An Approach to Cellular Automata Modeling in Modelica

Victorino Sanz   Alfonso Urquia

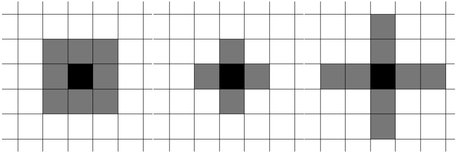
Departamento de Informática y Automática
ETSI Informática
{*vsanz,aurquia*}*@dia.uned.es*

EOOLT 2013, Nottingham

Outline

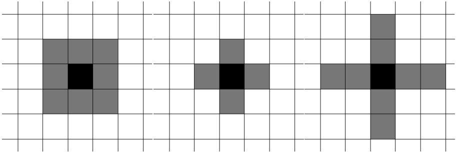
Cellular Automata

- Dynamic, discrete-time and discrete-space models
- Space represented as a grid of *cells*
- State updated using a transition function (*rule*)
- Different types of neighborhoods:



Cellular Automata

- Dynamic, discrete-time and discrete-space models
- Space represented as a grid of *cells*
- State updated using a transition function (*rule*)
- Different types of neighborhoods:



- Multiple domains (chemistry, medicine, economics, biology, ...)
- Microscopic approach to study fluid dynamics (LGCA, LBM)

Cellular Automata

- Formal specification using DEVS
  - Classic DEVS and Multicomponent DEVS (Zeigler)
  - Cell-DEVS (Wainer)
- Implementation of GOL using Modelica (Fritzson)

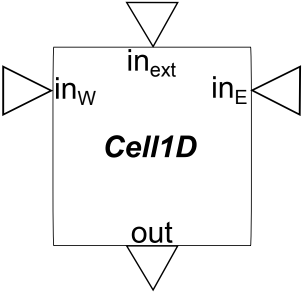
Cellular Automata

- Formal specification using DEVS
  - Classic DEVS and Multicomponent DEVS (Zeigler)
  - Cell-DEVS (Wainer)
- Implementation of GOL using Modelica (Fritzson)

Objective

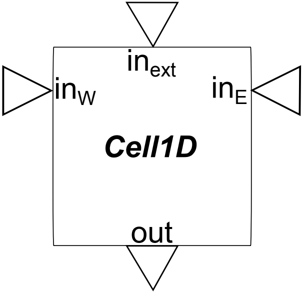
Facilitate the description of CA models in order to combine them with other Modelica models

One-dimensional Cell (*Cell1D*)



- Interface:
  - in_{ext} for initial inputs
  - in_W and in_E for state updates from neighbors
  - *out* to communicate the current state
- State variables:
  - *phase* ("active", "passive")
  - *sigma* (time advance)
  - *CS* (current cell state)
  - N_E and N_W (neighbors state)
- Formal specification in the manuscript

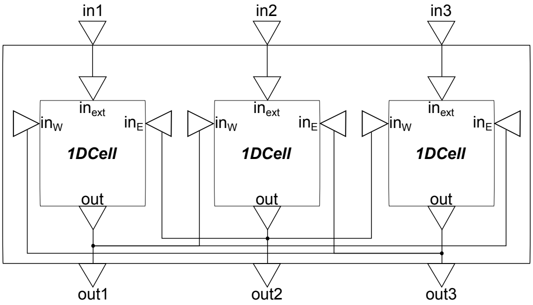
One-dimensional Cell (*Cell1D*)



- Interface:
  - in_{ext} for initial inputs
  - in_W and in_E for state updates from neighbors
  - *out* to communicate the current state
- State variables:
  - *phase* ("active", "passive")
  - *sigma* (time advance)
  - *CS* (current cell state)
  - N_E and N_W (neighbors state)
- Formal specification in the manuscript

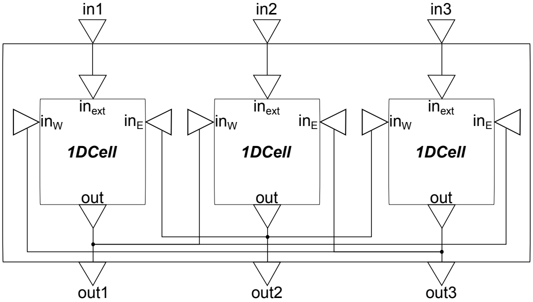
Analogous for two-dimensional cells (*Cell2D*), adding additional input ports and state variables for the new neighbors

One-dimensional Cellular Space (*CellSpace1D*)



- Array of N cells
- Moore's neighborhood
- Wrapped boundaries
- Input and output ports to cells

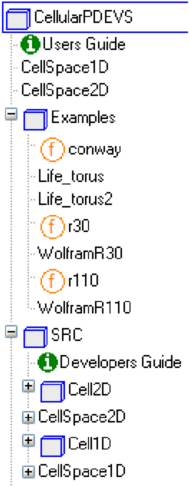
One-dimensional Cellular Space (*CellSpace1D*)



- Array of N cells
- Moore's neighborhood
- Wrapped boundaries
- Input and output ports to cells

Analogous for two-dimensional cellular spaces (*CellSpace2D*), increasing the size to $N \times N$, adding connections to new neighbors and 2D wrapped boundaries

CellularPDEVS Architecture



- User's area:
  - Documentation
  - *CellSpace1D* and *CellSpace2D* models used to develop new CA
  - Examples (rule 30, rule 110 and the Game of Life)
- Developer's area: internal implementation of cells and cellular spaces, and developer oriented documentation
- Implemented using the DEVSLib library
- Follows the presented formal specification

CellularSpaces

- Implemented as coupled DEVSLib models (array or matrix of interconnected individual cells)
- Connections follow the Moore's neighborhood with wrapped boundaries
- Automatically generates a graphical animation
- Initial state set using the Generator and DUP_N models from DEVSLib
- A replaceable function is used as transition function for each cell

```
function Rule
  input Integer s;
  input Integer[N] neighbors;
  output Integer sout;
algorithm
end Rule;
```

Development of New CA Models

- Extend the default cellular space model (*CellSpace1D* or *CellSpace2D*)
- Set the size and initial conditions of the cellular space
- Define the transition function

Development of New CA Models

- Extend the default cellular space model (*CellSpace1D* or *CellSpace2D*)
- Set the size and initial conditions of the cellular space
- Define the transition function

CellularPDEVS models can be combined with other Modelica models

- The state can be modified by sending a message to the desired cell (similarly to the initial message)
  - Continuous-time and discrete-time signals can be translated into messages (Quantizer, CrossUP and CrossDOWN)
- CA state can be observed using a variable of the cellular space

One-dimensional CA: Rule 30

$$00011110_2 = 30_{10}$$

```
model WolframR30
  extends CellSpace1D(
    Ssize = 20,
    init_cell = 10,
    redeclare replaceable
      function Rule = r30);
end WolframR30;
```

```
function r30
  input Integer s;
  input Integer[2] neighbors;
  output Integer sout;
protected
  Integer[2] n = neighbors;
algorithm
  if     n[2]==1 and s==1 and n[1]==1 then
    sout := 0;
  elseif n[2]==1 and s==1 and n[1]==0 then
    sout := 0;
  elseif n[2]==1 and s==0 and n[1]==1 then
    sout := 0;
  elseif n[2]==1 and s==0 and n[1]==0 then
    sout := 1;
  elseif n[2]==0 and s==1 and n[1]==1 then
    sout := 1;
  elseif n[2]==0 and s==1 and n[1]==0 then
    sout := 1;
  elseif n[2]==0 and s==0 and n[1]==1 then
    sout := 1;
  elseif n[2]==0 and s==0 and n[1]==0 then
    sout := 0;
  end if;
end r30;
```

One-dimensional CA: Rule 110

$$01101110_2 = 110_{10}$$

```
model WolframR110
  extends CellSpace1D(
    Ssize = 20,
    init_cell = 10,
    redeclare replaceable
      function Rule = r110);
end WolframR110;
```

```
function r110
  input Integer s;
  input Integer[2] neighbors;
  output Integer sout;
protected
  Integer[2] n = neighbors;
algorithm
  if     n[2]==1 and s==1 and n[1]==1 then
    sout := 0;
  elseif n[2]==1 and s==1 and n[1]==0 then
    sout := 1;
  elseif n[2]==1 and s==0 and n[1]==1 then
    sout := 1;
  elseif n[2]==1 and s==0 and n[1]==0 then
    sout := 0;
  elseif n[2]==0 and s==1 and n[1]==1 then
    sout := 1;
  elseif n[2]==0 and s==1 and n[1]==0 then
    sout := 1;
  elseif n[2]==0 and s==0 and n[1]==1 then
    sout := 1;
  elseif n[2]==0 and s==0 and n[1]==0 then
    sout := 0;
  end if;
end r110;
```

Two-dimensional CA: Game of Life

- Dead cell borns if 3 neighbors alive
- Living cell dies if less than 2 or more than three 3 neighbors alive
- No change otherwise

```
model Life_torus
  extends CellSpace2D(
    Ssize = 10,
    init_cells =  [1,2; 2,3; 3,1; 3,2; 3,3],
    redeclare replaceable
      function Rule = conway);
end WolframR110;
```

```
function conway
  input Integer s;
  input Integer[8] neighbors;
  output Integer sout;
protected
  Integer[8] n = neighbors;
algorithm
  sout := s;
  if s==0 then // dead, maybe borns
    if sum(n)==3 then
      sout := 1;
    end if;
  else // alive, maybe dies
    if (sum(n)<2 or sum(n)>3) then
      sout := 0;
    end if;
  end if;
end conway;
```

Two-dimensional CA: Game of Life

```
init_cells = [2,2; 2,4; 3,5; 4,5; 5,5; 6,3; 6,4; 6,5; 5,2]
```

Future work

- Extend the functionality of the library (boundaries, neighborhoods, interactive initialization, etc.)
- Model more complex systems (cement clinker cooler or PEM fuel cell)
- Performance evaluation

Conclusions

- CellularPDEVS facilitates the description of CA in Modelica
- Supports 1D and 2D cellular spaces
- Models are specified using the Parallel DEVS formalism
- Implemented using the DEVSLib library
- Facilitates the combination of CA with other Modelica models