MCS Project Part 2: Reversi Again

Laurent.Janssens@cs.kuleuven.be
Bart.bogaerts@cs.kuleuven.be

December 8, 2016

1 Introduction

In the MCS project, we will model the Reversi game[1].

- In this second part, you must model the same game in Event-B and use CTL and LTL to verify other properties of the game.

You will have to model the same game as in the first assignment, though one rule has changed to simplify the assignment: Players our now always allowed to pass instead of move, regardless of whether there are possible moves. Section 2 explains what you have to do for this part of the assignment. The project for this course must be completed individually.

2 Part 2: Modelling the game in Event-B

You are **obliged** to model this assignment in Event-B! In order to model this, you complete the skeleton present in the archive file on toledo. This second part is split up into four smaller parts.

- Complete the `Game_0` machine
- Complete the `Game_1` machine
- Do a refinement on the `Game_1` machine.
- Perform LTL/CTL checking on this refinement (`Game_2`) machine using ProB.

The skeleton is offered as a zip archive. To import it into Rodin, you need to create a new project and import the archive into that project.

For the first and second part you **cannot** add to or change the variables and contexts, or change or remove invariants, or remove or rename events, or add/remove/change event parameters. You do not have to worry about the proof obligations succeeding, these will not be checked. However take into account that they can help you in deciding whether your specification is correct.

[1]`https://en.wikipedia.org/wiki/Reversi`

2.1 Game Context

The Game context in the Event-B project contains a number of useful types and relations to be used while specifying the machines. This section will briefly introduce the symbols contained in this context.

xCoords The set of x-coordinates of positions

yCoords The set of y-coordinates of positions

Colours The set of colours: {White, Black}.

Positions The set of positions, i.e. the cartesian product of the xCoords and yCoords sets.

xOf A function mapping a position to its x-coordinate.

yOf A function mapping a position to its y-coordinate.

Next A function mapping a colour to the next colour, i.e. it maps white to black and vice versa.

NextTo A binary relation on positions (quaternary on coordinates) a tuple (x1, y1, x2, y2) is in the relation NextTo if the position (x1, y1) is next to the position (x2, y2).

Directions The set of possible directions. (The 8 possible values are specified a constants)

Neighbour A relation between positions, directions, and positions. Similar to the NextTo relation. A tuple (x1, y1, d, x2, y2) is in the relation Neighbour if starting from the position (x1, y1) in the direction d, you arive at position (x2, y2).

xDir A function mapping a direction to its component on the x-axis, e.g. xDir(Right) = 1.

yDir A function mapping a direction to its component on the y-axis, e.g. yDir(Up) = 1.

2.2 Complete the `Game_0` machine

For the first part of this assignment you have to complete the `Game_0` machine. To do this you have to add the proper guards and actions to the events of the machine.

In this abstraction of the Reversi game a player can play on any empty position or pass at any time. The game is over if a player passes and the next player chooses to end the game. At the end of the game a winner is chosen non-deterministically.

The machine consists of three variables:

CurrentPlayer is variable of type *Colour* equal to the colour of the certain player (White or Black).

PreviousPassed is a boolean variable, true if and only if the previous player passed.

GameOver is a boolean variable, true if and only if the game is over.

Winners is a set of colours, denoting which player(s) won the game. This set remains empty until GameOver becomes true.

FilledPos is a set of *Positions* containing all positions which contain a white or a black disk.

DO NOT CHANGE OR ADD ANYTHING TO THESE VARIABLES.

This machine consists of the following four events, apart from the INITIALISATION event:

Move happens when a disc is played on an empty position. This can only happen when the game is not over. This concludes a player's turn, and fills the given position.

Pass Enabled as long as the game isn't over, and if the previous player hasn't passed. Signifies the end of a player's turn.

EndGame Signifies the end of the game, decides the winner(On this level the winner is given non-deterministically, you'll have to fix this in the last refinement), and ends the game. This is enabled if and only if the game isn't over yet and the previous player passed.

Gameover Is only enabled when the game is over. It indicates the game is over and makes sure the machine can run forever.

NOTE that the pass action works differently from the previous assignment, as a player is always allowed to pass.

DO NOT ADD ANY EVENTS, only add guards and actions to the supplied events, and potential invariants, and leave the initialisation as is.

2.3 Complete the `Game_1` machine

For the second part of this assignment you have to complete the `Game_1` machine, a refinement of the `Game_0` machine. To do this you have to, again, add the proper guards and actions to the events of the machine. This machine is a refinement of the `Game_0` machine, so you must make sure you uphold the correct refinement relation between the two machines.

In this refinement restrictions are added to the moves a player is allowed to make. Whenever a player wants to make a move it first has to be checked to make sure it captures at least one of the opposing player's discs. Players can pass as long as they are not in the middle of checking whether a move is valid. The winner is still determined non-deterministically.

The machine consists of five additional variables:

Board is a function from the set of filled positions to colours, mapping each position with a disc on it to the colour of that disc.

CurrentMove The move currently being checked/executed. While a move is being checked this will be a currently unfilled position, where the player wants to play.

Captured is a set of positions which will be captured if the CurrentMove is played.

NeedsChecking is a relation between positions and directions, containing all the positions which have to be checked in a certain direction in order to deduce which positions will be captured by the CurrentMove.

Reachable is a relation between positions and directions, denoting which positions are reachable from the CurrentMove, in a certain direction. A position is reachable from the CurrentMove if it contains a disc of the other player's colour, and there are no empty positions or positions with discs of another colour between this position and the CurrentMove.

DO NOT CHANGE OR ADD ANYTHING TO THESE VARIABLES, only add guards and actions to the supplied events, and potential invariants, and leave the initialisation event as if.

This machine consists of the following 10 events, apart from the INITIALISATION event:

Move happens when the CurrentMove is actually made, this is only possible when the game is not over, no more moves need checking and at least one position will be captured. This concludes a player's turn, fills the CurrentMove's position, updates the board with the new move and the captured discs and clears the Captured set.

CheckMove Happens when the player wants to check if he/she can play on a certain position(x,y coordinates). This can only happen for empty positions and if no check is currently in progress, and no successful check has been completed (at least one position can be captured by the previous check). It adds all neighbouring positions, with their direction to the NeedsChecking set and set the CurrentMove to the given position.

CheckReachable Given a tuple of a position and direction in the NeedsChecking set, if the position has a disc of the opponent's colour, this position is considered reachable. As such it should be removed from the NeedsChecking set and added to the reachable set (with the given direction). The position next to the given one, in the given direction should then be added to the NeedsChecking set.

CheckNoValidEnd happens when a position is not a valid end for a chain (set of reachable positions in a direction), i.e. if the position is not a valid position or is empty. This removes the position from the NeedsChecking set and clears all Reachable positions in that direction.

CheckEndChain happens when a position that needs checking has a disc of the current player's colour. This marks all reachable positions in the given direction as captured and clears them from being reachable.

Pass Works just as before only, that it can't occur while a move is being checked, or if positions are determined to be capturable, i.e., the NeedsChecking, Reachable and Capturable sets need to be empty.

EndGame works exactly as before.

Gameover Works exactly as before.

2.4 Do a refinement on the `Game_1` machine

In the last part of this assignment you are asked to refine the `Game_1` machine.

The challenge for this part is as follows. Refine (Create a `Game_2` machine which refines Game_1) the `Game_1` machine so that you can determine the winner when the game ends. The rest of the game will function in the same way as in machine `Game_1`.

You can add variables (but don't change old ones) and add new (or refine old) events if necessary.

Tip: You can add two variables, one to represent the amount of white discs, and one to represent the amount of black discs currently on the board. These can then be updated whenever a move is made.

2.5 Perform LTL/CTL checking on the `Game_1` machine using ProB.

To do LTL/CTL checking on the `Game_1` machine follow the following steps:

1. Right-click the `Game_1` machine and under the `Prob Classic...` submenu, select `Open in Prob classic`. A prompt will appear saying the file exists and ask whether your want to overwrite it. Select Yes.
    - IF, you get a prompt saying ProB can not be found, go to the windows/preferences menu and under ProB/Prob Classic, select the path to your ProB installation. This is not the ProB plugin, but the actual application, as used in the exercise session on ProB.
2. Your machine will open in ProB in read-only mode, allowing you to specify and check LTL and CTL formulas as seen in the exercise sessions on ProB.
3. Use LTL and/or CTL to specify the following statements.

LTL/CTL

Check the following statements on your machine using LTL or CTL statements. **Add these statements to your report so we can verify them!** Remember that B syntax can be used in {curly brackets} in an LTL and CTL formula. The occurrence of an event is expressed as [E] for event E. Use the `ProB Classic ... - Open In ProB Classic` functionality in Rodin to open your final machine in ProB.

- Check whether it is possible for White to be the only winner.

- Check whether it is possible for the game to end in a draw.
- When the game is over it stays over forever.
- It is impossible that the game goes on forever.
- For the game to end there has to have been at least one pass event.
- After every CheckMove event there is always at least one CheckNoValidEnd and one CheckReachable event.
- When position (2,1) is coloured white it stays white unless it is captured.

3 Practical

The output of this part of the project consists of

1. A *concise* report in which you discuss design decisions.
   - The report should contain the time you spent on this part of the project. The expected time needed is 10 hours, this depends on your preparation during the exercise sessions.
2. A zip-file "solution.zip" containing all files in your Rodin project. Create this using Rodin's export option, which works the same as in Eclipse. Please make sure we can import your project from that .zip before submitting.
3. About the code:
   - You are obliged to start from our skeletons, and you are **not allowed** to change the given symbols or their interpretation. Remember that you **are allowed** to add new invariants to the intermediate machines, and new variables to the last refined machine.
   - Use well-chosen names for introduced symbols and variables, invariants, guards, and actions.
4. Grading of this part of the project is divided into three parts:
   - Just under half of the points are based on the first two parts of the assignment, i.e. the completion of the first two machines.
   - Just under half of the points are based on the last part, i.e. the LTL and CTL verifications.
   - The rest is based on the third part, i.e. the extra refinement.

3.1 Exporting your project

To export your project from Rodin, follow the following steps:

1. Right-click your project and select `Export...`
2. Select `Archive File`

3. Deselect your project folder, on the left (otherwise an additional folder is included in the zip, making it difficult to import)
4. Select **all** the files in your project, on the right. This makes sure only the files are present in the zip file.
5. Press finish
6. Create a new project and make sure you can import the archive file into it.

You are allowed to discuss this project with other people, but are not allowed to copy any code from others. This is not an open source project, i.e., you are also not allowed to put your code openly available on the web. If you want to use git, do not use public GitHub repositories (we will find them).

For any questions, do not hesitate to contact one of the assistants.

This project is again made individually. This project is expected to be completable in 10 hours or less. The deadline for submitting your solution on Toledo is the 23rd of December 2016, 23.59.

Good luck!