Implicational rewriting
User manual

Vincent Aravantinos

Hardware Verification Group
Concordia University

1 Introduction.

This document is the user manual of the "impconv" HOL Light library. It essentially provides four tactics:

– `IMP_REWRITE_TAC`,
– `CASES_REWRITE_TAC`,
– `TARGET_REWRITE_TAC`,
– `HINT_EXISTS_TAC`

The most useful ones are `IMP_REWRITE_TAC` and `TARGET_REWRITE_TAC`. *These tactics are so powerful that many proofs end up being combinations of these two tactics only.*

Installation. To make use of these tactics, just type in the following inside a HOL Light session:

```
> needs "target_rewrite.ml";;
```

2 `IMP_REWRITE_TAC`

Informal specification: given a theorem of the form:

$$\forall x_1 \cdots x_n.\ P \Rightarrow \forall y_1 \cdots y_m.\ l = r$$

implicational rewriting replaces any occurrence of l by r in the goal, *even if P does not hold.* This may involve adding some propositional atoms (typically instantations of P) or existentials, but in the end, you are (almost) sure that l is replaced by r.

Note: We use only first-order matching because higher-order matching happens to match "too much". An improvement would be to define a second version of the tactic using higher-order matching.

Remark 1. Contrarily to `REWRITE_TAC` or `SIMP_TAC`, the goal obtained by using implicational rewriting is generally *not* equivalent to the initial goal. This is actually what makes this tactic so useful: one often wants to make "irreversible" steps in a proof.

Tactic:

`IMP_REWRITE_TAC` : `thm list` $\to$ `tactic`
Given a list of theorems $[\mathtt{th_1};\cdots;\mathtt{th_k}]$ of the form $\forall \mathtt{x_1}\cdots\mathtt{x_n}.\ \mathtt{P} \Rightarrow \forall \mathtt{y_1}\cdots\mathtt{y_m}.\ \mathtt{l} = \mathtt{r}$, `IMP_REWRITE_TAC` $[\mathtt{th_1};\cdots;\mathtt{th_k}]$ applies as many implicational rewriting usin all theorems.

Use: Allows to make some progress when `REWRITE_TAC` or `SIMP_TAC` cannot. Namely, if the precondition P cannot be proved automatically, then these classic tactics cannot be used, and one must generally add the precondition explicitly using `SUBGOAL_THEN` or `SUBGOAL_TAC`. `IMP_REWRITE_TAC` allows to do this automatically. Additionnaly, it can add this precondition deep in a term, actually to the deepest where it is meaningful. Thus there is no need to first use `REPEAT STRIP_TAC` (which often forces to decompose the goal into subgoals whereas the user would not want to do so).

`IMP_REWRITE_TAC` can also be used like `MATCH_MP_TAC`, but, again, deep in a term. Therefore you can avoid the common preliminary `REPEAT STRIP_TAC`.

The only disadvantages with regards to `REWRITE_TAC`, `SIMP_TAC` and `MATCH_MP_TAC` are that `IMP_REWRITE_TAC` uses only first-order matching and is generally a little bit slower.

Bonus features:

- A theorem of the form $\forall x_1\cdots x_n.\ P \Rightarrow \forall y_1\cdots y_m.\ Q$ is turned into $\forall x_1\cdots x_n.\ P \Rightarrow \forall y_1\cdots y_m.\ Q = true$
  (this is the reason why `IMP_REWRITE_TAC` can be used as a replacement for `MATCH_MP_TAC`)
- A theorem of the form $\forall x_1\cdots x_n.\ P \Rightarrow \forall y_1\cdots y_m.\ \neg Q$ is turned into $\forall x_1\cdots x_n.\ P \Rightarrow \forall y_1\cdots y_m.\ Q = false$;
- A theorem of the form $\forall x_1\cdots x_n.\ l = r$ is turned into $\forall x_1\cdots x_n.\ true \Rightarrow l = r$
  (this is the reason why `IMP_REWRITE_TAC` can be used as a replacement for `REWRITE_TAC` and `SIMP_TAC`)
- A theorem of the form $\forall x_1\cdots x_n.\ P \Rightarrow \forall y_1\cdots y_k.\ Q\cdots \Rightarrow l = r$ is turned into $\forall x_1\cdots x_n, y_1\cdots y_k,\cdots P \wedge Q \wedge \cdots \Rightarrow l = r$;
- A theorem of the form $\forall x_1\cdots x_n.\ P \Rightarrow (\forall y_1^1\cdots y_k^1.\ Q_1\cdots \Rightarrow l_1 = r_1 \wedge \forall y_1^2\cdots y_k^2.\ Q_2\cdots \Rightarrow l_2 = r_2 \wedge\cdots)$ is turned into the list of theorems $\forall x_1\cdots x_n, y_1^1\cdots y_k^1,\cdots P \wedge Q_1 \wedge \cdots \Rightarrow l_1 = r_1$, $\forall x_1\cdots x_n, y_1^2\cdots y_k^2,\cdots P \wedge Q_2 \wedge \cdots \Rightarrow l_2 = r_2, \ldots$;

All these operations are combined. In practice, this entails that *several deduction steps can be applied using `IMP_REWRITE_TAC` with just a big list of theorems.*

2.1 Variant:

`SEQ_IMP_REWRITE_TAC` : `thm list` $\to$ `tactic`
Same as `IMP_REWRITE_TAC` but uses the provided theorems *sequentially* instead of

simultaneously: given a list of theorems $[\mathtt{th_1}; \cdots; \mathtt{th_k}]$ SEQ_IMP_REWRITE_TAC $[\mathtt{th_1}; \cdots; \mathtt{th_k}]$ applies as many implicational rewriting as it can with $\mathtt{th_1}$, then with $\mathtt{th_2}$, etc. When $\mathtt{th_k}$ is reached, start over from $\mathtt{th_1}$. Repeat till no more rewrite can be achieved.

Use: This addresses a problem which happens sometimes already with `REWRITE_TAC` or `SIMP_TAC`: one generally rewrites with one theorem, then with another, etc. and, in the end, once every step is done, he packs everything in a list and provides this list to `IMP_REWRITE_TAC`; but it then happens that some surprises happen at this point because the simultaneous use of all theorems does not yield the same result as their subsequent use. Note that this however slower than `IMP_REWRITE_TAC`. Therefore I advise to first use `IMP_REWRITE_TAC` and if it does not work like the subsequent use of single implicational rewrites then use `SEQ_IMP_REWRITE_TAC`.

3 CASES_REWRITE_TAC

Informal specification: given a theorem of the form:

$$\forall x_1 \cdots x_n.\ P \Rightarrow \forall y_1 \cdots y_m.\ l = r$$

case rewriting replaces any atom A containing an occurrence of l by $(P \Rightarrow A[l \to r]) \wedge (\neg P \Rightarrow A)$.

Tactic:
 `CASES_REWRITE_TAC` : $\mathtt{thm} \to \mathtt{tactic}$
Same usage as `IMP_REWRITE_TAC` but applies case rewriting instead of implicational rewriting. Note that it takes only one theorem since in practice there is seldom a need to apply this tactic subsequently with several theorems.

Use: Similar to `IMP_REWRITE_TAC`, but instead of assuming that a precondition holds, one just wants to make a distinction between the case where this precondition holds, and the one where it does not.

4 TARGET_REWRITE_TAC

5 HINT_EXISTS_TAC

6 Implementation details: implicational conversions