# Automated transformation of system of ordinary differential equations into Boolean models

**Project Report**

Radostina Misirkova

December 23, 2015

## 1 Background

Ordinary differential equations models as well as Boolean models are abstractions of reality, they focus on certain aspects of the underlying system. In order to connect both model approaches and use them in parallel a systematic transition scheme has been presented in the paper *Stötzel et. al.* (2015). The step-wise procedure starts with ODE and derives a Boolean model, which allows a more global analysis of the system to complement the results obtained from the ODE model. The aim of the project was to implement the translating step of this method based on an Euler-like discretization. The software was developed using *MATLAB*®[1]. The program reads the input ODE system and outputs a text file in *BoolNet*[2] format, representing the corresponding Boolean network.

In the following the mathematical background of the translating step is briefly described. The discretization of the continuous system is based on the structure and monotonicity behavior of the right hand side functions of the ODEs. The starting point is an autonomous system in the form

$$y'(t) = f(y,p), \quad y(0) = y_0$$

with time variable $t \in \mathbb{R}_{\geq 0}$, state vector $y(t) = (y_1(t), \ldots, y_n(t))^T \in \mathbb{R}^n$, model parameters $p \in \mathbb{R}^q$, and right hand side $f : \mathbb{R}^{n \times q} \mapsto \mathbb{R}^n$. In the following the parameter vector $p$ is omitted. All continuous variables $y_i(t)$ are normalized to the interval $[0,1]$ and the right hand side functions $f(y) = (f_1(y), \ldots, f_n(y))^T$ consist only of sums and products of *monotone* functions $F_{i,j}(y_j)$:

$$y_i'(t) = f_i(F_{i,1}(y_1), \ldots, F_{i,n}(y_n)), \quad i = 1, \ldots, n. \tag{1}$$

In addition each monotone function $F_{i,j}(y_j)$ only takes values in $[0,1]$. To derive the Boolean model all continuous variables $y_1, \ldots, y_n$ in the right hand side functions $f_i(y)$ are replaced by their discrete

[1] *MATLAB*® is a high-level language and interactive environment for numerical computation, visualization, and programming

[2] *BoolNet* is a software tool for assembling, analyzing and visualizing synchronous and asynchronous Boolean networks as well as probabilistic Boolean networks, the package is able to read in text file, where each line consists of a target gene and an update rule, separated by a comma. The first line of a file in *BoolNet* format is a header: targets, factors.

counterparts $x_1,\ldots,x_n$. Thereby, every monotonically increasing function $F_{i,j}(y_j)$ is mapped to $x_j$, and every monotonically decreasing function $F_{i,j}(y_j)$ is mapped to $(1-x_j)$, using the operator:

$$T(F) := \begin{cases} Id & \text{, if } F \text{ is monotonically increasing,} \\ 1-Id & \text{, if } F \text{ is monotonically decreasing.} \end{cases}$$

In this way the right hand sides $f_i(y_1,\ldots,y_n)$ in (1) are mapped to discrete counterparts $h_i(x_1,\ldots,x_n)$,

$$f_i(F_{i,1}(y_1),\ldots,F_{i,n}(y_n)) \mapsto f_i(\underbrace{TF_{i,1}(x_1)}_{x_1 \text{ or } 1-x_1},\ldots,\underbrace{TF_{i,n}(x_n)}_{x_n \text{ or } 1-x_n}) =: h_i(x_1,\ldots,x_n),$$

for all $i = 1,\ldots,n$. The sign of $h_i(x)$ then determines the update functions for the Boolean variables:

$$g_i : \{0,1\}^n \to \{0,1\} : x \mapsto g_i(x) := \begin{cases} 1 & \text{, if } sgn(h_i(x)) > 0 \\ x_i & \text{, if } sgn(h_i(x)) = 0\,, \\ 0 & \text{, if } sgn(h_i(x)) < 0 \end{cases} \quad i = 1,\ldots,n \tag{2}$$

Based on the above definitions the translation step can be interpreted as an Euler discretization yielding the successor state $x^{k+1}$ for a current state $x^k$ in the form

$$x_i^{k+1} = g_i(x^k) = x_i^k \dot{+} sgn(h_i(x_1^k,\ldots,x_n^k)),$$

where $\dot{+}$ denotes the operation $0\dot{+}(-1) = 0$ and $1\dot{+}1 = 1$ and the usual addition in all other cases.

## 2 Implementation

The software contains four *MATALB* functions: the main function *odeToBNet.m* and three help functions (*loadEquations.m, loadODEscript.m, isODEscript.m*).

The input set of ODEs can be read either from a standardized text file format or from a *MATLAB* program file. In the first case each line of the file consists of a differential equation, where the parameter values are substituted in the right hand side functions. The function *loadEquations.m* is able to read in the text file line by line storing the variable names in a cell array and the right hand side of the equations to a second cell array.

When reading ODE input system from a *MATLAB* program file keywords are required to identify parameter values and equations lines in the file. The *loadODEscript.m* helper function reads the file line by line and recognize parameters between keywords: `% parameters` and `% end of parameters`. Equations are enclosed by keywords: `% equations` and `% end of equations`. Comment lines therebetween are allowed. The parameters are stored to a cell array and the set of equations to a second one. Afterwards the values of the parameters are assigned to the equations. The variable names are stored in an additional cell array.

In the next step regular expressions are used to match functions in the right hand side. It has been assumed that the monotone functions in the input ODE are parenthesized. So whenever a monotone function occurs in an equation it has to be put in parentheses, such that both the nominator and the denominator are inside parentheses. In order to find such fraction expression a regular expression of the form is used:

```
'\([a-z0-9+\-*.\s^]+\)/\([a-z0-9+\-*.\s^]+\)'.
```

Biochemical kinetics like Hill kinetics, Michaelis-Menten kinetics, Goldbeter-Koshland kinetics are in form of fraction expressions. An additional regular expression is used to match remaining expressions inside parantheses:

```
'\([a-z0-9+\-*.\s\^]+\)'.
```

After determining, if the matched expressions are monotonic increasing or monotonic decreasing (see function *monotonic* bellow), they get replaced by the corresponding boolean variable (see function *func_rep* bellow).

```
function val = monotonic(fnc,var)
% determines if the expression in the input function fnc is a monotonically
% increasing or monotonically decreasing function of the variable var
% output argument of type string: 'incr' or 'decr'
    try
        % substitutes var with value 0.5
        m = subs(fnc,var,'0.5');
    catch
        % in case of division by zero, substitution with 0.4
        m = subs(fnc,var,'0.4');
    end
    try
        n = subs(fnc,var, '1');
    catch
        n = subs(fnc,var, '0.9');
    end
    if (m <= n)
        val = 'incr';
    else
        val = 'decr';
    end
end
```

```
function str = func_rep(eq,mon_fnc)
% replaces a monotone function: mon_fnc by its descrete counterpart
% depending on whether the function is monotonically decreasing
% or monotonically increasing
    %a string with values 'incr' or 'decr'
    s = monotonic(mon_fnc,get_var(mon_fnc));
    %the discrete counterpart of the variable
    boolVar = get_boolVar(get_var(mon_fnc));

    if strcmp(s,'incr') == 1
        % monotonically increasing function mapping
        str = strrep(eq,mon_fnc,boolVar);
   else
         % monotonically decreasing function mapping
        str = strrep(eq,mon_fnc,['(1-' boolVar ')']);
    end
end
```

The remaining continuous variables in the equations represent, for example mass actions kinetics and are substituted directly by the corresponding boolean variables. After performing this step, all occurring variables are replaced by their boolean counterparts and the right hand sides consist of sums and products of the boolean variables.

The value of the right hand side functions now can either be negative, positive or equal to zero and it determines the update rule for the boolean variables. This values have to be evaluated for all possible boolean states. For each equation a truth table is created of size $n := 2^m$, where $m$ is the number of variables in the current equation. Than, the value of the function is computed for all $n$ possible states. According to the sign of this value the update of the boolean variable is determined using the update function $g$, defined in the background section.

```
function next = g(sgn_vec,current)
% the update function g
% computes the update value of variable for given
next = zeros(1,numel(sgn_vec));
        for n=1:numel(sgn_vec)

            if sgn_vec(n)==0
                next(n)=current(n);

            else if sgn_vec(n) > 0
                    next(n) = 1;
                else next(n) = 0;
                end
            end

        end
    end
```

The truth tables for each equations can be printed in the command window of *MATLAB* within the main for-loop of the function *odeToBNet*. Additionally they are used to write an output file in *BoolNet* text format for further analysis. Each truth table is converted to a logical function in disjunctive normal form (*DNF*). The procedure finds all rows in the table that ends with 1, then takes the values of the occurring variables from each respective column. If the value of a variable is 0, the complement of the variable is taken, otherwise the variable itself is taken:

```
function arr = get_LogicVar(row,var_cell)
% substitutes the values in a table row with logical variables
% the output is a cell array
    arr = {};

    for j=1:(numel(var_cell))
        if row.(var_cell{j})==0
            x = ['!' char(var_cell{j})]; % '!' for negation

        else x = char(var_cell{j});
        end

        arr = [arr x];
    end
end
```

Afterwards the variables are joined by logical *AND*. In this way the terms of the *DNF* expression are determined and all terms resulting of a table are joined by logical *OR*.

```
cell_arr = get_LogicVar(row,varInFnc);
% joins the variables in cell_arr  with logical AND
str = strjoin(cell_arr,' & ');
str_eq=[str_eq str];
```

Finally the logical equation are written into the output file:

```
% write the logical equation in the text file, joined by logical OR
fprintf(fileID,'%s, %s\n',cur_var,strjoin(str_eq,' | '));
```

## 3 Conclusion

In this report a possible implementation of a translating scheme for ODE system into a Boolean model has been presented. The application of the software is feasible only if the following input requirements are met: all continuous variables in the ODE system are normalized to the interval $[0,1]$; the right hand side functions consist of sums and products of monotone functions; the monotone functions only take values in $[0,1]$. The user of the software has to check if these assumptions are fulfilled and to parenthesize the monotone functions occurring in the right hand side as described in the implementation section. It is reasonable to automatize this preprocessing step by implementing a procedure, which additionally check for global monotonicity. Instead of parenthesizing, a method for decomposing the right hand side into expressions representing monotone functions would lead to more user friendly application of the software. In order to extend the scope of the software it is recommended to implement *SBML*[3] import functionality.

## 4 Application example

The following example demonstrates the conversion from ODE system into Boolean model using the software. The cascade model for the mitotic oscillator is taken from *Goldbeter* (1991).

[3] *SBML* is a model representation format and infrastructure to foster interactions between qualitative modeling formalisms and tools.

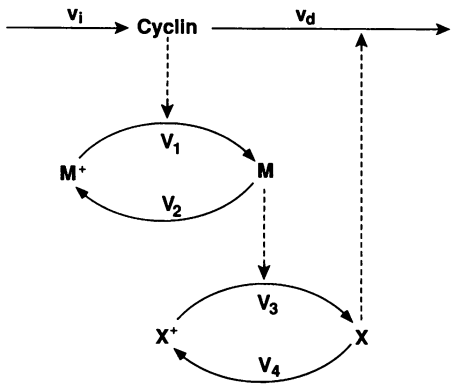


**Figure 1:** From *Goldbeter* (1991). Cyclin is synthesized at a constant rate $v_i$ and triggers the transformation of inactive $M^+$ into active *M cdc2 kinase*; a kinase reverts this modification. In the second cycle of the phosphorylation-dephosphorylation cascade, *cdc2* kinase elicits the transition from the inactive $X^+$ into the active $X$ form of a protease that degrades cyclin; the activation of cyclin protease is reverted by a phosphatase . $V_i$ $(i = 1-4)$ denotes the effective maximum rate of each of the four converter enzymes; $v_d$ denotes the maximum rate of cyclin degradation by protease $X$

ODE model:

$$\text{CYCLIN} \quad \frac{dC}{dt} = \underbrace{v_i}_{\text{synthesis}} - \underbrace{v_d X \left[\frac{C}{K_d + C}\right]}_{\text{protease degradation}} - \underbrace{k_d C}_{\text{degradation}}$$

$$\text{MPF} \quad \frac{dM}{dt} = \underbrace{V_{M1}\left[\frac{C}{K_C + C}\right]\left[\frac{(1-M)}{K_1 + (1-M)}\right]}_{\text{cyclin-stimulated activation}} - \underbrace{V_2\left[\frac{M}{K_2 + M}\right]}_{\text{inactivation}}$$

$$\text{PROTEASE} \quad \frac{dX}{dt} = \underbrace{MV_{M3}\left[\frac{(1-X)}{K_3 + (1-X)}\right]}_{\text{activation by MPF}} - \underbrace{V_4\left[\frac{X}{K_4 + X}\right]}_{\text{inactivation}}.$$

The *MATLAB* file shown bellow contains the system of ODEs and can be used for numerical simulations:

```
% ODE system of the minimal cascade model for the mitotic oscillator involving
% cyclin and cdc2 kinase: BioModels#3, model name "Goldbeter1991-Min Mit Oscil"
function dy = odecyc(t,y)
% parameters
K1 = .02;
K2 = .02;
K3 =.01;
K4 = .01;
Kc=.5;
Kd=.001;
kd=.046;
```

```
V2=2;
V4=.7;
vs=.06;
vd=.25;
VM1=4;
VM3=1;
% end of parameters

dy=zeros(3,1);
% equations
% cyclin
dy(1) = vs - vd*y(3)*(y(1))/(Kd + y(1))-kd*y(1);
% M-phase-promoting factor
dy(2) = (y(1)*VM1)/(Kc + y(1))*(1-y(2))/(K1+1-y(2))-(V2*y(2))/(K2+y(2));
% protease
dy(3) = y(2)*VM3*(1-y(3))/(K3+1-y(3))-V4*(y(3))/(K4+y(3));
% end of equations
```

The ODE model has been stored as *ode.m* file and the function *odeToBNet.m* can be called in the command window as:

```
>> odeToBNet('ode.m')
```

In the following the equations after the transition step as well as the corresponding tables are shown:

```
x1 = .06 - .25*x3*x1-.046*x1;
x2 = x1*(1-x2)-x2;
x3 = x2*1*(1-x3)-.7*x3;
```

| x1 | x3 | sgn | x1_next |
|---|---|---|---|
| 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | -1 | 0 |

| x1 | x2 | sgn | x2_next |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | -1 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | -1 | 0 |

| x2 | x3 | sgn | x3_next |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | -1 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | -1 | 0 |

The resulting output file in *BoolNet* text format can be used for further analysis of the system with the *BoolNet* package.

```
targets, factors
x1, !x1 & !x3 | !x1 & x3 | x1 & !x3
x2, x1 & !x2
x3, x2 & !x3
```

## References


Stötzel C., Röblitz S., and Siebert H. (2015), Complementing ODE-Based System Analysis Using Boolean Networks Derived from an Euler-like Transformation, PLoS ONE 10(10): e0140954. doi:10.1371/journal.pone.0140954.

Goldbeter A. (1991), A minimal cascade model for the mitotic oscillator involving cyclin and cdc2 kinase, Goldbeter A. Proc. Natl. Acad. Sci. U.S.A. 1991; 88(20):9107 – 11.