The go-ban of Cachan

C. Paulin & B. Werner

The problem

We consider a 3x3 go-ban (board) on which 9 stones are laid. These stones are two-colored (one side black, one side white).

At each step, one can turn over all stones from one given line or one given column.

One would like to check whether some given configurations can or cannot be reached from a given initial stat of the board.

Modelizing stones

Using the model of the type `bool`, define a type `color` which has exactly two elements `White` and `Black`

First one defines, by cases the function `turn_color` of type `color`$\rightarrow$`color` which inverts the colors.

Test the defined function:

```
Eval Compute in (turn_color White)
```

Triplets polymorphes

The representation of the board uses a polymorphic type of triples:

Open a section (`Section` *`name_of_the_section`*) where you introduce a variable `M:Set`.

Define the type `triple` of triples of objects of type `M` with one constructor: `Triple` of type `M` $\rightarrow$ `M` $\rightarrow$ `M` $\rightarrow$ `triple`.

The function `triple_x` which to `x`:`M` associates the triple formed of three `x` is written:

```
Definition triple_x:M→triple:=[x](Triple x x x).
```

Fonctions sur les triplets

Pour accéder aux composantes du triplet on définit un type :

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

Define the function `proj_triple`:`pos → triple → M` which returns the corresponding component of the triple.

We now give ourselves a function `f` of type M→M.

Define `triple_map`:`triple → triple` which applies `f` to each component of the triple.

Define `triple_map_select`:`pos → triple → triple` which applies `f` to the designated component.

Close the section of triples (`End` *`name_of_the_section`*)

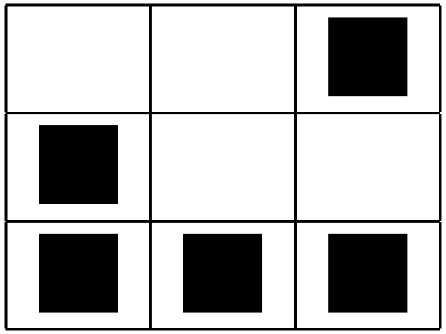
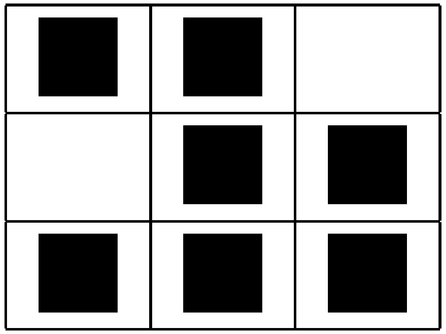
Observe the type of the defined objects.

Modelise the board

Define the type `board` as a triple of triples of objects of type `color`.

Define `white_board` as the board whose stones are all white.

We introduce two states:

`start` =  `target` = 

Functions over boards

Define `proj_board`:`pos` $\rightarrow$ `pos` $\rightarrow$ `board` $\rightarrow$ `color` the function which returns the color of one given stone.

Define `return_line`, `return_col`:`pos` $\rightarrow$ `board` $\rightarrow$ `board` which turns over respectively the stones of one given line and column.

Test these functions on examples.

Moves

Define a relation `move`:`board` $\rightarrow$ `board` $\rightarrow$ `Prop` such that `(move b1 b2)` is true when `b2` is obtained by returning one line or one column of `b1`.

One can either chose an iductive definition or use the logical connectors.

In both cases, one will define:

```
move_line:(b1:board)(p:pos)(move b1 (return_line p b1))
move_col: (b1:board)(p:pos)(move b1 (return_col p b1)).
```

Multiple moves

Define a relation `moves`:`board` → `board` → `Prop` such that `(moves b1 b2)` is true when `b2` is obtained by 0, 1 or several moves starting from `b1`.

Prove the following lemmas:

`moves_init`:`(b1:board)(moves b1 b1)`

`move_moves`:`(b1,b2:board)(move b1 b2) → (moves b1 b2)`

`moves_trans`: `(b1,b2,b3:board)`

`(moves b1 b2)→(moves b2 b3)→(moves b1 b3)`

Results

Establish `(moves start target)`
and `~(moves start white_board)`.

To show that `~(moves b1 b2)` it is sufficient to exhibit a type `M` and a function `f`:`board`→`M` such that:

`(b1,b2:board)(moves b1 b2)→(f b1)=(f b2)`
and `(f b1)` is different from `(f b2)`.