Appendix 8

Jugs testing

In this appendix we give the results of execution of various abstract definitive machine versions of the jugs program.

Appendix 8.1 Output of messages

In this appendix we list an adm jugs program that prints messages to indicate what action it is performing.

```
entity jugs(_capA,_capB)
{
definition
   capacityA = _capA, capacityB = _capB,
   contentA = 0, contentB = 0,
   emptyA = (contentA == 0),
   fullA = (contentA == capacityA),
   emptyB = (contentB == 0),
   fullB = (contentB == capacityB),
   avail[1] = !fullA, avail[2] = !fullB, avail[3] = !emptyA,
   avail[4] = !emptyB, avail[5] = (avail[6] || avail[7]),
   avail[6] = (avail[2] && avail[3]),
   avail[7] = (avail[1] && avail[4]),
   updating = FALSE
}

entity user()
definition
   x = rand(5), y = |x|
action
   !updating && (y == 0)
print ("pressed 1") -> init_pour(1) ; updating = TRUE,
   !updating && (y == 1)
print ("pressed 2") -> init_pour(2) ; updating = TRUE,
   !updating && (y == 2)
print ("pressed 3") -> init_pour(3) ; updating = TRUE,
   !updating && (y == 3)
print ("pressed 4") -> init_pour(4) ; updating = TRUE,
   !updating && (y == 4)
print ("pressed 5") -> init_pour(5) ; updating = TRUE,
   TRUE
print ("A: ",contentA,", B: ",contentB) -> y = |x|
}

entity init_pour (_input)
{
definition
   option = _input
action
   option == 1
print ("initialise fill A") -> pour(1) ; delete init_pour(_input),
   option == 2
print ("initialise fill B") -> pour(2) ; delete init_pour(_input),
   option == 3
print ("initialise empty A") -> pour(3) ; delete init_pour(_input),
   option == 4
print ("initialise empty B") -> pour(4) ; delete init_pour(_input),
   option == 6
print ("initialise pour A to B")
                                -> pour(6) ; delete init_pour(_input),

   option == 7
print ("initialise pour B to A")
                                -> pour(7) ; delete init_pour(_input),
   (option == 5) && !avail[5]
print ("can't perform pour")    -> updating = FALSE ;
```

```
                                        delete init_pour(_input),
   (option == 5) && avail[6]
print ("configure pour A to B")
                      -> contentB = |contentA + contentB| - contentA ;
                         option = 6,
   (option == 5) && avail[7]
print ("configure pour B to A")
                     -> contentB = |contentA + contentB| - contentA ;
                        option = 7
}

entity pour(_option)
{
action
   (_option == 1) && avail[1]
                         -> contentA = |contentA| + 1,
   (_option == 1) && !avail[1]
                         -> updating = FALSE ; delete pour(_option),
   (_option == 2) && avail[2]
                         -> contentB = |contentB| + 1,
   (_option == 2) && !avail[2]
                         -> updating = FALSE ; delete pour(_option),
   (_option == 3) && avail[3]
                         -> contentA = |contentA| - 1,
   (_option == 3) && !avail[3]
                         -> updating = FALSE ; delete pour(_option),
   (_option == 4) && avail[4]
                         -> contentB = |contentB| - 1,
   (_option == 4) && !avail[4]
                         -> updating = FALSE ; delete pour(_option),
   (_option == 6) && avail[6]
                         -> contentA = |contentA| - 1,
   (_option == 6) && !avail[6]
                         ->   updating   =   FALSE   ;   contentB=|contentB|   ;
                            delete pour (_option),
   (_option == 7) && avail[7]
                         -> contentA = |contentA| + 1,
   (_option == 7) && !avail[7]
                         -> updating = FALSE ; contentB=|contentB| ;
                            delete pour(_option)
}

jugs(5,7)
user()
start
```

Appendix 8.2 Execution of message program

In this appendix we list a test of the jugs program of Appendix 8.1.

```
Script started on Wed Sep  6 20:38:23 1989
snow!mike am < jugs.print
am> compiling jugs(_capA,_capB)
am> compiling user()
am> compiling init_pour(_input)
am> compiling pour(_option)
am> instantiating jugs(5,7)
am> instantiating user
am> starting simulation
#
 A: 0, B: 0
#
 pressed 2
 A: 0, B: 0
#
 A: 0, B: 0  initialise fill B
#
 A: 0, B: 0
#
 A: 0, B: 1
#
 A: 0, B: 2
#
 A: 0, B: 3
#
 A: 0, B: 4
#
 A: 0, B: 5
#
 A: 0, B: 6
#
 A: 0, B: 7
#
 pressed 1
 A: 0, B: 7
#
 A: 0, B: 7
 initialise fill A
#
 A: 0, B: 7
#
 A: 1, B: 7
#
 A: 2, B: 7
#
 A: 3, B: 7
#
 A: 4, B: 7
#
 A: 5, B: 7
#
 pressed 5
 A: 5, B: 7
#
 A: 5, B: 7
 can't perform pour
    •
```

```
    •       later
    •
#
 A: 5, B: 0
#
 pressed 5
 A: 5, B: 0
#
 A: 5, B: 0
 configure pour A to B
#
 A: 5, B: 0
 initialise pour A to B
#
 A: 5, B: 0
#
 A: 4, B: 1
#
 A: 3, B: 2
#
 A: 2, B: 3
#
 A: 1, B: 4
#
 A: 0, B: 5
#
    •
    •       later
    •
 A: 0, B: 5
#
 pressed 2
 A: 0, B: 5
#
 A: 0, B: 5
 initialise fill B
#
 A: 0, B: 5
#
 A: 0, B: 6
#
 A: 0, B: 7
#

script done on Wed Sep  6 20:40:14 1989
```

Appendix 8.3 Output of pictures

In this appendix we describe an adm jugs program that prints a graphical representation of the state, instead of printing messages about what is being done. It involves an extra entity, which is specific to jug A having a capacity of five units and jug B having a capacity of seven units. This is graphically depicted by the `pic()` entity:

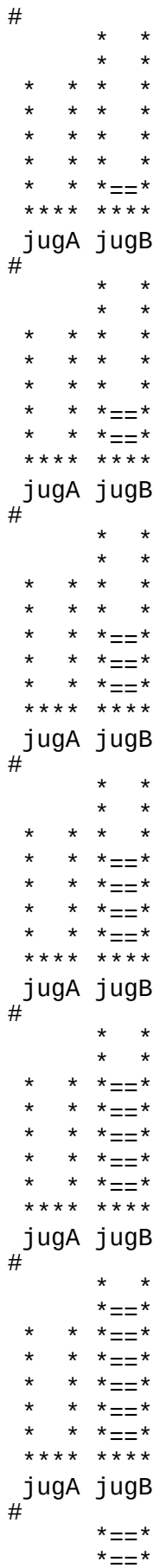
```
entity pic() {
action
   (contentB == 7) print ("     *==*") ->,
   (contentB < 7) print ("     *  *") ->,
   (contentB >= 6) print ("     *==*") ->,
   (contentB < 6)  print ("     *  *") ->,
   (contentA == 5) && (contentB >= 5) print ("*==* *==*") ->,
   (contentA == 5) && (contentB <5)   print ("*==* *  *") ->,
   (contentA < 5) && (contentB >= 5)  print ("*  * *==*") ->,
   (contentA < 5) && (contentB <5)    print ("*  * *  *") ->,
   (contentA >= 4) && (contentB >= 4) print ("*==* *==*") ->,
   (contentA >= 4) && (contentB <4)   print ("*==* *  *") ->,
   (contentA < 4) && (contentB >= 4)  print ("*  * *==*") ->,
   (contentA < 4) && (contentB < 4)   print ("*  * *  *") ->,
   (contentA >= 3) && (contentB >= 3) print ("*==* *==*") ->,
   (contentA >= 3) && (contentB < 3)  print ("*==* *  *") ->,
   (contentA < 3) && (contentB >= 3)  print ("*  * *==*") ->,
   (contentA < 3) && (contentB < 3)   print ("*  * *  *") ->,
   (contentA >= 2) && (contentB >= 2) print ("*==* *==*") ->,
   (contentA >= 2) && (contentB < 2)  print ("*==* *  *") ->,
   (contentA < 2) && (contentB >= 2)  print ("*  * *==*") ->,
   (contentA < 2) && (contentB < 2)   print ("*  * *  *") ->,
   (contentA >= 1) && (contentB >= 1) print ("*==* *==*") ->,
   (contentA >= 1) && (contentB < 1)  print ("*==* *  *") ->,
   (contentA < 1) && (contentB >= 1)  print ("*  * *==*") ->,
   (contentA < 1) && (contentB < 1)   print ("*  * *  *") ->,
   TRUE print ("**** ****") ->,
   TRUE print ("jugA jugB") ->
}
```

The procedural action to print the contents of the jugs in text form is removed from the `user()` entity of Appendix 8.1, and the `pic()` entity instantiated with the other entities.

Appendix 8.4 Execution of pictures program

In this appendix we present a short segment of the results of executing the program described in Appendix 8.3. The actions performed are the same as in Appendix 8.2.

```
Script started on Wed Sep  6 20:35:53 1989
snow!mike am < jugs.picture
am> compiling pic()
am> compiling jugs(_capA,_capB)
am> compiling user()
am> compiling init_pour(_input)
am> compiling pour(_option)
am> instantiating pic
am> instantiating jugs(5,7)
am> instantiating user
am> starting simulation
#
       *  *
       *  *
 *  * *  *
 *  * *  *
 *  * *  *
 *  * *  *
 *  * *  *
 **** ****
 jugA jugB
#
       *  *
       *  *
 *  * *  *
 *  * *  *
 *  * *  *
 *  * *  *
 *  * *  *
 **** ****
 jugA jugB
 pressed 2
#
       *  *
       *  *
 *  * *  *
 *  * *  *
 *  * *  *
 *  * *  *
 *  * *  *
 **** ****
 jugA jugB
 initialise fill B
#
       *  *
       *  *
 *  * *  *
 *  * *  *
 *  * *  *
 *  * *  *
 *  * *  *
 **** ****
 jugA jugB
```


jugA jugB
jugA jugB
jugA jugB
jugA jugB
jugA jugB
jugA jugB
jugA jugB

```
 *   * *==*
 *   * *==*
 *   * *==*
 *   * *==*
 *   * *==*
 **** ****
 jugA jugB
```

```
script done on Wed Sep  6 20:38:03 1989
```

Appendix 8.5 Allowing input

In this appendix we describe an adm jugs program which awaits input from the user during simulation. This allows a student to try to achieve a specified target. The `user()` entity is changed to become:

```
entity user()
definition
   value = 6
action
   !updating && (value == 6) print ("Next input, please") -> stop,
   !updating && (value == 1)
       -> init_pour(1) ; value = 6 ; updating = TRUE,
   !updating && (value == 2)
       -> init_pour(2) ; value = 6 ; updating = TRUE,
   !updating && (value == 3)
       -> init_pour(3) ; value = 6 ; updating = TRUE,
   !updating && (value == 4)
       -> init_pour(4) ; value = 6 ; updating = TRUE,
   !updating && (value == 5)
       -> init_pour(5) ; value = 6 ; updating = TRUE
}
```

This stops the simulation whenever an input can be accepted, to allow the user to select an option by defining `value`. The other entities are unchanged.

Appendix 8.6 Execution of input program

In this appendix we show how the program of Appendix 8.5 can be used to allow a specified target of three units to be achieved

```
Script started on Wed Sep  6 20:44:23 1989
snow!mike cat -u jugs.input - | am
am> compiling pic()
am> compiling jugs(_capA,_capB)
am> compiling user()
am> compiling init_pour(_input)
am> compiling pour(_option)
am> instantiating pic
am> instantiating jugs(5,7)
am> instantiating user
am> starting simulation
#
      *  *
      *  *
 *  * *  *
 *  * *  *
 *  * *  *
 *  * *  *
 *  * *  *
 **** ****
 jugA jugB
 Next input, please
* forced termination after iteration number 1
am> define value = 1 ;
defining value
am> cont
continuing simulation
#
      *  *
      *  *
 *  * *  *
 *  * *  *
 *  * *  *
 *  * *  *
 *  * *  *
 **** ****
 jugA jugB
#
      *  *
      *  *
 *  * *  *
 *  * *  *
 *  * *  *
 *  * *  *
 *  * *  *
 **** ****
 jugA jugB
 initialise fill A
#
```

-
- *later*
-

```
#
      *  *
      *  *
 *==* *  *
 *==* *  *
 *==* *  *
 *==* *  *
 *==* *  *
 **** ****
 jugA jugB
 Next input, please
* forced termination after iteration number 9
am> define value = 5 ;
defining value
am> cont
continuing simulation
#
   •
   •     later
   •
      *  *
      *  *
 *  * *==*
 *  * *==*
 *  * *==*
 *  * *==*
 *  * *==*
 **** ****
 jugA jugB
 Next input, please
* forced termination after iteration number 10
am> define value = 1 ;
defining value
am> cont
continuing simulation
#
   •
   •     later
   •
      *  *
      *  *
 *==* *==*
 *==* *==*
 *==* *==*
 *==* *==*
 *==* *==*
 **** ****
 jugA jugB
 Next input, please
* forced termination after iteration number 9
am> define value = 5 ;
defining value
am> cont
continuing simulation
#

      *  *
      *  *
 *==* *==*
 *==* *==*
 *==* *==*
 *==* *==*
```

```
 *==* *==*
 **** ****
 jugA jugB
#
      *  *
      *  *
 *==* *==*
 *==* *==*
 *==* *==*
 *==* *==*
 *==* *==*
 **** ****
 jugA jugB
 configure pour A to B
#
      *  *
      *  *
 *==* *==*
 *==* *==*
 *==* *==*
 *==* *==*
 *==* *==*
 **** ****
 jugA jugB
 initialise pour A to B
#
      *  *
      *  *
 *==* *==*
 *==* *==*
 *==* *==*
 *==* *==*
 *==* *==*
 **** ****
 jugA jugB
#
      *  *
      *==*
 *  * *==*
 *==* *==*
 *==* *==*
 *==* *==*
 *==* *==*
 **** ****
 jugA jugB
#
```

```
      *==*
      *==*
 *  * *==*
 *  * *==*
 *==* *==*
 *==* *==*
 *==* *==*
 **** ****
 jugA jugB
#
```

```
      *==*
      *==*
 *   * *==*
 *   * *==*
 *==* *==*
 *==* *==*
 *==* *==*
 **** ****
 jugA jugB
 Next input, please
* forced termination after iteration number 7 am>
script done on Wed Sep  6 20:46:11 1989
```

Appendix 8.7 Allowing concurrent activity

So far we have used an `updating` variable to flag when an option can be selected. In this appendix we describe a program which does not have this feature, so the user can select options whilst a previously selected option is being implemented.

```
entity screen()
definition
   button[1] = FALSE, button[2] = FALSE, button[3] = FALSE,
   button[4] = FALSE, button[5] = FALSE
}

entity user()
definition
   x = rand(5), y = 4, delay = rand(3)
action
   delay && (y == 0) && !button[1]
print ("pressing Fill A") -> button[1] = TRUE,
   delay && (y == 0) && button[1]
print ("releasing fill A") -> button[1] = FALSE ; y = |x|,
   delay && (y == 1) && !button[2]
print ("pressing fill B") -> button[2] = TRUE,
   delay && (y == 1) && button[2]
print ("releasing fill B") -> button[2] = FALSE ; y = |x|,
   delay && (y == 2) && !button[3]
print ("pressing empty A") -> button[3] = TRUE,
   delay && (y == 2) && button[3]
print ("releasing empty A") -> button[3] = FALSE ; y = |x|,
   delay && (y == 3) && !button[4]
print ("pressing empty B") -> button[4] = TRUE,
   delay && (y == 3) && button[4]
print ("releasing empty B") -> button[4] = FALSE ; y = |x|,
   delay && (y == 4) && !button[5]
print ("pressing pour") -> button[5] = TRUE,
   delay && (y == 4) && button[5]
print ("releasing pour") -> button[5] = FALSE ; y = |x|
}


entity dialogue_manager () {
definition
   !updating && button[1] && avail[1] -> updating = TRUE ; pour (1),
   !updating && button[2] && avail[2] -> updating = TRUE ; pour (2),
   !updating && button[3] && avail[3] -> updating = TRUE ; pour (3),
   !updating && button[4] && avail[4] -> updating = TRUE ; pour (4),
   !updating && button[5] && avail[6]
                     -> contentB = |contentA + contentB| - contentA ;
                        updating = TRUE ; pour(6),
   !updating && button[5] && avail[7]
                    -> contentB = |contentA + contentB| - contentA ;
                       updating = TRUE ; pour(7)
}
```

The `pour()` entity is unchanged. The following instantiations are performed:

```
pic() jugs(5,7) user() screen() dialogue_manager()
```

Appendix 8.8 Execution of concurrent activity program

In this appendix we show the results of executing the program of Appendix 8.7. A button push and release which occurs whilst a previous option is being implemented is ignored.

```
Script started on Wed Sep  6 20:33:28 1989
am> compiling pic()
am> compiling jugs(_capA,_capB)
am> compiling screen()
am> compiling user()
am> compiling dialogue_manager()
am> compiling pour(_option)
am> instantiating pic
am> instantiating jugs(5,7)
am> instantiating user
am> instantiating screen
am> instantiating dialogue_manager
am> starting simulation
#
       *  *
       *  *
 *  * *  *
 *  * *  *
 *  * *  *
 *  * *  *
 *  * *  *
 **** ****
 jugA jugB
#
       *  *
       *  *
 *  * *  *
 *  * *  *
 *  * *  *
 *  * *  *
 *  * *  *
 **** ****
 jugA jugB
 pressing fill B
#
   •
   •       later
   •
       *  *
       *  *
 *  * *  *
 *  * *  *
 *  * *  *
 *  * *  *
 *  * *  *
 **** ****
 jugA jugB
 releasing fill B
#
       *  *
       *  *
 *  * *  *
 *  * *  *
 *  * *  *
```

```
 *  * *  *
 *  * *==*
 **** ****
 jugA jugB
#
      *  *
      *  *
 *  * *  *
 *  * *  *
 *  * *  *
 *  * *==*
 *  * *==*
 **** ****
 jugA jugB
 pressing pour
#
      *  *
      *  *
 *  * *  *
 *  * *  *
 *  * *==*
 *  * *==*
 *  * *==*
 **** ****
 jugA jugB
 releasing pour
   •
   •      later
   •
#
      *==*
      *==*
 *  * *==*
 *  * *==*
 *  * *==*
 *  * *==*
 *  * *==*
 **** ****
 jugA jugB
#
      *==*
      *==*
 *  * *==*
 *  * *==*          the pour instruction is ignored, and the simulation
 *  * *==*          awaits the next button press by the user() entity
 *  * *==*
 *  * *==*
 **** ****
 jugA jugB

script done on Wed Sep  6 20:35:40 1989
```