# In the Name of God

## Reactive Classes:

### Autonomous Car:

- what does it do??
  - Goes all the way up
  - Goes all the way down
  - Sometimes needs to wait because the shouldStop signal is issued

  - Know rebecs: Infrastructure
  - constructor argument: nothing!
  - Message servers:

```
149     msgsrv canAutoMove(boolean canMove) {
150         shouldStop = canMove;
151         asked = true;
152     }
153
154     msgsrv live() {
155         move();
156     }
157
```

- **Methods:**

```
void move() {

    if (asked || firstStep) {

        firstStep = false;
        infra.isAutoAllowed(line, block);
        asked = false;

        if (!shouldStop) {
            if (up && block == MAX_BLOCK_NUM) {
                block--;
                up = !up;
            }

            if (!up && block == 0) {
                up = !up;
                block++;
            }

            if (up && block < MAX_BLOCK_NUM && block > 0) {
                block++;
            }

            if (!up && block < MAX_BLOCK_NUM && block > 0) {
                block--;
            }
        }

        shouldStop = true;
    }

    self.live();
}
```

- **Queue capacity: 3**
- **The live method was added to the actor's class to avoid deadlocks**

## Manned Car:

## -what does it do??

- Goes wherever it wants
- Known Rebecs: Infrastructure
- Methods:

```
void move() {

    if (asked || firstStep) {

        firstStep = false;
        infra.isMannedAllowed(line, block);
        asked = false;

        if (shouldChange) {
            line = !line;
        } else {
            self.randomMove();
        }

        if (up && block == MAX_BLOCK_NUM) {
            block--;
            up = !up;
        }

        if (!up && block == 0) {
            up = !up;
            block++;
        }

        if (up && block < MAX_BLOCK_NUM && block > 0) {
            block++;
        }

        if (!up && block < MAX_BLOCK_NUM && block > 0) {
            block--;
        }

    }

    self.live();
}
```

- Message servers:

```
222
223     msgsrv canMannedMove(boolean canMove) {
224         shouldChange = canMove;
225         asked = true;
226     }
227
228     msgsrv live() {
229         move();
230     }
231
232     msgsrv randomMove() {
233         up = ?(true, false);
234         randomMoved = true;
235     }
236
```

## Infrastructure:

## -what does it do??

- Known Rebecs: Infrastructure
- Queue Capacity: 6
- Methods: doesn't have any!
- Message servers:

```
83
84      msgsrv isAutoAllowed(boolean line, int block, int autoId) {
85          autoCars[autoId] = ((Auto)sender);
86          ctrl.isAutoAllowed(line, block, autoId);
87      }
88
89      msgsrv canAutoMove(boolean canMove, int autoId) {
90          autoCars[autoId].canAutoMove(canMove);
91      }
92
93      msgsrv isMannedAllowed(boolean line, int block, int mannedId) {
94          mannedCars[mannedId] = ((Manned) sender);
95          ctrl.isMannedAllowed(line, block, mannedId);
96      }
97
98      msgsrv canMannedMove(boolean canMove, int mannedId) {
99          mannedCars[mannedId].canMannedMove(canMove);
100     }
```

## Controller:

- Know rebecs: Infrastructure, Autonomous, Manned
- Constructor argument:
- Message server: checkAccident → is called whenever a message is sent to either of thee 2 channels and checks the condition of the moving cars and if they are about to have an accident, a stop signal is issued to the autonomous car or the changeLine signal is issued to the manned car
- StateVars:

```
8
9       statevars {
10          boolean [10] shouldStop;
11          boolean [10] shouldChangeLine;
12
13          boolean [10] autoLine;
14          boolean [10] mannedLine;
15
16          int [10] autoBlock;
17          int [10] mannedBlock;
18      }
19
```

- Check method:

```
35
36      void check() {
37          for (int i = 0; i < MAX_BLOCK_NUM; i++) {
38
39              for(int j = 0; j < MAX_BLOCK_NUM; j++) {
40
41                  if (autoLine[i] != mannedLine[j] && (autoBlock[i] == mannedBlock[j]+1 || autoBlock[i] == mannedBlock[j]+1) ) {
42                      shouldStop[i] = true;
43                  }
44
45                  if (autoLine[i] == mannedLine[j] && (autoBlock[i] == mannedBlock[j]+1 || autoBlock[i] == mannedBlock[j]+1) ) {
46                      shouldStop[i] = true;
47                      shouldChangeLine[j] = true;
48                  }
49
50              }
51          }
52      }
53
```

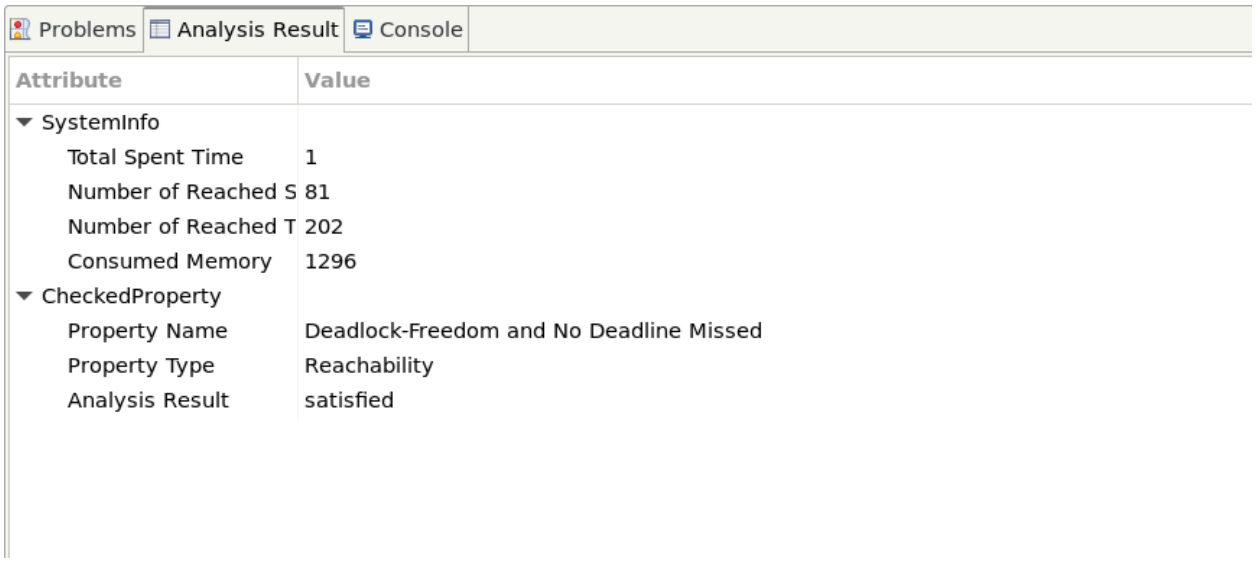
- **Message servers:**

```
53
54    msgsrv isAutoAllowed(boolean line, int block, int autoId) {
55        autoBlock[autoId] = block;
56        autoLine[autoId] = line;
57
58        check();
59        ((Infrastructure)sender).canAutoMove(shouldStop[autoId], autoId);
60    }
61
62    msgsrv isMannedAllowed(boolean line, int block, int mannedId) {
63        mannedBlock [mannedId] = block;
64        mannedLine [mannedId] =line;
65
66        check();
67        ((Infrastructure)sender).canMannedMove(shouldChangeLine[mannedId], mannedId);
68    }
69
```

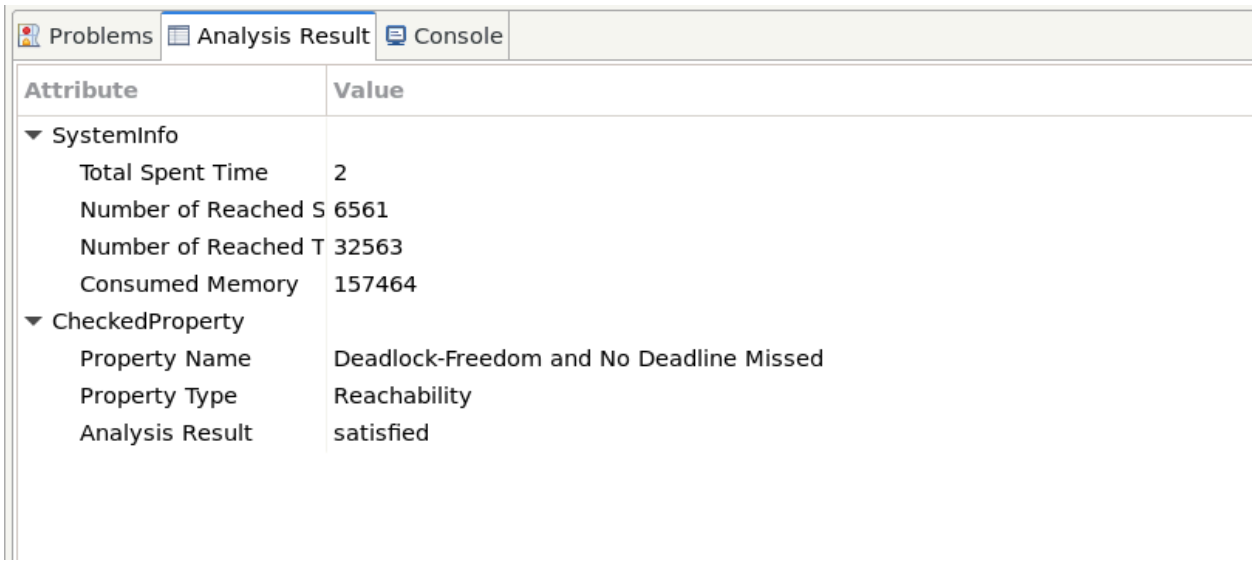
```
44
45    msgsrv isAutoAllowed(boolean line, int block) {
46        check();
47        ((Infrastructure)sender).canAutoMove(shouldStop);
48    }
49
50    msgsrv isMannedAllowed(boolean line, int block) {
51        check();
52        ((Infrastructure)sender).canMannedMove(shouldChangeLine);
53    }
```

- **Queue capacity: 6**

## Results for 1 autonomous car and 1 manned car:

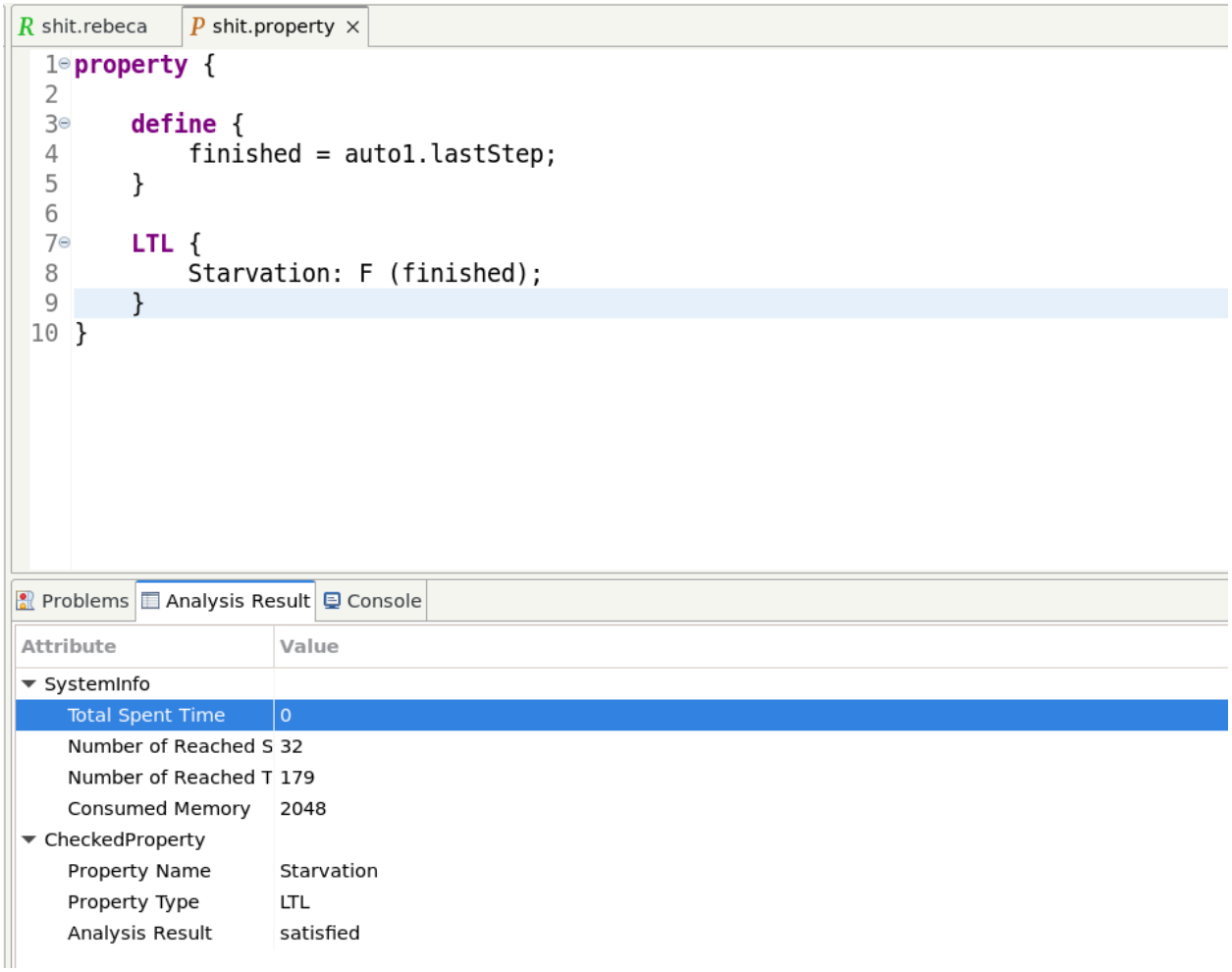



## Results for 2 autonomous cars and 2 manned cars:



## Properties:

1. Each autonomous car should finish its path: (for 1 car)




With 2 cars:

shit.rebeca | shit.property ×

```
1  property {
2
3      define {
4          finished1 = auto1.lastStep;
5          finished2 = auto2.lastStep;
6      }
7
8      LTL {
9          Starvation1: F (finished1);
10         Starvation2: F (finished2);
11     }
12 }
13
```

Problems | Analysis Result | Console

| Attribute | Value |
|---|---|
| SystemInfo | |
| Total Spent Time | 1 |
| Number of Reached S | 965 |
| Number of Reached T | 14071 |
| Consumed Memory | 69480 |
| CheckedProperty | |
| Property Name | Starvation2 |
| Property Type | LTL |
| Analysis Result | satisfied |

2. Safety property:

```
property {

    define {
        finished1 = auto1.lastStep;
        //finished2 = auto2.lastStep;

        isSafetyViolated = ctrl.safetyCheck;
    }

    LTL {

        Starvation1: F (finished1);
        //Starvation2: F (finished2);

        Safety: G(!isSafetyViolated);
    }
}

```

Problems | Analysis Result | Console

| Attribute | Value |
|---|---|
| ▼ SystemInfo | |
| Total Spent Time | 0 |
| Number of Reached S | 189 |
| Number of Reached T | 491 |
| Consumed Memory | 12096 |
| ▼ CheckedProperty | |
| Property Name | Safety |
| Property Type | LTL |
| Analysis Result | satisfied |

## Is safety violated was defined in the controller environment:

```
void check() {
    for (int i = 0; i < MAX_BLOCK_NUM; i++) {

        for(int j = 0; j < MAX_BLOCK_NUM; j++) {

            if (autoLine[i] != mannedLine[j] && (autoBlock[i] == mannedBlock[j]+1 || autoBlock[i] == mannedBlock[j]+1) ) {
                shouldStop[i] = true;
            }

            if (autoLine[i] == mannedLine[j] && (autoBlock[i] == mannedBlock[j]+1 || autoBlock[i] == mannedBlock[j]+1) ) {
                shouldStop[i] = true;
                shouldChangeLine[j] = true;
            }

            if (autoLine[i] == mannedLine[j] && autoBlock[i] == mannedBlock[j]) {
                safetyCheck = true;
            }
        }
    }
}
```