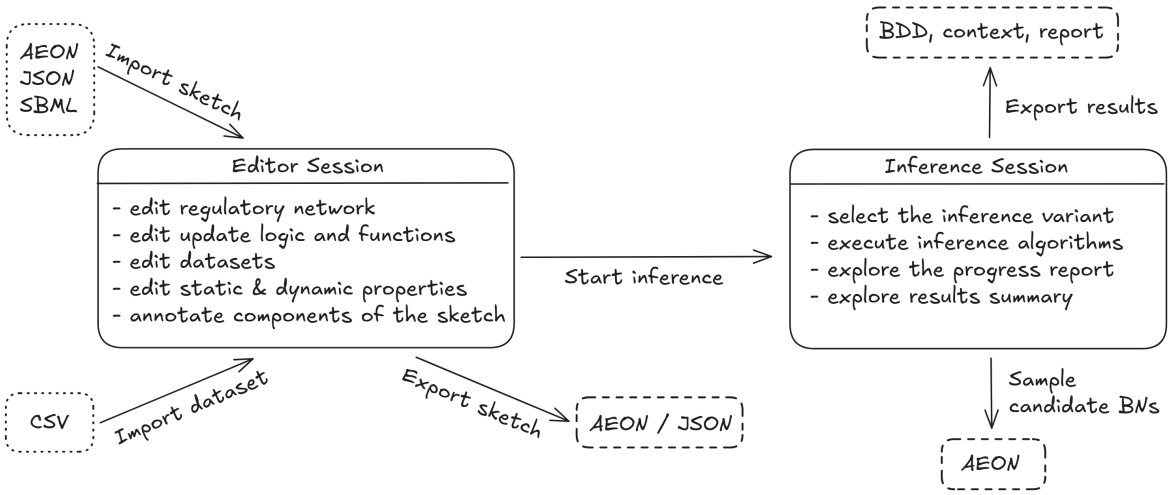
SketchBook Manual

Contents

Intro to SketchBook

SketchBook is a tool for inference of Boolean networks (BNs) using the framework of Boolean network sketches [2]. BN sketch is a concept for combining several types and sources of knowledge with experimental data. It allows us to apply inference algorithms to find all admissible BNs. Particularly, a BN sketch consists of an influence graph, a partially specified Boolean network (PSBN), and sets of required static and dynamic properties of the model.

Specifically, SketchBook is a multiplatform desktop application that offers a graphical interface for both editing all components of the sketch and running the inference process. SketchBook consists of two main sessions or windows: The interactive *editor session* and an *inference session*, as illustrated by the workflow below.



General workflow of SketchBook.

In the editor window, you can edit PSBNs, load datasets with observations, and create various kinds of static and dynamic properties. Once you have created the sketch, you can open the inference window. You can then run the inference algorithms and explore the set of admissible BNs that satisfy all the properties.

This manual covers the following:

- We give short instructions on the setup of SketchBook in Section 1.
- We describe the sketch editor session of the tool and its features in Section 2.
- Inference session and results are described in Section 3.
- Supported formats for datasets, sketches, and syntax for various expressions are discussed in Section 4.

1 Running SketchBook

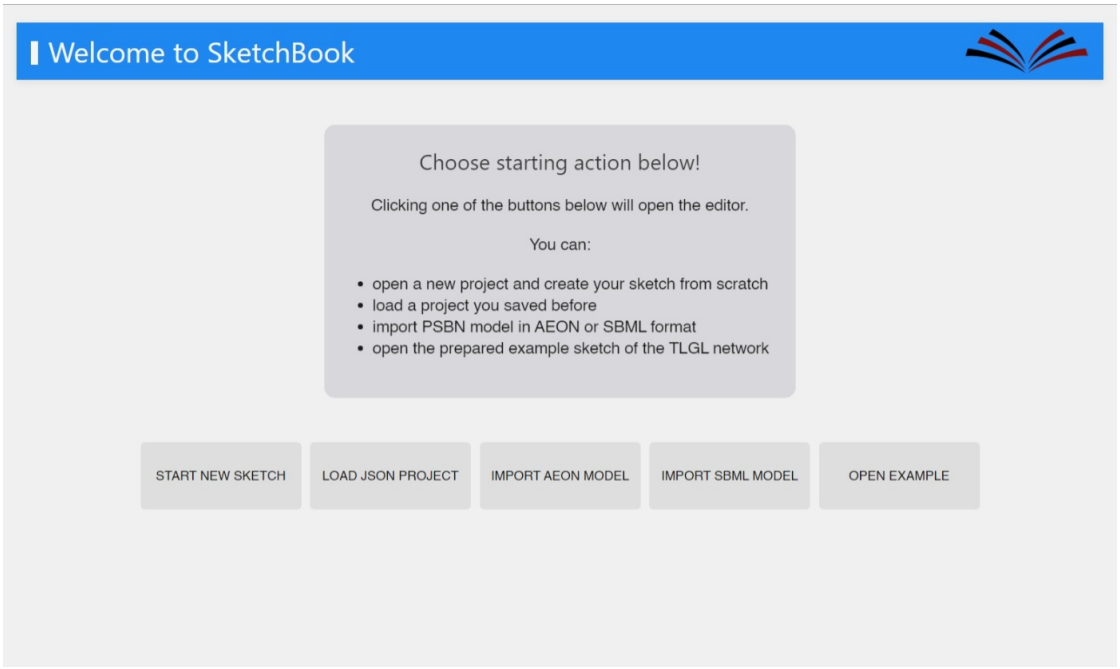
This section provides general instructions for installing and setting up SketchBook. The tool's repository with the latest version of the source code (that is licensed under MIT license) is freely available at our GitHub: `https://github.com/sybila/biodivine-sketchbook`.

We provide pre-built binaries and installation files for the application in the release section of the repository. To start using Sketchbook, choose the latest release and download a binary for your operating system. You choose between `.app` and `.dmg` for **macOS**, `.AppImage`, `.deb` and `.rpm` for **Linux**, or `.exe` and `.msi` for **Windows**. If you need a different pre-built binary for a specific platform, let us know!

Note that the binaries are not signed with official developer certificates, so macOS and Windows will most likely require you to grant special permissions to run the app. On newer versions of macOS, the message is that the app is *corrupted*. This is still the same issue regarding app certificates. You should be able to "enable" the app by running `xattr -c /path/to/biodivine_sketchbook.app`.

Alternatively, the desktop application can also be built directly from the repository's source code. To do this, please consult the *Development guide* of the repository Readme. Note that the local build requires additional dependencies to be installed.

After successfully starting the application, you should see the following screen:



Initial screen of SketchBook.

2 Sketch editor session

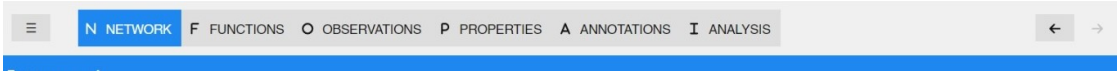
The sketch editor is the primary session of the tool that allows to edit all components of the sketch. The tool starts with a simple initial screen where you can select first action. This can be loading a sketch in various formats, creating a new empty sketch, or choosing a prepared example. Choose one of the buttons and continue:

Available buttons at the initial screen.

The following subsections then summarize the options available in the editor.

2.1 General navigation

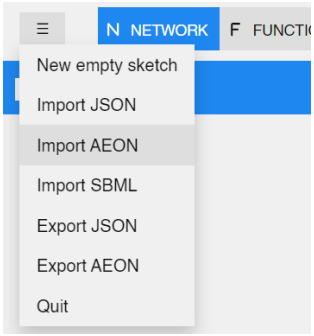
At the top of the screen, there is a popup menu, a list of tabs, and an undo-redo button.



Navigation bar at the top of the screen.

Each tab offers different functionality, as discussed below. You can select any of them by clicking on the tab button. Tabs can also be "locked," allowing two tabs to be displayed side by side. To do so, click on the small “lock button” at the bottom left of the screen.

The menu offers options regarding import/export. You can import sketches in JSON, AEON, or SBML format, and we also offer export in JSON and AEON. More details on the formats is in Section 4. Note that importing new sketch results in losing current data (you’ll get a warning).

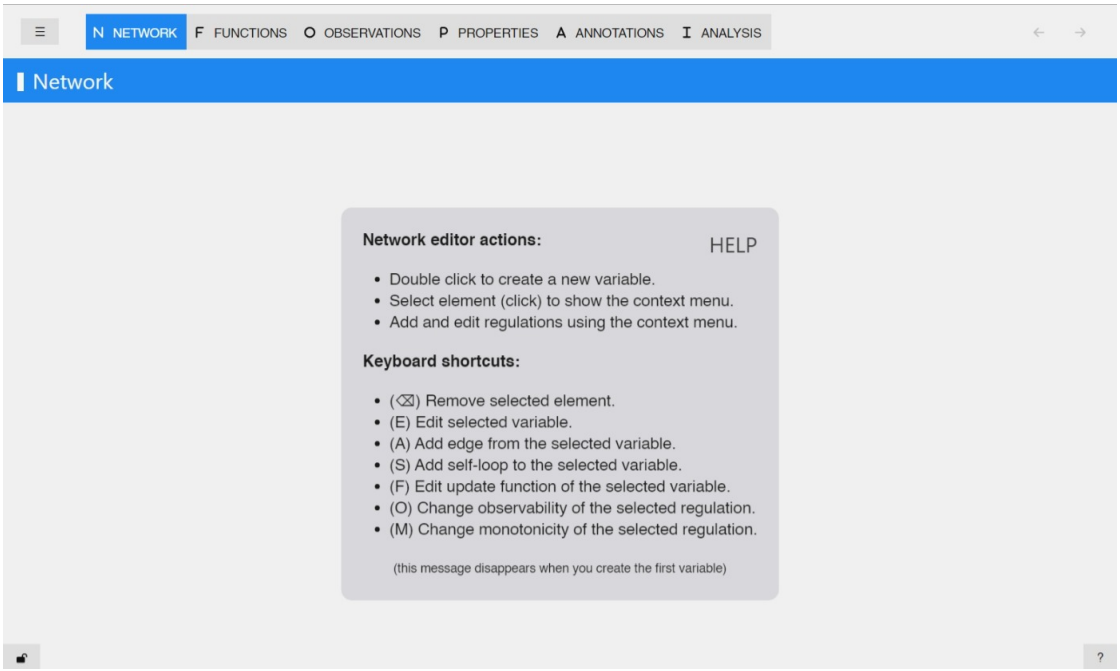


Menu with all the options.

Lastly, you can use the undo and redo buttons in the top right corner to reset or reapply your last actions.

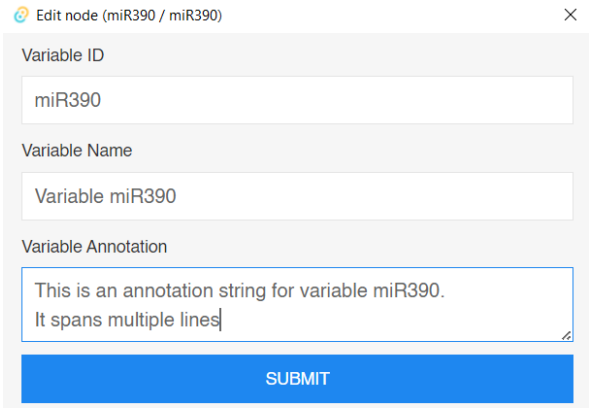
2.2 Editing regulatory network

The editor starts with the *Network* tab opened. If a new project is started, a help message is displayed. The message disappears after you start editing the network, but you can reopen it by hovering over the “question mark” button in the lower-right corner of the screen. The user can then edit or load the network.



Empty network tab of the editor session with a help message.

A new variable can be added by double-clicking at the free space. You can drag variables to achieve any kind of layout. Then, you can select a variable to get a context menu with options to add regulations, edit the variable, or edit its update function. Regulation can be added by clicking the corresponding "+" button and dragging the arrow that appears towards another variable. The option for editing a variable opens a dialogue window where you can edit its name, ID, and annotation.



Example of a dialogue window to edit variable.

You can also select a regulation and customize it. We offer customizing two kinds of regulatory properties – *monotonicity* and *observability*.

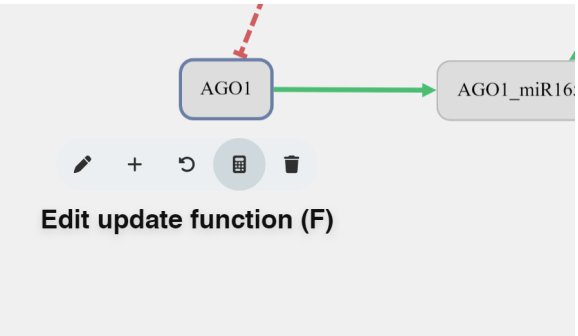
Monotonicity allows us to specify what type of effect the interaction has. The monotonicity of the regulation is reflected by its colour. The options are:

- **activation** for positive monotonicity (green colour)
- **inhibition** for negative monotonicity (red colour)

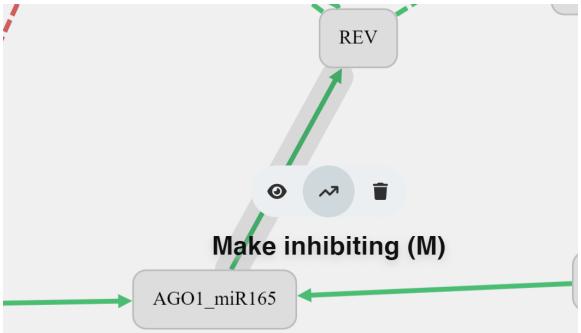
- **dual** for non-monotonous effect (blue colour)
- **unknown** for undefined (grey color)

Essentiality allows to specify whether the regulation must always have an effect on its target, or whether it may be "redundant". Essentiality is reflected by different line styles. The options are:

- **essential** for regulations that must always have an effect (solid line)
- **unknown** regulations that may or may not have an effect (dashed line)
- **non-essential** for regulation being completely redundant (dotted line)

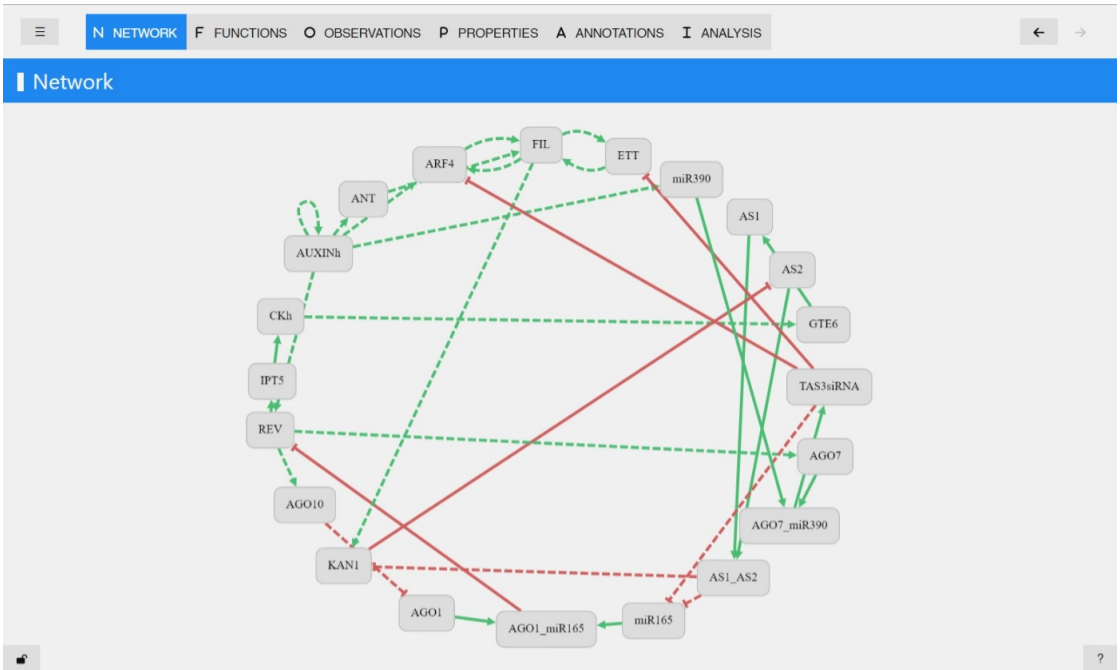


Variable context menu.



Regulation context menu.

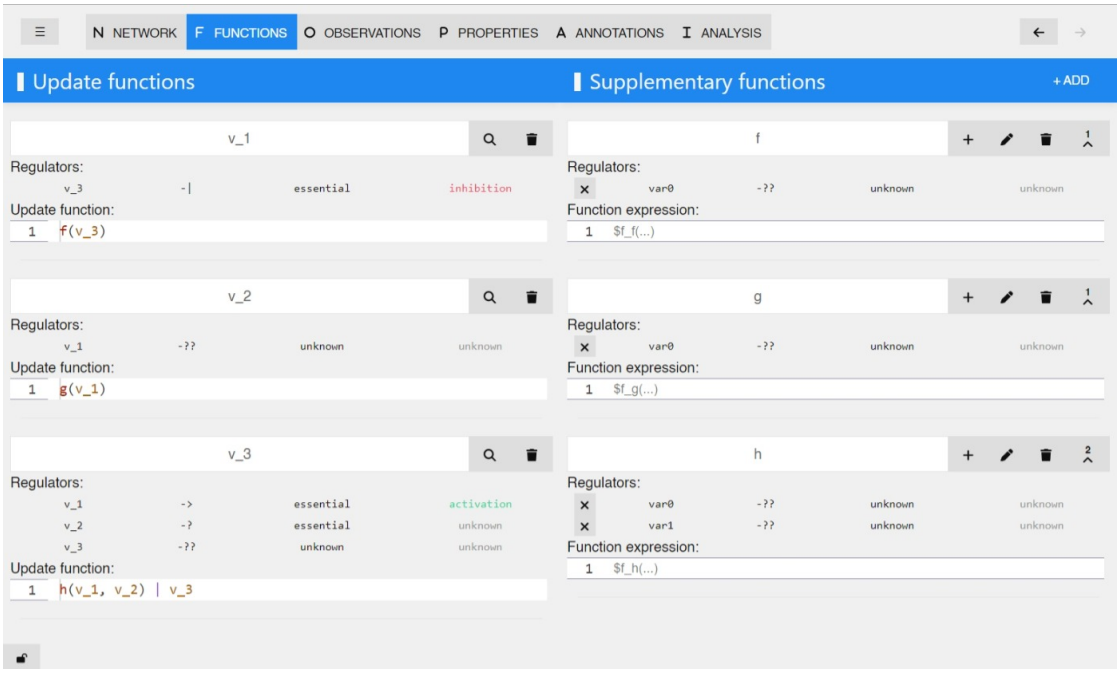
The fully edited network can look like the following:



Network tab with an example of a created regulation network.

2.3 Editing update functions

Update rules and supplementary functions can be edited within the *Functions* tab. On the left side of the screen, you can see a list of variables, their regulators and their *update functions*. On the right side, you can create additional *supplementary functions* that can be used inside the update function expressions.

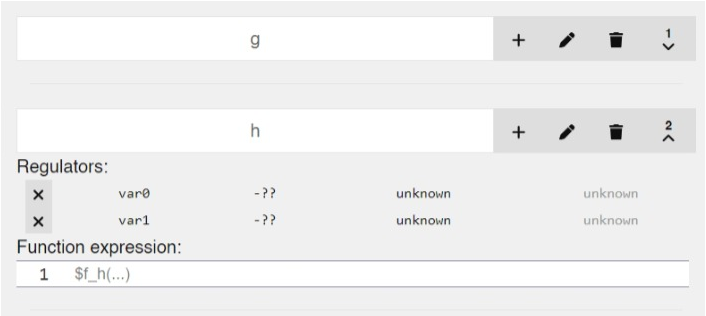


Functions tab with partially specified update functions for three variables and three supplementary functions.

For each variable, you can edit the type of regulations by clicking on the monotonicity or essentiality value. You can also edit the update function expression. The format of update functions is discussed in Section 4.1.1.

List of regulators and update expression for variable v_3.

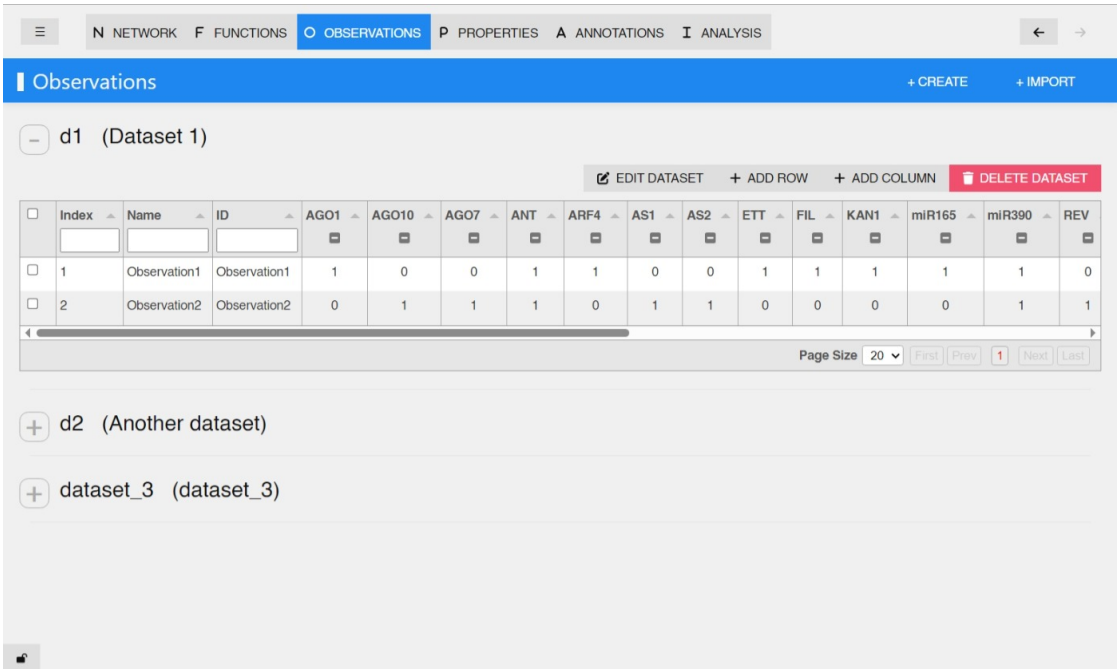
You can add new supplementary functions at the top bar by clicking the `+ADD` button. You must always add the function before using it in any expression. For each function, you can increment its arity, edit its details (same as for variable, via external dialogue), or delete it. Arguments of the function can be deleted to decrement its arity. The details can be hidden by collapsing the function.



Collapsed function g with arity 1 and expanded h with arity 2.

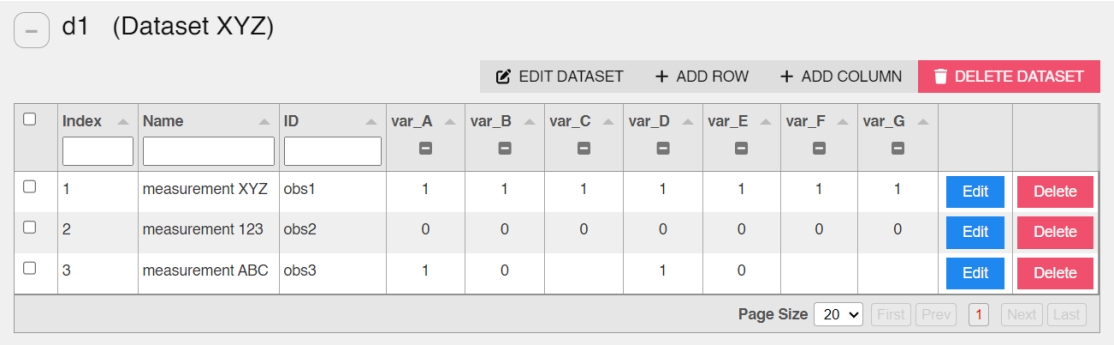
2.4 Loading and editing datasets

Datasets of observations can be edited in the *Observations* tab. You can either import a `CSV` dataset (in the format discussed in Section 4) or create one from scratch. You can either expand the dataset to see its content in a table-based editor (by clicking "+" button next to its ID) or keep it collapsed.



Observations tab with three datasets, one of them being expanded.

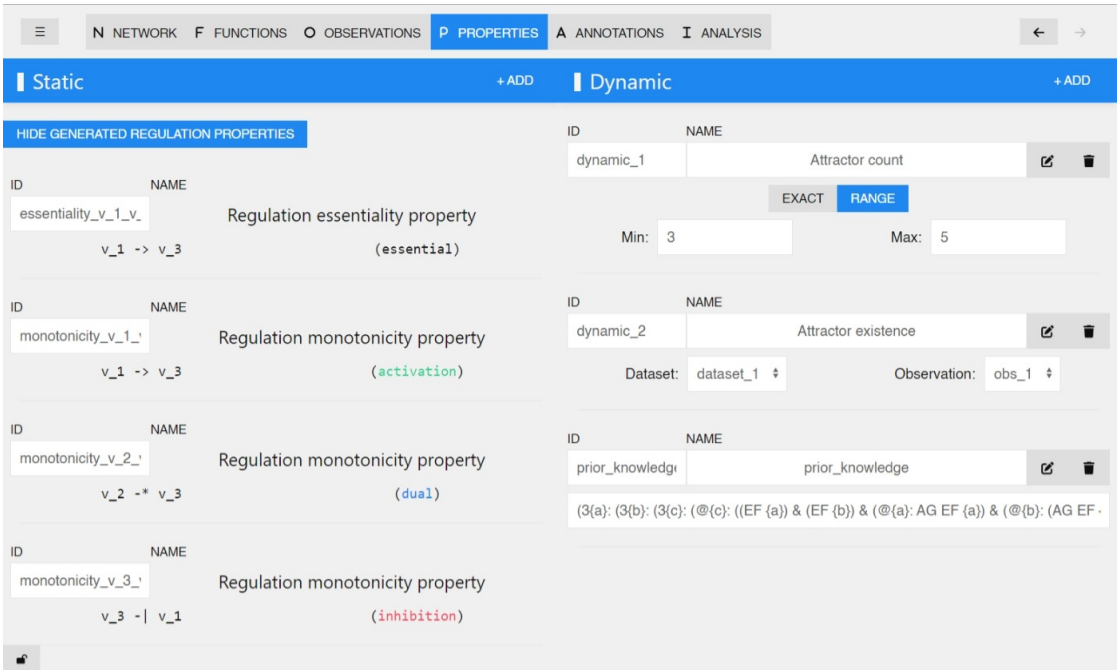
The table-based editor allows editing of all aspects of the dataset. The dataset's metadata (name, ID, and variable names) can be edited by selecting the edit dataset option. Observations can be edited by clicking on the table's fields or by selecting the blue edit button. You can delete both observations or datasets by clicking one of the pink buttons.



Observations tab with three datasets, one of them being expanded.

2.5 Editing properties

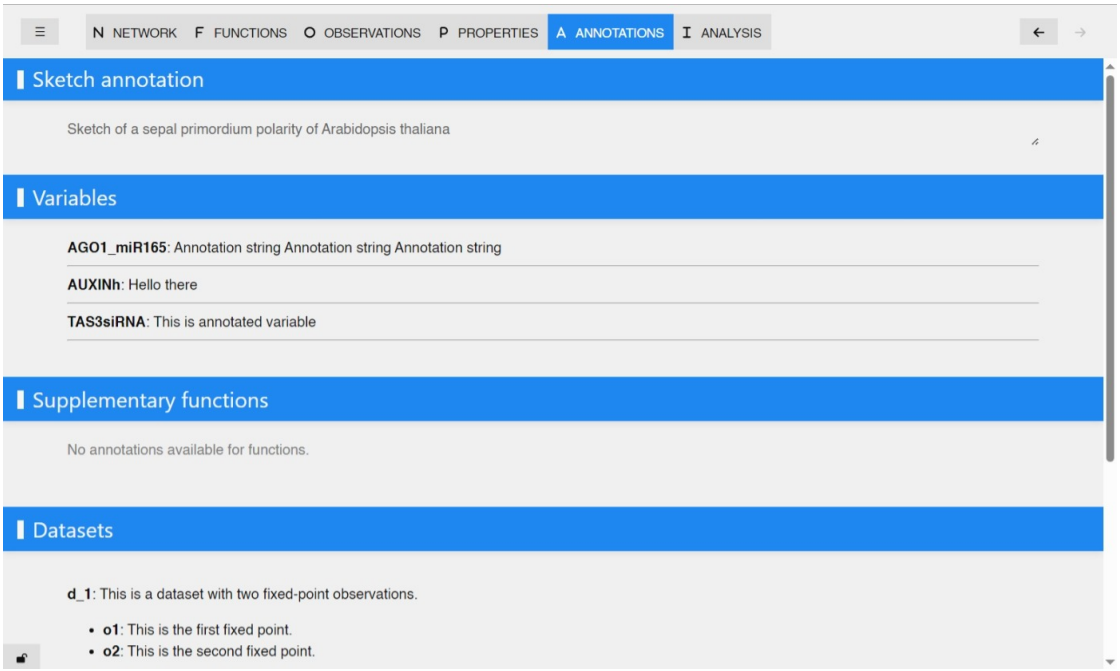
Properties tab serves to create and edit properties.



Properties tab with four static properties (automatically derived from the selected regulation properties) and three dynamic properties (showcasing some of the different templates).

2.6 Displaying annotations list

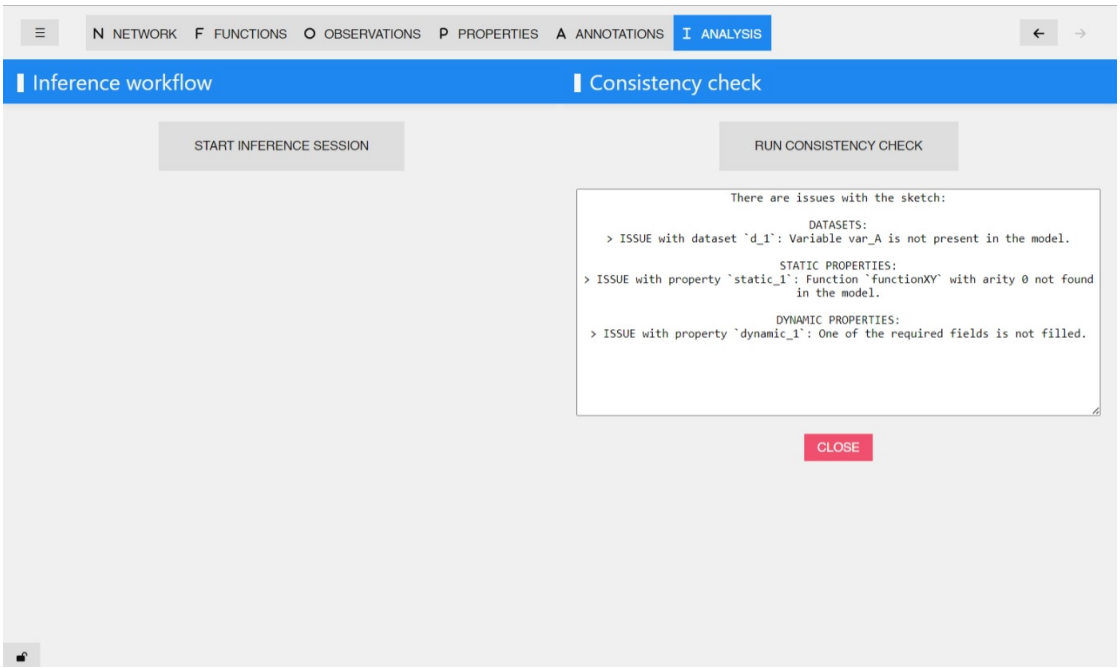
All of the annotations created throughout the sketch are summarized in the *Annotations* tab. Annotations are divided according to the entity type.



Annotations tab with annotations divided by entity types.

2.7 Starting analysis

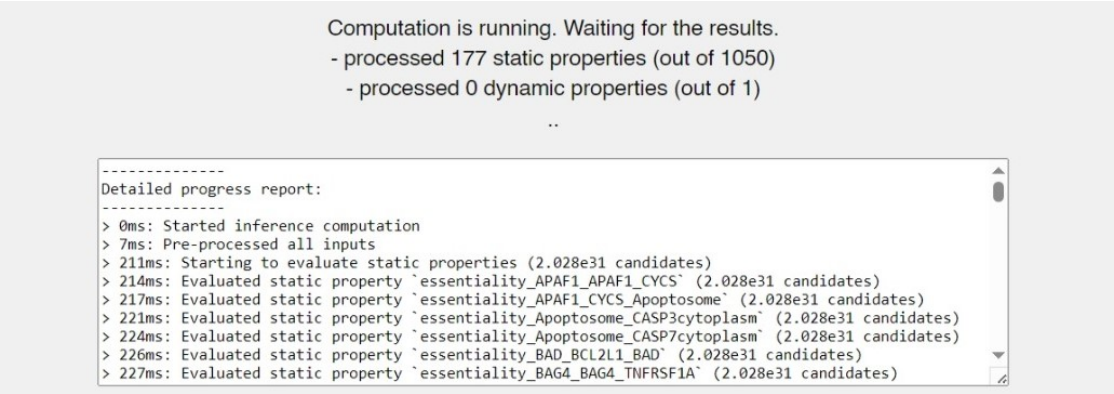
The last part of the editor session is the *Analysis* tab. The current state is very simple; the user can start the inference window or explicitly check the consistency of the sketch.



Analysis tab with a button to start inference window and run consistency check. An example of inconsistencies is shown in the text window.

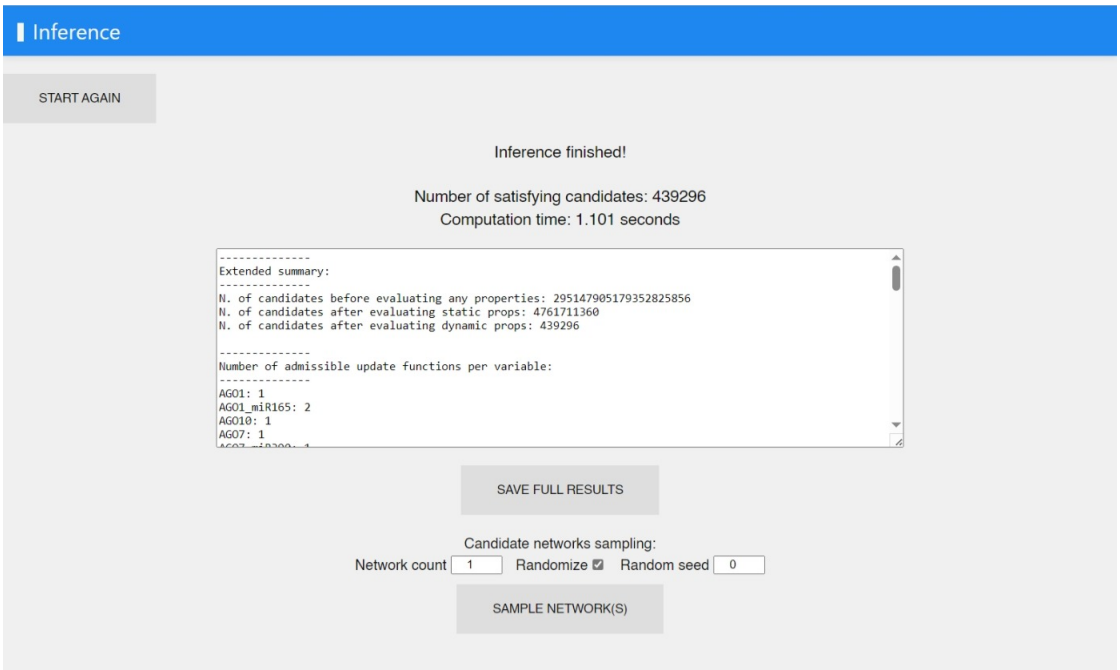
3 Inference session

After starting the new inference window, you can run the inference. You can choose from two options: running the full inference with all properties or performing a preliminary step – a partial "static" inference with static properties only. Once you run the inference, you'll see the progress summary and a report.



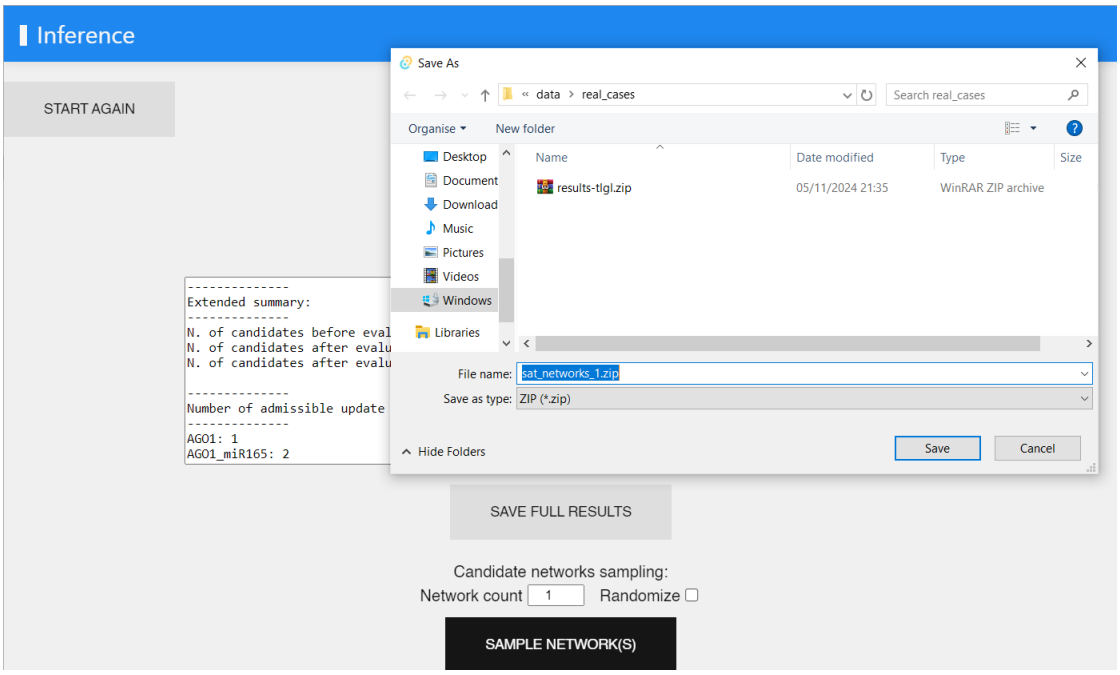
An example of the intermediate progress output during computation.

Once the computation finishes, you'll see a screen with results summary and options, as shown below.



Inference window with a summary and buttons to ex- port/sample the results.

Sampling admissible networks Finally, the user can randomly sample the fully specified admissible candidate networks (as `.aeon` files). To generate witness networks, click on the `SAMPLE NETWORK(S)` button. You can select how many witnesses to generate and also select a random sampling (the default sampling is deterministic). By clicking the button, a file save dialogue will open. Witness files are saved in a `.zip` archive.



Example of a sampling dialogue.

4 Sketch format

In the first part of this section, we describe our syntax for various logical expressions that users may utilize when writing update functions or properties from scratch. We then continue by describing the formats that the tool supports for the import/export of datasets and the whole sketch.

Before going into details, let's discuss some general rules regarding identifiers. Various entities (variables, functions, ...) of the sketch can have IDs, names, and annotations. An ID is a unique combination of alphanumeric characters and underscores; an ID should not start with a number. The name is an arbitrary combination of characters; we only require it not to contain newlines. Annotation is a completely arbitrary string.

4.1 Logical expressions

4.1.1 Syntax of update functions

Each variable has an update function that dictates its behaviour. Update functions are logical expressions in the format given by the following grammar. Note that *varId* is an ID of some variable and *fnId* is an ID of a function.

$$
\begin{aligned}
\mathit{function} &= \texttt{(}\ \mathit{function}\ \texttt{)} \mid \mathit{const} \mid \mathit{varId} \mid \mathit{fnSymbol} \mid \texttt{!}\mathit{function} \mid \mathit{function}\ \mathit{OP}\ \mathit{function} \\
\mathit{const} &= \texttt{true} \mid \texttt{false} \\
\mathit{OP} &= \texttt{\&} \mid \texttt{|} \mid \texttt{=>} \mid \texttt{<=>} \\
\mathit{fnSymbol} &= \mathit{fnId}\texttt{(}\mathit{args}\texttt{)} \\
\mathit{args} &= \mathit{function} \mid \mathit{args},\ \mathit{args}
\end{aligned}
$$

4.1.2 Syntax of generic dynamic properties

Apart from creating dynamic properties using pre-defined templates, users can define generic properties using logic HCTL.

The following grammar defines our textual format for HCTL formulae:

$$
\begin{aligned}
\mathit{formula} &= \texttt{(}\ \mathit{formula}\ \texttt{)} \mid \mathit{const} \mid \mathit{propName} \mid \mathit{var} \mid \mathit{unaryOp}\ \mathit{formula} \\
&\quad \mid \mathit{formula}\ \mathit{binaryOp}\ \mathit{formula} \\
&\quad \mid \mathit{hybridOp}\ \mathit{var}\ \texttt{:}\ \mathit{formula} \\
\mathit{const} &= \texttt{true} \mid \texttt{false} \\
\mathit{var} &= \texttt{\{}\mathit{varName}\texttt{\}} \\
\mathit{unaryOp} &= \sim \mid \texttt{AX} \mid \texttt{EX} \mid \texttt{AF} \mid \texttt{EF} \mid \texttt{AG} \mid \texttt{EG} \\
\mathit{binaryOp} &= \texttt{\&} \mid \texttt{|} \mid \texttt{=>} \mid \texttt{<=>} \mid \texttt{\^{}} \mid \texttt{AU} \mid \texttt{EU} \\
\mathit{hybridOp} &= \texttt{!} \mid \texttt{3} \mid \texttt{V} \mid \texttt{@} \mid \texttt{\textbackslash bind} \mid \texttt{\textbackslash exists} \mid \texttt{\textbackslash forall} \mid \texttt{\textbackslash jump}
\end{aligned}
$$

Note that *varName* stands for an HCTL variable, while *propName* stands for an atomic proposition. Each *propName* must correspond to a valid BN variable ID (this is checked). Only alphanumeric characters and underscores can appear in *varName* and *propName*. Note that $\sim$ corresponds to the negation and `^` to the xor operator. For hybrid operators, we allow specifying them by name prefixed by a backlash or using the following symbols: `!` for the bind operator, `3` for the existential quantifier, `V` for the universal quantifier, and `@` for the jump operator. Correspondence with other logical and temporal operators is straightforward.

The operator precedence is the following (the lower, the stronger):

- unary operators (both negation and temporal): 1
- binary temporal operators: 2
- Boolean binary operators: and=3, xor=4, or=5, imp=6, eq=7
- hybrid operators: 8

However, it is strongly recommended to use parentheses wherever possible. Note that formulae must not contain free variables.

For example, formula $\exists x.\forall y(@_y.\,\mathbf{EF}(x \wedge (\downarrow z.\,\mathbf{AX}\,z)))$ could be written as:

```
\exists {x}: \forall {y}: (\jump {y}: EF ({x} & (\bind {z}: AX {z})))
```

4.1.3 Syntax of generic static properties

Similarly, you can also design your own generic static properties. For these, we use the following first-order logic (FOL) syntax.

$$
\begin{aligned}
formula &= \texttt{(}\ formula\ \texttt{)} \mid const \mid var \mid fnSymbol \mid \texttt{!}formula \\
&\mid formula\ binaryOp\ formula \\
&\mid quantifier\ varList\ \texttt{:}\ formula \\
const &= \texttt{true} \mid \texttt{false} \\
binaryOp &= \texttt{\&} \mid \texttt{|} \mid \texttt{=>} \mid \texttt{<=>} \mid \texttt{\^{}} \\
fnSymbol &= fnId\texttt{(}args\texttt{)} \\
args &= function \mid args,\ args \\
quantifier &= \texttt{3} \mid \texttt{V} \mid \texttt{\textbackslash exists} \mid \texttt{\textbackslash forall} \\
varList &= var \mid var, varList
\end{aligned}
$$

Note that *var* stands for a FOL variable, and *fnId* is an ID of a function. Only alphanumeric characters and underscores can appear in *var* and *fnId*. For quantifiers, we allow specifying them by name prefixed by a backlash or using the following symbols: `3` for the existential quantifier, and `V` for the universal quantifier. This is the same as for the HCTL syntax.

For example, the monotonicity formula $\forall x, y.\ f(x, y, 1) \rightarrow f(x, y, 0)$ could be written as:

```
\forall x, y: f(x, y, 1) => f(x, y, 0)
```

4.2 Datasets

In the observations tab of the tool's editor, users can load binarized datasets from a CSV file. The first line of the CSV describes the dataset's variables, while the remaining lines describe individual observations. Let's look into an example:

```
ID, A, B, C
o1, 1, 0, 1
o2, 0, *, 0
```

This dataset contains two observations *o1* and *o2* over three variables *A*, *B*, and *C*. Values can be binary *1* or *0*, or a * character for unknown value.

4.3 Sketch formats

SketchBook currently supports three formats for importing or exporting data – a fully custom JSON format, a novel extension of AEON format, and an SBML format. For both JSON and AEON, we support both the export and import of the whole sketch. For the SBML format, we allow importing standard PSBN models (there is no standardized extension to cover properties or datasets at the moment).

4.3.1 Extended AEON format

The original AEON format allowed specifying variables (simple IDs), regulations, functions, and layout. More recently, it was extended with custom model annotations that can be used to represent additional information in a structured way. This format is still compatible with all tools using the AEON format, as these tools simply ignore the additional annotation lines. For example, you can directly import models created in `AEON` [1], and the other way around.

The influence graph of the PSBN is described as a list of edges, where each edge is encoded as:

```
regulator edge_type target
```

Here, `regulator` and `target` are the names of the network variables, and the `edge_type` is the type of influence, corresponding to one of $\{$`->`, `-|`, `-?`, `->?`, `-|?`, `-??`$\}$. Arrow `->` denotes activation, `-|` inhibition and `-?` is for unspecified monotonicity. Finally, an extra `?` signifies a non-observable regulation (a regulation that may or may not affect the target).

A particular update function for variable `A` is written as: "`$A: function`". The syntax of update functions is discussed in Section 4.1.1.

The layout position of variable nodes is written as "`#position:VAR:X,Y`", where `VAR` is the variable's ID, and X, Y are float positions (such as 42.42).

Our extension of the AEON format uses the annotations to represent all the additional information contained in the sketch. Each AEON annotation consists of a unique path-like identifier and a value. In our case, the path is simple – an entity type and ID. Note that IDs must always be unique in the sketch. Entity types can be *variable*, *function*, *static_property*, *dynamic_property*, or *dataset*. The value is a JSON-serialized struct of a given type (similar to our custom JSON format).

4.3.2 Custom JSON format

TODO:

4.3.3 SBML format

TODO:

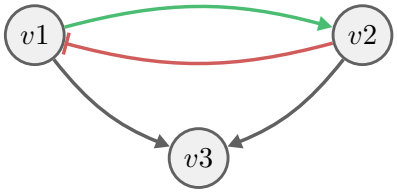
4.3.4 Example

TODO:

Let us have a PSBN model given by the following influence graph and update functions.

Variable `v1` positively influences (regulates) `v2`; `v1` negatively influences (regulates) `v2`; and both `v1` and `v2` influence `v3` (but the influences might be arbitrary). The update function for `v3` is not fully specified, but instead given by a *function* symbol **f** of arity two.

We further add the required HCTL properties for the existence of (arbitrary) fixed-point attractor, $\exists x.\, @_x(\mathbf{AX}\, x))$, and for the existence of a fixed point reachable from any state, $phi = \exists x. \forall y (@_y.\, \mathbf{EF}(x \wedge (\downarrow z.\, \mathbf{AX}\, z)))$.



$F_1(x) = \neg v2$
$F_2(x) = v1$
$F_3(x) = \mathbf{f}(v1, v2)$

The input file containing this whole model and properties may look like this:

```
#!  dynamic_property: #`3{x}: @{x}: (AX {x})`#
#!  dynamic_property:  phi: #`3{x}: V{y}: (@{y}: EF ({x} & (!{z}: AX {z})))`#
v2 -| v1
v1 -> v2
v2 -? v3
v1 -? v3
$v1: !v2
$v2: v1
$v3: f(v1, v2)
```

References

[1] Beneš, N., et al.: AEON: attractor bifurcation analysis of parametrised boolean networks. In: Computer Aided Verification. Lecture Notes in Computer Science, vol. 12224, pp. 569–581. Springer (2020)

[2] Beneš, N., et al.: Boolean network sketches: a unifying framework for logical model inference. Bioinformatics **39**(4) (04 2023)