Appendix 6

Door controller testing

This appendix contains a description of the definitive program developed in chapter 5 for controlling the door of an elevator. Parameters were used in transforming the LSD specification for the door controller into an adm program. We give a listing of one definitive program, and a sample of its execution. To change the inappropriate behaviour exhibited by the program we change one of the parameters of the transformation to produce a second program. A sample of the execution of this program shows behaviour which is consistent with the system being modelled.

The appendix is organised into four parts. Appendix 6.1 contains the program of Figure 5.3 which was the result of transforming the LSD specification. Appendix 6.2 is a sample of the output generated by execution of the program of Appendix 6.1. Appendix 6.3 analyses the results of Appendix 6.2, and demonstrates the reasoning which is carried out in deciding how to correct the erroneous behaviour. Appendix 6.4 gives a sample of the output of a second program, which now exhibits the behaviour intended by the modeller of the system.

The purpose of the appendix is not to produce a working simulation, but instead to demonstrate how the simulation can be used to provide useful information. For this reason we do not develop a completely working simulation, but rather indicate how this would be done.

Appendix 6.1 The door control program

This is the door control entity of Figure 5.3, which was developed in §5.4.1:

```
entity door_control() {
definition
   open_light = false, can_open = !moving[1] && !open[1],
   moving[1] = |moving|, waited[1] = |waited|,
   open[1] = |open|, hold[1] = |hold|,
   init_flag[1] = true, only_one[1] = |rand(3)|,
   prob[1] = 4, level[1] = 0,
   average[1] = 3, even = true,
   selected[1] = 0
action
   init_flag[1] -> hold = false ; init_flag[1] = false,
   even && !waited[1] && can_open && (only_one[1] == 1)
   && (level[1] == 0) && (selected[1] == 0)
        -> level[1] = 1 ; selected[1] = 1,
   even && (level[1] == 1) && (rand(4) == 1) && (selected[1] == 1)
print ("Opening the door because lift has stopped...")
        -> open = true ; level[1] = 2,
   even && (level[1] == 2) && (rand(4) == 1) && (selected[1] == 1)
print ("...and illuminating the open light")
        -> open_light = true ; level[1] = 0 ; selected[1] = 0,

   even && hold[1] && can_open && (only_one[1] == 2)
   && (level[1] == 0) && (selected[1] == 0)
        -> level[1] = 1 ; selected[1] = 2,
   even && (level[1] == 1) && (rand(4) == 1) && (selected[1] == 2)
print ("The hold button has been pressed: opening door...")
        -> open = true ; level[1] = 2,
   even && (level[1] == 2) && (rand(4) == 1) && (selected[1] == 2)
print ("...and illuminating the open light")
        -> open_light = true ; level[1] = 0 ; selected[1] = 0,

   even && !hold[1] && open[1] && waited[1] && (only_one[1] == 3)
   && (level[1] == 0) && (selected[1] == 0)
        -> level[1] = 1 ; selected[1] = 3,
   even && (level[1] == 1) && (rand(4) == 1) && (selected[1] == 3)
print ("Closing door...")
        -> open = false ; level[1] = 2,
   even && (level[1] == 2) && (rand(4) == 1) && (selected[1] == 3)
print ("...and turning off the open light")
        -> open_light = false ; level[1] = 0 ; selected[1] = 0,

   !even && (rand(3) == 1) -> moving[1] = |moving|,
   !even && (rand(3) == 1) -> waited[1] = |waited|,
   !even && (rand(3) == 1) -> open[1] = |open|,
   !even && (rand(3) == 1) -> hold[1] = |hold|,
   !even && (selected[1] == 0) -> only_one[1] = |rand(3)|,
   true -> even = !|even|
}


entity environment () {
definition
   moving = true, hold = false,
   open = false, waited = false
}
```

Appendix 6.2 Execution of the first door control program

The results of executing the program of Appendix 6.1 are now given:

```
Script started on Tue Aug 29 07:00:34 1989
emerald!mike cat -u lift - | am -i40
```

execute the commands in the file "`lift`*" and then await input. Set* `iterations` *to 40*

```
am> compiling door_control()
am> compiling environment()
am> instantiating door_control
am> instantiating environment
am> l en
```

list the entities in the program store P

```
                ENTITY LIST
                ***********
entity door_control() {        (0 parameters)
DEFINITION open_light = FALSE, can_open = (!moving[1]&&!open[1]), moving[1] =
|moving|,  waited[1]  =  |waited|,  open[1]  =  |open|,  hold[1]  =  |hold|,
init_flag[1] = TRUE, only_one[1] = |rand(3)|, prob[1] = 4, level[1] = 0,
average[1] = 3, even = TRUE, selected[1] = 0
ACTION
init_flag[1] ->
                               hold = FALSE ;
                               init_flag[1] = FALSE,
(((((even&&!waited[1])&&can_open)&&(only_one[1]==1))&&(level[1]==0))&&(selecte
d[1]==0)) ->
                               level[1] = 1 ;
                               selected[1] = 1,
(((even&&(level[1]==1))&&(rand(4)==1))&&(selected[1]==1))
print ("Opening the door because lift has stopped...") ->
                               open = TRUE ;
                               level[1] = 2,
(((even&&(level[1]==2))&&(rand(4)==1))&&(selected[1]==1))        print("...and
illuminating the open light") ->
                               open_light = TRUE ;
                               level[1] = 0 ;
                               selected[1] = 0,
(((((even&&hold[1])&&can_open)&&(only_one[1]==2))&&(level[1]==0))&&(selected[
1]==0)) ->
                               level[1] = 1 ;
                               selected[1] = 2,
(((even&&(level[1]==1))&&(rand(4)==1))&&(selected[1]==2))
print("The hold button has been pressed: opening door...") ->
                               open = TRUE ;
                               level[1] = 2,
(((even&&(level[1]==2))&&(rand(4)==1))&&(selected[1]==2))        print("...and
illuminating the open light") ->
                               open_light = TRUE ;
                               level[1] = 0 ;
                               selected[1] = 0,
((((((even&&!hold[1])&&open[1])&&waited[1])&&(only_one[1]==3))&&(level[1]==0))
&&(selected[1]==0)) ->
                               level[1] = 1 ;
                               selected[1] = 3,
(((even&&(level[1]==1))&&(rand(4)==1))&&(selected[1]==3))       print("Closing
door...") ->
                               open = FALSE ;
                               level[1] = 2,
(((even&&(level[1]==2))&&(rand(4)==1))&&(selected[1]==3))
print("...and turning off the open light") ->
                               open_light = FALSE ;
                               level[1] = 0 ;
```

```
                                selected[1] = 0,
(!even&&(rand(3)==1)) ->
                                moving[1] = |moving|,
(!even&&(rand(3)==1)) ->
                                waited[1] = |waited|,
(!even&&(rand(3)==1)) ->
                                open[1] = |open|,
(!even&&(rand(3)==1)) ->
                                hold[1] = |hold|,
(!even&&(selected[1]==0)) ->
                                only_one[1] = |rand(3)|,
TRUE ->                         even = !|even|
} 1 instances
entity environment() {           (0 parameters)
DEFINITION
moving = TRUE, hold = FALSE, open = FALSE, waited = FALSE
} 1 instances
                 END OF ENTITY LIST
                 ******************
am> l ds
```

list the variables in the definition store D

```
                   DEFINITION STORE
                 ****************
Variable# 1: open_light = FALSE
Variable# 2: can_open = (!moving[1]&&!open[1])
Variable# 3: moving[1] = |moving|
Variable# 4: waited[1] = |waited|
Variable# 5: open[1] = |open|
Variable# 6: hold[1] = |hold|
Variable# 7: init_flag[1] = TRUE
Variable# 8: only_one[1] = |rand(3)|
Variable# 9: prob[1] = 4
Variable# 10: level[1] = 0
Variable# 11: average[1] = 3
Variable# 12: even = TRUE
Variable# 13: selected[1] = 0
Variable# 14: moving = TRUE
Variable# 15: hold = FALSE
Variable# 16: open = FALSE
Variable# 17: waited = FALSE
                 END OF DEFINITION STORE
                 ***********************
am> l as
```

list the actions in the action store A

```
                ACTION STORE
                ************
Action# 1:init_flag[1] ->
                                hold = FALSE ;
                                init_flag[1] = FALSE
Action#
2:(((((even&&!waited[1])&&can_open)&&(only_one[1]==1))&&(level[1]==0))&&(selec
ted[1]==0)) ->
                                level[1] = 1 ;
                                selected[1] = 1

Action#             3:(((even&&(level[1]==1))&&(rand(4)==1))&&(selected[1]==1))
print("Opening the door because lift has stopped...") ->
                                open = TRUE ;
                                level[1] = 2
Action#             4:(((even&&(level[1]==2))&&(rand(4)==1))&&(selected[1]==1))
print("...and illuminating the open light") ->
                                open_light = TRUE ;
```

```
                              level[1] = 0 ;
                              selected[1] = 0
Action#
5:(((((even&&hold[1])&&can_open)&&(only_one[1]==2))&&(level[1]==0))&&(selected
[1]==0)) ->
                              level[1] = 1 ;
                              selected[1] = 2
Action# 6:(((even&&(level[1]==1))&&(rand(4)==1))&&(selected[1]==2)) print("The
hold button has been pressed: opening door...") ->
                              open = TRUE ;
                              level[1] = 2
Action#           7:(((even&&(level[1]==2))&&(rand(4)==1))&&(selected[1]==2))
print("...and illuminating the open light") ->
                              open_light = TRUE ;
                              level[1] = 0 ;
                              selected[1] = 0
Action#
8:((((((even&&!hold[1])&&open[1])&&waited[1])&&(only_one[1]==3))&&(level[1]==0
))&&(selected[1]==0)) ->
                              level[1] = 1 ;
                              selected[1] = 3
Action#           9:(((even&&(level[1]==1))&&(rand(4)==1))&&(selected[1]==3))
print("Closing door...") ->
                              open = FALSE ;
                              level[1] = 2
Action#
10:(((even&&(level[1]==2))&&(rand(4)==1))&&(selected[1]==3))      print("...and
turning off the open light") ->
                              open_light = FALSE ;
                              level[1] = 0 ;
                              selected[1] = 0
Action# 11:(!even&&(rand(3)==1)) ->
                              moving[1] = |moving|
Action# 12:(!even&&(rand(3)==1)) ->
                              waited[1] = |waited|
Action# 13:(!even&&(rand(3)==1)) ->
                              open[1] = |open|
Action# 14:(!even&&(rand(3)==1)) ->
                              hold[1] = |hold|
Action# 15:(!even&&(selected[1]==0)) ->
                              only_one[1] = |rand(3)|
Action# 16:TRUE ->
                              even = !|even|
                 END OF ACTION STORE
                 *******************
am> l in                      list the current instantiations
                 INSTANCES
                 *********
door_control ()
environment ()
                 END OF INSTANCES

am> status             print the status of the control variables for execution
nflag = TRUE                                         print notifiable errors
aflag = TRUE                                         print avoidance errors
silent = FALSE             print prompt and execution cycle indicator ("#")
iterations = 40                               stop after 40 execution cycles
am> load runset                                            load the run set
loading run set
#
* loaded run set
am> l runset                                               list the run set
```

```
                RUN SET
                *******
```
the next commands to be executed

```
hold = FALSE ;
init_flag[1] = FALSE ;
even = !TRUE
                 END OF RUN SET
                 **************
am> define moving = false ;
defining moving
```
supply a redefinition indicating that the elevator is not moving

```
am> define only_one[1] = 1 ;
defining only_one[1]
```
make sure that guard 1 in door_control() is chosen next

```
am> load runset
```
load the run set

```
loading run set
#
* loaded run set
am> l runset
```
list the run set

```
                RUN SET
                *******
hold = FALSE ;
init_flag[1] = FALSE ;
level[1] = 1 ;
selected[1] = 1 ;
even = !TRUE
```
guard 1 has been evaluated as true, and now the first command list will be executed

```
                 END OF RUN SET
                 **************
am> cont
```
continue with the simulation

```
continuing simulation
#
#
#
#
#
#
#
 Opening the door because lift has stopped...
```
procedural action

```
#
#
#
#
#
#
#
#
#
#
#
 ...and illuminating the open light
```
procedural action

```
#
#
#
#
#
#
#
#
#
#
#
#
#
```

```
#
#
#
#
#
#
#
#
#
#
* 40 iterations successfully completed
```
stops after 40 execution cycles

```
am> define waited = true ;
```
the door has been open long enough

```
defining waited
am> cont
continuing simulation
```

Having shown the use of the # symbol to indicate the end of each execution cycle, we now introduce a more concise (unimplemented) notation for the remainder of this and future appendices. We preface each procedural action output with the number of the execution cycle in which it occurred, and omit the # symbol.

```
(51) Closing door...
(53) ...and turning off the open light

 * 40 iterations successfully completed

am> define hold = true ;
```
the hold button has been pressed

```
defining hold
am> cont
continuing simulation

(99) The hold button has been pressed: opening door...
(105) ...and illuminating the open light
(109) The hold button has been pressed: opening door...

* 40 iterations successfully completed

am>

script done on Tue Aug 29 07:02:19 1989
```

Appendix 6.3 Analysis of results

It was seen in Appendix 6.2 that when the hold button was pressed the command to open the door was executed twice. This is not the intended behaviour of the system described by the specification, and can be for one of two reasons: the specification is incorrect or inappropriate parameters have been used in the transformation process (§8.7.2). By the application of simple cognitive reasoning, as opposed to consideration of the adm program, it is seen that according to the LSD specification the elevator door should not be opened a second time, because the variable `can_open` appears in the guard, and `can_open` is false when `open` is false. It is therefore the parameters of the transformation that need to be changed in order to permit the appropriate synchronisation by perception..

In considering what would be appropriate parameters it is again cognitive reasoning which is used: if the door controller had perceived that the door was open then it would not attempt to re-open it, so it cannot have known. In other words, the value of `open` is not being propagated fast enough from the owning `environment()` entity to the `door_control()` entity. Looking at the program, the guarded command which updates the value of open is

```
(!even&&(rand(3)==1)) -> open[1] = |open|
```

The door controller immediately perceive changes in the value of `open`, with the guard:

```
!even -> open[1] = |open|
```

which updates `open[1]`, the perceived value of `open`, in every execution cycle in which `!even` is true (i.e. `even` is false). This means that in every execution cycle in which `!even` is false (i.e. `even` is true) `open[1]` will have the authentic value of `open`, which is a sufficient condition to guarantee that the value of `open[1]` is always authentic when it is used by guarded commands which have come from the LSD specification.

Appendix 6.4 Execution of the second door control program

In this appendix we present the results of executing the definitive program for the door controller which updates the value of `open[1]` in every execution cycle when `even` is false.

```
Script started on Tue Aug 29 07:07:30 1989
emerald!mike cat -u lift - | am -i40
am> compiling door_control()
am> compiling environment()
am> instantiating door_control
am> instantiating environment
am> l as                               list the actions in the action store A
                 ACTION STORE
                 ************
Action# 1:init_flag[1] ->
                               hold = FALSE ;
                               init_flag[1] = FALSE
Action#
2:(((((even&&!waited[1])&&can_open)&&(only_one[1]==1))&&(level[1]==0))&&(selec
ted[1]==0)) ->
                               level[1] = 1 ;
                               selected[1] = 1
Action# 3:(((even&&(level[1]==1))&&(rand(4)==1))&&(selected[1]==1))
print("Opening the door because lift has stopped...") ->
                               open = TRUE ;
                               level[1] = 2
Action# 4:(((even&&(level[1]==2))&&(rand(4)==1))&&(selected[1]==1))
print("...and illuminating the open light") ->
                               open_light = TRUE ;
                               level[1] = 0 ;
                               selected[1] = 0
Action#
5:(((((even&&hold[1])&&can_open)&&(only_one[1]==2))&&(level[1]==0))&&(selected
[1]==0)) ->
                               level[1] = 1 ;
                               selected[1] = 2
Action# 6:(((even&&(level[1]==1))&&(rand(4)==1))&&(selected[1]==2))
print("The hold button has been pressed: opening door...") ->
                               open = TRUE ;
                               level[1] = 2
Action# 7:(((even&&(level[1]==2))&&(rand(4)==1))&&(selected[1]==2))
print("...and illuminating the open light") ->
                               open_light = TRUE ;
                               level[1] = 0 ;
                               selected[1] = 0
Action#
8:((((((even&&!hold[1])&&open[1])&&waited[1])&&(only_one[1]==3))&&(level[1]==0
))&&(selected[1]==0)) ->
                               level[1] = 1 ;
                               selected[1] = 3
Action# 9:(((even&&(level[1]==1))&&(rand(4)==1))&&(selected[1]==3))
print("Closing door...") ->
                               open = FALSE ;
                               level[1] = 2




Action#
10:(((even&&(level[1]==2))&&(rand(4)==1))&&(selected[1]==3))      print("...and
turning off the open light") ->
```

```
                              open_light = FALSE ;
                              level[1] = 0 ;
                              selected[1] = 0
Action# 11:(!even&&(rand(3)==1)) ->
                              moving[1] = |moving|
Action# 12:(!even&&(rand(3)==1)) ->
                              waited[1] = |waited|
Action# 13:!even ->
                              open[1] = |open|
```
this command has a new guard

```
Action# 14:(!even&&(rand(3)==1)) ->
                              hold[1] = |hold|
Action# 15:(!even&&(selected[1]==0)) ->
                              only_one[1] = |rand(3)|
Action# 16:TRUE ->
                              even = !|even|
                 END OF ACTION STORE
                 *******************
am> load runset
loading run set
#
* loaded run set
am> l runset

                RUN SET
                *******
hold = FALSE ;
init_flag[1] = FALSE ;
even = !TRUE
                 END OF RUN SET
                 **************
am> define moving = false ;
```
the elevator has stopped

```
defining moving
am> define only_one[1] = 1 ;
defining only_one[1]
am> load runset
loading run set
#
* loaded run set
am> l runset

                RUN SET
                *******
hold = FALSE ;
init_flag[1] = FALSE ;
level[1] = 1 ;
selected[1] = 1 ;
even = !TRUE
                 END OF RUN SET
                 **************
am> start
starting simulation

(15) Opening the door because lift has stopped...
(19) ...and illuminating the open light

* 40 iterations successfully completed
am> define waited = true ;
```
the door has been open for long enough

```
defining waited
am> cont
continuing simulation

(67) Closing door...
```

```
* 40 iterations successfully completed
am> cont
continuing simulation

(89) ...and turning off the open light

* 40 iterations successfully completed
am> define hold = true ;                the hold button has been pressed
defining hold
am> cont
continuing simulation

(141) The hold button has been pressed: opening door...
(145) ...and illuminating the open light

* 40 iterations successfully completed
am> define hold = false ;               the hold button has been released
defining hold
am> cont
continuing simulation

* 40 iterations successfully completed
am> cont
continuing simulation

(207) Closing door...
(213) ...and turning off the open light

* 40 iterations successfully completed
am> define moving = true ;              simultaneously the elevator starts moving...
defining moving
am> define hold = true ;                ...and the hold button is pressed
defining hold
am> cont
continuing simulation

(249) The hold button has been pressed: opening door...
(251) ...and illuminating the open light

* 40 iterations successfully completed
am>

script done on Tue Aug 29 07:10:29 1989
```

In the final 40 execution cycles the elevator door was opened as a result of pressing the hold button, even though the elevator was moving. The specification shows that a precondition for opening the door is that the elevator not be moving, so the value of `moving` is not being propagated sufficiently quickly. We can remedy this behaviour using the technique employed in Appendix 6.3 to produce a new definitive program, whose execution we can again study. This process can continue until the simulation appears to result in appropriate behaviour. At this point further verification of the LSD specification would require other techniques to be developed.