Formalization of Tic-Tac-Toe in Path Semantics

by Sven Nilsen, 2017

This is a practice problem that people can use to get familiar with path semantical notation.

```
type player := nat ∧ (< 2)
type opt[a] := none() | some(a)

move : board → board
slot : nat ∧ (< 3) × nat ∧ (< 3) → board → opt[player]

is_full : board → bool
has_won : player → board → bool
can_play : board → bool

player_turn : board → player
next_player : player → player
moves_made : board → nat
moves_made_by_player : player → board → nat

next_player := \(x : player) = (x + 1) % 2
moves_made := \(b : board) = ∑ i, j { if slot(i, j)(b) == none() {0} else {1} }
is_full := \(b : board) = moves_made(b) == 9
has_won := \(x : player) = \(b : board) =
        ∀ i, j { slot(i, j)(b) == some(x) } ∨
        ∀ i, j { slot(j, i)(b) == some(x) } ∨
        ∀ i { slot(i, i)(b) == some(x) } ∨
        ∀ i { slot(i, 2 – i)(b) == some(x) }
moves_made_by_player := \(x : player) = \(b : board) =
        ∑ i, j { if slot(i, j)(b) == some(x) {1} else {0} }
can_play := \(b : board) = ¬is_full(b) ∧ ¬has_won(0)(b) ∧ ¬has_won(1)(b)

move{[can_play] true}[moves_made] => inc
move{[can_play] true}[players_turn] => next_player
move{[can_play] true ∧ [player_turn] x}[moves_made_by_player(x)] => inc
move{[can_play] true ∧ [slot(i, j)] some(x)} : [slot(i, j)] some(x)
```

Task 1: Prove that when one player is making a move, the number of moves made by the other player stays constant. You do not need a completely rigorous proof (Hint: The total moves made should be sum of …).

Task 2: Draw a dependency graph between normal functions (not paths). Which 2 functions besides `move` require implementation? Why can they not be implemented in the formalization?