

INE22 : Preuves de Programmes – TP n° 2

Damiers

On rappelle la correspondance suivante :

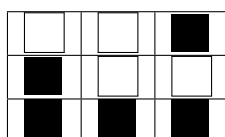
Tactique	Règle(s) de déduction
<code>assumption</code>	Axiome
<code>intro, intros</code>	\Rightarrow -intro, \forall -intro, \neg -intro
<code>apply</code>	\Rightarrow -élim, \forall -élim, \neg -élim
<code>split</code>	\wedge -intro, \top -intro
<code>left, right</code>	\vee -intro ₁ , \vee -intro ₂
<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 la tactique `trivial` clôt 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.

On dispose d’un damier de 3 cases sur 3, sur lequel sont posés 9 jetons bicolores (une face blanche et une face noire).



À chaque étape on peut soit retourner les pions d’une ligne, soit retourner les pions d’une colonne. Voici un exemple de suite d’étapes valides :



On cherche à modéliser ce jeu, et à savoir si certaines configurations sont ou non atteignables à partir d'une configuration initiale donnée.

Pour travailler, ouvrez le fichier `tp2.v`, et remplissez les trous laissés dans le code en fonction de vos réponses aux questions successives.

Exercice 1 (Modélisation des pions)

1. En s'inspirant du type `bool`, définir un type `color` ayant exactement deux valeurs `White` et `Black`.
2. Définir par cas la fonction `turn_color` de type `color→color` qui inverse les couleurs.

3. Tester la fonction définie :

```
Eval compute in (turn_color White)
```

Exercice 2 (Triplets polymorphes) La modélisation du damier utilise un type *polymorphe* de triplets.

On ouvre une section (*Section Polymorphic_triples*) dans laquelle on se donne une variable `M :Set`.

1. Définir le type `triple` des triplets d'objets de type `M` avec un (seul) constructeur : `Triple` de type `M → M → M → triple`.
2. On définit la fonction `triple_x` qui construit un triplet répétant (3 fois) la même valeur `x` donnée en argument.

Exercice 3 (Fonctions sur les triplets) Pour accéder aux composantes du triplet on définit un type :

```
Inductive pos : Set := A : pos | B : pos | C : pos.
```

1. Définir `proj_triple : pos → triple → M` qui renvoie la valeur du triplet en la position donnée.
2. On se donne maintenant une fonction `f` de type `M→M`. Définir `triple_map : triple → triple` qui applique la fonction `f` à chaque composante du triplet.
3. Définir `triple_map_select : pos → triple → triple` qui applique la fonction `f` à la composante désignée du triplet.
4. Fermer la section sur les triplets (*End Polymorphic_triples*)
Observer le type des objets définis.

Exercice 4 (Modéliser le damier)

1. Définir le type `board` comme étant un triplet de triplet d'objets de type `color`. On considérera par la suite qu'un damier est un triplet de lignes.
2. Définir `white_board` le damier dont tous les pions sont blancs.
3. Définir les deux configurations :

`start` =

			■
■			
■	■	■	

`target` =

■	■		
	■	■	■
■	■	■	

4. Définir `proj_board : pos → pos → board → color` la fonction qui calcule la couleur d'un point du damier.
5. Définir `turn_line, turn_col : pos → board → board` qui retournent respectivement une ligne et une colonne du damier.
Tester ces fonctions sur des exemples.

Exercice 5 (Déplacements)

1. Définir une relation `move : board → board → Prop` telle que :
`(move b1 b2)` est vrai lorsque `b2` est obtenu en retournant une ligne ou une colonne de `b1`.
On pourra choisir une définition inductive ou bien utiliser les connecteurs logiques.
2. On définira dans chacun des cas :
`move_line : forall1 (b1 : board)(p : pos), move b1 (turn_line p b1)`
`move_col : forall (b1 : board)(p : pos), move b1 (turn_col p b1).`
3. Définir une relation `moves : board → board → Prop` telle que :
`(moves b1 b2)` est vrai lorsque `b2` est obtenu par 0, 1 ou plusieurs déplacements à partir de `b1`.
4. On prouvera les lemmes suivants :
`moves_init : forall b1 : board, moves b1 b1`
`move_moves : forall b1 b2 : board, move b1 b2 → moves b1 b2`
`moves_trans : forall b1 b2 b3 : board,`
`(moves b1 b2) → (moves b2 b3) → (moves b1 b3)`

Exercice 6 (Conclusion)

1. Établir le lemme `(moves start target)`.
2. Pour établir que `~(moves b1 b2)` commencer par trouver un type `M` et une fonction `f : board → M` telle que :
`forall b1 b2 : board, moves b1 b2 → (f b1) = (f b2)`
et que `(f b1)` ne soit pas égal à `(f b2)`.
3. En déduire que `~(moves start white_board)`.