

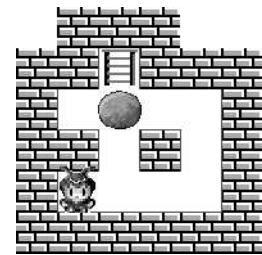


COMP 524-06A

Topics in Software Engineering

Maze Project

A maze consists of a rectangular grid of *squares*. Each square can be accessible, i.e., a passage, or inaccessible, i.e., a wall. Accessible squares can be free, or contain one of the following items: the *hero*, an *exit*, a *rock*, a *key*, a *door*, or a *gate*. The hero can move into free squares, or into squares containing a door or gate if the appropriate key has been picked up before. Rocks can be pushed to the square behind provided that there is space. The objective is to find a sequence of moves automatically by which the hero can reach an exit and thus escape from the maze.



A detailed specification of the problem in Z is given below.

The Task

The goal of this project is to use model checking to solve as many mazes as possible. A set of sample mazes is available. They are to be translated into the input language of NuSMV and VALID, and an appropriate question needs to be found. Then the model checker can be used to compute a counterexample that can be interpreted as a way of escaping from the given maze.

The work will be carried out as a group project by all COMP524-06A students together.

Overall, the following tasks need to be carried out.

- a) Read and understand the Z specification of the maze given below.
- b) Confirm that the refined version of the Z specification is equivalent to the original specification.
- c) Find a way of translating a maze into NuSMV and VALID code and create some sample models.
- d) Write a computer program that takes a maze description file as input and produces an appropriate NuSMV and VALID model as output, which can then be used to solve the maze.
- e) Compare the performance of the two model checkers by solving a set of sample mazes.
- f) Write a report detailing all the tasks carried out in the project.

The Maze in Z

The maze is represented in an abstract way, independently from the precise structure of positions and directions. We will only assume that there is some set of positions of squares, and a set of possible directions of movements. Also, there is a set of lock codes to tell how keys are related to doors and gates.

$$[POS, DIR, LOCK]$$

The following definition describes the concept of neighbouring squares in an abstract way. The only thing we need to know about neighbourhood is that there is a one-to-one function that, given a square and a direction, tells us the square reachable in that direction. Only a few assumptions are associated with this function, namely, a square is not neighbour of itself, and one does not return to a square by going twice in the same direction.

| |
|--|
| $neighbour : POS \times DIR \rightarrow POS$ |
| $\forall p_1 : POS \bullet \forall p_2 : POS \bullet \forall d : DIR \bullet$ $neighbour(p_1, d) = neighbour(p_2, d) \Rightarrow p_1 = p_2$ |
| $\forall p : POS \bullet \forall d : DIR \bullet neighbour(p, d) \neq p$ |
| $\forall p_1 : POS \bullet \forall p_2 : POS \bullet \forall d : DIR \bullet$ $neighbour(p_1, d) = p_2 \Rightarrow neighbour(p_2, d) \neq p_1$ |

Each accessible square can be free, or contain a hero, an exit, a rock, a key, a door, or a gate. Keys, doors, and gates also contain information about their lock code.

$$CONTENTS ::= free \mid hero \mid exit \mid rock \mid$$

$$door\langle\langle LOCK \rangle\rangle \mid gate\langle\langle LOCK \rangle\rangle \mid key\langle\langle LOCK \rangle\rangle$$

The state of the maze consists of a set of squares, represented as a map from positions of squares to their current contents, and the hero's inventory, consisting of the set of keys picked up. Walls are not represented explicitly, instead, it is assumed that the *square* function is undefined for inaccessible squares.

| |
|--|
| $Maze$ |
| $square : POS \rightarrow CONTENTS$ $keys : \mathbb{F} LOCK$ |
| $\#\{p : POS \mid square(p) = hero\} \leq 1$ |

The initial state of a maze can be an arbitrary configuration, however, there must be exactly one hero, and it is assumed that no keys have been picked up at the beginning.

| |
|---|
| $InitMaze$ |
| $\Delta Maze$ |
| $keys' = \emptyset$ $\exists_1 p : POS \bullet square'(p) = hero$ |

Now, the possible moves of the hero are defined. The following schema describes the part that is common to all moves. Each move involves a *source* square, containing the hero, an accessible *target* square, and a *direction* of movement connecting these two squares.

GeneralMove

$\Delta Maze$

$dir : DIR$

$src : POS$

$dest : POS$

$src \in \text{dom } square$

$dest \in \text{dom } square$

$square(src) = hero$

$neighbour(src, dir) = dest$

There are five possibilities of movement in the maze. The hero can *walk* into a free square, *escape* via an exit, *pick up* a key, *unlock* a door or gate, or *push* a rock. Each case is specified by its own schema.

Walk

GeneralMove

$square(dest) = free$

$square' = square \oplus \{src \mapsto free, dest \mapsto hero\}$

$keys' = keys$

Escape

GeneralMove

$square(dest) = exit$

$square' = square \oplus \{src \mapsto free\}$

$keys' = keys$

Pickup

GeneralMove

$k : LOCK$

$square(dest) = key(k)$

$square' = square \oplus \{src \mapsto free, dest \mapsto hero\}$

$keys' = keys \cup \{k\}$

Unlock

GeneralMove

$k : LOCK$

$k \in keys$

$square(dest) \in \{door(k), gate(k)\}$

$square' = square \oplus \{src \mapsto free, dest \mapsto hero\}$

$keys' = keys$

Push

GeneralMove

behind : *POS*

$behind \in \text{dom } square$

$behind = \text{neighbour}(dest, dir)$

$square(dest) = rock$

$square(behind) \neq rock$

$\forall k : LOCK \bullet square(behind) \neq gate(k)$

$square' = square \oplus \{src \mapsto free, dest \mapsto hero, behind \mapsto rock\}$

$keys' = keys$

Finally, the overall movement schema is obtained by combining the five possibilities of movement and hiding the auxiliary variables.

$$\begin{aligned} Move \cong & Walk \setminus (dir, src, dest) \vee \\ & Escape \setminus (dir, src, dest) \vee \\ & Pickup \setminus (dir, src, dest, k) \vee \\ & Unlock \setminus (dir, src, dest, k) \vee \\ & Push \setminus (dir, src, dest, behind) \end{aligned}$$

A Refined Specification

When using model checkers, it is important to represent the state space using as few bits as possible. Therefore, it is considered too expensive to maintain the set of keys in a separate data structure. This is also not necessary, if we note that we could immediately remove all doors when the corresponding key is picked up. With gates, this is not so easy, but we can at least mark them as *unlocked*.

The following new contents structure leads to a refined representation of the maze. First, the new state *unlocked_gate* is added to the possible contents of each square.

$$\begin{aligned} CONTENTS2 ::= & free_2 \mid hero_2 \mid exit_2 \mid rock_2 \mid \\ & door_2 \langle\langle LOCK \rangle\rangle \mid gate_2 \langle\langle LOCK \rangle\rangle \mid unlocked_gate_2 \mid key_2 \langle\langle LOCK \rangle\rangle \end{aligned}$$

Now the set of keys is removed from the maze state, and the general move schema is rewritten accordingly.

Maze2

$square_2 : POS \rightarrow CONTENTS2$

$\#\{p : POS \mid square_2(p) = hero_2\} \leq 1$

GeneralMove2

$\Delta Maze_2$

$dir_2 : DIR$

$src_2 : POS$

$dest_2 : POS$

$src_2 \in \text{dom } square_2$

$dest_2 \in \text{dom } square_2$

$square_2(src_2) = hero_2$

$neighbour(src_2, dir_2) = dest_2$

In the *Walk* and *Escape* schemas, it suffices to drop the reference to the *keys* set. The *pickup* schema now is more complicated, because it has to modify all doors affected by the key picked up. The *unlock* schema now only affects gates, and the *push* schema has to be modified to take unlocked gates into account.

Pickup2

GeneralMove2

$k_2 : LOCK$

$square_2(dest_2) = key_2(k_2)$

$square'_2 = square_2 \oplus (\{src_2 \mapsto free_2, dest_2 \mapsto hero_2\} \cup$

$\{p : POS \mid square_2(p) = door_2(k_2) \bullet p \mapsto free_2\} \cup$

$\{p : POS \mid square_2(p) = gate_2(k_2) \bullet p \mapsto unlocked_gate_2\})$

Unlock2

GeneralMove2

$square_2(dest_2) = unlocked_gate_2$

$square'_2 = square_2 \oplus \{src_2 \mapsto free_2, dest_2 \mapsto hero_2\}$

Push2

GeneralMove2

$behind_2 : POS$

$behind_2 \in \text{dom } square_2$

$behind_2 = neighbour(dest_2, dir_2)$

$square_2(dest_2) = rock_2$

$square_2(behind_2) \notin \{rock_2, unlocked_gate_2\}$

$\forall k : LOCK \bullet square_2(behind_2) \neq gate_2(k)$

$square'_2 = square_2 \oplus \{src_2 \mapsto free_2, dest_2 \mapsto hero_2, behind_2 \mapsto rock_2\}$

File Format

A simple text file format for mazes has been defined, and a couple of example mazes are available in this format from the course home page. Each file lists the squares of the maze as terms with the following syntax.

`X/Y. or X/Y - Contents.`

where `X` and `Y` identify the position of the square, and `Contents`, if present, specifies an item on the square. Possible contents are `hero`, `exit`, `rock`, `key(Id)`, `door(Id)`, and `gate(Id)`. The `Id` of a key, door, or gate is a number or an identifier representing the lock type, which defines how keys fit doors and gates. Inaccessible squares are not listed.

As an example, the problem situation given on the first page may be represented by the following file.

```
2/4 - exit.
1/3.  2/3 - rock.  3/3.  4/3.
2/2.  4/2.
1/1 - hero.  2/1.  3/1.  4/1.
```

There are only four directions of movement for these mazes, north, south, east, and west.

Deliverables

By the due date, the group should hand in a written report that includes the following.

- A detailed proof of the correctness of the refinement relationship between the two maze structures.
- A detailed description of how a maze is translated into NuSMV and VALID models.
- Source code of computer programs to convert maze descriptions into NuSMV and VALID models. These programs are to be demonstrated their use to solve some sample mazes.
- An evaluation of which mazes could be solved by which model checker, and a comparison of performance.

A final project demonstration will be held at the due date, during which each student will show the part of the work that they have done.

Due date: Wednesday 5 July 2006, 11:00 A.M.