ECE 18-649
Final Project Report

December 4, 2013

Group # 9

Wenhui Hu
Yichao Xue
Yujia Wang

Overview

- Content
  - Project statistics (Wenhui Hu)
  - Design of Dispatcher
    - Sequence Diagram (Wenhui Hu)
    - Requirements (Wenhui Hu)
    - Statecharts (Yujia Wang)
    - Code (Yujia Wang)
    - Testing (Yichao Xue)
  - Lessons learned (Yichao Xue)
  - Suggestion (Yichao Xue)

Project Statistics

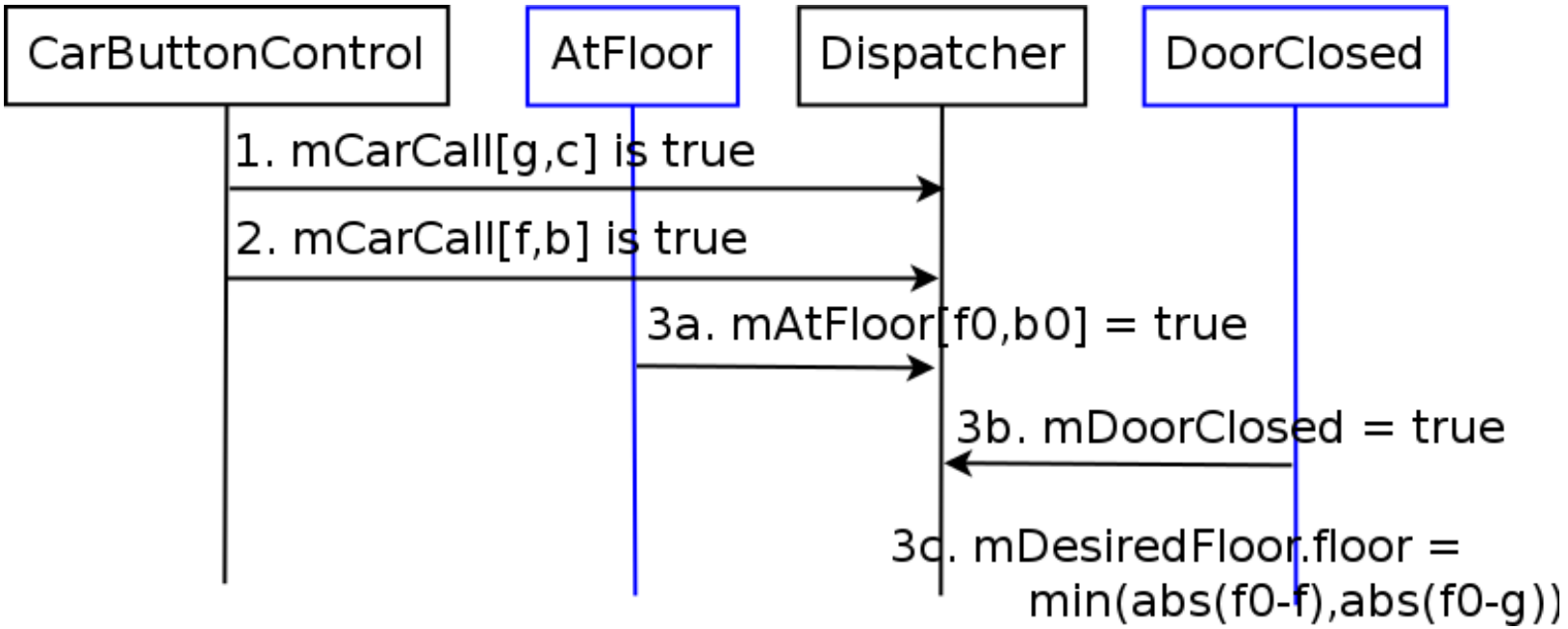
# of	Midterm	Final
Scenarios/Sequence Diagrams	20	20
Total arcs in sequence diagrams	131	133
Requirements	51	78
State charts/controllers	7	7
Total states in state charts	26	35
Total arcs in state charts	32	78
Non-comment code	1392(62%)	2542(68%)
Test files written	28	39
Change log entries	21	35
Peer review	79	107
Defects found via peer review	47	88
Defects found via test & others	69	163

Design of Dispatcher

- Sequence Diagram
- Requirements
- State Chart
- Code
- Testing

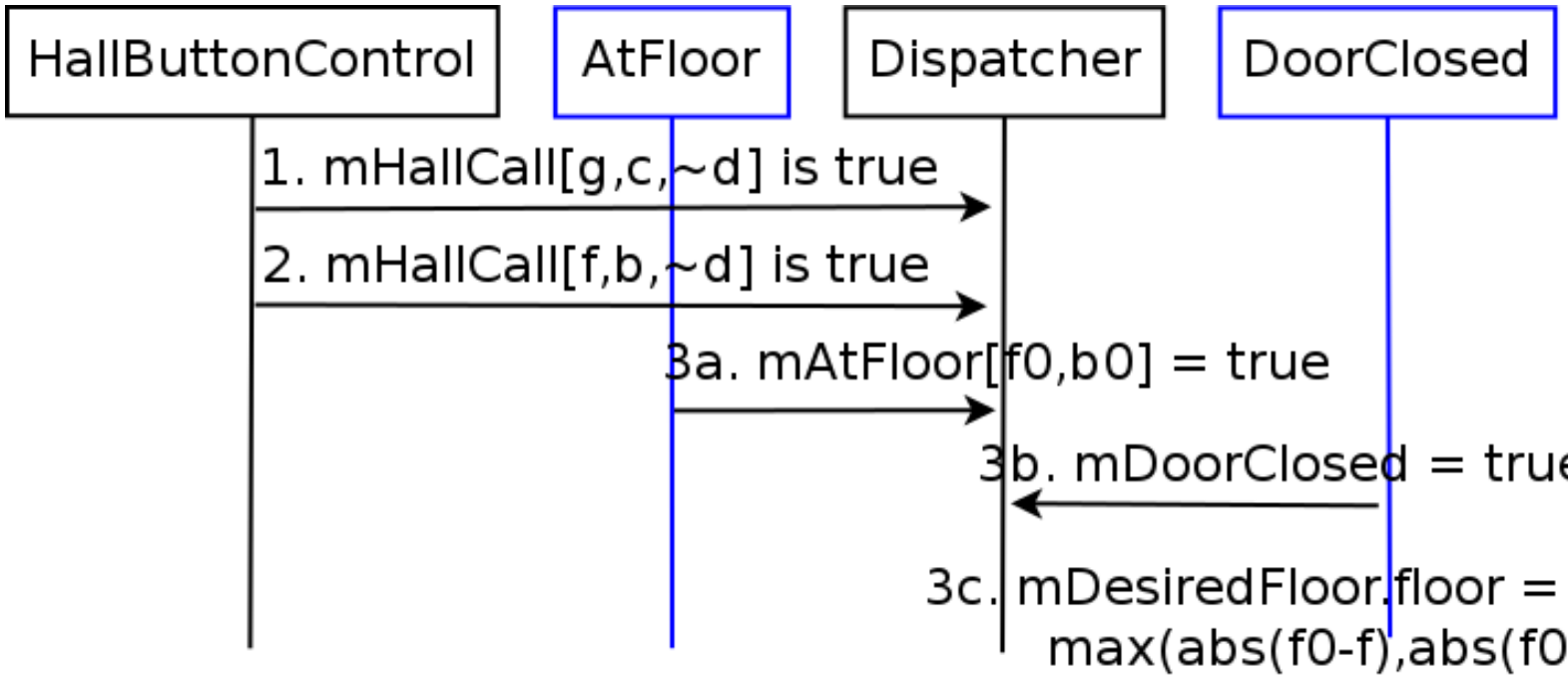
Sequence Diagram

- Scenario 7C
  - Elevator determines desired floor when there are 2 car calls for different floors. Both calls have the same desired direction as current direction.
- Sequence Diagram



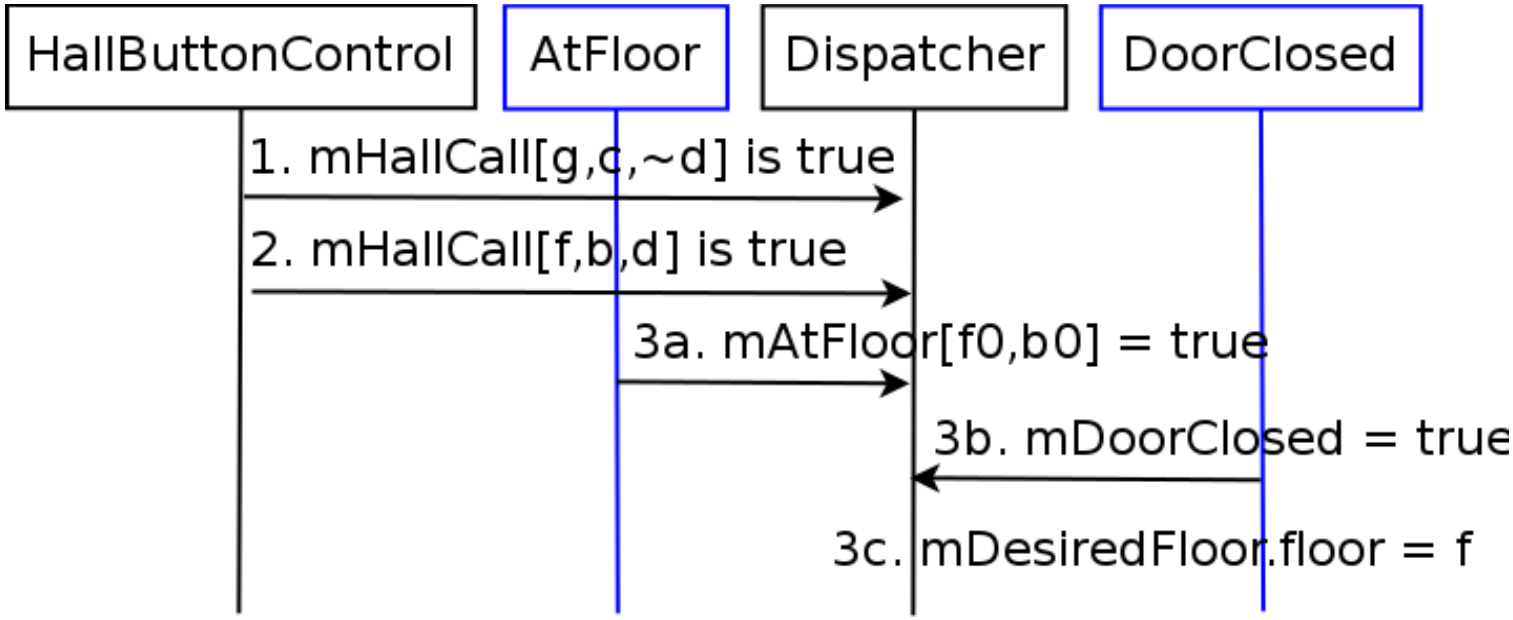
Sequence Diagram

- Scenario 7D
  - Elevator determines desired floor when there are 2 hall calls for different floors. Both calls have the opposite desired direction as current direction.
- Sequence Diagram



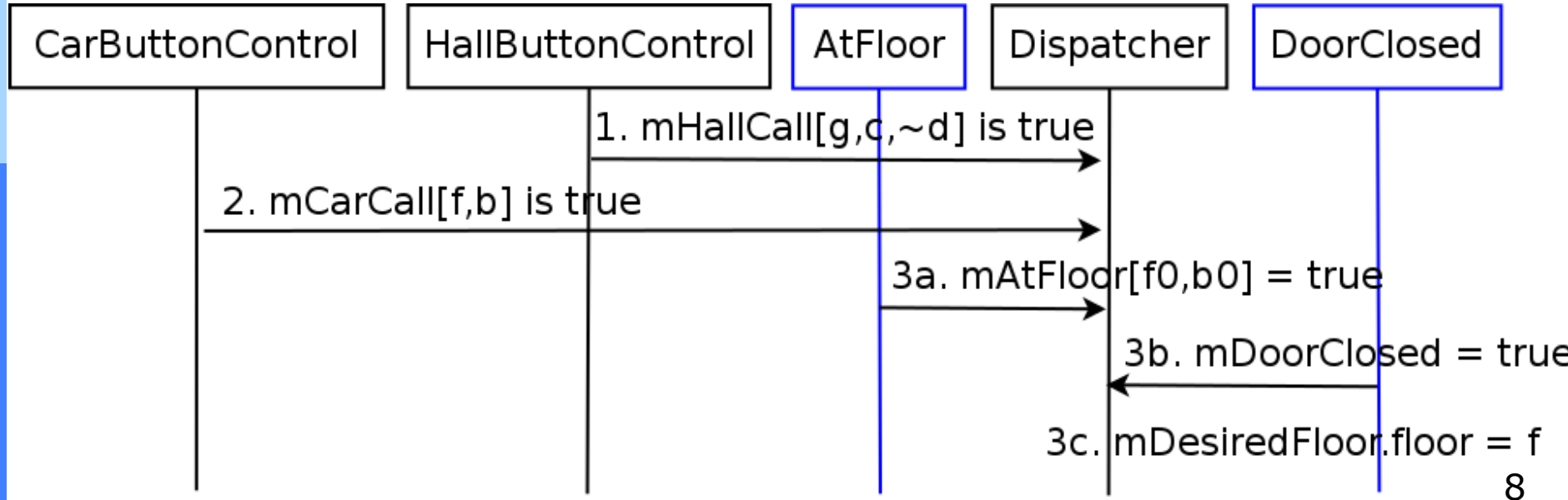
Sequence Diagram

- Scenario 7E
  - Elevator determines desired floor when there are 2 hall calls for different floors. One has the same desired direction as current direction, the other has the opposite.
- Sequence Diagram



Sequence Diagram

- Scenario 7F
  - Elevator determines desired floor when there are 1 car call and 1 hall call from different floors. Car call has the same desired direction as current direction, hall call has opposite desired direction as current direction.
- Sequence Diagram



Time-triggered Requirements

- 11.8 If CurrentDirection is UP and mDriveSpeed.s > level speed and mCarCall[f1, b1] is true and mCarCall[f2, b2] is true and CurrentFloor <= f1 < f2, then
  - 11.8.1 Target shall be set to the **nearest** floor of calls
  - 11.8.2 DesireDirection shall be set to UP

- 11.9 If CurrentDirection is UP and mDriveSpeed.s > level speed and mCarCall[f1, b1] is true and mHallCall[f2, b2, *] is true and CurrentFloor <= f1 < f2, then
  - 11.9.1 Target shall be set to the **nearest** floor of calls
  - 11.9.2 DesireDirection shall be set to UP

- 11.10 If CurrentDirection is UP and mDriveSpeed.s > level speed and mHallCall[f2, b2, UP] is true and CurrentFloor <= f1
  - 11.10.1 Target shall be set to the **nearest** floor of calls
  - 11.10.2 DesireDirection shall be set to UP

Time-triggered Requirements

- 11.11 If CurrentDirection is UP and mDriveSpeed.s > level speed and mCarCall[f1, b1] is true and mCarCall[f2, b2] is true and f2 < CurrentFloor <= f1 and other mCarCall[f,b] or mHallCall[f,b,*] is false, which f > CurrentFloor
  - 11.11.1 Target shall be set to the **farthest** floor of calls
  - 11.11.2 DesireDirection shall be set to DOWN

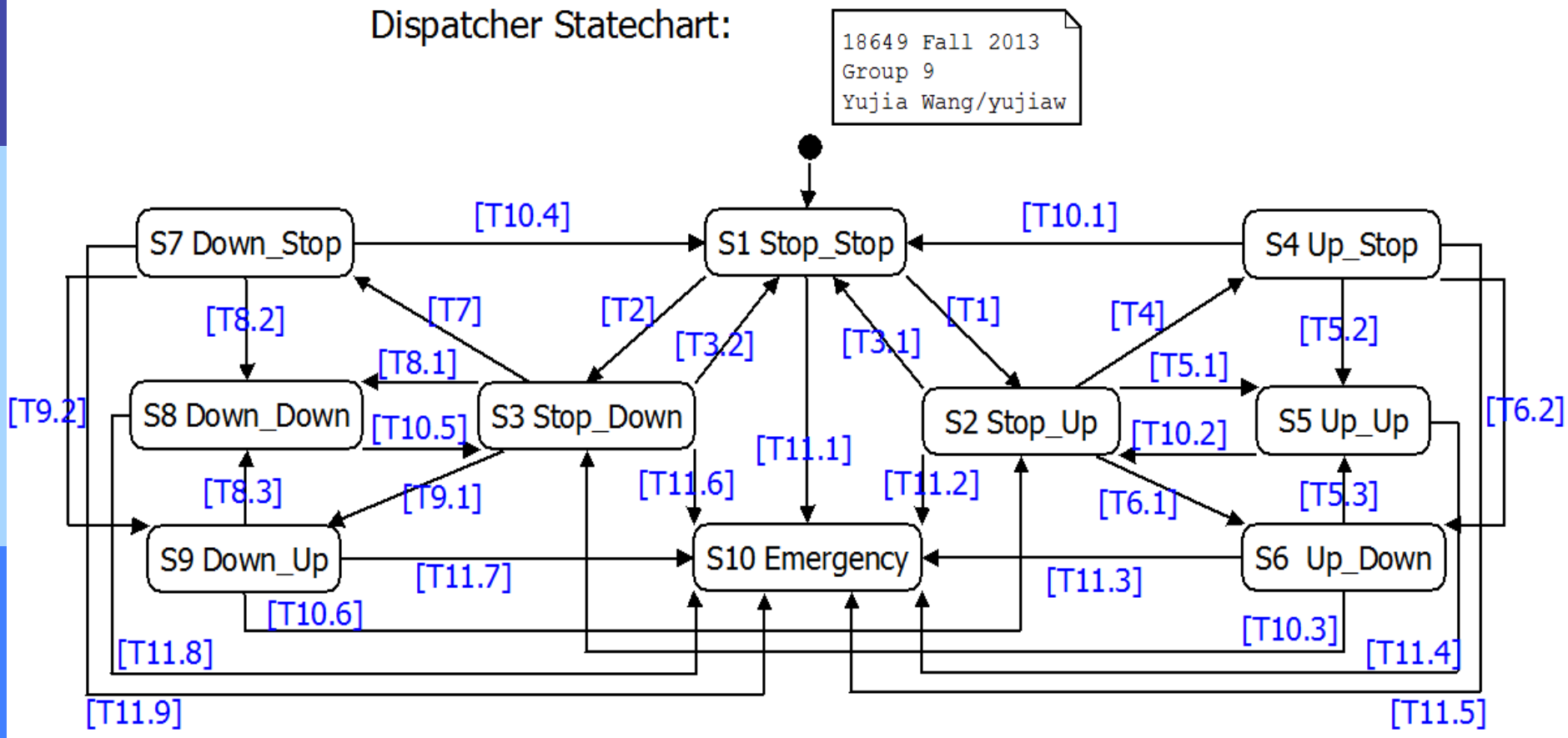
- 11.12 If CurrentDirection is UP and mDriveSpeed.s > level speed and mCarCall[f1, b1] is true and mHallCall[f2, b2, DOWN] is true and f2 < f1 and CurrentFloor <f1 and other mCarCall[f,b] or mHallCall[f,b,*] is false, which f > CurrentFloor
  - 11.12.1 Target shall be set to the **farthest** floor of calls
  - 11.12.2 DesireDirection shall be set to DOWN

Time-triggered Requirements

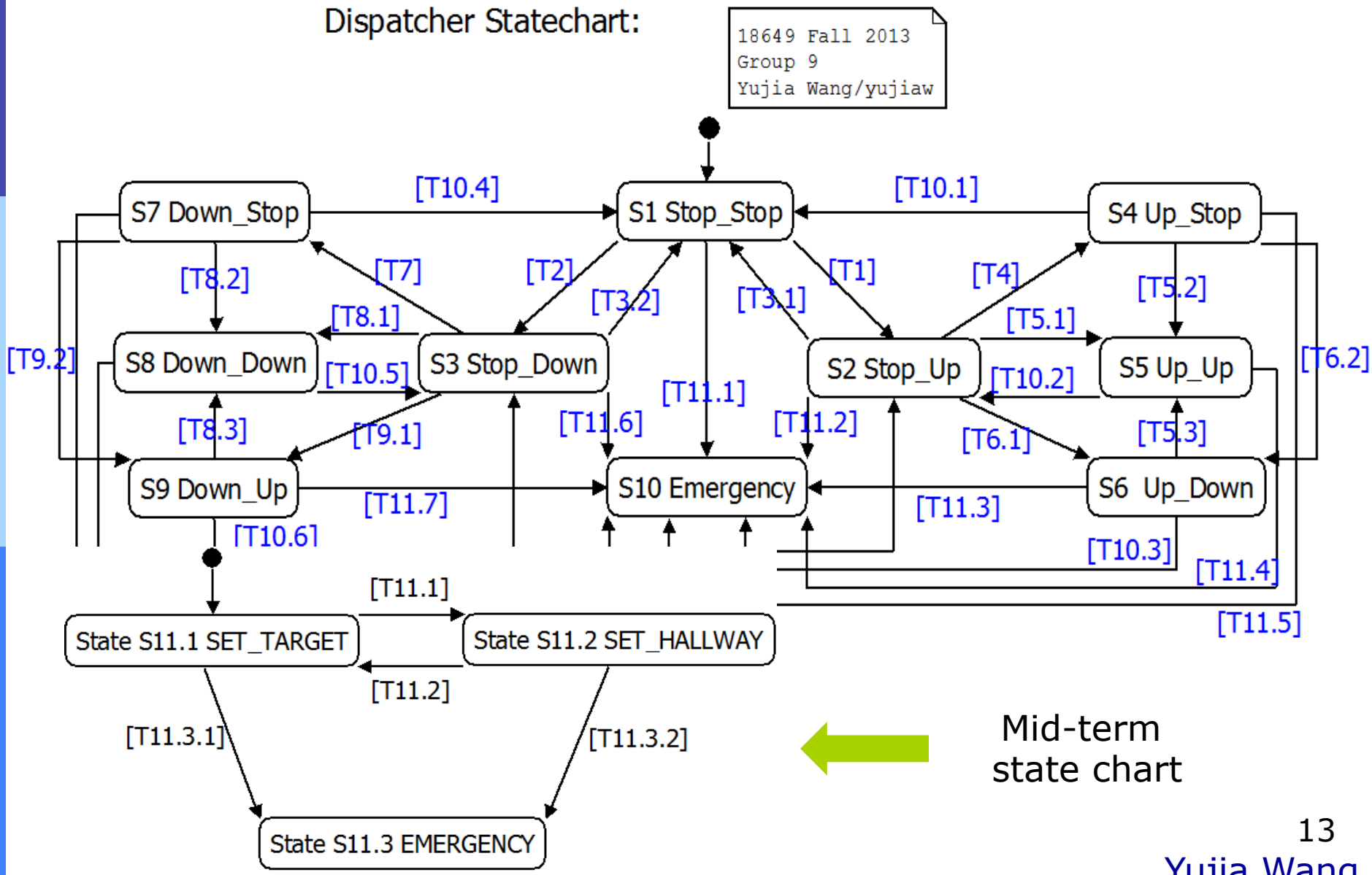
- 11.13 If CurrentDirection is UP and mDriveSpeed.s > level speed and mCarCall[f1, b1] is true and mHallCall[f2,b2,*] is true and f2 < CurrentFloor < f1 and other mCarCall[f,b] or mHallCall[f,b,*] is false, which f > CurrentFloor
  - 11.13.1 Target shall be set to the **farthest** floor of calls
  - 11.13.2 DesireDirection shall be set to DOWN

- 11.14 If CurrentDirection is UP and mDriveSpeed.s > level speed and mHallCall[f2,b2,DOWN] is true and CurrentFloor <= f1 and other mCarCall[f,b] or mHallCall[f,b,*] is false, which f > CurrentFloor
  - 11.14.1 Target shall be set to the **farthest** floor of calls
  - 11.14.2 DesireDirection shall be set to DOWN

State Chart


Dispatcher Statechart:
18649 Fall 2013
Group 9
Yujia Wang/yujiaw
S7 Down_Stop
S1 Stop_Stop
S4 Up_Stop
S8 Down_Down
S3 Stop_Down
S2 Stop_Up
S5 Up_Up
S9 Down_Up
S10 Emergency
S6 Up_Down
[T10.4]
[T10.1]
[T8.2]
[T7]
[T2]
[T3.2]
[T3.1]
[T1]
[T4]
[T5.2]
[T8.1]
[T5.1]
[T9.2]
[T10.5]
[T10.2]
[T6.2]
[T11.1]
[T8.3]
[T9.1]
[T11.6]
[T11.2]
[T6.1]
[T5.3]
[T11.7]
[T11.3]
[T10.6]
[T10.3]
[T11.8]
[T11.4]
[T11.9]
[T11.5]

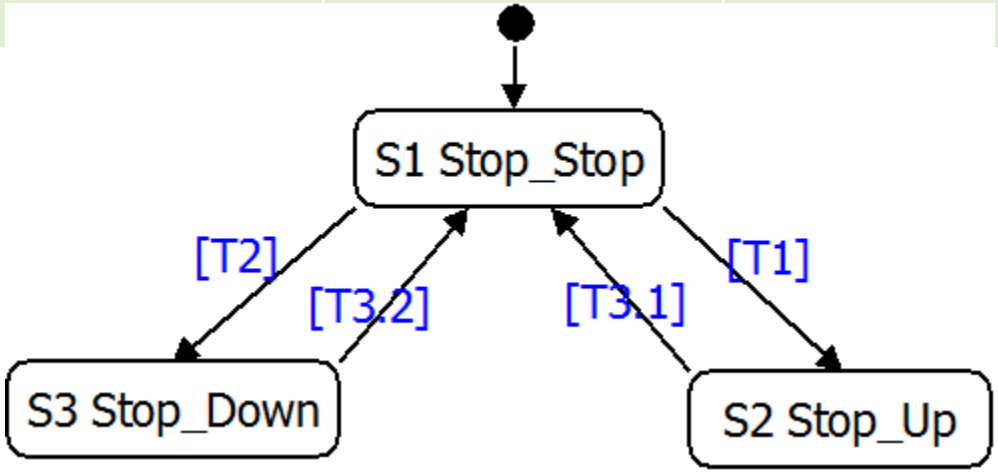
State Chart



Mid-term state chart

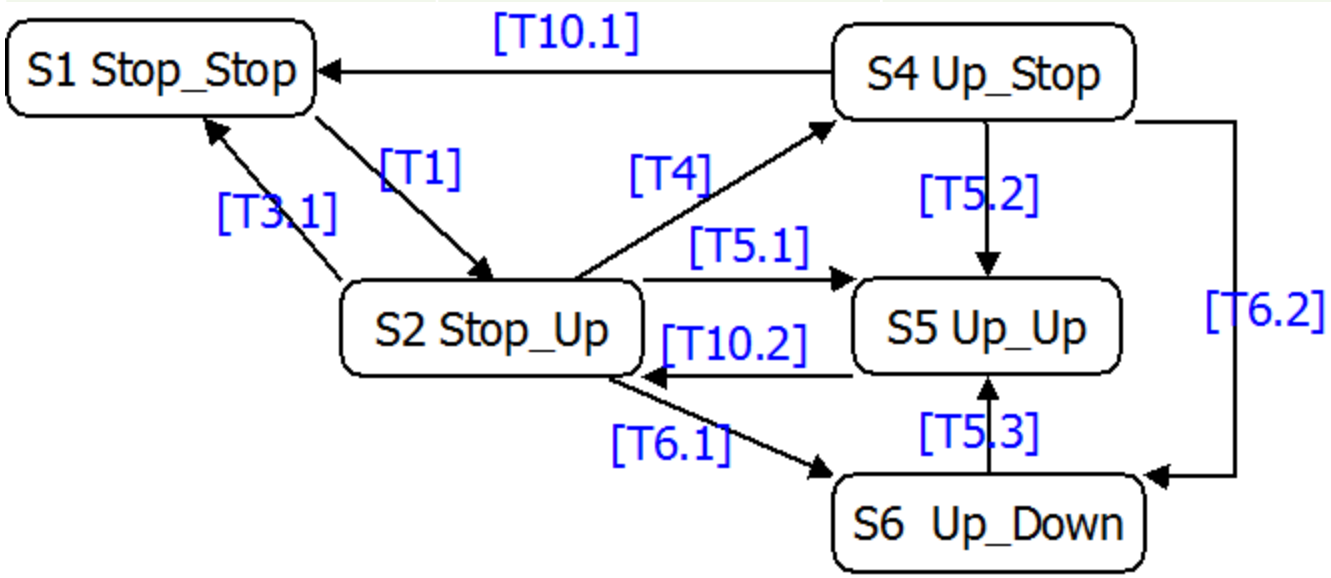
State Description

State	Description	Actions
S1 Stop_Stop	Car is stopped, DesiredDirection is Stop	set Target to **None**, set DesiredHallway to CurrentHallway, set DesiredDirection to Stop
S2 Stop_Up	Car is stopped, DesiredDirection is Up	set Target to the **nearest** floor of calls, set DesiredHallway to CurrentHallway, set DesiredDirection to Up
S3 Stop_Down	Car is stopped, DesiredDirection is Down	set Target to the **nearest** floor of calls, set DesiredHallway to CurrentHallway, set DesiredDirection to Down



State Description – cont.

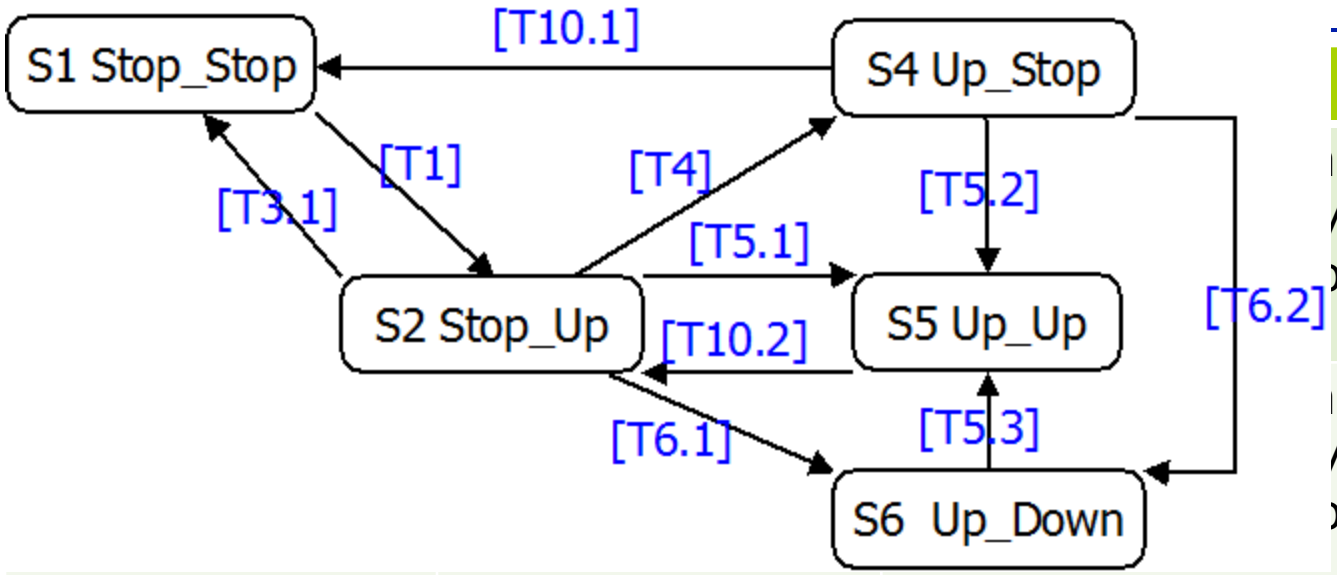
State	Description	Actions
S4 Up_Stop	Car is moving up, DesiredDirection is Stop	set Target to the **nearest** floor of calls, set DesiredHallway to CurrentHallway, set DesiredDirection to Stop
S5 Up_Up	Car is moving up, DesiredDirection is Up	set Target to the **nearest** floor of calls, set DesiredHallway to CurrentHallway, set DesiredDirection to Up



arthest floor of calls,
y to CurrentHallway,
on to Down

t DesiredHallway to
Direction to Stop

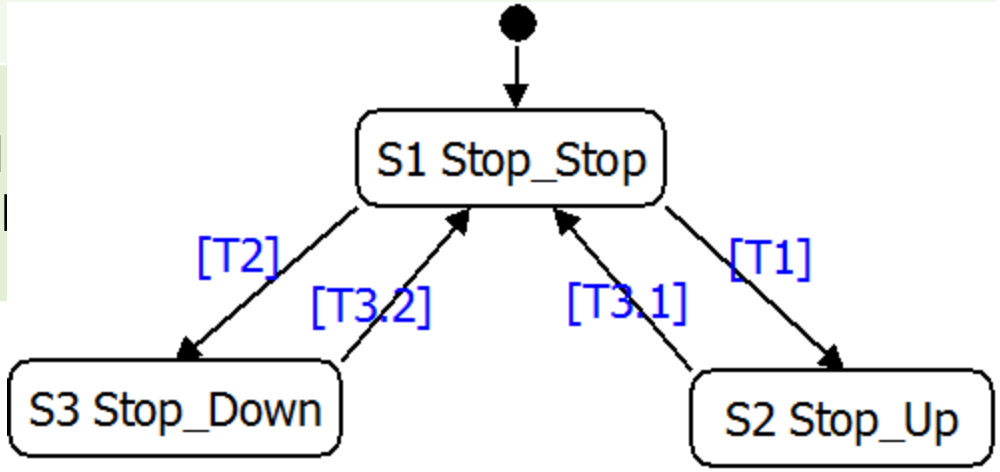
State Description – cont.



		earest floor of calls, to CurrentHallway, on to Stop
		earest floor of calls, to CurrentHallway, on to Up
S6 Up_Down	Car is moving up, DesiredDirection is Down	set Target to the **farthest** floor of calls, set DesiredHallway to CurrentHallway, set DesiredDirection to Down
S10 Emergency	Doors are not closed between floors	set Target to 1, set DesiredHallway to None, set DesiredDirection to Stop

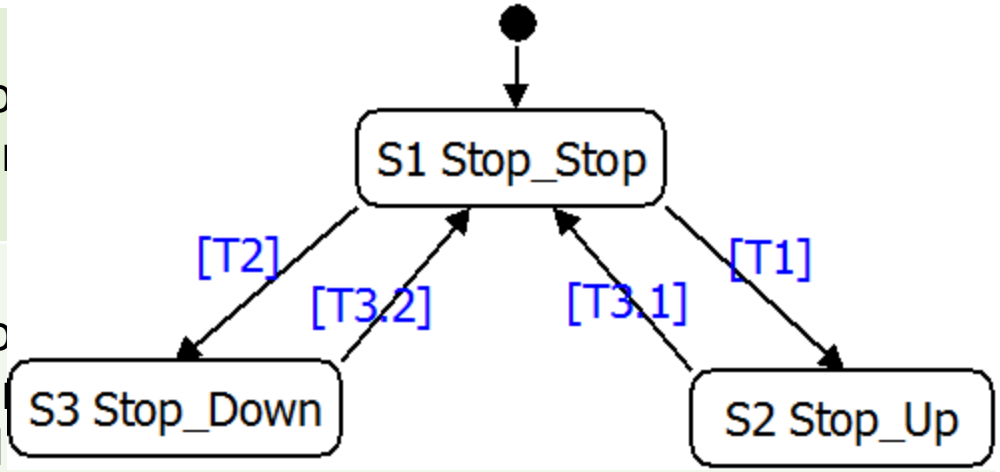
Transitions

Transition	Guard
T1	all mDoorClosed ← true **AND** (mCarCall or mHallCall above CurrentFloor ← true **OR** mHallCall[CurrentFloor, *, UP] ← true)
T2	all mDoorClosed ← true **AND** (mCarCall or mHallCall below CurrentFloor ← true **OR** mHallCall[CurrentFloor, *, DOWN] ← true) **AND** all mCarCall or mHallCall above CurrentFloor ← false
T3.*	all mDoorClosed **AND** all mCarCal **AND** all mHallCal



Transitions

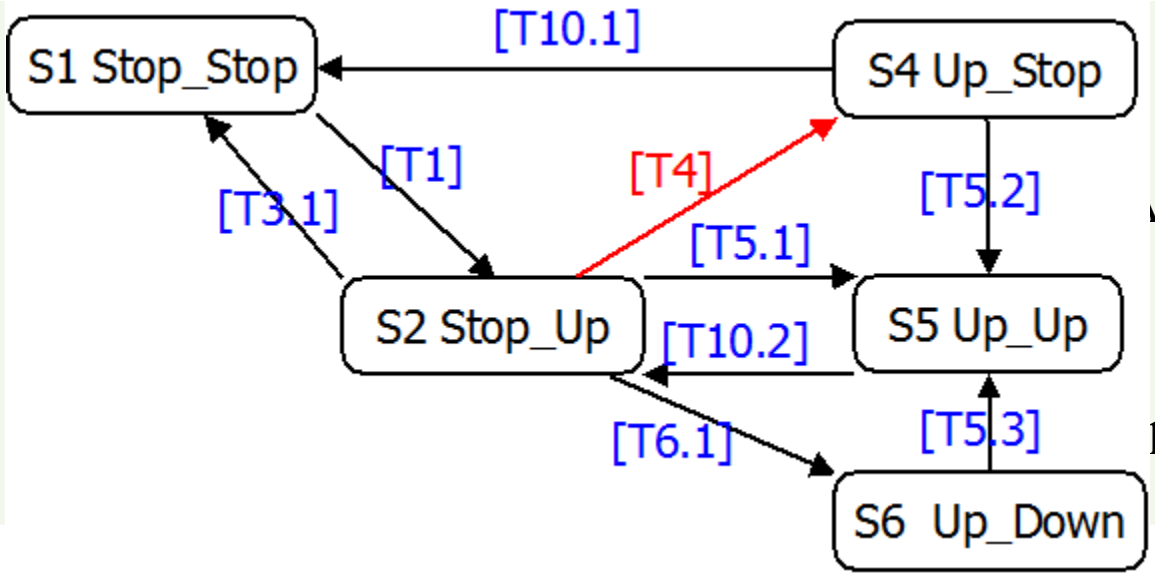
Transition	Guard
T1	all mDoorClosed **AND** (mCarCall o **OR** mHallCall[Cu
T2	all mDoorClosed **AND** (mCarCall o **OR** mHallCall[Cu **AND** all mCarCal false
T3.*	all mDoorClosed ← true **AND** all mCarCall ← false **AND** all mHallCall ← false



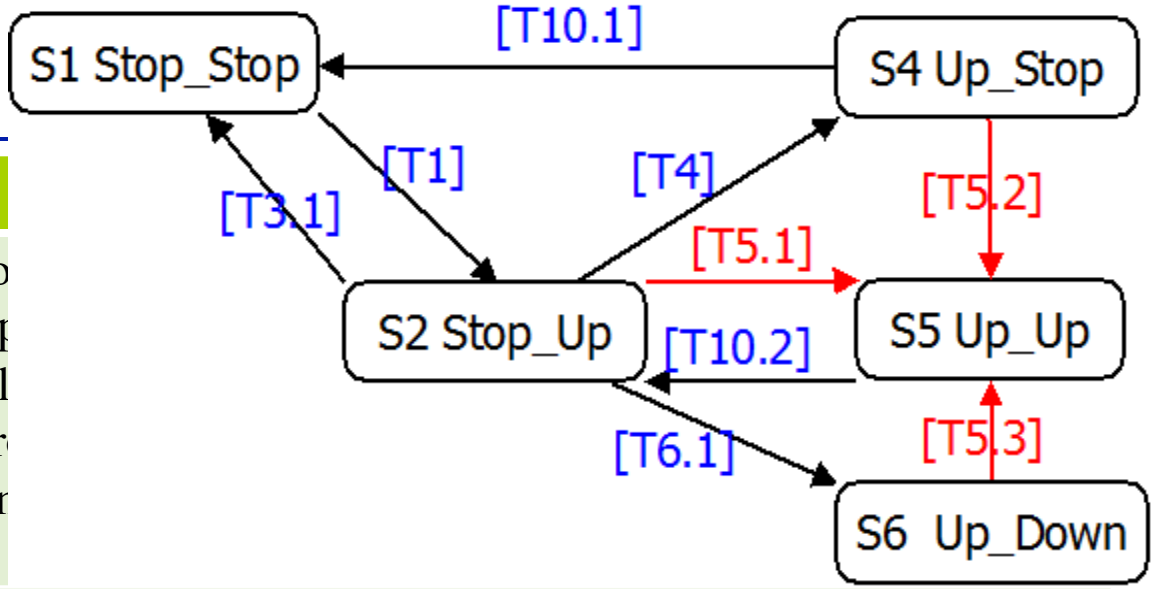
Transitions – cont.1

Transition	Guard
T4	CurrentDirection ← UP **AND** mDriveSpeed.s > Leveling_Speed **AND** mCarCall[f, b] ← true **AND** f >= CurrentFloor **AND** all other mCarCall and mHallCall ← false
T5.*	CurrentDirection ← UP **AND** mDriveSpeed.s > Leveling_Speed **AND** ((mCarCall[f1, b1] ← true *AND* mCarCall[f2, b2] ← true AND CurrentFloor <= f1 < f2) *OR* (mCarCall[f1, b1] ← true *AND* mHallCall[f2, b2, *] ← true AND CurrentFloor <= f1 < f2) *OR* (mHallCall[f1, b1, UP] ← true *AND* CurrentFloor <= f1))

Transitions – cont.1

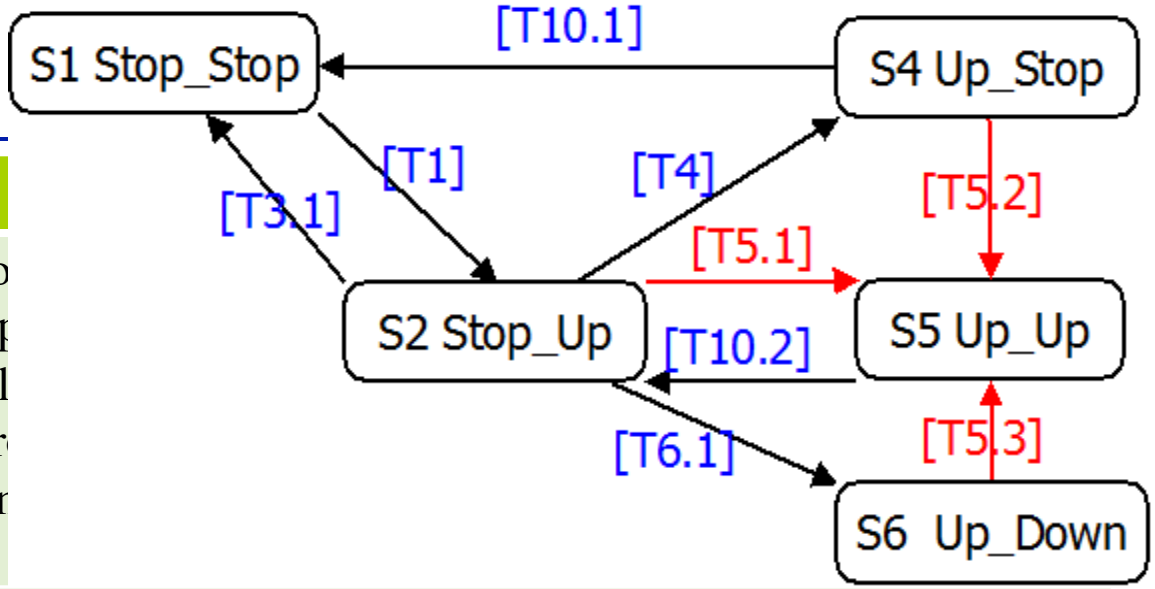
Transition	Guard
T4	CurrentDirection ← UP **AND** mDriveSpeed.s > Leveling_Speed **AND** mCarCall[f, b] ← true **AND** f >= CurrentFloor **AND** all other mCarCall and mHallCall ← false
T5.*	

Transitions



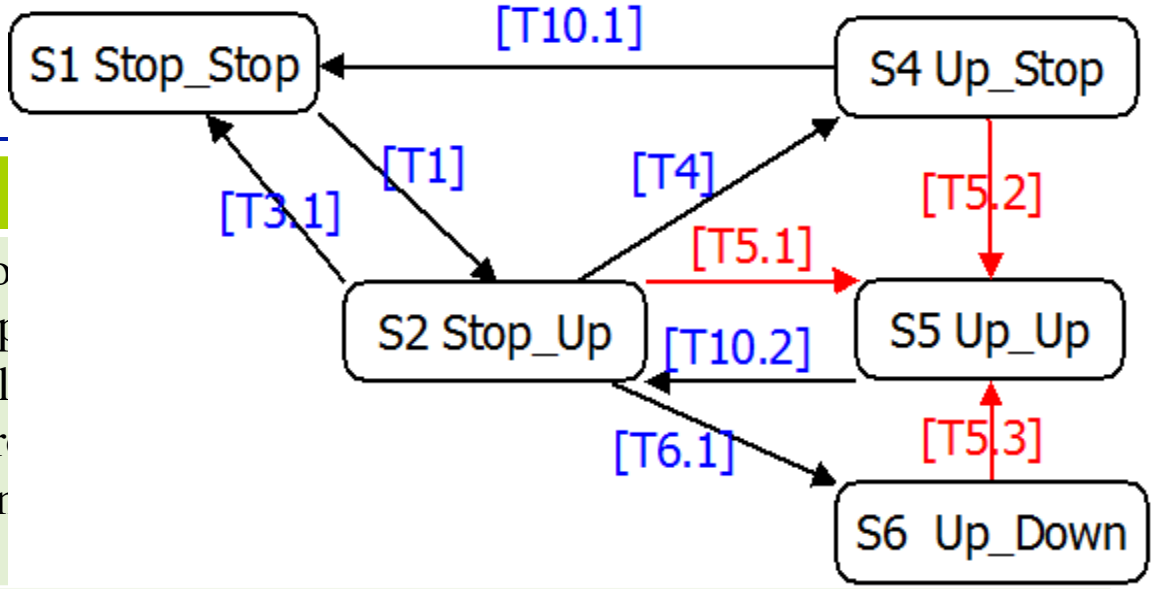
Transition	Guard
T4	CurrentDirectio **AND** mDriveSp **AND** mCarCall **AND** f >= Curr **AND** all other r
T5.*	CurrentDirection ← UP **AND** mDriveSpeed.s > Leveling_Speed **AND** ((mCarCall[f1, b1] ← true *AND* mCarCall[f2, b2] ← true AND CurrentFloor <= f1 < f2) *OR* (mCarCall[f1, b1] ← true *AND* mHallCall[f2, b2, *] ← true AND CurrentFloor <= f1 < f2) *OR* (mHallCall[f1, b1, UP] ← true *AND* CurrentFloor <= f1))

Transitions



Transition	Guard
T4	CurrentDirectio **AND** mDriveSp **AND** mCarCall **AND** f >= Curr **AND** all other r
T5.*	CurrentDirection ← UP **AND** mDriveSpeed.s > Leveling_Speed **AND** ((mCarCall[f1, b1] ← true *AND* mCarCall[f2, b2] ← true AND CurrentFloor <= f1 < f2) *OR* (mCarCall[f1, b1] ← true *AND* mHallCall[f2, b2, *] ← true AND CurrentFloor <= f1 < f2) *OR* (mHallCall[f1, b1, UP] ← true *AND* CurrentFloor <= f1))

Transitions



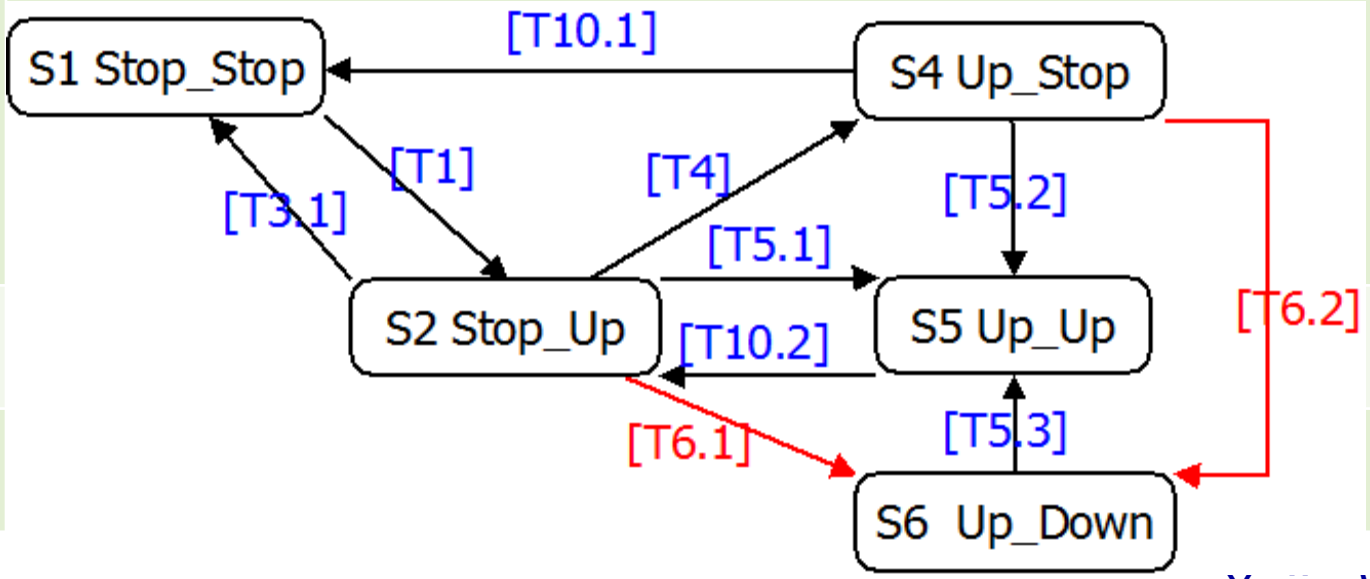
Transition	Guard
T4	CurrentDirectio **AND** mDriveSp **AND** mCarCall **AND** f >= Curr **AND** all other r
T5.*	CurrentDirection ← UP **AND** mDriveSpeed.s > Leveling_Speed **AND** ((mCarCall[f1, b1] ← true *AND* mCarCall[f2, b2] ← true AND CurrentFloor <= f1 < f2) *OR* (mCarCall[f1, b1] ← true *AND* mHallCall[f2, b2, *] ← true AND CurrentFloor <= f1 < f2) *OR* (mHallCall[f1, b1, UP] ← true *AND* CurrentFloor <= f1))

Transitions – cont. 2

Transition	Guard
T6.*	CurrentDirection ← UP **AND** mDriveSpeed.s > Leveling_Speed **AND** ((mCarCall[f1, b1] ← true *AND* mCarCall[f2, b2] ← true AND f2 < CurrentFloor <= f1 *AND* other mCarCall or mHallCall above CurrentFloor ← false) *OR* (mCarCall[f1, b1] ← true *AND* mHallCall[f2, b2, DOWN] ← true AND f2 < f1 *AND* f1 > CurrentFloor *AND* other mCarCall or mHallCall above CurrentFloor ← false) *OR* (mCarCall[f1, b1] ← true *AND* mHallCall[f2, b2, *] ← true *AND* f1 > CurrentFloor > f2 *AND* other mCarCall or mHallCall above CurrentFloor ← false) *OR* (mHallCall[f1, b1, DOWN] ← true *AND* CurrentFloor <= f1 *AND* other mCarCall or mHallCall above CurrentFloor ← false))
T10.*	mDriveSpeed.s <= Leveling_Speed **AND** at least mAtFloor[f, b] ← true
T11.*	any mDoorClosed[b, r] ← false **AND** all mAtFloor[f, b] ← false

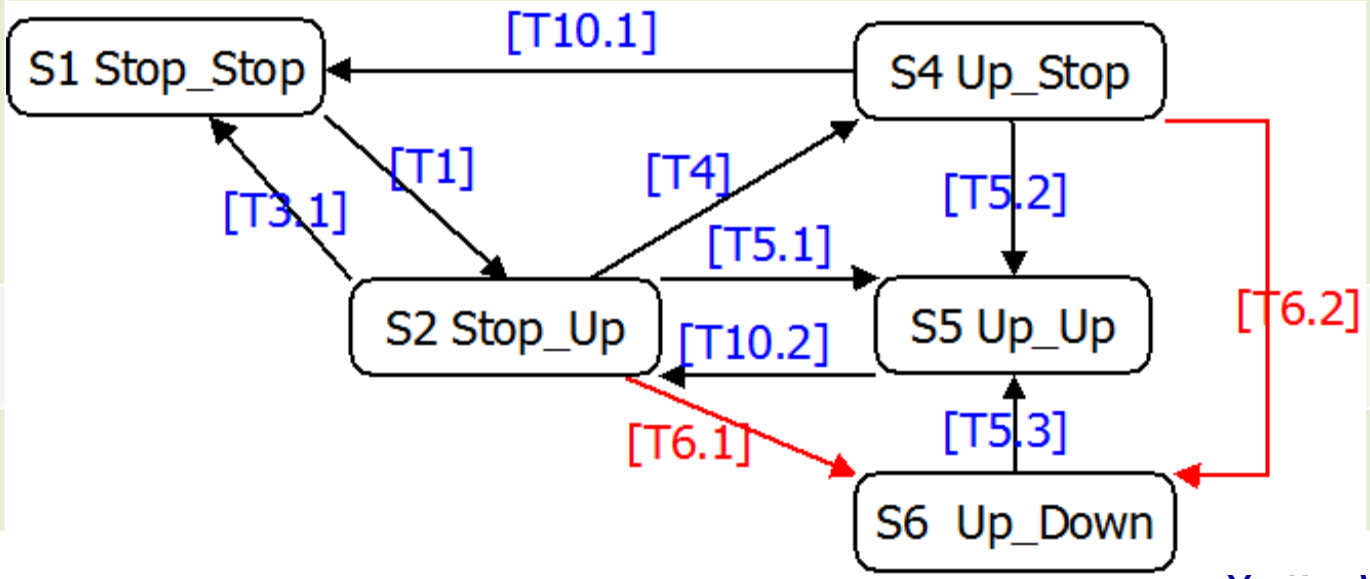
Transitions – cont. 2

Transition	Guard
T6.*	CurrentDirection ← UP **AND** mDriveSpeed.s > Leveling_Speed **AND** ((mCarCall[f1, b1] ← true *AND* mCarCall[f2, b2] ← true AND f2 < CurrentFloor <= f1 *AND* other mCarCall or mHallCall above CurrentFloor ← false) *OR* (mCarCall[f1, b1] ← true *AND* mHallCall[f2, b2, DOWN] ← true AND f2 < f1 *AND* f1 > CurrentFloor *AND* other mCarCall or mHallCall above CurrentFloor ← false)
T10.*	
T11.*	

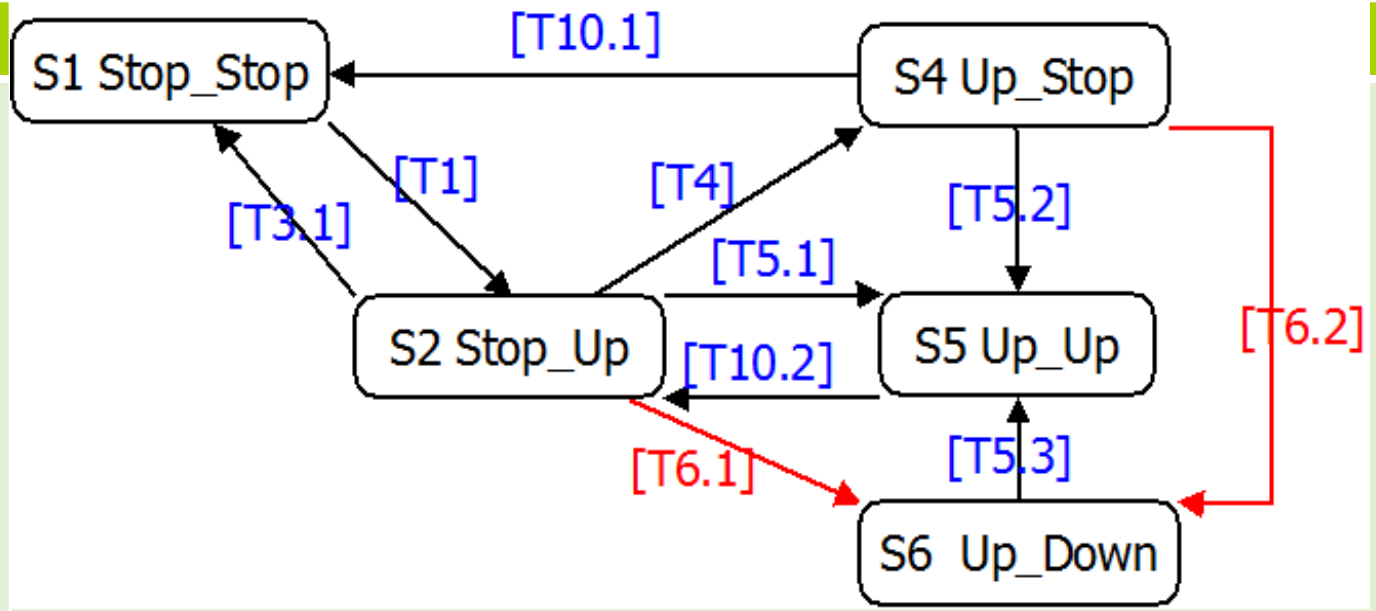


Transitions – cont. 2

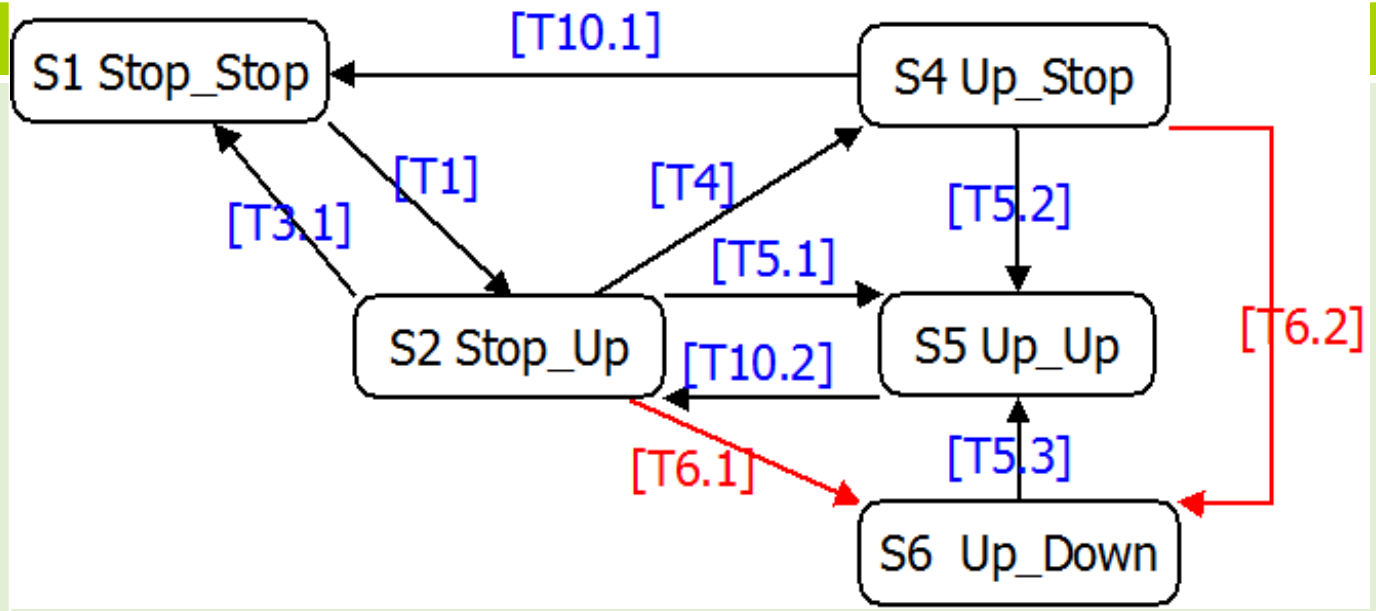
Transition	Guard
T6.*	CurrentDirection ← UP **AND** mDriveSpeed.s > Leveling_Speed **AND** ((mCarCall[f1, b1] ← true *AND* mCarCall[f2, b2] ← true AND f2 < CurrentFloor <= f1 *AND* other mCarCall or mHallCall above CurrentFloor ← false) *OR* (mCarCall[f1, b1] ← true *AND* mHallCall[f2, b2, DOWN] ← true AND f2 < f1 *AND* f1 > CurrentFloor *AND* other mCarCall or mHallCall above CurrentFloor ← false)
T10.*	
T11.*	



Transitions – cont. 2

Transition	
T6.*	 *OR* (mCarCall[f1, b1] ← true *AND* mHallCall[f2, b2, *] ← true *AND* f1 > CurrentFloor > f2 *AND* other mCarCall or mHallCall above CurrentFloor ← false) *OR* (mHallCall[f1, b1, DOWN] ← true *AND* CurrentFloor <= f1 *AND* other mCarCall or mHallCall above CurrentFloor ← false))
T10.*	mDriveSpeed.s <= Leveling_Speed **AND** at least mAtFloor[f, b] ← true
T11.*	any mDoorClosed[b, r] ← false **AND** all mAtFloor[f, b] ← false

Transitions – cont. 2

Transition	
T6.*	 *OR* (mCarCall[f1, b1] ← true *AND* mHallCall[f2, b2, *] ← true *AND* f1 > CurrentFloor > f2 *AND* other mCarCall or mHallCall above CurrentFloor ← false) *OR* (mHallCall[f1, b1, DOWN] ← true *AND* CurrentFloor <= f1 *AND* other mCarCall or mHallCall above CurrentFloor ← false))
T10.*	mDriveSpeed.s <= Leveling_Speed **AND** at least mAtFloor[f, b] ← true
T11.*	any mDoorClosed[b, r] ← false **AND** all mAtFloor[f, b] ← false

Code – helper function

```
public int getNearestPressedFloor(int floor, Direction d, int count, boolean
     ignoreFlag){
…
if(d == Direction.UP){
    if(ignoreFlag == true) floor ++;
    for (int f = floor; f <= Elevator.numFloors && i<count; f++) {
          for (Hallway h : Hallway.replicationValues) {
                if(isPressed(f, h) && nearestFloor != f){
                nearestFloor = f;
                i++;
                }
          }
     }
}
…
if(i == count)  return nearestFloor;
else return MessageDictionary.NONE;
}
```

Calculate ***n***th nearest pressed car call floor, starting from floor **floor**, counting in Direction **d**, ***n*** = **count**, **ignoreFlag** is a flag whether to ignore current floor's hall call.

Code – helper function

```
private Commit commitPoint(int floor, int CarLevelPosition, double speed,
     Direction d) {
double floorPosition = (floor - 1) * 5 * ONETOMILLI;
double brakeDistance = speed*speed / (2* DECELERATION) * ONETOMILLI;
double allowance = 600;
switch (d) {
case STOP:
    return Commit.NOTREACHED;
case UP: {
    int estimatedPosition = (int) (floorPosition – brakeDistance allowance);
    if (estimatedPosition > CarLevelPosition)
          return Commit.NOTREACHED;
    else
          return Commit.REACHED;
}
case DOWN:…
default:…
}
```

Check if car has reached CommitPoint for floor **floor**, at speed **speed**, in Direction **d**.

Code – helper function

```
private int getNearestCommitableFloor(int currentFloor, int
    CarLevelPosition, int s, Direction d){
double speed = s/100.0;
if (speed <= SLOW_SPEED && mAtFloor.getCurrentFLoor() != -1) return
    (int)Math.round(mCarLevelPosition.getPosition()/1000.0/5.0) + 1;
switch(d){
case UP:
for(floor = currentFloor; floor <= Elevator.numFloors; floor ++){
    if (commitPoint(floor, CarLevelPosition, speed, d) ==
     Commit.NOTREACHED)
            return floor;
}break;
case DOWN: ..
case STOP: return CurrentFLoor;
default: return MessageDictionary.NONE;
}
```

Calculate nearest commitable floor starting from floor **currentFloor**, at speed **s**, moving in Direction **d**.

Code – state Stop_Up

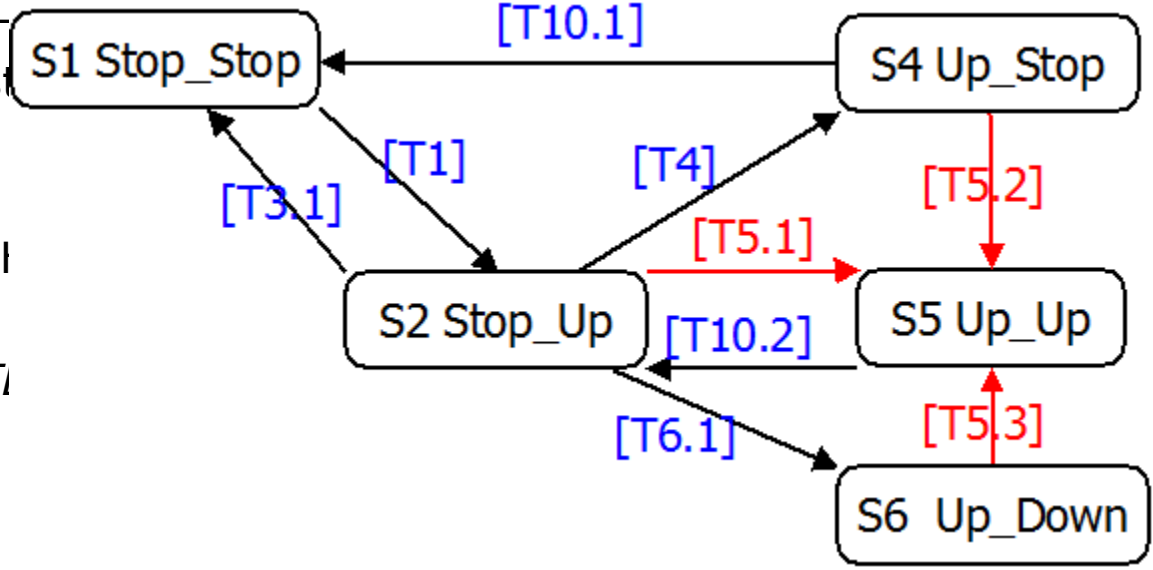
```
//S2  'Stop_Up'
if(mDoorClosedArrayFront.getBothClosed() == true &&
    mDoorClosedArrayFront.getBothClosed() == true){
    countDown = SimTime.subtract(countDown, period);
}else
    countDown = waitingTime;
    //ignore current floor hall call after waitingTime
    mDesiredFloor.setFloor(target);
    currentHallway = mAtFloor.getCurrentHallway();
    mDesiredFloor.setHallway(currentHallway);
    mDesiredFloor.setDirection(Direction.UP);

if (mAtFloor.getCurrentFloor() != MessageDictionary.NONE)
    currentFloor = mAtFloor.getCurrentFloor();

commitableFloor = getNearestCommitableFloor(currentFloor,
     mCarLevelPosition.getValue(), mDriveSpeed.getSpeed(), Direction.UP);
```

Code – state Stop_Up

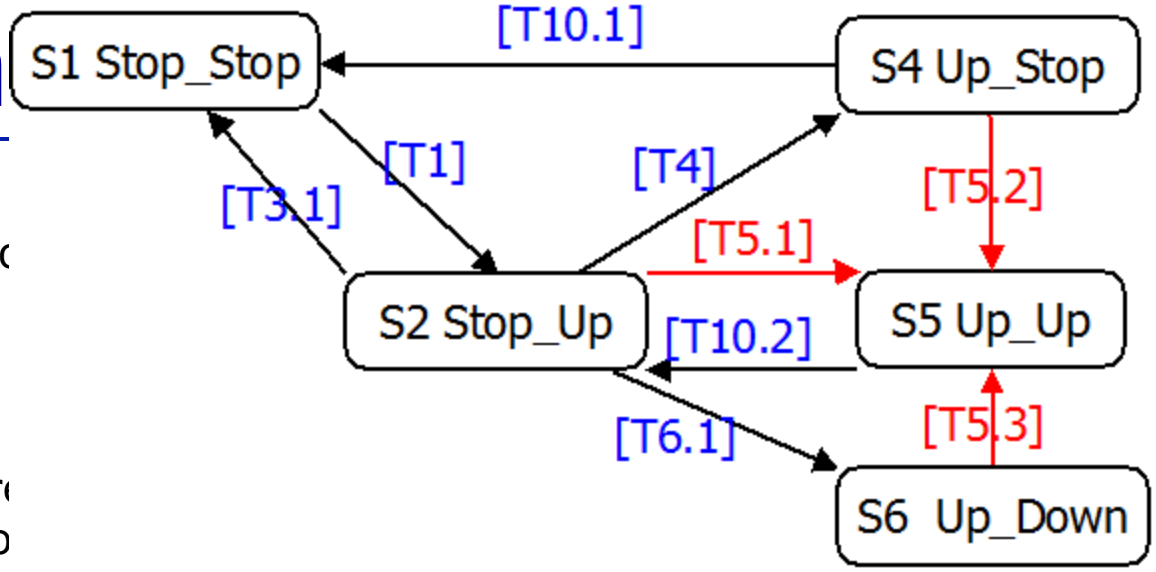
```
int nearestCarCallFloor = mCarCall.getNearestPressedFloor(commitableFloor,
    Direction.UP, 1, isIgnoringCurrentFloorCall());
int nearestHallCallUpFloor =
    mHallCall.getNearestPressedFloor(commitableFloor, Direction.UP, 1,
    Direction.UP, isIgnoringCurrentFloorCall());
int nearestHallCallDownFloor =
    mHallCall.getNearestPressedFloor(commitableFloor, Direction.UP, 1,
    Direction.DOWN, isIgnoringCurrentFloorCall());
//set target to nearest car call or hall call floor
target = computeNearestFloor(nearestCarCallFloor, nearestHallCallUpFloor,
    Direction.UP);
if(target == MessageDictionary.NONE){
    target = computeNearestFloor(nearestCarCallFloor, nearestHallCallFloor,
    Direction.UP);
}
```

Code – transition T5.*

```
//#transition 'T5.1'
if(currentDirection==Direction.UP && mDriveSpeed.getSpeed() >
     MLEVEL_SPEED &&

    ((nearestCarCallFloor != -1&& secondNearestCarCallFloor != -1
    && currentFloor <= nearestCarCallFloor && nearestCarCallFloor <
     secondNearestCarCallFloor)

    || (nearestCarCallFloor != -
    && currentFloor <= nearest
     nearestHallCallFloor)

    ||(currentFloor <= nearestH

    CurrentState = State.STATI
```



Code – tran



//#transition 'T5.1'

```
if(currentDirection==Directic
      MLEVEL_SPEED &&

      ((nearestCarCallFloor !=
      && currentFloor <= near
       secondNearestCarCallFlo

      || (nearestCarCallFloor != -1&& nearestHallCallFloor != -1
      && currentFloor <= nearestCarCallFloor && nearestCarCallFloor <
       nearestHallCallFloor)

      ||(currentFloor <= nearestHallCallUpFloor))

      CurrentState = State.STATE_UP_UP;
```

Testing for Dispatcher

- Unit Test
  - 10 states and 31 transitions tested
  - 120 assertion passed, 0 failed
  - Defects found in unit test
- Integration Test
  - Dispatcher tested in 7 integration tests
  - All these 7 integration tests passed
  - Defects found in integration test
- Acceptance Test
  - 6 fatal defects found in acceptance test
  - Defects usually found with a certain seed

Lessons Learned

- Lessons Learned
  - Leave enough time to submit project
    - Unexpected network issue
  - Scripts are very helpful
    - Very efficient when doing tests
  - Fix peer review feedback early
    - Earlier fix your bugs, less cost you will pay
  - In person meetings are more productive

Suggestion

- Suggestions for next year students
  - Meet and split the work right after Friday's class
  - Submit your project early
  - Use grading rubric to check your work
  - Project shall be checked by at least two members
    - Really need "double" members to do "double" check
  - Use a version control tool
    - Github or SVN

Questions?