Aeon Manual

1 What does Aeon do **1**

2 Getting Aeon running **1**
2.1 Running pre-compiled binaries 1
2.2 Building from source 2
2.3 Startup 2

3 Model description **2**

4 Graphical user interface **3**
4.1 Model editor 3
4.2 Compute engine panel and analysis control 4
4.3 Model panel 4
4.4 Import and export 5
4.5 Model format and update function syntax 5
4.6 Expected output 5

January 2020
https://sybila.fi.muni.cz/

1 What does **Aeon** do

As a member of BioDivine suite, **Aeon** (Analysis & Exploration of Networks) is a parallel tool for creating, editing, and analysing parametrised Boolean network models; specifically, it provides means of analysis of model's bifurcations—qualitative changes in behaviour, which are originating in, typically small, changes of parameters. Details on the underlying theory can be found in [1].

2 Getting **Aeon** running

The tool implementation consists of two components: the *compute engine*, and the web-based, user-facing GUI application (the *client*). A typical use of the tool requires a local installation of the compute engine, which is accessed from the client. The client can be also stored locally, or hosted remotely, with no change in functionality between the two cases. The online version of the client is accessible from `https://biodivine.fi.muni.cz/aeon`; for offline use, the client application can be downloaded from `https://github.com/sybila/biodivine-aeon-client`. The client application can be used to create and edit parametric models without the compute engine being installed. The client does not connect to the internet. The engine can be obtained as a pre-compiled executable (for all major desktop platforms) or as a Rust source code. Because the client is accessing the engine via `http` connection in which the engine acts as a server, it is possible to access the engine remotely, assuming sufficient network configuration—this is useful when the computation is delegated to a suitable powerful hardware.

Client	
online access	`biodivine.fi.muni.cz/aeon/`
offline download	`github.com/sybila/biodivine-aeon-client/`
Engine	
source, executables	`github.com/sybila/biodivine-aeon-server/releases/`

2.1 Running pre-compiled binaries

Pre-compiled executables for multiple platforms are available at `https://github.com/sybila/biodivine-aeon-server/releases`. After downloading and running the corresponding file, the engine will be accessible from the client application and ready for use. The relevant executables can be also downloaded through the links listed in the client application under the *compute engine* panel, described in Section 4.2. Preparing the executable on Linux:

```
$ unzip aeon-compute-engine-linux.zip && chmod +x aeon-compute-engine
```

2.2 Building from source

The engine source code, written in the Rust programming language and licensed under the MIT License, is freely available for download. To compile the software, one needs to install the Rust toolchain – `rustup`, and download the actual source code.

- `rustup` – `https://www.rust-lang.org/tools/install`
- *Compute engine* – `https://github.com/sybila/biodivine-aeon-server`

When the Rust toolchain is installed following the instructions on its website, the engine can be compiled using the `$ cargo +nightly build` command in the root of the directory. After successful compilation, running `$ cargo run` will start up the engine.

2.3 Startup

By default, the engine uses the localhost address and the port 8000 to run on. If the port is available, the engine will report the address and the port number on which it is running.

```
Rocket has launched from http://localhost:8000
```

The default server address and port will work in most cases; however, should the automatic assignment fail, manual configuration is possible through the environment variables `AEON_ADDR` and `AEON_PORT`. For example, setting a different port number would look like this (on Linux/Mac):

```
$ export AEON_PORT=3485
```

After the engine has been properly configured and it's up and running, the client will automatically establish a connection on its startup. If it is already running in the web browser, clicking on the *Connect* button under the *compute engine* panel will link the two, and the tool will be ready to be used.

3 Model description

The Aeon does use parametrised Boolean network models. A Boolean network can be seen as a directed graph

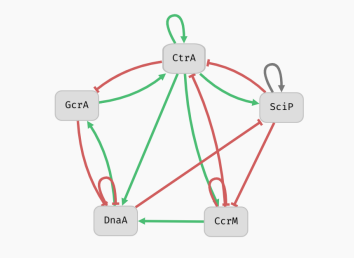

Figure 1: A simple Boolean network as displayed in Aeon– model adopted from [3].

4 Graphical user interface

The client, running in a web browser, provides a user-friendly graphical interface, that enables one to create, edit, and visualise Boolean network models on the one hand, and allows for interfacing with the engine, supervising the computation, and visualisation of the results on the other. Models are drawn and displayed on the large editor canvas. At any time, pressing and holding the H key will display the help window.

4.1 Model editor

Adding a variable Double-clicking the empty space of the editor canvas will create a new variable, which is automatically assigned a fresh name.

Renaming a variable Clicking on a variable will bring up the variable menu (Figure 4). Choosing the *Edit name* option (hotkey E) opens the *Model* panel, in which the variable can be renamed.

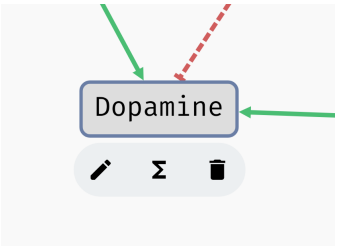

Figure 2: Node menu visible on a clicked variable

Editing variable's update function The same menu (Figure 4) enables one to change the update function of a given variable by clicking the Σ *Edit update function* button, or using the hotkey F. An update function has to comply with the syntax defined in the Section 4.5. If the engine is connected, a syntax check will occur on edit.

Creating a new regulation To create a new regulation between existing variables, click and drag the + plus icon from the regulator onto the regulated variable. The new, grey-coloured arrow signifies a regulation with an unspecified effect on the regulated variable. This effect can be made explicit by editing the update function of that variable, or by specifying the regulation kind.

Editing regulation kind and observability As described in Section 3, regulation can be either *inhibiting*, *activating*, or left unspecified (*monotonicity off*). Besides the regulation kind, each of the regulations can be either *observable* or *non-observable*. Editing these options is done via the *Edge menu*, invoked by simply clicking on a particular edge. Apart from the button, the regulation kind can be toggled using hotkey `M`, its observability with hotkey `O`. The regulators of a variable can be used as arguments in the parametric update function of that variable.

Removing variables and regulations Removing a variable is again done using the node context menu (Figure 4), or using the *backspace* hotkey; similarly for the regulations using the edge menu.

4.2 Compute engine panel and analysis control

This panel is used to control the engine and to observe its state. Particularly, it is responsible for establishing a link between the engine and the client, presenting the state of the computation to the user, or launching and cancelling the computational tasks. The address output by the engine executable has to match the address that is set in this panel.

During an ongoing computation, the tool provides partial results of the bifurcation function, should the user choose to inspect them (see Figures 4.2 and 4.6).

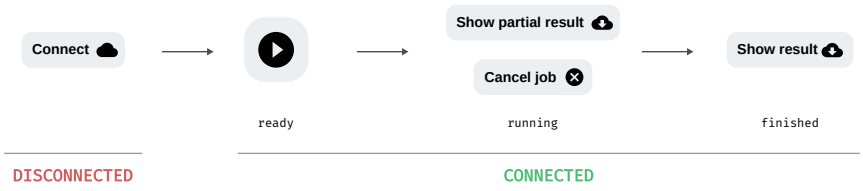

Figure 3: A state diagram of the engine control panel

4.3 Model panel

The model panel provides means to create and edit the model as well. This panel shows the model's basic properties, such as the number of variables and regulations, the parameter state space size, and the state transition graph size. Employing this panel, models can be named and described as well. Because the complexity of the underlying algorithm is doubly exponential in the number of regulators of a given node, the panel shows the maximal indegree of the model's variables to indicate an approximate duration of the potential computation.

This panel contains loaded model in a form of list, where each cell contains one variable and a list of its regulators.

When a variable is hovered over with a cursor, the correspondence between the list view and the graphical view is presented visually as an outline.

4.4 Import and export

Boolean network models can be imported and exported using the *Import/Export* panel. The available options include the in-house `.aeon` format described below, and the standard SBML format[2]. Import and export of SBML is available only when the client is connected to the engine. If the target web browser has enabled HTML5 local storage features, the last-used model can also be restored from the previous browser sessions using this option.

4.5 Model format and update function syntax

Models are in this format:

Aeon file	::=	*Regulation*
	|	*Update fn decl*
	|	*Meta*
	|	*Aeon file*\n*Aeon file*
Update fn decl	::=	`$` *Name* `:` *Update fn*
Meta	::=	`#` *Key* `:` *Value*
Regulation	::=	*Name* ␣ *Arrow* ␣ *Name*
Arrow	::=	*Kind* | *Kind*`?`
Kind	::=	`->` | `-|` | `-?`
Update fn	::=	`true` | `false` | *Name* | *Uninterpreted fn*
	|	`!`*Update fn*
	|	`(`*Update fn* *Op* *Update fn*`)`
Op	::=	`&` | `|` | `=>` | `<=>`
Uninterpreted fn	::=	*Name*`(`*Parameters*`)`
Parameters	::=	*Name* | *Parameters*`,` *Parameters*

Update function syntax
Only names of the can be used as function parameters.

4.6 Expected output

Examination of the bifurcation function Result

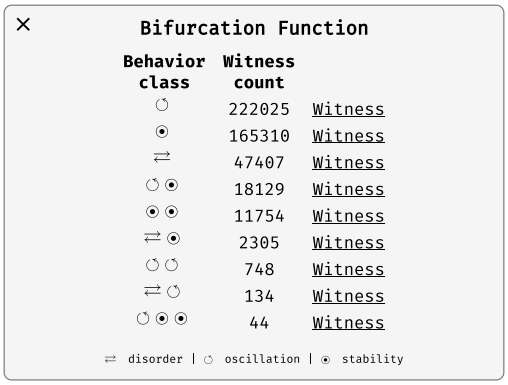

Figure 4: An example of a result, representing a bifurcation function

Witness inspection Partition of the parameter space of parametrizations exhibiting the same behaviour

References

[1] Nikola Beneš et al. "Formal Analysis of Qualitative Long-Term Behaviour in Parametrised Boolean Networks". In: *Formal Methods and Software Engineering (ICFEM 2019)*. Springer, 2019, pp. 353–369.

[2] Claudine Chaouiya et al. "SBML qualitative models: a model representation format and infrastructure to foster interactions between qualitative modelling formalisms and tools". In: *BMC systems biology* 7.1 (2013), p. 135.

[3] Ismael Sánchez-Osorio, Carlos A. Hernández-Martínez, and Agustino Martínez-Antonio. "Modeling Asymmetric Cell Division in Caulobacter crescentus Using a Boolean Logic Approach". In: *Asymmetric Cell Division in Development, Differentiation and Cancer*. Ed. by Jean-Pierre Tassan and Jacek Z. Kubiak. Cham: Springer International Publishing, 2017, pp. 1–21.