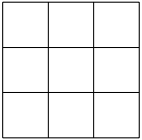
Formal Specification F28FS2, Lecture 14
An example: noughts and crosses (tic-tac-toe)

Jamie Gabbay

March 19, 2014

Noughts and crosses

This game is played on a 3x3 board:



Each cell may be empty, or contain a nought O, or contain a cross X.

How shall we model this in Z?

The board

There are plenty of methods, but the one I favour is this:

Declare a type $STATE ::= \mathsf{N} \mid \mathsf{O} \mid \mathsf{X}$.

(N stands for 'empty' or 'nothing'; E might be better but I like to see the 'NOX' because it reminds me of nitrogen oxide.)

Then we can model the type of possible states of a board as follows:

$CELL = 1..3 \times 1..3$

$BOARDSTATE = CELL \rightarrow STATE$

The *BoardState* schema

$$\begin{array}{|l}
\hline \textit{BoardState} \\
\textit{boardState} : \textit{BOARDSTATE} \\
\hline \\
\hline
\end{array}$$

The schema predicate here is 'True'; let's make it visible:

$$\begin{array}{|l}
\hline \textit{BoardState} \\
\textit{boardState} : \textit{BOARDSTATE} \\
\hline \top \\
\hline
\end{array}$$

This tells us that any value of *boardState* is an acceptable state of the board.

(Do you agree? What about the board state consisting of a column of Os on the left and a column of Xes on the right? Do we care?)

Initialising the board

Usually the board is started with all cells set to empty. This suggests the following initialisation schema:

InitBoard

$$boardState' : BOARDSTATE$$

$$\forall c : CELL \bullet boardState'(c) = \mathrm{N}$$

Warning: all of the following are incorrect!

InitBoard

$$boardState : BOARDSTATE$$

$$\forall c : CELL \bullet boardState(c) = \mathrm{N}$$

InitBoard

$$\Delta BoardState$$

$$\forall c : CELL \bullet boardState'(c) = \mathrm{N}$$

InitBoard

$$boardState' : BOARDSTATE$$

$$boardState'(c) = \mathrm{N}$$

InitBored

$$\exists x : LECTURE \bullet \neg understood(x)$$

Initialisation

We could spice things up and ask the user to provide the initial state (e.g. resuming a previous played game):

$$\begin{array}{l} \textit{InitBoard} \\ \hline \textit{boardState}', \textit{initState}? : \textit{BOARDSTATE} \\ \hline \textit{boardState}' = \textit{initState}? \\ \hline \end{array}$$

We could initialise to a random initial state (e.g. if this was some kind of weather simulation):

$$\begin{array}{l} \textit{InitBoard} \\ \hline \textit{boardState}' : \textit{BOARDSTATE} \\ \hline \\ \hline \end{array}$$

Exercise: write an initialisation schema that inputs $c : \textit{CELL}$ and $s : \textit{STATE}$ that is not N, and initialises the board with all cells empty except for c which has state s.

Moves

Players can play moves. If nought plays, they place a nought in a cell that was previously empty.

Here is how I would do it:

NoughtPlays

$\Delta BoardState$

$$\exists\, c : CELL \bullet$$
$$boardState(c) = \mathsf{N} \land boardState'(c) = \mathsf{O} \land$$
$$\forall\, c' : CELL \mid c' \neq c \bullet boardState(c) = boardState'(c)$$

This stuff is easy, if you have the right mindset.

Exercise: close this window and write a schema *CrossPlays*.

Recognise a winning state

Let's write a predicate to recognise if *boardState* : *BOARDSTATE* represents a winning state for player O. So we need to recognise a column, row, or diagonal line of Os in *boardState*.

This is not difficult. There are only eight possibilities and we could run through them; literally checking each possible line by hand.

That would be boring. Can we think of something more elegant? Have a go.

My attempt on the next slide ...

Recognise a winning state

$\exists\, i, j, i', j', i'', j'' : 1..3 \bullet$	
$\#\{(i,j),(i',j'),(i'',j'')\} = 3 \wedge$	3 cells
$i''{-}i' = i'{-}i \wedge j''{-}j' = j'{-}j \wedge$	in a line
$\mathit{boardState}(\{(i,j),(i',j'),(i'',j'')\}) = \{\mathrm{O}\}$	creative use of relational application

How far along are we?

We still can't represent an actual game.

For that we need e.g. some notion of alternating moves.

We could stick with schemas and enrich *BoardState* with an extra variable $nextToMove : \mathrm{O} \mid \mathrm{X}$ (initialised to O, I believe).

We could model a game as a partial function from $\mathbb{N}$ to *BOARDSTATE*, along with a bunch of consistency conditions.

We could model a game as an element of *BOARDSTATE seq*, likewise with consistency conditions.

Any of these would be fine.