Biological Regulatory Inferential Additive Network (BRIAN) Specification

M Batsaikhan, Luke Young-Xu

Introduction

The human body is a carefully engineered biological system, consisting of approximately 30 trillion cells — all which must perform a diverse and dynamic set of functions to support daily living. But how do all these cells know which role to perform and successfully carry out their functions? The answer lies in biological regular networks, or BRNs, consisting of interconnected cellular components, such as genes, proteins, and metabolites, that interact through regulatory mechanisms to control the expression of genes, cellular activities, and organismal functions.

With a single cell in humans often containing over 10,000 different types of protein and over 20,000 genes, constructing a BRN becomes a graph-navigating problem which can quickly evolve to be a computational nightmare. This is where Biological Regulatory Inferential Additive Network (BRIAN) language hopes to offer improvements in the workflow of a lab scientist by automatically constructing possible biological regulatory networks from the user's input of simple existing relationships between cellular components discovered through experimental work. The language would be capable of translating this experimentally established relationship data into an adjacency matrix representation of a BRN, each row/column representing a specific component and indices representing relationships. Ultimately, the BRIAN language could use this matrix to find new relationships between components as well as test the feasibility of relationships between specified components during evaluation and present these results back to the user.

As a programming language, BRIAN's strength lies in its ease of use: programs from a text file consist of a list of observed relationships between cellular components, which does not require the user to be acquainted with any level of coding. In this sense programs are extremely modular in which new experimental results can simply be appended to an already existing program. The language itself can also identify logical conflicts in the relations provided in the program through its evaluation of the adjacency matrix (described in further detail below).

Design Principles

On the technical side, BRIAN hopes to be a highly scalable and flexible language in the following manner:

- Since relationships between cellular components are ultimately stored in an adjacency matrix, adding new relationships between existing components has a very low storage and computational cost. Additionally, adding new components is performed by simply expanding the existing matrix. As only a single relationship between two components can exist (i.e. unknown, activates, or inhibits), a matrix of ints is sufficient to represent the BRN.
- BRIAN accommodates all existing proteins, genes, phenotypes (as well as an [other] category for components such as small RNAs or metabolites) and allows interactions between all these types freely. The language also does not strictly define the type of relationship between two cellular nodes and can consider the downstream effects of activation and inhibition between any two components.

Aesthetically, BRIAN aims to be a clear and expressive language. Since BRIAN is a strongly-typed language, it would help users avoid confusion of variables. For example, insulin is both a protein and a gene, using BRIAN, the user can be sure which variable is under consideration at all times. However, in the user's input this distinction can easily be made by following the common biological syntax of gene names starting with a lowercase letter and protein names with an uppercase.

Examples

Sample Program #1:
Input:

```
Run with: dotnet run txt/Example1.txt (from /lang)
or: dotnet run --project lang/lang.fsproj lang/txt/Example1.txt (from /code)
    Example1.txt file:
    "Protein_A activates gene_b
    gene_b activates Protein_A
    gene_b activates Protein_C
    Protein_C inhibits Protein_D
    Protein_D inhibits Protein_D"
```

Output (in addition to graphical output):

```
Successful parse.
New found relations:
Protein_A activates Protein_A
Protein_A activates Protein_C
Protein_A inhibits Protein_D
gene_b activates gene_b
gene_b inhibits Protein_D
```

Sample Program #2:
Input:

```
dotnet run txt/Example2.txt
    Example2.txt file:
    "gene1 inhibits Protein1
    Protein2 activates *phenotype1
    Protein2 inhibits gene3
    *phenotype2 activates gene3
    #unknown_siRNA inhibits gene4-a"
```

Output (in addition to graphical output):

```
Successful parse.
New found relations:
No new relations found
```

Sample Program #3:
Input:

```
dotnet run txt/Example3.txt
    Example3.txt file:
    "A activates B
    B inhibits C
    A activates C"
```

Output (in addition to graphical output):

```
Successful parse.
New found relations:
Contradiction Found At: A activates C
```

Sample Program #4:
Input:

```
dotnet run txt/Example4.txt
    Example4.txt file:
    "A activates b
    C inhibits b
    C ? A
    A inhibits d"
```

Output (in addition to graphical output):

```
Successful parse
New found relations:
No new relations found
----------------
Expected new relations for C activates A
Contradiction Found At: C activates b

Expected new relations for C inhibits A
C activates d

Conclusion: C may inhibit A or have no effect
```

Language Concepts

A typical user is expected to be familiar with cellular components, such as proteins, genes and phenotypes, which serve as primitives for BRIAN. When the user inputs a text file describing observed relationships between components, BRIAN constructs a fitting BRN and infers new relationships between the variables. In essence, BRIAN's evaluator is designed to consider all possible combinations between the identified primitives and offer any new found ones as a result.

The combining forms of cellular components mainly include more abstract interactions (e.g. activation, repression), or if these are not known, the user can request a query to understand what is the interaction between their desired pair of components. Upon receiving a correct file of relationship descriptions from the user, BRIAN parses and stores all relationships between cellular components in a list of relations.

Syntax

$$
\begin{aligned}
\langle gene \rangle &::= \langle \alpha \in \{a...z\} \rangle \langle \alpha \in \{a...z\} \cup \{A...Z\} \cup \{\mathbb{Z}\} \cup \{\_, -\} \rangle^* \\
\langle protein \rangle &::= \langle \alpha \in \{A...Z\} \rangle \langle \alpha \in \{a...z\} \cup \{A...Z\} \cup \{\mathbb{Z}\} \cup \{\_, -\} \rangle^* \\
\langle phenotype \rangle &::= * \langle \alpha \in \{a...z\} \cup \{A...Z\} \cup \{\mathbb{Z}\} \cup \{\_, -\} \rangle^+ \\
\langle other \rangle &::= \# \langle \alpha \in \{a...z\} \cup \{A...Z\} \cup \{\mathbb{Z}\} \cup \{\_, -\} \rangle^+ \\
\langle variable \rangle &::= \langle gene \rangle \\
&\quad | \langle protein \rangle \\
&\quad | \langle phenotype \rangle \\
&\quad | \langle other \rangle \\
\langle relation \rangle &::= \langle variable \rangle \text{␣activates␣} \langle variable \rangle \\
&\quad | \langle variable \rangle \text{␣inhibits␣} \langle variable \rangle \\
&\quad | \langle variable \rangle \text{ ? } \langle variable \rangle \\
\langle sequence \rangle &::= \langle relationship \rangle^+
\end{aligned}
$$

Semantics

Syntax	Abstract Syntax	Meaning
<gene>	Variable of string	Gene is a primitive of type string. Gene is a subtype of the primary Variable primitive and must start with a lowercase letter.
<protein>	Variable of string	Protein is a primitive of type string. Protein is a subtype of the primary Variable primitive and must start with a capital letter.
<phenotype>	Variable of string	Phenotype is a primitive of type string. Phenotype is a subtype of the primary Variable primitive and must start with a "∗" character.
<other>	Variable of string	Other is a primitive of type string. Other is a subtype of the primary Variable primitive and must start with a "#" character.
<v1>␣ activates ␣<v2>	Activation of Variable * Variable	Activation indicates the relationship between two given variables, specifically v1 activates v2. Activation is a subtype of the primary Relation combining form.
<v1>␣ inhibits ␣<v2>	Inhibition of Variable * Variable	Inhibition indicates the relationship between two given variables, specifically v1 inhibits v2. Inhibition is a subtype of the primary Relation combining form.
<v1> ? <v2>	Query of Variable * Variable	Query indicates the user's request to find out about the relationship between two given variables, v1 and v2. Query is a subtype of the primary Relation combining form.
<Relation>$^{+}$	List of Relation	Sequence contains a list of all Relations present in a given BRN. Sequence must take at least one Relation to be valid.

About Evaluation

Aside from the program which is contained in a .txt file, no additional input is provided.

To understand how evaluation works for BRIAN, let's consider our first example. When we call

```
eval test1.txt
```

BRIAN parses the text file and classifies the cellular components into their appropriate primitive types.

```
Example1.txt file:
    "Protein_A activates gene_b
    gene_b activates Protein_A
    gene_b activates Protein_C
```

```
Protein_C inhibits Protein_D
Protein_D inhibits Protein_D"
```

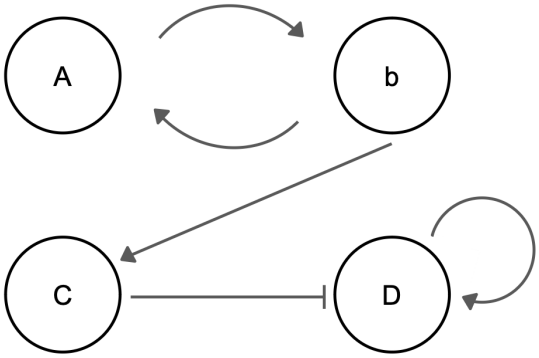
After parsing, BRIAN compiles a list of all unique cellular components, which in our case we have four:

```
[Protein "Protein_A"; Gene "gene_b"; Protein "Protein_C"; Protein "Protein_D"]
```

Then, a matrix (call it P) is constructed from all the parsed relationships:

$$P = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 2 \end{pmatrix}$$

, where 1 signifies activation, 2 signifies inhibition and 0 means no relationship. The P matrix successfully captures the following BRN graph:



Using the P matrix, we are now ready to search for new relationships using matrix multiplication. The next step is using matrix multiplication to find new relationships. This operates under the biological logic of:

```
(A activates B, B activates C) implies A activates C
    In the matrix: 1 * 1 = 1
(A activates B, B inhibits C) implies A inhibits C
    In the matrix: 1 * 2 = 2
(A inhibits B, B activates C) implies A inhibits C
    In the matrix: 2 * 1 = 2
(A inhibits B, B inhibits C) implies A activates C
    In the matrix: 2 * 2 = 4, 4 is then converted to 1 (activation)
```

Since the underlying relationship between cellular components does not change, we may model this traversal problem as a quasi-Markov Chain process. The transition matrix for our MC model would contain information on existing relationships, which we may probabilistically interpret as a sure event that state i would go to state j. So, at each step (of multiplying P to itself), we capture the transfer of relationship ij to relationship jk in our next step, which is true to how BRN relationships function.

However, to avoid the summation of multiple values, each row and column of a matrix can only contain one value before being multiplied by itself. Following this logic, the following matrix derivations are made by zeroing certain values:

$$\text{Matrix1} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\text{Matrix2} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 \end{pmatrix}$$

$$\text{Matrix3} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\text{Matrix4} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 \end{pmatrix}$$

And the results of multiplication:

$$\text{Matrix1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\text{Matrix2} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4 \end{pmatrix}$$

$$\text{Matrix3} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\text{Matrix4} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4 \end{pmatrix}$$

The new values from these matrices can be added back to the original (inhibition (2) on diagonal values (representing self-inhibition) result in a 4 in the same place after squaring; in this case we do not consider this a contradiction).

$$P' = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 2 \\ 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 2 \end{pmatrix}$$

From this new matrix, the following relationships are found:

```
New found relations:
Protein_A activates Protein_A
Protein_A activates Protein_C
gene_b activates gene_b
gene_b inhibits Protein_D
```

Using this new graph, the same matrix derivation and multiplication process can be repeated to find new relationships, up to $log_2(n)$ times where n is the number of unique elements. This results in the final matrix:

$$P' = \begin{pmatrix} 1 & 1 & 1 & 2 \\ 1 & 1 & 1 & 2 \\ 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 2 \end{pmatrix}$$

From the final matrix, the following relationships are found:

```
New found relations:
Protein_A activates Protein_A
Protein_A activates Protein_C
Protein_A inhibits Protein_D
gene_b activates gene_b
gene_b inhibits Protein_D
```

Remaining Work

Further development for BRIAN as a programming language would look as following:

- Addition of other relationships: Currently, BRIAN covers relatively abstract relations, i.e. activation and inhibition. But sometimes experimental data can only contain biochemical modifications (e.g., phosphorylation, ubiquitination, acetylation, methylation) from which a more abstract relationship can be derived through comparison of multiple downstream outcomes — in the future, this would be a challenging but a helpful feature to implement.
- Graphical representation of variables and relationships: The future version of BRIAN should be able to offer a visual display of the network given by the user. Given enough time to explore and experiment with F#'s graphical libraries, this feature is sure to give BRIAN a nice visual flair and make the relationship predictions more intuitive to the user.

Both of these features were included in the original draft. However, during development of BRIAN the authors felt developing a query feature (which was not in the original draft) would be a more valuable addition. Development of this feature was therefore prioritized with the remaining time.