Serving Robot - Formal Z Specification

Verification of Safe and Secure Systems Using Formal Specification

Andreas Zinkl

Summer Term 2017

Technical Computer Science
Ostbayerische Technische Hochschule Regensburg
June 23, 2017

Contents

1 Introduction

The project "Serving Robot" is about a NXC Lego Mindstorm Robot, which works in a restaurant as a helping robot. The robot helps employees by serving drinks and meals autonomously. The robot is equipped with sensors for route calculation.

The restaurant contains the bar and kitchen as well as three tables. The restaurant is split in a grid-layout. The grid-layout defines unique coordinates for every field in the map. The bar, kitchen and the tables are saved in the map as the start position. The map will be used for the navigation of the robot. The navigation algorithm calculates the shortest route to the destination table. The tables are ordered with different priorities. The priority of each table represents the table number. The tables are served in the order of the highest to the lowest priority.

The robot can just serve one table at a time. That means, that the robot needs to go back to the bar or kitchen after serving one table. Each table will be marked with one physical and unique coloured line. The robot has a color sensor which recognises the coloured line. If the robot reached the table, he turns around and drives the same way from the table back to the bar or kitchen. The start of the robot needs to be done by a user. This person just needs to press the start button on the robot and the robot starts to work.

The restaurant itself is a closed environment. That means, that the robot cannot leave the restaurant. The definition of the restaurant map is done by a text file, which is created by the user. The definition of the map in the text file contains the character '#' as symbols for borders, the character 'S' for the start position and numbers for the table. At the end the map looks like this:

```
##############
#     S      #
##   #####    #
#    11       #
#             #
#   ####   2  #
#      #   2  #
#      #      #
#   ##########
#   #         #
#   #   33    #
#   #         #
#             #
##############
```

Figure 2: *The map of the restaurant.*

2 Types and Global Variables

The following definitions define important types and global variables which are used over the whole project. The Type DIRECTION represents a command for the robot. Those commands will be sent to the robot and the robot will proceed them. The FreeType FIELD represents all characters which can be set on the map. This can be the position of the robot, the start position of the robot, the position of the borders as well as the positions of each table. Those Types needs to be global, because those are used from the hardware too.

The startposition of the robot is clearly specified and stays the same the whole process. This position also specifies the position of the kitchen / bar which is located as the standard robot startposition. The map of the restaurant is also defined in static values. The room has a static height and width and will not change while the system is running. The specified speed will be sent within a command to the robot. The specified speed limits will always stay the same.

$$[DIRECTION]$$

$$FIELD ::= robotposition \mid startpoint \mid tableone \mid tabletwo \mid tablethree \mid border \mid free \mid cutlery$$

$$startposition : (\mathbb{N} \times \mathbb{N}) \pfun FIELD$$
$$cutlerydesk : (\mathbb{N} \times \mathbb{N}) \pfun FIELD$$
$$mapWidth : \mathbb{N}_1$$
$$mapHeight : \mathbb{N}_1$$

$$fullspeed : \mathbb{P}\,\mathbb{Z}$$
$$halfspeed : \mathbb{P}\,\mathbb{Z}$$
$$stopspeed : \mathbb{P}\,\mathbb{Z}$$
$$\text{disjoint}\langle fullspeed, halfspeed, stopspeed\rangle$$

$$forward : \mathbb{P}\,DIRECTION$$
$$left : \mathbb{P}\,DIRECTION$$
$$right : \mathbb{P}\,DIRECTION$$
$$turn : \mathbb{P}\,DIRECTION$$
$$\text{disjoint}\langle forward, left, right, turn\rangle$$

3 System State Schema

3.1 Defining the System State

The System State defines the main state of the serving robot. This contains all necessary values like the map and all fields on the *map*. The map itself is split into various types of fields. Just one type of Fields, called *free*, can be used to navigate the robot through the restaurant to the destination table. Tables are a kind of border. A table would be an obstacle which can't be passed without any interaction problems between obstacle and the robot. Borders work like real borders and display walls on the map. The robot needs to circumvent those obastacles. On each serving track, the robot needs to stop at the *cutlerdesk*. There he picks up the clean cutler which he takes with him to serve the food and drinks. He will also stop at there while driving back from the table to the kitchen, because he will store the used and dirty cutler at the desk for the cleaners.

The track, how the robot needs to drive will be stored as a route. The route is a sequence of coordinates which specify all fields, the robot needs to pass, to get to the destination. The connections between those coordinates are stored in a so called value. The value *connections* specify, the link between each field and is necessary to calculate the route to the destination. This also means, that the connection exists not only of *freefields*. It also contains the *tables*, which are necessary to identify the destination coordinates of a route.

While the robot is driving, he counts the passed fields in a value called *drovenFields*. This value is needed to localize the current position, called *robotpos*, in the map and also to check, which command needs to be executed next. The current position is updated after each driving operation and always up to date.

ServingRobotState

$$
\begin{array}{l}
map, robotpos, freefields, borders, tables : (\mathbb{N} \times \mathbb{N}) \pfun FIELD \\
drovenFields : \mathbb{N} \\
connections : (\mathbb{N} \times \mathbb{N}) \rel (\mathbb{N} \times \mathbb{N}) \\
leftwheelspeed : \mathbb{Z} \\
rightwheelspeed : \mathbb{Z} \\
\hline
\langle freefields, borders, robotpos \rangle \text{ partition } map \\
\# map = (mapWidth * mapHeight) \\
tables \subset borders \\
\text{disjoint} \langle tables, borders \rangle \\
robotpos \subset freefields \\
\{tableone, tabletwo, tablethree\} \subseteq \text{ran } tables \\
robotpos \in \text{dom } map \\
\# robotpos = 1 \\
map(startposition) = startpoint \\
drovenFields <= \# route \\
(\text{dom } freefields \cup \text{dom } talbes) \subseteq \text{ran } connections \\
\text{dom } freefields \subseteq \text{dom } connections
\end{array}
$$

3.2 Initialization of the System State

The System State needs an initial state. This state defines the robot without any maps and positions. The robot is standing still in a room. The robot gets its position after starting the system of the robot and defining a map. This needs to be done manually. The robot will be able to start his work, after the system start.

InitServingRobotState

$ServingRobotState'$

$map' = \varnothing$

$connections' = \varnothing$

$freefields' = \varnothing$

$borders' = \varnothing$

$tables' = \varnothing$

$robotpos' = startposition$

$drovenFields' = 0$

$leftwheelspeed = stopspeed$

$rightwheelspeed = stopspeed$

4 Operations

The running system for a working restaurant robot needs some important operations and functions. Those define the work logic of the robots and how efficient the robots proceeds his tasks.

4.1 Route Calculation

The calculation of the route is one of the most important operations, the robot needs to do. The robot can only drive to his destination with a valid route. This valid route is always the best and shortest possible way. It navigates the robot to the tables, where he can serve drinks for the guests. The robot needs to move around the borders and obstacles of the restaurant. The tables on the map can be passed. This is possible because of the small size of the robot, which won't causes interaction problems with the guests. The robot calculates the route, by using the start position and destination coordinates. Those locations are free fields and should be connected by other free fields.

The *route*! will always be the shortest way between the start and the cutlery desk as well as between the cutlery desk and the destination point. For this reason, we can define, that there is no sequence *betweenRoute* between the start and the destination which is longer than the current *betweenRoute*. That means, that we drive through the minimum amount of fields between the given start and destination point. If the given destination is the table, then the calculated destination will be the last *free* field in front of the table.

Every time, the robot needs to serve a table or comes back from a table to the kitchen, he needs to go the cutlery desk. The cutlery desk is necessary to get the cutlery and plate, which will be used for the meal serving and to store the used cutlery for the cleaners.

RouteCalculation

$$
\begin{array}{l}
\Xi ServingRobotState \\
to?, from? : \mathbb{N} \times \mathbb{N} \\
route! : \mathrm{iseq}(\mathbb{N} \times \mathbb{N}) \\
\hline
(from? \in \mathrm{dom}\, startposition \land to? \in \mathrm{dom}\, tables) \\
\quad \lor (from? \in \mathrm{dom}\, tables \land to? \in \mathrm{dom}\, startposition) \\
(from?, cutlerydesk) \in connections^{+} \land (cutlerydesk, to?) \in connections^{+} \\
\\
\neg\ \exists\, betterRoute : \mathrm{iseq}(\mathbb{N} \times \mathbb{N}) \mid \\
\qquad head\ betterRoute = from? \land cutlerydesk \in betterRoute \land last\ betterRoute = to? \\
\quad \bullet\ \#\, betterRoute < \#\, route! \\
\\
\forall\, i : 1..(\#\, route! - 2) \\
\quad \mid route!(i) \in \mathrm{dom}\, freefields \\
\qquad \land\ route!(i+1) \in \mathrm{dom}\, freefields \\
\qquad \land\ 1 < route!^{\sim}(\!|\{cutlerydesk\}|\!) < \#\, route - 1 \\
\qquad \land\ (route!(1) \in startposition \land route!(\#\, route!) \in \mathrm{dom}\, tables) \\
\quad \bullet\ (route!(i), route!(i+1)) \in connections
\end{array}
$$

4.2 Serve a Table

If the robot gets the route, he will not know how to drive. The solution for this problem is a sequence of commands. Such commands tell the robot what he needs to do, to reach the destination. The command also calculates the new destination of the robot. The commands are split into driving left, right and forward. The robot does not need to drive backwards. Instead of driving backwards, the robot just turns around 180 degrees and starts driving forward.

$$Drive == DriveLeft \land DriveRight \land DriveForward \land TurnAround$$

4.2.1 Drive Left

Driving left has 4 special cases. Those ones can be calculated by using the destination field, the actual position and the last droven field. The driving task introduces a state change, which includes the change of the robot position, the speed of the robot as well as the new map. The generated command will be sent to robot with 2 bits, represented by the DIRECTION Type. The current position needs always to be updated. The reason therefore is, because we need to know at which position the robot is standing.

DriveLeft

$$\Delta ServingRobotState$$
$$cmd! : DIRECTION$$
$$route? : \text{iseq}(\mathbb{N} \times \mathbb{N})$$

$$drovenFields < \# route?$$
$$leftwheelspeed < (rightwheelspeed/2)$$
$$\forall\, m, n : \mathbb{N}_1$$
$$\quad | \ (m < mapWidth \land n < mapHeight) \land$$
$$\quad\quad (\ [(m, n+1) = route?(drovenFields - 1) \land (m+1, n) = route?(drovenFiels + 1)] \lor$$
$$\quad\quad\quad [(m-1, n) = route?(drovenFields - 1) \land (m, n+1) = route?(drovenFields + 1)] \ \lor$$
$$\quad\quad\quad [(m, n-1) = route?(drovenFields - 1) \land (m-1, n) = route?(drovenFields + 1)] \lor$$
$$\quad\quad\quad [(m+1, n) = route?(drovenFields - 1) \land (m, n-1) = route?(drovenFields + 1)] \ )$$
$$\quad\quad \bullet\ (m, n) = route?(drovenFields)$$

$$map' = (map \oplus \{route?(drovenFields) \mapsto freefield,$$
$$\qquad\qquad route?(drovenFields') \mapsto robotposition\})$$

$$drovenFields' = drovenFields + 1$$
$$rightwheelspeed' = fullspeed$$
$$leftwheelspeed' = halfspeed$$
$$cmd! = left$$

4.2.2 Drive Right

Driving right has, as well as driving left, 4 special cases. Those ones can be also calculated by using the destination field, the actual position and the last droven field. The new introduced ServingRobotState includes all changes like in the driving left task. The calculated command will be sent to the robot, to move into the correct position. Driving right is in this way similar to the driving left state.

DriveRight

$$
\begin{array}{l}
\Delta ServingRobotState \\
cmd! : DIRECTION \\
route? : \mathrm{iseq}(\mathbb{N} \times \mathbb{N}) \\
\hline
drovenFields < \# route? \\
rightwheelspeed < (leftwheelspeed/2) \\
\forall\, m, n : \mathbb{N}_1 \\
\quad \mid (m < mapWidth \land n < mapHeight) \land \\
\qquad ([(m, n+1) = route?(drovenFields - 1) \land (m-1, n) = route?(drovenFiels + 1)] \lor \\
\qquad [(m-1, n) = route?(drovenFields - 1) \land (m, n-1) = route?(drovenFields + 1)] \lor \\
\qquad [(m, n-1) = route?(drovenFields - 1) \land (m+1, n) = route?(drovenFields + 1)] \lor \\
\qquad [(m+1, n) = route?(drovenFields - 1) \land (m, n+1) = route?(drovenFields + 1)]) \\
\quad \bullet\ (m, n) = route?(drovenFields) \\
\\
map' = (map \oplus \{route?(drovenFields) \mapsto freefield, \\
\qquad\qquad route?(drovenFields') \mapsto robotposition\}) \\
\\
drovenFields' = drovenFields + 1 \\
leftwheelspeed' = fullspeed \\
rightwheelspeed' = halfspeed \\
cmd! = right
\end{array}
$$

4.2.3 Drive Forward

If the robot should drive forward, the robot will always stay on the same row or column on the map. For this calculation, we look also on the last field, the actual position and the destination field. Driving forward introduces like every task a new system state. This is necessary, because the position will change even if the wheelspeed may won't.

DriveForward

$\Delta ServingRobotState$

$cmd! : DIRECTION$

$route? : \text{iseq}(\mathbb{N} \times \mathbb{N})$

$drovenFields < \# route?$

$leftwheelspeed = rightwheelspeed$

$\forall\, m, n : \mathbb{N}_1$

$\quad | \ (m < mapWidth \land n < mapHeight) \land$

$\quad ([(m, n-1) = route?(drovenFields - 1) \land (m, n+1) = route?(drovenFiels + 1)] \lor$

$\quad\quad [(m-1, n) = route?(drovenFields - 1) \land (m+1, n) = route?(drovenFields + 1)] \ \lor$

$\quad\quad [(m, n+1) = route?(drovenFields - 1) \land (m, n-1) = route?(drovenFields + 1)] \ \lor$

$\quad\quad [(m+1, n) = route?(drovenFields - 1) \land (m-1, n) = route?(drovenFields + 1)] \ )$

$\quad \bullet \ (m, n) = route?(drovenFields)$

$map' = (map \oplus \{route?(drovenFields) \mapsto freefield,$

$\qquad\qquad route?(drovenFields') \mapsto robotposition\})$

$drovenFields' = drovenFields + 1$

$leftwheelspeed' = fullspeed$

$rightwheelspeed' = fullspeed$

$cmd! = forward$

4.2.4 Destination reached - Turn Around

After reaching the destination, the robot needs to drive back the whole route. For this, the robot just turns around and uses a new route for the drive back home. This new route will be calculated after the turn. The new route mostly is just just the reverse old route, because this is also the fastest way. The values, which display the robot and route state (meaning the amount of droven fields), will be set to the initial value. The new route will be passed like the old route before, just reverse. Also this tasks introduces a new ServingRobotState. The map stays the same, but the route will be reversed and the wheelspeed, as well as the drovenFields will be changed for this task.

TurnAround

$\Delta ServingRobotState$

$cmd! : DIRECTION$

$route? : \mathrm{iseq}(\mathbb{N} \times \mathbb{N})$

$drovenFields = \# route?$

$rightwheelspeed = -leftwheelspeed$

$map' = map$

$leftwheelspeed' = -fullspeed$

$rightwheelspeed' = fullspeed$

$drovenFields' = 0$

$cmd! = turn$