```
c===========================================================
c     ca1dr1: Implements 1D 2-state (k=2), range-1 (r=1)
c     cellular automata with "dead" boundary conditions.
c
c     usage: ca1dr1 <rule> [<nsite> <ngen> <init option>]
c
c     Output is character-based, thus ’nsite’ is currently
c     restricted to 78 for viewing on standard ’terminals’.
c
c     Formalism follows:
c
c        Wolfram, Rev. Mod Phys, v55, 601-644 (1983)
c===========================================================
      program          ca1dr1

      implicit         none

      integer          iargc,        i4arg,        roll

c-----------------------------------------------------------
c     Command-line arguments.
c-----------------------------------------------------------
      integer          rule,         nsite,        ngen,
     &                 init_option

c-----------------------------------------------------------
c     Maximum number of sites.
c-----------------------------------------------------------
      integer          maxnsite
      parameter      ( maxnsite = 78 )

c-----------------------------------------------------------
c     Storage for two generations of sites.
c-----------------------------------------------------------
      integer          c(maxnsite,2)
      integer          n,            np1
```

```
c-----------------------------------------------------------
c     Update rule.  For 2-state, range-1 rules there are
c     eight possible site+nearest-neighbor configurations,
c     conveniently represented as a 3-bit binary number
c     000-111 (binary) or 0-7 (decimal).
c-----------------------------------------------------------
      integer         range,        nupdate
      parameter     ( range = 1,    nupdate = 8 )
      integer         update(0:nupdate)

c-----------------------------------------------------------
c     Locals.
c----------------------------------------------------------
      integer         i,            isite,        igen

c-----------------------------------------------------------
c     Argument parsing.
c----------------------------------------------------------
      if( iargc() .lt. 1 ) go to 900
      rule        = i4arg(1,-1)
      if( rule .lt. 0 .or. rule .gt. 255 ) go to 900
      nsite       = i4arg(2,maxnsite)
      if( nsite .lt. 3   .or.  nsite .gt. maxnsite )
     &    nsite = maxnsite
      ngen        = i4arg(3,60)
      init_option = i4arg(4,0)

c-----------------------------------------------------------
c     Construct (decode) update fcn from rule #.
c----------------------------------------------------------
      do i = 0 , nupdate
         if( and(rule,2**i) .ne. 0 ) then
            update(i) = 1
         else
            update(i) = 0
         end if
      end do
```

```
c-----------------------------------------------------------
c     Check that update is quiescent and left-right
c     symmetric.
c----------------------------------------------------------
      if( update(0) .ne. 0 ) then
         write(0,*) 'ca1dr1: Rule ', rule, ' not quiescent'
         stop
      end if
      if( update(1) .ne. update(4)  .or.
     &    update(3) .ne. update(6) ) then
         write(0,*) 'ca1dr1: Rule ', rule, ' not symmetric'
         stop
      end if
c-----------------------------------------------------------
c     Initialize configuration:
c
c     init_option = 0   ->  Each site live with 50% prob.
c     init_option = 1   ->  Center site live, others dead.
c-----------------------------------------------------------
      n   = 1
      np1 = 2
      if(      init_option .eq. 0 ) then
         c(1,n)       = 0
         do isite = 2 , nsite - 1
            c(isite,n) = roll(2) - 1
         end do
         c(nsite,n)   = 0
      else if( init_option .eq. 1 ) then
         call ivloadsc(c(1,n),nsite,0)
         c(nsite/2,n) = 1
      else
         write(0,*) 'ca1dr1: Unimplemented initialization '//
     &              'option ', init_option
         stop
      end if
c-----------------------------------------------------------
c     Character-oriented output of initial configuration.
c-----------------------------------------------------------
      call charout(c(1,n),nsite,' ','*',6)
```

```
c-----------------------------------------------------------
c     Update loop.
c-----------------------------------------------------------
      do igen = 2 , ngen
c-----------------------------------------------------------
c        Left boundary site stays dead.
c-----------------------------------------------------------
         c(1,np1) = 0
         do isite = 2 , nsite -1
c-----------------------------------------------------------
c           Index into update rule by encoding left-to-right
c           nearest-neighbor (r=1) group of sites as 3 bit
c           binary number (i.e. o o o -> 0, o o * -> 1, ...
c           * * o -> 6, * * * -> 7)
c-----------------------------------------------------------
            c(isite,np1) = update(4 * c(isite-1,n) +
     &                            2 * c(isite,  n) +
     &                                c(isite+1,n))
         end do
c-----------------------------------------------------------
c        Right boundary site stays dead.
c-----------------------------------------------------------
         c(nsite,np1) = 0
c-----------------------------------------------------------
c        Output new configuration.
c-----------------------------------------------------------
         call charout(c(1,np1),nsite,' ','*',6)
c-----------------------------------------------------------
c        Slightly tricky way to 'swap' two values which
c        are always either (1,2) or (2,1).
c-----------------------------------------------------------
         np1 = 3 - np1
         n   = 3 - n
      end do
      stop
 900  continue
         write(0,*) 'usage: ca1dr1 <rule> '//
     &              '[<nsite> <ngen> <init option>]'
      stop
      end
```

```
c-----------------------------------------------------------
c     Dumps character representation of bit vector.
c-----------------------------------------------------------
      subroutine charout(bv,n,char0,char1,unit)
         implicit     none

         integer      n,          unit
         integer      bv(n)
         character*1  char0,      char1

         character*78 buffer
         integer      ln,         i

         if( n .ge. 0 ) then
            ln = min(n,78)
            do i = 1 , ln
               if( bv(i) .eq. 0 ) then
                  buffer(i:i) = char0
               else
                  buffer(i:i) = char1
               end if
            end do
            write(unit,*) buffer(1:ln)
         end if
         return
      end
```

```
c------------------------------------------------------------
c     Returns uniformly distributed random integer chosen
c     from 1 to n.
c------------------------------------------------------------
      integer function roll(n)
         implicit     none

         real*8       rand

         integer      n

         roll = min(n,1 + int(n * rand()))

         return
      end
c------------------------------------------------------------
c     Loads integer vector with scalar.
c------------------------------------------------------------
      subroutine ivloadsc(v,n,sc)
         implicit     none

         integer      n,       sc
         integer      v(n)

         integer      i

         do i = 1 , n
            v(i) = sc
         end do

         return
      end
```

```
c===========================================================
c     History: ca1dr1
c
c     ca1dr2t: Implements 1D 2-state (k=2), range-2 (r=2)
c     totalistic cellular automata with "dead" boundary
c     conditions.
c
c     usage: ca1dr2t <rule> [<nsite> <ngen> <init option>]
c
c     Output as in 'ca1dr1'.
c
c     Formalism follows:
c
c        Wolfram, Rev. Mod Phys, v55, 601-644 (1983)
c===========================================================
      program         ca1dr2t

      implicit        none

      integer         iargc,        i4arg,       roll
      logical         iseqivv

c-----------------------------------------------------------
c     Command-line arguments.
c-----------------------------------------------------------
      integer         rule,         nsite,       ngen,
     &                init_option

c-----------------------------------------------------------
c     Maximum number of sites.
c-----------------------------------------------------------
      integer         maxnsite
      parameter     ( maxnsite = 78 )

c-----------------------------------------------------------
c     Storage for two generations of sites.
c-----------------------------------------------------------
      integer         c(maxnsite,2)
      integer         n,            np1
```

```fortran
c------------------------------------------------------------
c     Update rule.  For 2-state, range-2 totalistic rules
c     there are 6 possible site+nearest-neighbor
c     configurations (total of 0 to 5 sites alive).
c------------------------------------------------------------
      integer         range,        nupdate
      parameter     ( range = 2,    nupdate = 6 )
      integer         update(0:5)

c------------------------------------------------------------
c     Locals.
c-----------------------------------------------------------
      integer         i,            isite,        igen,
     &                nalive

c------------------------------------------------------------
c     Argument parsing.
c-----------------------------------------------------------
      if( iargc() .lt. 1 ) go to 900
      rule        = i4arg(1,-1)
      if( rule .lt. 0 .or. rule .gt. 255 ) go to 900
      nsite       = i4arg(2,maxnsite)
      if( nsite .lt. 3   .or.  nsite .gt. maxnsite )
     &    nsite = maxnsite
      ngen        = i4arg(3,60)
      init_option = i4arg(4,0)

c------------------------------------------------------------
c     Construct (decode) update fcn from rule #.
c-----------------------------------------------------------
      do i = 0 , nupdate - 1
         if( and(rule,2**i) .ne. 0 ) then
            update(i) = 1
         else
            update(i) = 0
         end if
      end do
```

```
c-----------------------------------------------------------
c     Check that update is quiescent, left-right symmetry
c     automatic with totalistic rules.
c----------------------------------------------------------
      if( update(0) .ne. 0 ) then
         write(0,*) 'ca1dr2t: Rule ', rule, ' not quiescent'
         stop
      end if

c-----------------------------------------------------------
c     Initialize configuration:
c
c     init_option = 0   ->  Each site live with 50% prob.
c     init_option = 1   ->  Center site live, others dead.
c-----------------------------------------------------------
      n   = 1
      np1 = 2
      if(      init_option .eq. 0 ) then
         call ivloadsc(c(1,n),2,0)
         do isite = 3 , nsite - 2
            c(isite,n) = roll(2) - 1
         end do
         call ivloadsc(c(nsite-1,n),2,0)
      else if( init_option .eq. 1 ) then
         call ivloadsc(c(1,n),nsite,0)
         c(nsite/2,n) = 1
      else
         write(0,*) 'ca1dr2t: Unimplemented initialization '//
     &              'option ', init_option
         stop
      end if
c-----------------------------------------------------------
c     Character-oriented output of initial configuration.
c-----------------------------------------------------------
      call charout(c(1,n),nsite,' ','*',6)
```

```
c-----------------------------------------------------------
c     Update loop.
c-----------------------------------------------------------
      do igen = 2 , ngen
c-----------------------------------------------------------
c        2 left boundary sites stay dead.
c-----------------------------------------------------------
         call ivloadsc(c(1,np1),2,0)
         do isite = 3 , nsite - 2
c-----------------------------------------------------------
c           Index into update rule by counting all live
c           sites in range-2 neighborhood of site.
c-----------------------------------------------------------
            nalive = 0
            do i = -range , range
               nalive = nalive + c(isite+i,n)
            end do
            c(isite,np1) = update(nalive)
         end do
c-----------------------------------------------------------
c        2 right boundary sites stay dead.
c-----------------------------------------------------------
         call ivloadsc(c(nsite-1,np1),2,0)
c-----------------------------------------------------------
c        Quit if configuration is static.
c-----------------------------------------------------------
         if( iseqivv(c(1,n),c(1,np1),nsite) ) then
            write(0,*) 'ca1dr2t: Configuration is static '//
     &                 'after ', igen, ' generations'
            stop
         end if
c-----------------------------------------------------------
c        Output new configuration.
c-----------------------------------------------------------
         call charout(c(1,np1),nsite,' ','*',6)

         np1 = 3 - np1
         n   = 3 - n
      end do
      stop

 900  continue
         write(0,*) 'usage: ca1dr2t <rule> '//
     &              '[<nsite> <ngen> <init option>]'
      stop

      end

c-----------------------------------------------------------
c     Tests two integer vectors for equality.
c-----------------------------------------------------------
      logical function iseqivv(v1,v2,n)
         implicit     none

         integer      n
         integer      v1(n),        v2(n)

         integer      i

         iseqivv = .true.
         do i = 1 , n
            if( v1(i) .ne. v2(i) ) then
               iseqivv = .false.
               return
            end if
         end do

         return

      end
```