Symbolic and Algebraic Methods for Modeling, Analysis, Design and Implementation of Discrete Systems
Lecture notes for a seminar at ITM*14 May 1993

Roger Germundsson
Div. of Automatic Control
Dept. of Electrical Engineering
Linköping University
S-581 83 Linköping
Email: roger@isy.liu.se

May 6, 1993

1 Introduction

Consider:

- The operation of an automated manufacturing plant.
- The flawless functioning of computer and telephone networks.
- The daily operation of a large airport terminal with all its interactions between “jobs” and “resources”.
- The startup, shutdown or error recovery procedures of a large processing plant.
- Traffic control systems, i.e. traffic lights for cars and trains.
- A VLSI digital circuit.

These are all examples of man made systems that have a significant discrete component. We would like to be able to model, analyze, design and implement such systems. Within the control society these go by the name “Discrete Event Dynamic Systems”. Depending on the objective of study, i.e. timing, resource or correctness properties, the class of models under consideration will differ. See figure 1 for a rough classification of the model classes used in control theory.

This paper will deal with un timed deterministic or nondeterministic models and they will all be modeled as polynomial difference equations.

1.1 The Big Picture

We can essentially summarize the remainder of this document as in figure 2.

---

*Institutet för Tillämpad Matematik
## Deterministic vs. Stochastic

<table>
<thead>
<tr>
<th>Time Type</th>
<th>Deterministic</th>
<th>Stochastic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timed</td>
<td>- Timed Petri Nets</td>
<td>- Stochastic Petri Nets</td>
</tr>
<tr>
<td></td>
<td>- Timed Finite Automata</td>
<td>- Queueing Networks</td>
</tr>
<tr>
<td></td>
<td>- Min-max Algebra</td>
<td>- GSMP</td>
</tr>
<tr>
<td>Un timed</td>
<td>- Petri Nets, Grafcet</td>
<td>- Finite Markov Processes</td>
</tr>
<tr>
<td></td>
<td>- Finite Automata</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- CCS, CSP, FRP, PA</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- VHDL, Signal, Estrelle</td>
<td></td>
</tr>
</tbody>
</table>

### Figure 1: Descriptions used in connection with DEDS

### Figure 2: Summary of this document. In this picture MD$_i$ denotes a modeling domain, ID$_j$ denotes an implementation domain and PDS denotes polynomial dynamical systems (over finite fields).
In figure 2 we receive a model of the system in question expressed within some modeling domain \( MD_i \). We then translate this to our internal format which is polynomial (dynamical) systems over finite fields. This object can then be analyzed and we may add further constraints to do an actual design. At the end of all this we would like to obtain some form of implementation of our controller. This amounts to translating to some implementation domain \( ID_j \).

2 Polynomial Dynamical Systems over Finite Fields

We will model discrete dynamic systems as a polynomial dynamical systems over some finite field \( F_q \). These systems take the form:

\[
\begin{align*}
x^{+} &= f(x, u) \\
y &= g(x, u)
\end{align*}
\]

where \( u \) is the input, \( y \) is the output, \( x \) is the state and \( x^{+} \) is the next state. Finally \( f \) and \( g \) are vectors of polynomials in \( F_q[x_1, x_2, \ldots, x_n, u_1, u_2, \ldots, u_m] \).

2.1 Polynomials over Finite Fields

We will polynomial manipulations to analyze and design systems. We then need to make sure that we can actually represent our objects of interest in terms of polynomials.

Lemma 2.1 The polynomial ring \( F_q[x_1, x_2, \ldots, x_n] \) is functionally complete.

The proof is given since we will make use of this construction repeatedly.

Proof 2.1 Let \( f : F_q^n \rightarrow F_q \) be any function, then

\[
f_p(x) = \sum_{\xi \in F_q^n} L_\xi(x) f(\xi) \in F_q[x]
\]

where \( L_\xi(x) = L_{\xi_1}(x_1) \cdots L_{\xi_n}(x_n) \) and

\[
L_{\xi_i}(x_i) = \prod_{\nu \in F_q \setminus \{\xi_i\}} \frac{(x_i - \nu)}{(\xi_i - \nu)} = \begin{cases} 
1 & , x = \xi_i \\
0 & , x \neq \xi_i 
\end{cases}
\]

is the Lagrange interpolating polynomial. We then have \( f(\xi) = f_p(\xi) \) for all \( \xi \in F_q^n \).

2.2 Key Results

There is then a large body of results concerning the structure of this ring as well as algorithmic methods of computing properties, see [2, 5]. In particular we have an automatic way of generating proof systems for propositional and multiple valued logics, see [3, 4].

From now onwards the principal object of study will be ideals in the ring

\[
R_q = F_q[x_1, x_2, \ldots, x_n]/\langle x_1^n - x_1, \ldots, x_n^n - x_n \rangle
\]

1 These are usually denoted Galois fields and the number of elements \( q \) has to be a power of a prime \( q = p^q \).
2 A simple example of a polynomial in e.g. \( F_2[x, y] \) is \( 4x^3 + y^{12}x \)
3 A polynomial ring \( F[x_1, \ldots, x_n] \) is functionally complete iff every function \( R^n \rightarrow R \) can be realized as a polynomial. In particular the ring \( \mathbb{Z}_4[x, y] \), where \( \mathbb{Z}_4 = \{0, 1, 2, 3\} \) is an example of a functionally incomplete ring.
Basically this is the ring where there is an exact match between the set of functions \( (\mathbb{F}_q^n \rightarrow \mathbb{F}_q) \) and the set of polynomials, but see [2] for details.

Let \( \mathcal{A}(x) = \langle a_1(x), \ldots, a_m(x) \rangle \) be an ideal we can then compute a principal representation for this ideal through:

\[
\mathcal{A}(x) = \langle 1 - (1 - a_1(x)^{q-1}) \cdots (1 - a_m(x)^{q-1}) \rangle
\]

Let \( \mathcal{A}(x) = \langle a(x) \rangle, \mathcal{B}(x) = \langle b(x) \rangle \) and \( \mathcal{C}(x,y) = \langle c(x,y) \rangle \) be ideal in appropriate rings.

We can now define an algebraic language:

\[
\mathcal{A}(x) \wedge \mathcal{B}(x) = \langle 1 - (1 - a(x)^{q-1})(1 - b(x)^{q-1}) \rangle
\]

\[
\mathcal{A}(x) \vee \mathcal{B}(x) = \langle a(x)b(x) \rangle
\]

\[
\neg \mathcal{A}(x) = \langle 1 - a(x)^{q-1} \rangle
\]

\[
(\forall y). \mathcal{C}(x,y) = \bigwedge_{w \in \mathbb{F}_q} \mathcal{C}(x,w)
\]

\[
(\exists y). \mathcal{C}(x,y) = \bigvee_{w \in \mathbb{F}_q} \mathcal{C}(x,w)
\]

The purpose of this language is to simplify algorithm formulation and can be understood by checking the varieties\(^4\) of all the expressions\(^5\):

\[
V(\mathcal{A}(x) \wedge \mathcal{B}(x)) = V(\mathcal{A}(x)) \cap V(\mathcal{B}(x))
\]

\[
V(\mathcal{A}(x) \vee \mathcal{B}(x)) = V(\mathcal{A}(x)) \cup V(\mathcal{B}(x))
\]

\[
V(\neg \mathcal{A}(x)) = \mathbb{F}_q^n \setminus V(\mathcal{A}(x))
\]

\[
V((\forall y). \mathcal{C}(x,y)) = \bigcap_{w \in \mathbb{F}_q} V(\mathcal{C}(x,w))
\]

\[
V((\exists y). \mathcal{C}(x,y)) = \bigcup_{w \in \mathbb{F}_q} V(\mathcal{C}(x,w))
\]

Finally, given a polynomial dynamical system

\[
x^+ = f(x, u)
\]

we define

\[
\mathcal{F}(x, u, x^+) = \langle x_1^+ - f_1(x, u), \ldots, x_n^+ - f_n(x, u) \rangle
\]

which is the ideal corresponding to the state equations of our process model. This ideal basically captures the state evolution behavior in one step. The ideal \( \mathcal{F}(x, u, x^+) \) is one of the main building blocks when doing analysis and design.

### 3 Modeling

The main objective in this section is to obtain a polynomial dynamical system as a translation from several other modeling domains. This can of course be repeated for many more domains than the ones presented in this paper, but this is a sampling of these.

---

\(^4\) The variety of an ideal is the set of common zeros, i.e.

\[
V(\langle f_1(x), \ldots, f_m(x) \rangle) = \{ u \in \mathbb{F}_q^n : f_1(u) = \cdots = f_m(u) = 0 \}
\]

\(^5\) In computer science terms, this would be the semantics of the language.
3.1 Finite Automata

3.1.1 Basic Description

Finite automata (FA) come in many flavors, but basically they can all be thought of as a system:

\[
x(k + 1) = f(x(k), u(k)) \quad (2)
\]
\[
y(k) = g(x(k), u(k)) \quad (3)
\]

where \( f : X \times U \to X \) and \( g : X \times U \to Y \) and \( X, U, Y \) are all finite sets. These machines go by several special names such as Moore/Mealy automata or deterministic/nondeterministic finite automata, but these are all special cases of this class.

3.1.2 Mapping to PDS

To obtain a representation of a FA we need to encode each of the sets \( X, U, Y \) in \( \mathbb{F}_q^n, \mathbb{F}_q^i, \mathbb{F}_q^o \) for some suitable choices of \( q \) and \( n, m, o \). Given \( p \) the minimal choices are given by:

\[
n = \lceil \log_q |X| \rceil, \quad i = \lceil \log_q |U| \rceil, \quad o = \lceil \log_q |Y| \rceil
\]

Suppose that we also have the encodings\(^6\)

\[
\phi_X : X \to \mathbb{F}_q^n, \quad \phi_U : U \to \mathbb{F}_q^i, \quad \phi_Y : Y \to \mathbb{F}_q^o
\]

By using this encoding we can obtain functions:

\[
f_p : \mathbb{F}_q^n \times \mathbb{F}_q^i \to \mathbb{F}_q^n, \quad g_p : \mathbb{F}_q^n \times \mathbb{F}_q^i \to \mathbb{F}_q^o
\]

through the equations:

\[
f_p(\phi_X(x), \phi_U(u)) = \phi_X(f(x, u)), \quad \forall x \in X, \forall u \in U \quad (4)
\]
\[
g_p(\phi_X(x), \phi_U(u)) = \phi_Y(g(x, u)), \quad \forall x \in X, \forall u \in U \quad (5)
\]

Using the Lagrange interpolation of section 2 we immediately obtain polynomial representations of \( f_p \) and \( g_p \).

**Example 3.1** Suppose we have the following finite automata:

```
\begin{array}{ccc}
(f, g) & a_0 & a_1 \\
\hline
s_0 & (s_0, b_0) & (s_1, b_0) \\
s_1 & (s_2, b_0) & (s_1, b_0) \\
s_2 & (s_0, b_0) & (s_3, b_1) \\
s_3 & (s_3, b_0) & (s_3, b_0) \\
\end{array}
```

where we have:

\[
X = \{s_0, s_1, s_2, s_3\}, \quad U = \{a_0, a_1\}, \quad Y = \{b_0, b_1\}
\]

\(^6\)Or embeddings
The state transition map \( f \) and the output map \( g \) are given directly in the graph (left) and in
the table (right). In the graph (left) one should read
\[ s_i \xrightarrow{a_k=b_i} s_j \]
as \( f(s_i, a_k) = s_j \) and \( g(s_i, a_k) = b_i \).
We can map this to a system over \( \mathbb{F}_2 \) by e.g. the following encodings:
\[ \begin{align*}
\phi_X & : s_0 \mapsto [0,0], \quad s_1 \mapsto [0,1], \quad s_2 \mapsto [1,0], \quad s_3 \mapsto [1,1] \\
\phi_U & : a_0 \mapsto 0, \quad a_1 \mapsto 1 \\
\phi_Y & : b_0 \mapsto 0, \quad b_1 \mapsto 1
\end{align*} \]
Using equations (4)- (5) and the Lagrange interpolating polynomial in equation (1) we get:
\[ \begin{align*}
f_p(x, u) &= \left[ ux_1 + x_2 + ux_2 \right] \\
g_p(x, u) &= ux_1 + ux_1 x_2
\end{align*} \]

### 3.2 Boolean Systems

#### 3.2.1 Basic Description

By Boolean system we denote a dynamic system that has a system description of the form:
\[ \begin{align*}
x(k+1) &= f(x(k), u(k)) \\
y(k) &= g(x(k), u(k))
\end{align*} \]
where \( f \) and \( g \) are vectors of Boolean expressions\(^7\). The set of Boolean expressions
over the variables \( x_1, x_2, \ldots, x_n \), denoted by \( \mathbb{B}[x] \), and recursively defined through:
\[ \begin{align*}
0, 1, x_1, x_2, \ldots, x_n & \in \mathbb{B}[x] \\
e_1, e_2 & \in \mathbb{B}[x] \Rightarrow (-e_1), (e_1 \land e_2), (e_1 \lor e_2) & \in \mathbb{B}[x]
\end{align*} \]

#### 3.2.2 Mapping to PDS

Given a Boolean system we can obtain an equivalent polynomial system by mapping
the set of Boolean expressions to their corresponding polynomial expressions and
preserving functional equality. This map \( \phi : \mathbb{B}[x] \to \mathbb{R}_2[x] \) is recursively given below:
\[ \begin{align*}
\phi(0) &= 0 \\
\phi(1) &= 1 \\
\phi(x_i) &= x_i, \quad i = 1,2,\ldots, n \\
\phi(-e) &= 1 - \phi(e) \\
\phi(e_1 \land e_2) &= \phi(e_1) \phi(e_2) \\
\phi(e_1 \lor e_2) &= \phi(e_1) + \phi(e_2) + \phi(e_1) \phi(e_2)
\end{align*} \]

\(^7\)For each \( n \in \mathbb{Z}_+ \) there exist a \( 2^n \)-valued Boolean algebra, these are however not functionally complete
except for the case \( n = 1 \) which the most important case anyway. Hence we restrict ourselves to the 2
valued Boolean algebra otherwise known as switching algebra.
Example 3.2 Suppose we have the simple system:

\[
\begin{bmatrix}
  x_1^+ \\
  x_2^+
\end{bmatrix}
= \begin{bmatrix}
  (x_1 \vee x_2) \land (\neg u) \\
  x_2 \land u
\end{bmatrix}
\]

\[
y = x_1 \land u
\]

applying \( \phi \) to the right hand sides yield:

\[
\begin{bmatrix}
  x_1^+ \\
  x_2^+
\end{bmatrix}
= \begin{bmatrix}
  (1 - u)(x_1 + x_2 + x_1 x_2) \\
  u + x_2 + u x_2
\end{bmatrix}
\]

\[
y = u x_1
\]

Hence we get an equivalent polynomial dynamical system.

3.3 Grafcet

3.3.1 Basic Description

Grafcet is an industry standard graph-oriented description language for dealing with sequential and parallel processes. In essence it is an industrial adaptation of Petri nets, see [1] for more on this. The descriptive power is no greater than a finite state machine with timers, but it is a nice representation of parallel activities which is not transparent in finite automata.

3.3.2 Mapping to PDS

Example 3.3 Suppose we have the following Grafcet graph:

![Grafcet Diagram]

We can then obtain an equivalent polynomial system as:

\[
x_1^+ = x_1 + (x_1 - 1)x_2 x_4 u_3 + x_1 u_1
\]

\[
x_2^+ = x_2 + (x_2 - 1)x_1 u_1 + x_2 x_4 (u_3 + u_4 + u_3 u_4)
\]

\[
x_3^+ = x_3 + (x_3 - 1)x_1 u_1 + x_3 u_2
\]

\[
x_4^+ = x_4 + (x_4 - 1)x_3 u_2 + x_4 (u_3 + u_4 + u_3 u_4)
\]

This system has the same time evolution behavior as the Grafcet graph.

4 Analysis

By analysis we mean verifying or validating system properties. A general sampling of such properties include: reachability, observability, IO-equivalence, deadlock and liveness. Some of the algorithms for computing these are given below:
4.1 Reachability

Because of the way we model discrete systems we usually have only a small fraction of the potentially reachable states actually reachable from our initial state.

**Forward Reachable States:** The set of states reachable from some initial set \( I_i(x) \) in \( k \) steps or less:

\[
R^+_0(I_i(x)) = I_i(x) \\
R^+_k(I_i(x)) = R^+_{k-1}(I_i(x)) \vee (\exists \bar{x})(\exists \bar{u}) \cdot F(\bar{x}, \bar{u}, x) \wedge R^+_{k-1}(I_i(x))
\]

**Backward Reachable States:** The set of states that can reach \( I_i(x) \) in \( k \) steps or less:

\[
R^-_0(I_i(x)) = I_i(x) \\
R^-_k(I_i(x)) = R^-_{k-1}(I_i(x)) \vee (\exists \bar{x})(\exists \bar{u}) \cdot F(x, \bar{u}, \bar{x}) \wedge R^-_{k-1}(I_i(x))
\]

**Forced Backward Reachable States:** The set of states states that has to reach \( I_i(x) \) in \( k \) steps or less:

\[
FR^-_0(I_i(x)) = I_i(x) \\
FR^-_k(I_i(x)) = FR^-_{k-1}(I_i(x)) \vee (\exists \bar{x})(\forall \bar{u}) \cdot F(x, \bar{u}, \bar{x}) \wedge FR^-_{k-1}(I_i(x))
\]

In particular \( R^+_k(I_i(x)) \), \( R^-_k(I_i(x)) \), and \( FR^-_k(I_i(x)) \) are all well defined and computed as a finite fixed point of their respective iterations. This is because the ring in question is Artin.

4.2 Deadlock/Liveness

A problem unique to discrete systems is the possibility of having stuck states.

**Deadlocked States:** States that cannot be left whatever the control action:

\[
DL(x) = (\forall u). F(x, u, x)
\]

**Liveness States:** States where you are guaranteed to be able to get some desirable set of states. Suppose \( D(x) \) are the desirable set of states then the set of live states \( L(D(x)) \) are:

\[
L(D(x)) = R^-_{\infty}(D(x))
\]

5 Design

By design we mean computing a control policy from a system description and control constraints.

**Given:** A system

\[
x^+ = x + f(x, u)w
\]

where \( w \) is our control signal.

**Sought:** A *supervisor* \( u = k(x, u) \) that guarantees that property \( p(x) \) will always hold.

Note:

\[
p(x) = \begin{cases} 0 & \text{Property holds} \\ 1 & \text{Property false} \end{cases}
\]
Solution: No extra constraints:
\[ k(x, u) = p(x + f(x, u)) \]

Suppose that \( p(x) \) is an old set of constraint, we may then complement this set with further constraints to obtain \( \bar{p}(x) \) in several ways e.g.

- We want our system to have the liveness property:
  \[ \bar{p}(x) = p(x) \land L(D(x)) \]

- We do not want to get into locked states, i.e. were the only control action is to block every command:
  \[ \bar{p}(x) = p(x) \land R_{\infty}(p(x)) \]

All of the controls are also maximally permissive in the sense that as many trajectories as possible will be allowed in the system. This approach can then be extended to more complicated side constraints such as:

- Constraints on sequences of states
- Constraints on sequences of inputs

6 Implementation

Implementation is essentially the inverse of modeling, but we can also derive equivalent representations in less structured environments. Given a PDS

\[
\begin{align*}
x^+ &= f(x, u) \\
y &= g(x, u)
\end{align*}
\]

we can then define a mapping

\[ \psi : \text{PDS} \rightarrow \text{ID}_i \]

that maps a PDS to an equivalent representation in some implementation domain \( \text{ID}_i \). As this quite similar to the modeling section only on brief example will be given.

6.1 High-level programming language: C

Suppose we have only polynomials in \( \mathbb{F}_p[x, u] \) for some prime \( p \). We can then define \( \psi \) recursively:\n
\[
\begin{align*}
\psi(i) &= i, & i &= 1, 2, \ldots, p - 1 \\
\psi(x_i) &= x_i, & i &= 1, 2, \ldots, n \\
\psi(u_j) &= u_j, & j &= 1, 2, \ldots, m \\
\psi(e_1 e_2) &= (\psi(e_1) * \psi(e_2)) \% p \\
\psi(e_1 + e_2) &= (\psi(e_1) + \psi(e_2)) \% p
\end{align*}
\]

\[ ^8 \text{This translation is of course incomplete in that we would need to package PDSs as one object and we would also need initializations in the C code. These extras are however only extra baggage in this exposition.} \]
Example 6.1 Suppose we have the PDS in \( \mathbb{F}_5[x_1, x_2, u_1] \)

\[
\begin{bmatrix}
  x_1^+ \\
  x_2^+
\end{bmatrix} =
\begin{bmatrix}
  x_1 + 4x_2u_1 \\
  x_2 - 3u_1
\end{bmatrix}
\]

\[ y = -2x_1 \]

Applying \( \psi \) to the right hand sides of these equations we get:

\[
\begin{align*}
  x_1^+ &= (x_1 + ((4 * x_2) \% 5 * u_1) \% 5) \% 5 \\
  x_2^+ &= (x_2 - (3 * u_1) \% 5) \% 5 \\
  y &= (-2 * x_1) \% 5
\end{align*}
\]

Which should be the appropriate C-code fragment that implements the same state transition and output equations.

6.2 Other Implementation Domains

We could perform the same operation on for many other domains such as e.g.

- Low-level programming languages, e.g. assembler or PLC-code.
- Hardware implementations, e.g. VLSI descriptions such as VHDL-code.
- Relay, pneumatic, hydraulic implementations, this is particularly common in older control systems.

One might note that the implementation part essentially corresponds to the code generation part of a compiler and thus has the same advantages. I.e. we can get provably correct implementations, even when they are rather voluminous.

7 Conclusion

This has been a brief review of how one might use state equations in polynomials over finite fields to model discrete (dynamic) systems. In particular general symbolic (or compilation type) techniques was used in the modeling and implementation phases and algebraic techniques for the analysis and design phases.

Some of the appealing parts of this theory is that it specializes to linear systems and generalizes to nonlinear (polynomial) dynamical systems, i.e. systems of the form:

\[
\begin{align*}
  \dot{x} &= Ax + Bu \\
  y &= Cx + Du
\end{align*}
\]

and

\[
\begin{align*}
  \dot{x} &= f(x, u) \\
  y &= g(x, u)
\end{align*}
\]

This has however not been demonstrated in these pages.
References


