Chapter 1

Library ex2

The next two examples illustrate that different proofs lead to different extracts. The exact we get from ex1 is somewhat unreadable because our proof introduces new assumptions. The extract we get from better_ex1 is the straightforward one we would expect because we only manipulate our conclusion

```
Lemma ex1 : ∀ P Q : Prop, (P → Q) → (¬Q → ¬P).
Print ex1.
```

```
Lemma better_ex1 : ∀ P Q : Prop, (P → Q) → (¬Q → ¬P).
Print better_ex1.
```

The next example illustrates that it is sometimes necessary to keep the functions even we have already used them once, because we might need them more than once. For example the following proof uses g twice.

```
Lemma ex2 : ∀ P Q : Prop, (P → Q) → ¬ ¬ (~P ∨ Q).
Print ex2.
```

We now prove that peirce's law is equivalent to the law of excluded middle. The first proof *peirce_implies_lem_by_hand* is a tedious proof that shows that the first one implies the second one. We can see in *peirce_iff_lem_cool* that Coq can help us here because it provides decision procedures such as `tauto`.

```
Definition peirce := ∀ p q : Prop, ((p → q) → p) → p.
Definition lem := ∀ r : Prop, r ∨ ¬ r.
```

```
Theorem peirce_implies_lem_by_hand: peirce → lem.
```

```
Theorem peirce_iff_lem_cool: peirce ↔ lem.
```

The following example illustrates the use of set types. Coq does not have set types, it has what we often call sum types (the type of dependent pairs). Which means that in Coq we cannot discard the witness of the proposition we use to refine a given type. For example, a proof $\{x : nat \ \& \ 1 < x\}$ is a *nat* along with a proof that this *nat* is greater than 1. Nuprl has both set types and dependent sums. A member of the set type $\{x : nat \ \& \ 1 < x\}$ in Nuprl is a member of *nat* that is guaranteed to be greater than 1. Because Coq does not

have "true" set types, in the following statement, we have to use *projT1* to access the *nat* part of x (its first component).

```
Require Omega.
Lemma ex3 : ∀ x : {x : nat & 1 < x}, ∃ k, k < (projT1 x).
```