Math 4500 worksheet: Reverse engineering of polynomial dynamical systems

Goal. Find all models $F = (f_1, \ldots, f_n)$: that fit the partial data:

\[
\begin{align*}
\text{Input states: } & s_1, \ldots, s_m \in \mathbb{F}^n \\
\text{Output states: } & t_1, \ldots, t_m \in \mathbb{F}^n
\end{align*}
\]

with $F(s_i) = t_i$.

Here, each $f_i : \mathbb{F}_2^n \rightarrow \mathbb{F}_2$ can be assumed to be a Boolean polynomial, and updating each of these functions synchronously yields the finite dynamical systems (FDS) map $F : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^n$. That is, the equation $F(s_i) = t_i$ means

\[F(s_i) = (f_1(s_i), f_2(s_i), \ldots, f_n(s_i)) = (t_1, t_2, \ldots, t_m) = t_i.
\]

The set of all solutions (models) is called the model space:

\[F_1 \times \cdots \times F_n = \{(f_1, \ldots, f_n) \mid f_j(s_i) = t_{ij}\}.
\]

To find all solutions, we find each $F_j$ separately. Note that $F_j$ is the set of all local functions at node $j$ that fit the data:

\[F_j = \{f_j : f_j(s_1) = t_{ij}, \ldots, f_j(s_m) = t_{im}\}.
\]

To find $F_j$, we use that fact that it can be written as

\[F_j = f_j + I = \{f_j + h : h \in I\},
\]

where $f_j$ is any particular function in $F_j$, and $I$ is the set of polynomials that vanish on the data:

\[I = \{h : h(s_i) = 0 \text{ for all } i = 1, \ldots, m\}.
\]

Thus, to find $F_j$, we need to do two things:

1. Find the ideal $I$:
2. Find any polynomial $f_j$ that fits the data.

1. Finding $I$: Define $I(s_i)$ to be the set of polynomials that vanish on $s_i$:

\[
I(s_i) = \{\text{all polynomials } h_i \text{ such that } h_i(s_i) = 0\}
\]

\[
= \{(x_1 - s_{i1})g_1(x) + (x_2 - s_{i2})g_2(x) + \cdots (x_m - s_{im})g_m(x)\}
\]

\[
= (x_1 - s_{i1}, x_2 - s_{i2}, \ldots, x_m - s_{im})
\]

Clearly, the set $I$ of polynomials that vanish on all $s_i$ (for $i = 1, \ldots, m$) is simply

\[I = \bigcap_{i=1}^{m} I(s_i).
\]

2. Finding $f_j$: This method uses the "Chinese Remainder Theorem" for rings, though this is "hidden" in the background.

For each data point $s_i; i = 1, \ldots, m$, we’ll construct an “$r$-polynomial” that has the following property:

\[
(1) \quad r_i(x) = \begin{cases} 
1 & x = s_i \\
0 & x \neq s_i
\end{cases}
\]

Once we have the $r$-polynomials, then the polynomial $f_j(x)$ we seek will be

\[f_j(x) = t_{1j}r_1(x) + t_{2j}r_2(x) + \cdots + t_{mj}r_m(x).
\]

So, how do we find these $r$-polynomials? There are likely many such polynomials that work, but here’s a sure-fire way to construct them:

\[r_i(x) = \prod_{k=1, k \neq i}^{m} b_{ik}(x),
\]
where
\[ b_{ik}(x) = (s_{i\ell} - s_{k\ell})^{p-2}(x_{\ell} - s_{k\ell}) \]
and \( \ell \) is the first coordinate in which \( s_i \) and \( s_k \) differ.

This looks horrible! (But it’s not too bad.) Let’s try it. Consider the following time series in a 3-node system over \( \mathbb{Z}_5 \):

\[
\begin{align*}
    s_1 &= (2, 0, 0) \\
    s_2 &= (4, 3, 1) = t_1 \\
    s_3 &= (3, 1, 4) = t_2 \\
    (0, 4, 3) &= t_3
\end{align*}
\]

For reference, here are the input vectors \( s_i \) and output vectors \( t_i \):

\[
\begin{align*}
    s_1 &= (s_{11}, s_{12}, s_{13}) = (2, 0, 0), & t_1 &= (t_{11}, t_{12}, t_{13}) = (4, 3, 1), \\
    s_2 &= (s_{21}, s_{22}, s_{23}) = (4, 3, 1), & t_2 &= (t_{21}, t_{22}, t_{23}) = (3, 1, 4), \\
    s_3 &= (s_{31}, s_{32}, s_{33}) = (3, 1, 4), & t_3 &= (t_{31}, t_{32}, t_{33}) = (0, 4, 3).
\end{align*}
\]

Note that \( s_1 \) differs from \( s_2 \) and \( s_3 \) in the \( \ell = 1 \) coordinate, so this \( \ell \) will work for each of \( f_1 \), \( f_2 \), and \( f_3 \).

Let’s compute the first \( r \)-polynomial, which is:

\[
r_1(x) = b_{12}(x)b_{13}(x).
\]

Since we are working in \( \mathbb{Z}_5 \), we are taking the remainder of everything modulo 5. Particularly useful identities are: \( 0 = 5, -1 = 4, -2 = 3, -3 = 2, \) and \( -4 = 1 \). Using our formulas for \( b_{ij}(x) \), we compute:

\[
\begin{align*}
    b_{12}(x) &= (s_{11} - s_{21})^3(x_1 - s_{21}) = (2 - 4)^3(x_1 - 4) = -8(x_1 + 1) = 2x_1 + 2 \\
    b_{13}(x) &= (s_{11} - s_{31})^3(x_1 - s_{31}) = (2 - 3)^3(x_1 - 3) = -x_1 + 3 = 4x_1 + 3.
\end{align*}
\]

Therefore, the first \( r \)-polynomial is

\[
r_1(x) = b_{12}(x)b_{13}(x) = (2x_1 + 2)(4x_1 + 3) = 8x_1^2 + 14x_1 + 6 = 3x_1^2 + 4x_1 + 1.
\]

Your turn! Compute \( r_2(x) \) and \( r_3(x) \), and then use these to find \( f_1(x) \), \( f_2(x) \), and \( f_3(x) \). Note that you will need to compute the polynomials \( b_{21}(x) \), \( b_{23}(x) \), \( b_{31}(x) \), and \( b_{32}(x) \).

Before proceeding, check to make sure that each of these polynomials fits the data. In other words, for each \( j = 1, 2, 3 \), verify (do this now!) that

\[
    f_j(s_1) = f_j(2, 0, 0) = s_{1j}, \quad f_j(s_2) = f_j(4, 3, 1) = s_{2j}, \quad f_j(s_3) = f_j(3, 1, 4) = s_{3j}.
\]

To explore why this works, go back a step further, and verify that each \( r \)-polynomial satisfies the equation from (1).

Now that we have found \( f_1 \), \( f_2 \), and \( f_3 \), our “particular” solution that fits the data is \( f = (f_1, f_2, f_3) \), and our “general solution” (the model space) is the set

\[
    F_1 \times \cdots \times F_n = f + (I \times \cdots \times I) = (f_1 + I, \ldots, f_n + I).
\]
Further exploration. In this project, we will investigate a simple Boolean FDS, explore its phase space, and attempt to reverse engineer it given partial data.

Consider the following polynomial dynamical system:

\[ f_1(x_1, x_2, x_3) = \text{AND}(x_1, x_2) = x_1x_2 \]
\[ f_2(x_1, x_2, x_3) = \text{AND}(x_1, x_2, x_3) = x_1x_2x_3 \]
\[ f_3(x_1, x_2, x_3) = \text{AND}(x_1, x_2) = x_1x_2. \]

This is called an \textit{AND-network} because the Boolean functions can be written as logical AND functions.

Go to the Analysis of Dynamic Algebraic Models (ADAM) toolbox, at \texttt{http://adam.plantsimlab.org/}. Enter the functions above into the “Model Input” box and click the “Analyze” button. The state space should look like this:

\[
\begin{align*}
001 & \rightarrow 010 & 011 & \rightarrow 100 & 101 \\
& \rightarrow 000 & & \rightarrow 111 \\
& & & &
\end{align*}
\]

This graph literally encodes the entire function \( F = (f_1, f_2, f_3): \mathbb{F}_2^3 \rightarrow \mathbb{F}_2^3 \).

Let’s try to reverse engineer this network given partial data. In particular, let’s suppose that all we know is the following “piece” of the state space (elephant):

\[ s_1 = (1, 1, 0) \]
\[ s_2 = (1, 0, 1) = t_1 \]
\[ s_3 = (0, 0, 0) = t_2 \]
\[ (0, 0, 0) = t_3 \]

Follow the steps of the above example to find all FDS maps that fit this data. Naturally, you could “cheat” and use the OR functions above for \( f_1, f_2, \) and \( f_3 \), but try the \( r \)-polynomial method. Do you get the same particular solution?