

## INE22 : Preuves de Programmes – TP n° 3

**Damiers (suite et fin)**

On rappelle la correspondance suivante :

Tactique	Règle(s) de déduction
<code>assumption</code>	Axiome
<code>intro, intros</code>	$\Rightarrow$ -intro, $\forall$ -intro, $\neg$ -intro
<code>apply</code>	$\Rightarrow$ -élim, $\forall$ -élim, $\neg$ -élim
<code>split</code>	$\wedge$ -intro, $\top$ -intro
<code>left, right</code>	$\forall$ -intro <sub>1</sub> , $\forall$ -intro <sub>2</sub>
<code>exists</code>	$\exists$ -intro
<code>destruct</code>	$\wedge$ -gauche, $\vee$ -gauche, $\perp$ -gauche, $\top$ -gauche, $\exists$ -gauche

et le lien vers la documentation en ligne : <http://www.pps.jussieu.fr/~miquel/enseignement/mpri/>

En particulier, on consultera avec profit la documentation de la tactique *induction*. On rappelle également

- que les tactiques `trivial` et `auto` éliminent les buts “évidents” (identiques à une hypothèse dans le contexte, ...);
- qu’une égalité entre deux constructeurs différents (comme 0 et  $(Sn)$ ) est toujours absurde. Si une telle hypothèse rend le but évident (puisque cette hypothèse est nécessairement absurde), la tactique `discriminate` clôturera le but en question;
- que les constructeurs sont *injectifs* et qu’on exploite ce fait par la tactique `injection`, qui prend en argument une hypothèse d’égalité;
- qu’on *simplifie* un but par “calcul”, grâce à la tactique `simpl`.

Dans le sujet précédent, on a modélisé un jeu de damier. Il s’agissait de montrer que certaines configurations étaient ou non atteignables à partir d’une configuration de départ donnée, en suivant les règles du jeu.

Cette fois, il s’agit d’étudier de façon générale la relation qui lie deux configurations atteignables, et de donner une méthode systématique pour obtenir les preuves.

Pour travailler, ouvrez avec `coqide` le fichier `tp2.v`, que vous avez complété la séance dernière. A défaut, télécharger le fichier `ensta_tp2_corr.v` sur la page des enseignements et renommez le en `tp2.v`. Dans le menu Compile, choisissez Compile Buffer. Ceci a normalement créé un fichier `tp2.vo` dans votre répertoire courant.

Téléchargez le fichier à trous `tp3.v` sur la page des enseignements et complétez le en fonction des questions ci-dessous. Servez vous des (*\* commentaires \**) pour répondre aux questions qui ne sont pas des preuves ou définitions à compléter.

**Exercice 1 (Invariants)** Un invariant est une fonction sur les damiers qui est préservée par les jeux.

1. Définir un prédicat `invariant` qui qualifie les fonctions sur les damiers à valeur dans un type `M` et qui donnent la même valeurs à deux damiers en relation par `moves`.
2. Montrer le lemme `invariant_move` qui exprime que pour être un invariant, il suffit qu’une fonction `f` donne la même valeur à deux damiers en relation par `move`.
3. Montrer le lemme `invariant_not_moves`. Qu’exprime-t-il?

**Exercice 2 (Involutions)** Une involution est une fonction  $f$  telle que  $\forall x, f(f(x)) = x$

1. Définir un prédicat `involution` qui qualifie les fonctions involutives sur un type `M`.
2. Montrer que si `f` est une involution, alors `triple_map f` est aussi une involution, dans un lemme nommé `triple_map_inv`.
3. Montrer que si `f` est une involution, alors `triple_map_select f` est aussi une involution, dans un lemme nommé `triple_map_select_inv`.
4. Montrer que `turn_color` est une involution.
5. Montrer que `(turn_line_inv p)` est une involution quelle que soit `p`.
6. Montrer que `(turn_col_inv p)` est une involution quelle que soit `p`.

On rappelle qu'on peut exploiter une hypothèse d'égalité `H` en la *réécrivant* dans le but, grâce à la tactique `rewrite H`.

**Exercice 3 (Symétrie)** Une relation binaire  $R$  est symétrique si  $\text{forall } x, y, Rxy \Rightarrow Ryx$ .

1. Montrer que la relation `move` est symétrique.
2. Montrer que la relation `moves` est symétrique. On rappelle qu'on a montré le lemme de transitivité `moves_trans`, et que la correction donne des exemples de son utilisation.

**Exercice 4 (Forme normale)**

1. Définir la fonction `force_white_line` qui force une ligne `p` à avoir un pion blanc en première position, quitte à la retourner.
2. Définir la fonction `force_white_col` qui force une colonne `p` à avoir un pion blanc en première position, quitte à la retourner.
3. Montrer que ces transformations respectent les règles du jeu, en prouvant les lemmes `force_white_line_moves` et `force_white_col_moves`.
4. Définir la fonction `force_white` qui retourne lignes et colonnes d'un damier si nécessaire pour obtenir des pions blanc sur la première ligne et la première colonne du damier.
5. Montrer que `force_white` respecte les règles du jeu.
6. Montrer que deux damiers qui ont la même valeur par `force_white` sont en relation par `moves`.
7. Le lemme `move_force_eq` établit la réciproque en examinant tous les cas. On l'admettra car sa preuve demande au système un certain temps de calcul. Pourquoi? Le système Coq est doté d'un calcul numérique relativement efficace. Que pourrait-on imaginer pour éviter cet écueil?
8. Montrer dans le lemme `moves_force_eq` que `force_white` est un `invariant`.

**Exercice 5 (Décidabilité)** Le prédicat `dec_eq` exprime la décidabilité de l'égalité sur le type `M`. Quelle est la différence entre ce prédicat et une simple disjonction :

`forall x y, x = y ∨ ~ (x = y)`?

1. Montrer que `dec_eq` se transmet d'un type `M` au type `triple M`.
2. Montrer que les types `color` et `board` vérifient `dec_eq`.
3. Montrer le lemme `moves_dec`. Qu'exprime-t-il?

**Exercice 6 (Méthode calculatoire systématique)** On définit la fonction à valeur booléenne `moves_dec_fun` à partir du résultat précédent.

1. Montrer le lemme `moves_proof` qui exprime que si `moves_dec_fun` prend la valeur `true` sur deux arguments, alors ceux-ci sont en relation par `moves`.
2. Montrer le lemme `moves_proof` qui exprime que si `moves_dec_fun` prend la valeur `false` sur deux arguments, alors ceux-ci ne sont pas en relation par `moves`.
3. Que se passe-t-il dans les (nouvelles) preuves des lemmes `accessible_auto` et `not_accessible_auto`? En quoi a-t-on obtenu une méthode systématique de preuve?