `backup` are the real bread-and-butter operations that you invoke while looking for a proof using the subgoal package of the HOL system. Be familiar with their types.

Let's examine the history of a (successful) attempt to prove the following standard result of predicate calculus

$$\forall P Q. (\exists x. P(x) \wedge Q(x)) \Rightarrow ((\exists y. P(y) \wedge (\exists y. Q(y)))$$

Using the standard HOL encoding of special symbols, the goal can be pushed onto the goal stack as follows. Try it.

```
set_goal([], ‘‘!P Q. (?x:'a. P x /\ Q x)
              ==> (?y:'a. P y) /\ (?y:'a. Q y)‘‘);
```

The outermost construct in the above formula is universal generalization. It is appropriate to attack this goal with `GEN_TAC`. That approach is accomplished as follows.

```
expand GEN_TAC;
```

We wouldn't normally do it piecemeal like that. Rather, the following tactic is what we would use in this situation. To see the full effect of `REPEAT STRIP_TAC` instead, back up first.

```
backup();
expand (REPEAT STRIP_TAC);
```

Of the two subgoals that are generated, the first is attacked by suggesting that variable mentioned in the assumptions is the one to try.

```
expand (EXISTS_TAC ‘‘x:'a‘‘);
```

Now that the latest subgoal appears as an assumption the following rewriting tactic proves that subgoal.

```
expand (ASM_REWRITE_TAC []);
```

The system backs up to the remaining subgoal on the stack. Since it is similar to the one just proved, we try the same sequence of two tactics as follows:

```
expand (EXISTS_TAC ‘‘x:a’’ THEN ASM_REWRITE_TAC []);
```

The original goal is now proved so that we can access the new theorem, assigning it to the variable `thm`.

```
val thm = top_thm();
```

## 1.2 Proof Recording

The following single command is the way that we like to capture a discovered proof. We do it like this so that a proof can be replayed with a single interaction. (You can cut-and-paste this text from the file:

```
/dept/dcs/comp4100/public/Lab2/lab2.bits)
```

```
val thm =
  TAC_PROOF
  (([], ‘‘!P Q. (?x:a. P x /\ Q x)
        ==> (?y:a. P y) /\ (?y:a. Q y)’’,
    REPEAT STRIP_TAC THENL
      [EXISTS_TAC ‘‘x:a’’ THEN ASM_REWRITE_TAC [],
       EXISTS_TAC ‘‘x:a’’ THEN ASM_REWRITE_TAC []]);
```

While the above most accurately represents the successful operations that were performed during discovery, the following is an equivalent script. Why?

```
val thm =
  TAC_PROOF
  (([], ‘‘!P Q. (?x:a. P x /\ Q x)
        ==> (?y:a. P y) /\ (?y:a. Q y)’’,
    REPEAT STRIP_TAC THEN
      EXISTS_TAC ‘‘x:a’’ THEN
      ASM_REWRITE_TAC []);
```

## 2 Three Exercises

Don't be put off by the technical jargon of the logician. The theorems are ones you (should) know.

## 2.1 Disjunction is symmetric

$$\forall p q. (p \vee q) = (q \vee p)$$

## 2.2 Implication distributes over conjunction

$$\forall p q r. (p \Rightarrow (q \wedge r)) = ((p \Rightarrow q) \wedge (p \Rightarrow r))$$

## 2.3 A theorem about relations

Any relation that is irreflexive and transitive is also asymmetric.

$$\begin{aligned} \forall R. ((\forall x. \neg R(x, x)) \wedge (\forall x y z. R(x, y) \wedge R(y, z) \Rightarrow R(x, z))) \\ \Rightarrow (\forall x y. R(x, y) \Rightarrow \neg R(y, x)) \end{aligned}$$

# 3 Tactics You Should Know About

The following table does *not* provide more than a memory aid to what the most common tactics of HOL achieve (in appropriate circumstances).

ACCEPT_TAC	Use a given theorem to prove the goal
CONJ_TAC	Splits a conjunction in two
EQ_TAC	Splits an equality into 2 implications
DISCH_TAC	Assume the hypothesis of an implication
CONTR_TAC	Proves goal if assumptions inconsistent
DISJ1_TAC	Proceed with just the first disjunct
DISJ2_TAC	Proceed with just the second disjunct
GEN_TAC	Strip off a universal quantifier
EXISTS_TAC	Provide a witness for existential goal
STRIP_TAC	GEN_TAC, CONJ_TAC or DISCH_TAC
ASSUME_TAC	Gratuitously add an assumption
REWRITE_TAC	Simplify goal using a set of equalities
RES_TAC	Resolve assumptions against each other

Remember there are eight flavours of the rewriting tactic - distinguished by the use of qualifiers PURE\_, ONCE\_ and ASM\_ as prefixes (in that order).

For more information during a HOL session on DISJ2\_TAC, for example,

- Get its exact type by saying DISJ2\_TAC;
- Consult the lecture notes
- Ask for information with help "DISJ2\_TAC";