TME 3 : Prédicats inductifs sur les entiers et les listes

1 Nouvelles tactiques

Nous ajoutons cette semaine cinq nouvelles tactiques à notre catalogue :

- `destruct x` permet de faire une analyse par cas sur la structure de `x`. Ainsi, si `x` est un entier, il y aura deux cas à prouver : `x = 0` et `x = S x'`. `destruct` fera immédiatement la substitution et ne gardera pas l'égalité. Dans certains cas, il peut être intéressant de garder l'égalité, par exemple pour des réécritures futures. On peut dans ce cas utiliser `case_eq`.
- `inversion H` permet de déduire du prédicat inductif défini par H comment celui-ci a été crée, en appliquant les lemmes d'inversions mentionnés en cours.
- S'il y a une égalité `x = y` dans les hypothèses, `subst` éliminera `y` et remplacera ses occurences par `x`.
- `trivial` est capable de résoudre les preuves les plus simples (par exemple quand le but de preuve est une hypothèse).

Il est possible d'enchaîner l'application de différentes tactiques en utilisant le point virgule. Ainsi, `inversion H; subst` appliquera la tactique `subst` dans tous les sous-cas de preuve créés par l'inversion.

Notez aussi que la tactique `apply`, vue précédemment, peut aussi être utilisée sur les constructeurs des prédicats inductifs.

2 Prédicat inférieur ou égal sur les entiers

1. Définir la relation d'ordre inférieur ou égal sur les entiers, $\leq_{\mathbb{N}}$, comme un prédicat inductif sur les couples d'entiers. Il faudra utiliser les deux cas suivants :
   - tout entier est inférieur ou égal à lui même,
   - si $n \leq_{\mathbb{N}} m$, alors $n \leq_{\mathbb{N}} m + 1$.
2. Montrer que tout entier naturel est positif.
3. Montrer que `le` est stable par passage au successeur : pour tous les entiers `n` et `m`, `le n m` implique `le (S n) (S m)`.
4. Nous avions utilisé la semaine dernière le fait qu'il n'existe pas d'entier naturel strictement inférieur à 0. En faire la preuve, maintenant que la définition de `le` est connue. Pour cela, montrer que `~ (le (S n) 0)`.
5. Montrer que `le` est stable par l'application du prédécesseur, i.e. `le (S n) (S m) -> le n m`.
6. Monter que `le` est antisymétrique.

3 Prédicat de liste triée

Nous allons utiliser des définitions sur les entiers et les listes. Commencez par importer ces modules avec `Require Import Arith. Require Import List.`.

1. Définir un prédicat inductif affirmant qu'une liste d'entiers est triée. Il faudra utiliser les trois cas suivants :

— la liste vide est triée,
— la liste réduite à un seul élément est triée,
— si n_1 et n_2 désignent des entiers et l une liste d'entiers, si $n_1 \leq_{\mathbb{N}} n_2$ et que la liste contenant n_2 en tête et l ensuite est triée, alors la liste contenant n_1, puis n_2, puis les éléments de l est aussi triée.

N.B : Utilisez `<=` *sur les entiers plutôt que le prédicat* `le` *défini précédemment. Cela facilitera les preuves suivantes.*

2. Définir une fonction d'insertion d'un élément dans une liste supposée triée. On utilisera `leb` (aussi noté `<=?`), qui permet de *calculer* si un entier est inférieur à un autre.
3. Prouver que la fonction d'insertion définie précédemment préserve le caractère trié de la liste.

   Cette preuve est plus difficile et longue que les précédentes. N'hésitez pas à réfléchir à faire la preuve sur papier en parallèle.

   Vous pouvez utilisez des lemmes de la librairie standard pour obtenir des équivalences entre la propriété `x <= y` et le booléen `x <=? y`, tels que `leb_complete` et `leb_complete_conv`. N'hésitez pas à utiliser la commande `Search`.
4. Définir une fonction de tri par insertion.
5. Prouver que le tri par insertion crée une liste triée.
6. Que reste-t-il à prouver pour montrer que le tri par insertion est correct ?