MCS Project Part 2: Swap It

Simon.Marynissen@cs.kuleuven.be
Ingmar.Dasseville@cs.kuleuven.be

1 december, 2017

1 Introduction

In this part of the project you will have to model a game which is very alike the one in the first assignment. However, some rules have changed to make the exercise appropriate for Event-B. If some detail about the game is ommited, you can assume it works exactly like in part 1. 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 five smaller parts.

- Complete the `Machine0` machine: an abstraction of our game
- Complete the `Machine1` machine: a refinement of `Machine0` which includes the proper win conditions
- Add the `Machine2` machine: a refinement of `Machine1` which includes a scoring system
- Add the `Machine3` machine: a refinement of `Machine2` which splits up swapping tiles into 3 different actions
- Perform LTL/CTL checking on `Machine3` 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** change the variables and contexts (it is allowed to expand the context), 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. It is however expected that the ProB model check does succeed for your specification. Do not worry if this takes a few minutes.

A good reference manual for the syntax of Event-B formulas can be found here:
`http://wiki.event-b.org/images/EventB-Summary.pdf`

2.1 New Rules in Swap It

The game you are modelling has the same principle as Shift It: a number of coloured blocks are aligned on a grid and you have to move them around to reach a winning configuration. The moves and the winning conditions are changed however. Moves consist of swapping to adjacent blocks (instead of whole rows/columns). Unlike the original game there are no locked tiles. Games are won when every block has an adjacent block with the same colour. So if there is only one block of a particular colour it becomes impossible to win the game.

2.2 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.

colour The set of colours the squares can take: this is either red or green.

xCo The set of x-coordinates of positions

yCo The set of y-coordinates of positions

position The set of positions, i.e. the cartesian product of the xCo and yCo sets.

block Blocks are identified with a natural number between and height * width.

height The height of the field, for the purpose project, we only use a 2×2-field with height 2. However, your specification should not depend on this.

width The height of the field, for the purpose of this project, we only use a 2×2-field with width 2. However, your specification should not depend on this.

blockColour Every block has a fixed colour, this mapping is represented by blockColour

adjacent $p1 \mapsto p2$ is an element of this relation if the two positions $p1$ and $p2$ are adjacent to each other. Adjacency is only considered horizontally and vertically, not diagonally. The board does not wrap around the edges.

2.3 Complete the `Machine0` machine

For the first part of this assignment you have to complete the `Machine0` machine. To do this you have to add the proper guards and actions to the events of the machine.

In this abstraction of Swap It:

The machine consists of three variables:

posColour is variable of type *position* $\rightarrow$ *colour* describing the colour of every position at the current point in time.

won is a boolean variable, true if and only if the game has been won.

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

DO NOT CHANGE OR ADD ANYTHING TO THESE VARIABLES.

This machine consists of the of the INITIALISATION event and two others:

INITIALISATION At the initial time point the game has not been won nor is the game over. You can give any valid interpretation for the initialisation of posColour but it needs to be deterministic (given that the context has been initialised).

swap2 This event takes two arbitrary positions as arguments and switches their colours.

finishGame This event signifies the end of the game, it decides whether the game has been won by a parameter named `winning`, and ends the game. If this event has been executed once with a particular parameter, it stays enabled but only with the same parameter.

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

2.4 Complete the `Machine1` machine

For the second part of this assignment you have to complete the `Machine1` machine, a refinement of the `Machine0` 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 `Machine0` machine, so you must make sure you uphold the correct refinement relation between the two machines.

In this refinement, only adjacent blocks can be switched and the `finishGame` event has been split up into `lose` and `win` events. Losing can be done at any point during the game (except when you have already won). Winning can only be done when every block has an adjacent block with the same colour and the game has not been lost yet.

In this refinement no explicit mapping between positions and colours is present, only a mapping between blocks and positions. You will need to provide the right initialisation and gluing invariants to uphold the refinement relation.

2.5 Do a refinement on Machine1

Make a new refinement of `Machine1` called `Machine2`. In this refinement you only have to add a new variable named `score`. The score starts at 10. Every move the score needs to be subtracted by one. When the game is lost, the score gets set to 0. A move can only be done if the current score is non-zero.

2.6 Do a refinement on Machine2

Make a new refinement of `Machine2` called `Machine3` in which the `swapAdjacent` event is split up into three new events: `swap`, `select1` and `select2`. The `select` events select a particular block (given through an event parameter). The `switch` event can only be executed once two blocks have been selected. This event switches the two selected blocks and resets the selection.
Hint: you can use the integers to represent your selection and use -1 to represent an unselected block.

2.7 Perform LTL/CTL checking on the `Machine3` machine using ProB.

To do LTL/CTL checking on the `Machine3` machine follow the following steps:

1. Right-click the `Machine3` 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.

- It is impossible that the game goes on forever.
- Until the game is over, it is always possible to put a red block in the top left corner.
- Unless you've already won, you can always lose at the next timepoint.
- There is a single state in which it is possible to win with zero points and lose with zero points.
- When the game is over it stays over forever.
- Before every switch event there is always at least one `select1` and one `select2` event.

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 8 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 two parts:
   - 66% of the points are based on the specifications of the machines.
   - 33% of the points are based on the last part, i.e. the LTL and CTL verifications.

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 8 hours or less. The deadline for submitting your solution on Toledo is the 2nd of January 2018, 23.59.

Good luck!