Appendix 5

Blocks testing

In this appendix we describe an abstract definitive machine program to simulate the movement of blocks. The program uses a DoNaLD interface to present an animated picture of the blocks being moved.

Appendix 5 Blocks implementation

In this appendix we present an adm program to simulate the movement of blocks. For control purposes we have introduced new `driving` variables, but the program is essentially the one developed in §4.3.

```
entity handler(_block)
definition
driving[_block] = drivingL[_block] || drivingR[_block],
drivingL[_block] = holding[_block] && pushingL[_block],
drivingR[_block] = holding[_block] && pushingR[_block],
pushingL[_block] = FALSE,
pushingR[_block] = FALSE,
holding[_block] = FALSE
action
(!holding[_block]) -> holding[_block]=TRUE,
(y[_block]==0) && holding[_block] && (!driving[_block])
                             -> holding[_block]=FALSE,
(y[_block]==-1) && holding[_block] && (!driving[_block])
                             -> pushingL[_block]=TRUE,
drivingL[_block] -> pushingL[_block] = FALSE,
(y[_block]==1) && holding[_block] && (!driving[_block])
                            ->pushingR[_block]=TRUE,
drivingR[_block]  -> pushingR[_block] = FALSE
}
```

```
entity control(_blk1,_blk2)
{
definition
```

```
   y[_blk1], y[_blk2], x[_blk1]=rand(3), x[_blk2]=rand(3),
action
   true
print("pa, pb, ha, hb, dra, drb, dlb, dla, ss = ", pa, ",", pb,
",", ha, ",", hb, ",", dra, ",", drb, "," ,dlb, ",", dla, ",", ss)
      -> y[_blk1] = |x[_blk1]|-1; y[_blk2] = |x[_blk2]|-1,
         ha = holding[_blk1] ; hb = holding[_blk2] ;
         dra = drivingR[_blk1]; drb = drivingR[_blk2];
         dla = drivingL[_blk1]; dlb = drivingL[_blk2]
}

entity blockstate()
{
definition
   stringtaut = (!stringsnap) && ((pb-pa)==d),
   touching = (pb-pa)==1,
   stringsnap = FALSE,
   pa = -1, pb = 1, ha = 0, hb = 0, ss=0,
   dra = 0, dla = 0, drb = 0, dlb =0,
   d = 5                   //d is the length of the string
action
   (pb-pa)>d && !stringsnap -> stringsnap = TRUE; ss=1
}




entity blockmover(_blk1,_blk2)
{
action
   drivingL[_blk1] && !stringtaut -> pa = |pa|-1,
   !holding[_blk2] && drivingL[_blk1] && stringtaut
      -> pb=pa+d; pa=|pa|-1,
   drivingR[_blk1] && !touching -> pa = |pa|+1,
   !holding[_blk2] && drivingR[_blk1] && touching
      -> pb=pa+1; pa = |pa|+1,
   drivingL[_blk2] && !touching -> pb = |pb|-1,
   !holding[_blk1] && drivingL[_blk2] && touching
      -> pa=pb-1; pb = |pb|-1,
   drivingR[_blk2] && !stringtaut -> pb = |pb|+1,
   !holding[_blk1] && drivingR[_blk2] && stringtaut
      -> pa=pb-d; pb=|pb|+1
}
```

The following instantiations are made:

```
control(0,1)
handler(0)
handler(1)
blockstate()
blockmover(0,1)
```

Appendix 5.1 DoNaLD interface

An example of the output generated by the program of Appendix 5 is:

```
pa,pb,ha,hb,dra,drb,dlb,dla,ss = -1,1,0,0,0,0,0,0,0
pa,pb,ha,hb,dra,drb,dlb,dla,ss = -1,1,TRUE,TRUE,FALSE,FALSE,FALSE,
   FALSE,0
pa,pb,ha,hb,dra,drb,dlb,dla,ss = -1,1,TRUE,TRUE,TRUE,FALSE,FALSE,
   FALSE,0
pa,pb,ha,hb,dra,drb,dlb,dla,ss = 0,1,TRUE,TRUE,FALSE,TRUE,FALSE,
   FALSE,0
pa,pb,ha,hb,dra,drb,dlb,dla,ss = 0,2,FALSE,TRUE,FALSE,FALSE,FALSE,
   FALSE,0
pa,pb,ha,hb,dra,drb,dlb,dla,ss = 0,2,TRUE,TRUE,FALSE,TRUE,FALSE,
   FALSE,0
pa,pb,ha,hb,dra,drb,dlb,dla,ss = 0,3,TRUE,TRUE,FALSE,FALSE,FALSE,
   FALSE,0
pa,pb,ha,hb,dra,drb,dlb,dla,ss = 0,3,TRUE,TRUE,TRUE,TRUE,FALSE,
   FALSE,0
pa,pb,ha,hb,dra,drb,dlb,dla,ss = 1,4,TRUE,TRUE,FALSE,FALSE,FALSE,
   FALSE,0
```

and so on…

This output is clearly very uninformative, and so a graphical interface has been constructed for the program. The demonstration file which is executed contains the following command:

```
cat blocks | ./am -s -n -a > amlog  &
tail +0lf amlog | cat -u donald.init - | donald
```

The first command commences execution of the blocks adm program, with the output sent to a file called `amlog`. The second command takes the contents of `amlog` as it appears and passes it to the DoNaLD [Beynon et al

86] interpreter. The DonaLD interpreter is initialised with the file `donald.init`, which sets up the picture, and then reads the output generated by the `blocks` program. The picture on the screen is animated according to the changes in values of variables caused by the execution of the `blocks` program. The file `donald.init` contains:

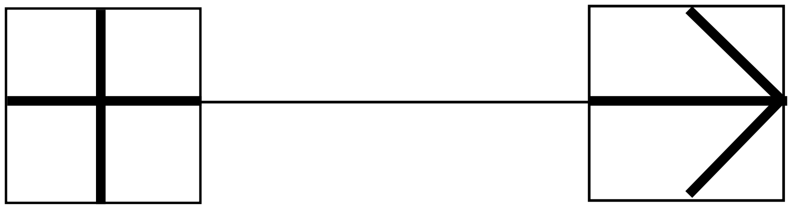
```
int pa, ha, dra,dla,ss                          declare these variables as DoNaLD variables

openshape block1                                describe block1
within block1 {
   point centre, NW,NE,SW,SE, N,E,S,W,X
   centre = +~/pa*100, 500
   NW = centre+,50
   SW = centre-,50
   NE = centre+,50
   SE = centre+,-50
   N = if ~/ha then (NE+NW) div 2 else centre
   S = if ~/ha then (SW+SE) div 2 else centre
   E = if ~/ha then (NE+SE) div 2 else centre
   W = if ~/ha then (NW+SW) div 2 else centre
   X = if ~/dra then E else if ~/dla then W else centre
   line Nl,Sl,El,Wl,NX,SX,WE
   Nl = [NW,NE]
   Sl = [SW,SE]
   El = [SE,NE]
   Wl = [SW,NW]
   WE = [W,E]
   NX = [N,X]
   SX = [S,X]   }
int pb, hb, drb,dlb
openshape block2                                describe block2
within block2 {
   point centre, NW,NE,SW,SE, N,E,S,W,X
   centre = +~/pb*100, 500
   NW = centre+,50
   SW = centre-,50
   NE = centre+,50
   SE = centre+,-50
   N = if ~/hb then (NE+NW) div 2 else centre
   S = if ~/hb then (SW+SE) div 2 else centre
   E = if ~/hb then (NE+SE) div 2 else centre
   W = if ~/hb then (NW+SW) div 2 else centre
   X = if ~/drb then E else if ~/dlb then W else centre
   line Nl,Sl,El,Wl,NX,SX,WE
   Nl = [NW,NE]
   Sl = [SW,SE]
   El = [SE,NE]
   Wl = [SW,NW]
   WE = [W,E]
   NX = [N,X]
   SX = [S,X]
}

line string                                                   describe the string
string =  [(block1/SE+block1/NE) div 2,
          if ss then (block1/SE+block1/NE) div 2 else
                                (block2/SW+block2/NW) div 2]

int TRUE, FALSE
TRUE =1
FALSE=0
```

The output of the blocks simulation, as stored in `amlog`, is then used to produce an animated representation of the blocks on the screen. This is difficult to demonstrate, so we here just give two possible pictures which can occur. Block A being held and block B being moved right:



and block A not being held and block B being moved left: