```
intros n. destruct n as [| n'].

intros n. induction n as [| n' IHn'].
  - rewrite <-IHn'.

assert (H: 0 + n = n). { reflexivity. }
```

On peut finir avec "[rewrite -> eq2. reflexivity.]" comme on l'a fait plusieurs fois auparavant. On peut obtenir le même effet en un seul pas en utilisant la tactique [apply] à la place apply eq2.

```
intros n m o H. inversion H. reflexivity. Qed.
```

Voici ce que l'on a vu :

- [intros] : déplace les hypothèses/variables du but vers le contexte

- [reflexivity] : fini une preuve (quand le but ressemble à [e = e])

- [apply] : prouve un but en utilisant une hypothèse, un lemme ou un constructeur

- [apply... in H] : applique une hypothèse, un lemme ou un constructeur à une hypothèse dans le contexte (raisonnement par chaînage avant)

- [apply... with...] : indique explicitement les valeurs pour les variables qui ne peuvent pas être déterminées par le filtrage

- [simpl] : simplifie le but

- [simpl in H] : simplifie une hypothèse

- [rewrite] : utilise une hypothèse d'égalité (ou un lemme) pour réécrire le but

- [rewrite ... in H]: utilise une hypothèse d'égalité (ou un lemme) pour réécrire une hypothèse

- [symmetry] : modifie un but de la forme [t=u] en [u=t]

- [symmetry in H] : modifie une hypothèse de la forme [t=u] en [u=t]

- [unfold] : remplace une constante définie par son coté droit
  dans le but

- [unfold... in H] : remplace une constante définie par son coté
  droit dans une hypothèse

*)
(**

- [destruct... as...] : analyse de cas sur les valeurs d'un type
  défini inductivement

- [destruct... eqn:...] : indique le nom d'une équation qui doit
  être ajoutée au contexte

- [induction... as...] : récurrence sur les valeurs d'un type
  défini inductivement

- [inversion] : raisonne par injectivité et distinction de
  constructeurs

- [assert (H: e)] (or [assert (e) as H]) : introduit un lemme
  local [e] et l'appelle [H].

- [generalize dependent x] : déplace la variable [x] (et tout ce
  qui en dépend) du contexte vers une hypothèse explicite dans le
  but *)