MCS Project Part 1: Shift It

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

20 october, 2017

1 Introduction

In the MCS project, we will model the Android game Shift It[1] (most of it).

The project consists of two parts:

- In this first part, you model the game in IDP. This should be done in the LTC formalism. We use this theory to verify some properties of the game.
- In the second part (in about a month), you model this same game in Event-B and use CTL and LTL to verify other properties of the game.

In Section 2 we describe the game in general. Section 3 explains what you have to do for the first part of the assignment. Section 4 explains some practicalities. The project for this course must be completed **alone**.

2 The game

For this year's project you are tasked with modelling the Android game Shift It. This is a sliding game played on a rectangular board. In the original game, only square boards are allowed. This is a one-player game.

Initially, on each *position* on the board lies exactly one *tile*. Each tile is divided into four *sides* along the diagonals. Each side has a *colour*. Some tiles are *locked* into the position. An example starting setup is given in Figure 1.

[1] `https://play.google.com/store/apps/details?id=com.gamegou.ShiftIt.Google`

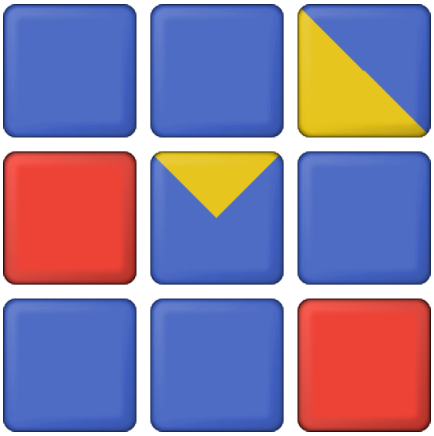

Figure 1: Example starting setup

2.1 Moves

There are essentially two types of moves:

Row moves The player selects a certain row and chooses a direction from {`left, right`}. Then the selected row will move *one* position in the chosen direction. In the original game, the player is allowed to move multiple positions in one move. Our solution, however, will not allow this. The tile that is off the board is placed at the empty position created by moving the row. A row move *cannot* be executed if the selected row contains a locked tile.

Column moves The player selects a certain column and chooses a direction from {`up, down`}. Then the selected column will move *one* position in the chosen direction. The tile that is off the board is placed at the empty position created by moving the column. A column move *cannot* be executed if the selected column contains a locked tile.

If there exists a valid move, the player *must* make a move.

2.2 End of the game

The game ends when the player cannot make a valid move or if for each colour that is present, the sides of that colour form a connected region. In the latter case, the player wins the game. Once the game has ended, the tiles do not change position. Two sides of the same colour that only touch in one corner are not connected. In Figure 2, every purple side is disconnected. Blue makes up 3 different connected colour regions.

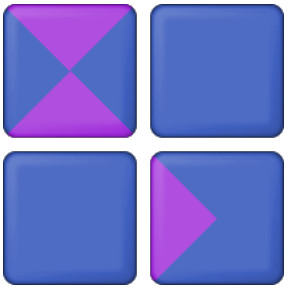

Figure 2: Connected colour regions

3 Part 1: Modelling the game in IDP

3.1 Build an LTC specification.

You are **obliged** to model this assignment in the LTC formalism! A structure of your LTC theory specifies a process of playing the game. There should be a one to one correspondence between the models of your LTC theory, and correct ways to play the game.

Do not add redundant constraints: each constraint should cut away a class of impossible worlds. For example, if f is a total function, the constraints $\forall x : \exists y : f(x) = y$ and $\forall x : \#\{y : f(x) = y\} \leq 1$ are useless. They follow from the fact that f is a total function. Adding useless constraints to your specification often makes it less readable, and less readable means less points.

3.2 Vocabulary

In order to model this, you use an extension of the following vocabulary:

```
LTCvocabulary ShiftItVoc {
    type Time isa int
    Start:Time
    partial Next(Time):Time
    type Co isa int
    type XCo isa Co
    type YCo isa Co
    type Pos constructed from { P(XCo, YCo) }
    type Dir constructed from { up, down, left, right }
    type Move constructed from { RM(YCo, Dir), CM(XCo, Dir) }
    type Tile
    type Side constructed from { S(Tile, Dir) }
```

```
    type Colour isa string

    partial MoveAt(Time):Move
    GetPos(Time, Tile):Pos
    Init_GetPos(Tile):Pos
    C_GetPos(Time, Tile):Pos
    CN_GetPos(Time, Tile):Pos
    IsLocked(Tile)
    Colouring(Side):Colour
    partial Init_Colouring_Full(Tile):Colour
    partial Init_Colouring(Side):Colour
    Reach(Time, Side, Colour)
    partial Ref(Colour): Side
    GameEnded(Time)
    Won(Time)
}
```

It is permitted, and advised, to extend this vocabulary with new predicates or functions but **no extra types**! You **cannot** change the names, arity, types, ... of the given predicates, types and functions, since our automatic grading is based on these properties.

The intended interpretation of the given symbols is as follows:

Time represents the time in your domain.

Start is the initial time point.

Next(t) is the successor time of t.

Co is the set of coordinates. These are integers.

XCo is the set of x-coordinates. These are integers. We use a standard mathematical axis system: bigger x coordinates are more to the right.

YCo is the set of y-coordinates. These are integers. We use a standard mathematical axis system: bigger y coordinates are higher.

Pos is the set of all positions (tuples of x and y coordinates) in the game.

Dir is a set of four directions {`left, right, up, down`}.

Move is the set of row and column moves. A row is determined by a y-coordinate and a column is determined by an x-coordinate.

Tile is the set of tiles.

Side is the set of all sides (tuples of t and d, where t is a tile and d is a direction.

Colour is a set consisting of strings representing colours. For the visualisation to work Colour should be a subset of {`"blue"`,`"red"`,`"green"`, `"yellow"`,`"orange"`,`"purple"`}. However, any other colour can be used as well, if one has no need to us the visualisation. Therefore, you should not specify this in your theory.

MoveAt(t) is the move played directly after time t (if there is a move). So MoveAt(Start) would be the first move.

GetPos(t, a) is the position of tile a at time t.

Init_GetPos, C_GetPos, CN_GetPos are standard LTC predicates to encode the initial state of GetPos and the causal effects.

IsLocked(a) holds if and only if tile a is locked.

Colouring(s) is the colour of side s.

Init_Colouring_Full(t) is the colouring of the full tile t. To ease the specification of the colouring of sides in a structure, we introduce two partial functions `Init_Colouring_Full` and `Init_Colouring`. The function `Init_Colour_Full(t)` specifies a colour for all sides of t simultaneously. This has lower precedence than `Init_Colouring`, i.e. if there is a side s of this tile such that `Init_Colouring(s)` is defined, then it should be coloured with `Init_Colouring(s)` instead.

Init_Colouring(s) specifies the colour of side s. See `Init_Colouring_Full` above for more information.

Reach(t, s, c) holds if and only if at time t the side s is in the colour c connected region of `Ref(c)`. This will be useful to define when the player has won.

Ref(c) is the largest side with colour c with the following order: $S(t_1, d_1) < S(t_2, d_2)$ if one of the following three cases occur:

- $Y(Start, t_1) < Y(Start, t_2)$
- $Y(Start, t_1) = Y(Start, t_2)$ and $X(Start, t_1) < X(Start, t_2)$
- $t_1 = t_2$ and $d_1 < d_2$,

where $down < left < right < up$ and $X(Start, t)$ and $Y(Start, t)$ denote the x-coordinate and the y-coordinate of the tile t at the start. Ref is defined partially since not all colours have to occur in a given game. In that case, `Ref(c)` is undefined.

GameEnded(t) holds iff the game is ended at time t, i.e. there are no possible moves or for each colour present, the sides with that colour form a connected region or the game was ended in a previous time.

Won(t) holds iff at time t that for each colour present, the sides with that colour form a connected region or `Won(s)` for s a previous time.

The IDP IDE supports autocompletion (CTRL+SPACE), use this to save yourself some time.

3.3 Verifications

Please, verify the following claims about your theory in the context of a given finite structure with finite time (use the verification procedures in the skeleton for this, i.e. implement them). The IDP manual[2] has a useful list of procedures.

1. It is possible to win the game.
2. If the game has not ended at the start and at some other time it is ended, then the game is won at that time.
3. For all colours c, the number of sides reached under c is maximal when the game is won.
4. It is possible to win the game without breaking up any colour connected component (it is allowed to merge colour components of course)
5. It is possible to win the game at a time t such that the position of any tile at Start is next(in one of the 4 principal directions) to the position at t or at the position at t.
6. When moving a column in opposite directions in subsequent time steps, the resulting board is the same as the initial board.

3.4 Visualisation

A function to visualise one run of the game (a ShiftItVoc-structure) is provided. In order to run it, you need to use the idp-ide (cfr exercise sessions). The visualisation can be found in the file visualisation.idp. In order to visualise a ShiftItVoc-structure `struct`, you use must include the visualisation file by adding `include "visualisation.idp"`. Subsequently, you must call `initVisualisation()` to initialise the visualisation. To, then, finally visualise `struct` you must run `visualiseStruct(struct)`. An example is given in skeleton.idp. Note that if you would run this example with an empty theorem, you will get weird visualisation, e.g. locked tiles that are in the wrong initial position.

Some explanation:

[2] `https://dtai.cs.kuleuven.be/krr/files/bib/manuals/idp3-manual.pdf`

- Locked tiles are visualised with 4 screws drawn on top of them.
- Arrows are shown next to the row or column that will be moved pointing in the direction of the intended movement.

4 Practical

The output of this part of the project consists of a `.zip` file with file name `LastnameFirstname` (where Lastname and Firstname are your actual names) containing

1. A *concise* report in which you discuss design decisions.
   - For each predicate symbol `p` and function symbol `f` you introduce, the report should contain a detailed description of its intended interpretation in the following format (which we also used above):
     - `Pred(x,y)` is true if and only if ⟨some condition on x and y here⟩,
     - `Func(x,y,z)` is ⟨some description here⟩.
   - The report should also contain the time you spent on this part of the project. The expected time needed is 10 to 15 hours.
2. A file "solution.idp", containing your IDP specification but without `include visualisation.idp` and such that the IDP IDE does not complain about your code (syntax).

About the code:

- You are obliged to start from `skeleton.idp`, and you are **not allowed** to change the given symbols in ShiftItVoc. Remember that you **are allowed** (and advised) to add new symbols to the vocabulary ShiftItVoc.
- You are **not allowed** to rename any theory, procedure and vocabulary as they are used for automated grading. The name of the structure `exampleS` can be changed since the automated grading uses other structures to test your solution.
- For testing purposes, you are allowed to edit the example structure `exampleS` to test other situations. In order to make it easily computable it is advised to not take boards with more than 16 tiles (4×4) (especially on windows, where there is no support for XSB).
- Use well-chosen names for introduced symbols and variables.

- Make sure there are no warnings, except for the *Verifying and/or autocompleting structure* warnings!
- Choose your ontology wisely. An important part of our quotation consists of checking whether your theory is a good and readable representation of the domain knowledge.
- Write comments! Clearly specify the meaning of every line in your theories!
- Use good indentation.
- Specify the type when quantifying a variable.

This assignment is graded mostly using automated tests (to see whether your specification matches the rules described in this document. Some points are given for readability of the code and automatic verifications.

You are allowed to discuss this project with other people, but are not allowed to copy any code from other students. 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.

You submit the result on Toledo before Sunday 19/11/2017, 23.59.

Good luck!