Macchiato
A Simple and Scriptable Petri Nets Implementation
Version 1-6-1

Basic User Manual

(Spyder)

Dr. Mark James Wootton

Macchiato [makˈkjaːto]*, from the Italian, meaning* "spotted"*,* "marked" *or* "stained"*, in reference to a latte macchiato, which resembles a Petri Net place with a token.*

Contents

1   Introduction

Aim of this document is to provide comprehensive instructions for the use of the basic functionality provided by Macchiato for the creation and simulation of Petri Net [1, 2] models. For a detailed guide to all available features in Macchiato, please refer to the advanced user guide provided as a separate document.

The use of Macchiato for Petri Nets can been seen in the following publication:

- Mark James Wootton, John D. Andrews, Adam L. Lloyd, Roger Smith, A. John Arul, Gopika Vinod, M. Hari Prasad, Vipul Garg. Risk Modelling of Ageing Nuclear Reactor Systems, *Annals of Nuclear Energy*, volume 166, 108701, 2022. doi.org/10.1016/j.anucene.2021.108701

Please consider referencing this paper in work in which Macchiato has been used.

2   Dependences

- Python 3 [3]
    - NumPy [4] – only required by analysis scripts
    - Matplotlib [5] – only required by analysis scripts
- Microsoft Visio [6] – only required for graphical Petri Net construction tool

3   How to Run Macchiato Simulations in Spyder

In the following instructions, the use of Macchiato will be demonstrated within Spyder [7]. Users comfortable working with the command line directly may refer to the advanced manual.

Create a folder called `Macchiato` in a convenient location, and extract the downloaded files there. It is not advised to run simulations in the same directory as Macchiato itself. Instead, create a new folder to work in for each model.

For our demonstration, we will save Macchiato to `C:\Macchiato` and perform simulations in `C:\MacchiatoSimulations`. Create equivalent folders on your computer. A new folder is created by right clicking in the empty space of a File Explorer window to open the folder menu and selecting *"Folder"* from the *"New"* submenu, as seen in figure 1.

We will be using a simple Petri Net which contains examples of the various transition and arc types available. This Petri Net is visualised in figure 2.

Open Spyder and click the button highlighted in figure 3. Open the file `Macchiato.py` from where you saved it, as seen in figure 4.

Once open, Macchiato's source code will be visible on screen, see figure 5.

For this demonstration, we will be using the file `Example.mpn`, which describes the Petri Net seen in figure 2. If you want to view its contents in Spyder, you can open it from from the same dialogue box as you used to open Macchiato,
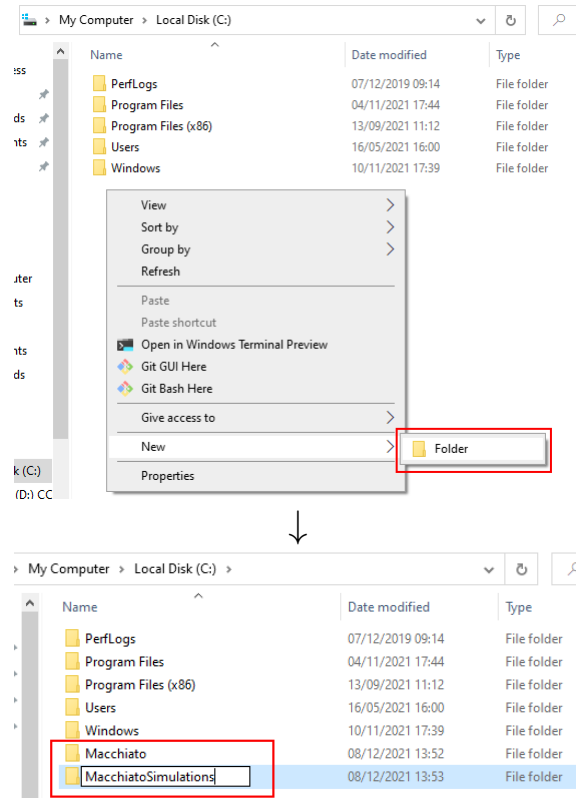
Figure 1: Create a folder to save Macchiato's file in and another to run the example Petri Net simulations.

i.e. by clicking the highlighted button in figure 5, but remember to enable the *"All files (*)"* option by selecting in from the drop-down menu seen in the bottom-right of figure 6. Alternatively, you can use any plain text editor. How to create your own Petri Nets for Macchiato is discussed in detail in section §4 *How to Create a Petri Net Model for Macchiato*.

By default, Macchiato prints minimal information to the console, giving only the time of the beginning and end of a batch of simulation. To see more information about each simulation, you can enable verbose mode, which we will do in our first example. Verbose is useful for debugging Petri Net models, but should not be used for large batches of simulation, as it can increase run times substantially.

We need to set the configuration for our simulations, so go to the top menu, and from *"Run"* select *"Configurations per file..."*, see figure 7.

Set the command line options to be file path of example file (in our case, `C:\Macchiato\Example.mpn`), followed by `1` (to run one simulation) and `-v` (to enable verbose mode), and set the working directory to be the folder you created earlier to run the simulations (in our case, `C:\MacchiatoSimulations`), then press *"Run"*, as seen in figure 8.

Macchiato will produce detailed output in the console, seen in figure 9, and you may scroll through this to inspect the behaviour of the simulation.

Files will be produced in the working directory we selected. There you will find a
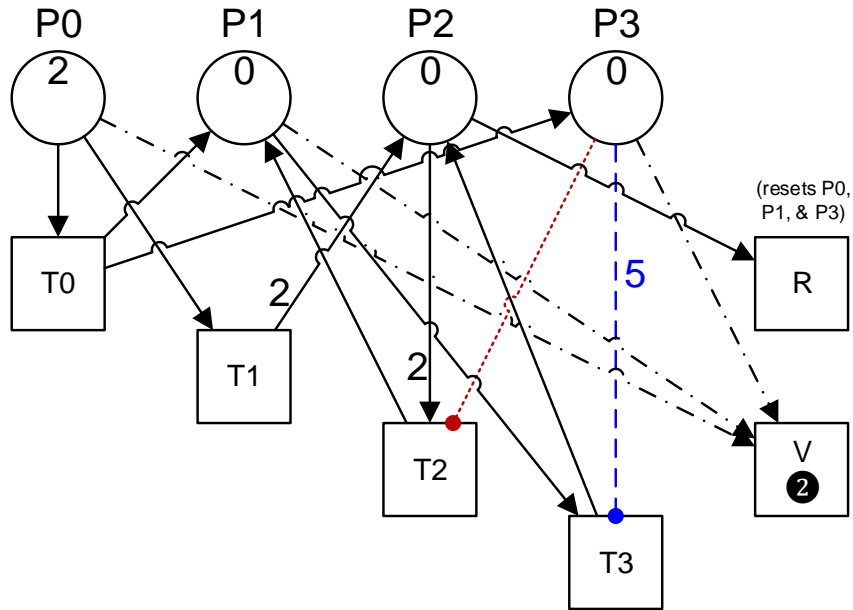
Figure 2: Illustration of the example Petri Net. Standard arcs are black solid-lines with arrowheads, inhibit arcs are red dotted lines with circular heads, and place conditional arcs are blue dashed lines with circular heads. Arcs without an explicit label have a weight of 1. The dot-dash effect has been added to the arcs of the voting transition manually.

file called `Test_Summary.txt` containing an overview of the simulation batch, and a subfolder called `Test`, which will hold three files:

- `Macchiato_PetriNet_Places_1.csv` – the number of tokens held by each place over the course of the simulation

- `Macchiato_PetriNet_Trans_1.csv` – the number of times each transition has fired over the course of the simulation

- `Macchiato_PetriNet_TransList_1.csv` – the sequence of transition firings in the simulation

You can open and inspect them, either with a plain text editor, or with spreadsheet software such as Microsoft Excel or LibreOffice Calc etc.

In the next example, we will run a batch of 1000 simulations using the same example[1]. Open the configuration menu again and this time change `1` to `1000` and remove the `-v` to disable verbose mode, see figure 10.

When you set these simulations running, you will notice that the only console outputs are the notifications giving the start and end time of the batch[2], see figure 11, and on completion, you will see that in the `Test` folder, there are three

---

[1]Petri Nets often contain probabilistic features, so it is necessary to sample them in sufficiently large numbers to achieve convergence. You may find that you have to run batch sizes several orders of magnitude larger than those in this demonstration.

[2]Note that for large batches of simulations, run times may extend into several hours, or even days in extreme cases.

files for each of the 1000 simulations. Keep hold of these files – in §5 *Analysis of Simulation Results*, we will use these output files as part of the demonstrations.

If you need to restrict the output in the `Macchiato_PetriNet_Places_*.csv` and `Macchiato_PetriNet_Trans_*.csv` files to a limited selection of places or transitions, you can do by adding the flags `-p` and `-t` to the command line options. As shown in figures 12, the labels of the places and transitions should then be listed after their respective flag.
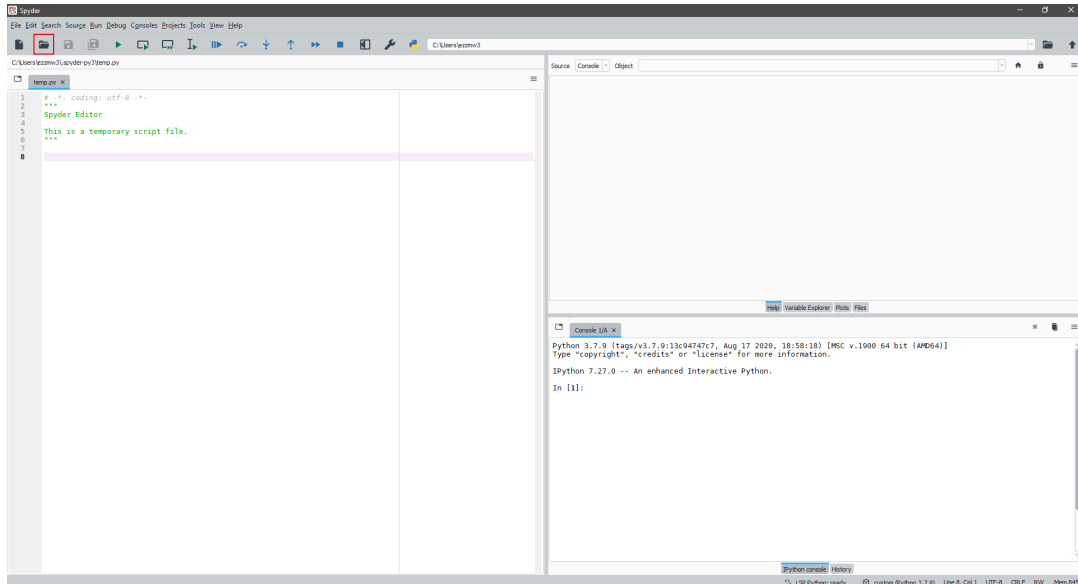

Figure 3: A new fresh Spyder window. Click the *"Open file"* button.
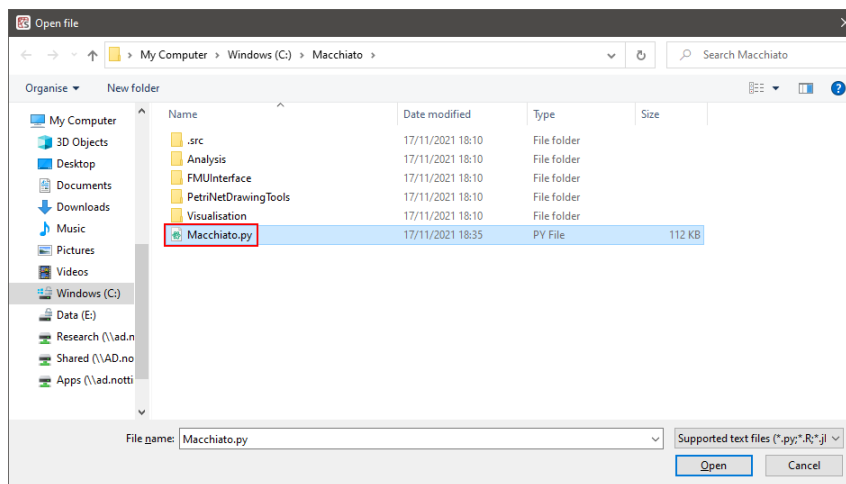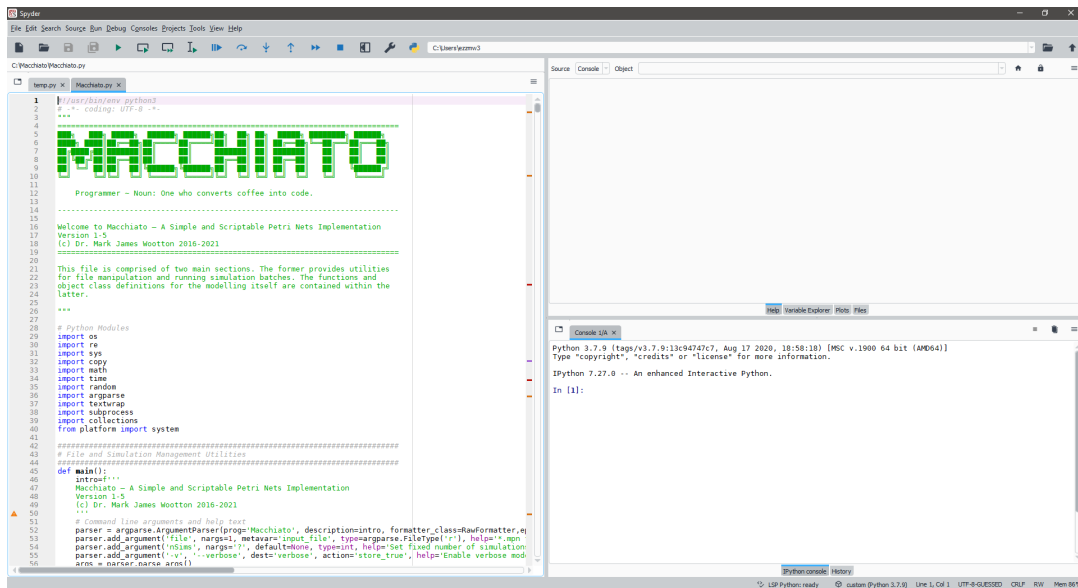

Figure 4: Opening Macchiato in Spyder.
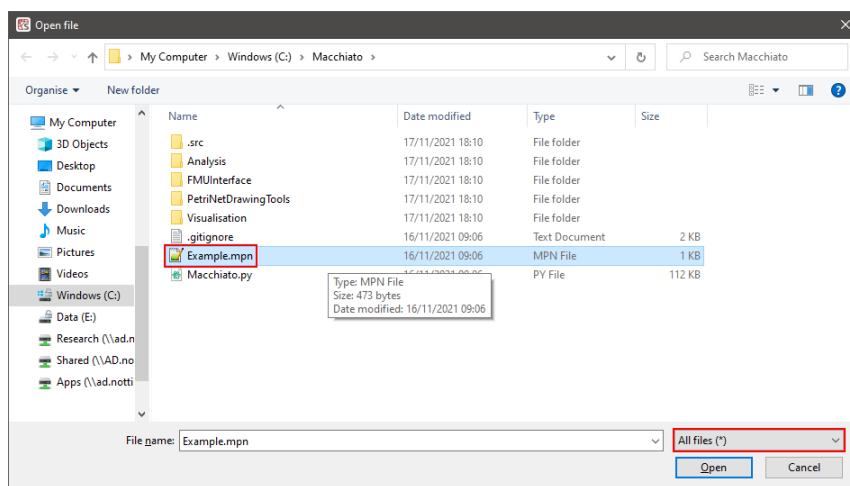
4

Figure 5: Macchiato open in Spyder.

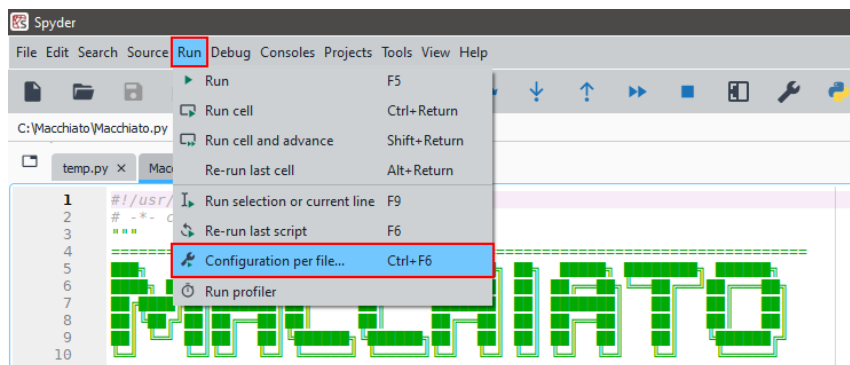Figure 6: Opening the example file in Spyder.

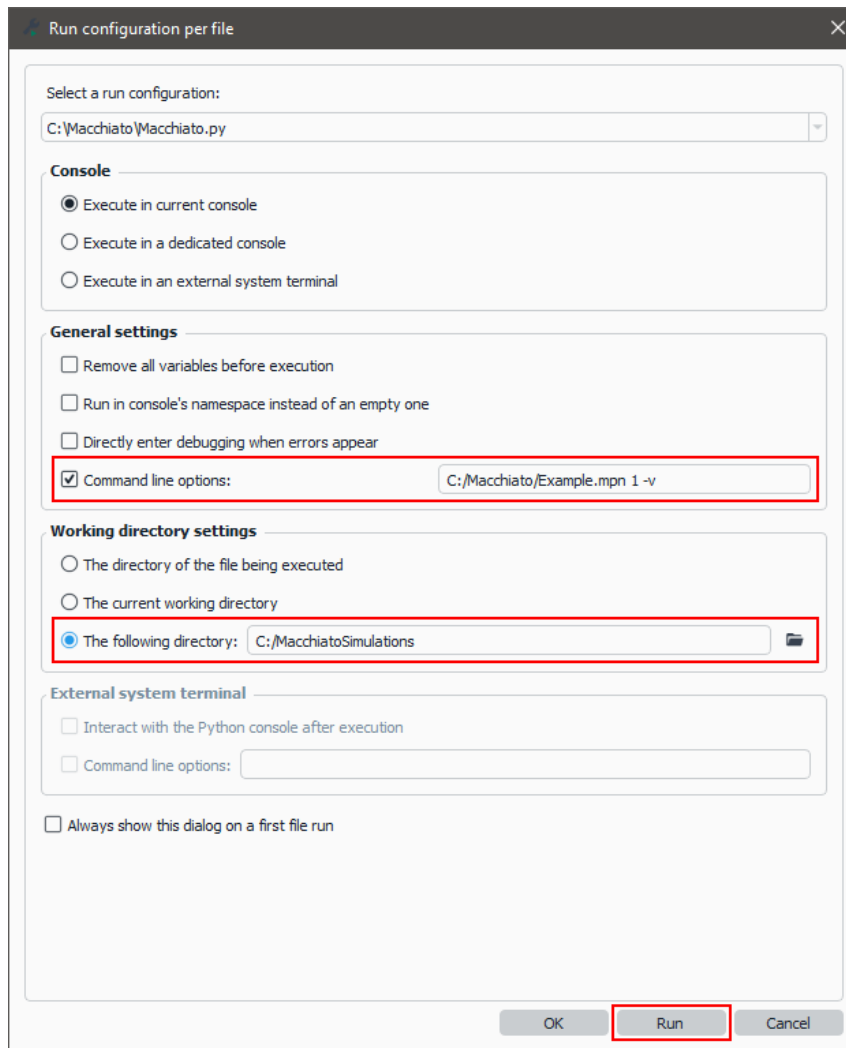Figure 7: Click on *"Configurations per file..."* from the *"Run"* tab of the menu.

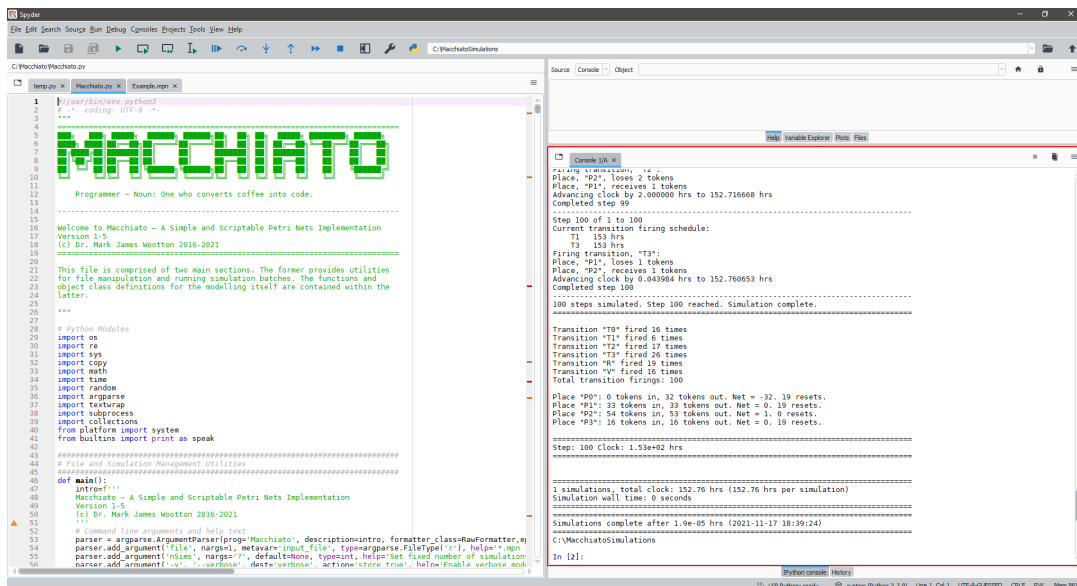Figure 8: Configuration to run one simulation in verbose mode.

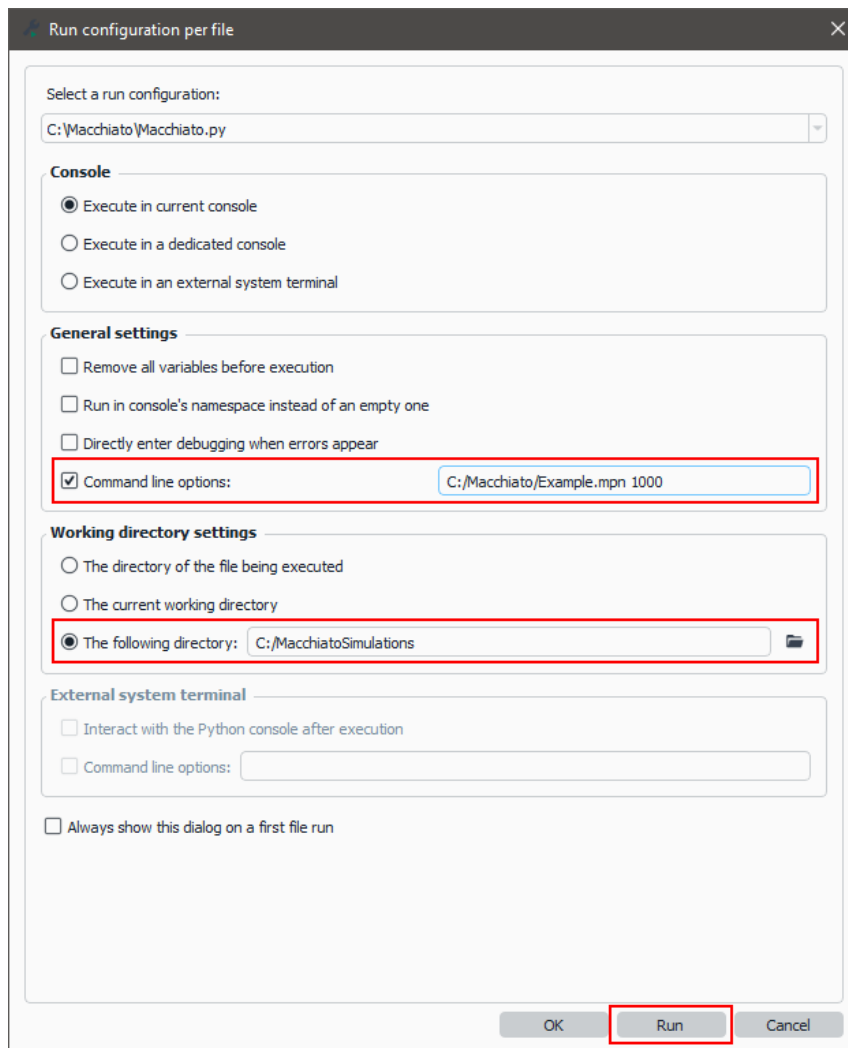Figure 9: Console output from one simulation in verbose mode.

Figure 10: Configuration to run 1000 simulations.

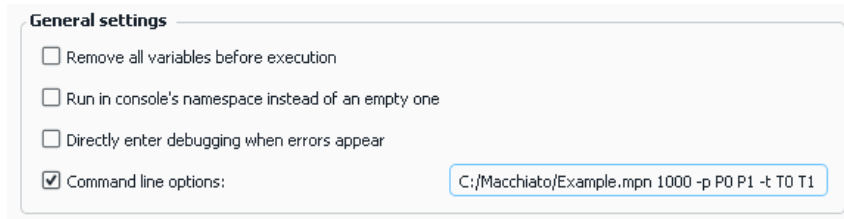Figure 11: Console output from 1000 simulations without verbose mode.

Figure 12: Configuration to run 1000 simulation with the output of `Macchiato_PetriNet_Places_*.csv` and `Macchiato_PetriNet_Trans_*.csv` respectively restricted to the places `P0` and `P1`, and transitions `T0` and `T1`.

4 How to Create a Petri Net Model for Macchiato

A Petri Net description in `*.mpn` format may be created textually, as outlined in §4.1 *Macchiato Petri Net Files (*.mpn)*, or via the tools seen in §4.2 *Graphical Petri Net Construction with Microsoft Visio*. Building Petri Nets with the graphical tools is generally much quicker, easier, and less prone to error than writing out the model by hand, but it is recommended that you read through both methods to understand the role of each parameter.

4.1 Macchiato Petri Net Files (`*.mpn`)

4.1.1 Example

In figure 13 is an example `*.mpn` file to demonstrate the usage of the key features of Macchiato. Indentation or any initial white space will be ignored, meaning that the user can arbitrarily align the contents the input file, such as is most convenient in clearly outlining the structure of the Petri Net. The structure and syntax of `*.mpn` files will be discussed in the next subsection.

4.1.2 Structure

An `*.mpn` should be comprised of three sections – the simulation parameters, the places list, and the transitions list. To mark the beginning of of the latter two, the lines **Places** and **Transitions** must respectively appear in the file. Any input on a line after a `#` character is treated as a comment and ignored. An example file is seen in §4.1.1 *Example*.

4.1.3 Simulation Parameters

name The label given to the Petri Net and used in output directories

unit The units of time to be used by the Petri Net (Default is `hrs`)

runMode The mode of integration to be used for simulation (Default is `schedule`). Changing this parameter is not recommended.

maxClock A simulation is terminated if its simulated period exceeds this parameter (Default is 10^6 **units** of time). Be aware that this does not mean that a simulation cannot exceed this value. If a simulation with **maxClock** set to 10 has a clock value of 9 and its next transition due to fire after a delay of 5, it will reach a clock value of 14 on the next step, at which point the sim-

```
# Petri Net Parameters
        name Test
        units hrs
        runMode schedule
        visualise None
        dot False

# Run Parameters
        maxClock 1E3
        maxSteps 100
        simsFactor 1

# Build Petri Net
Places
        P0 2
        P1
        P2
        P3


Transitions
        T0:lognorm:1:1 IN P0 OUT P1 P3
        T1:weibull:1:0.5 IN P1 OUT P2:2
        T2:delay:2 IN P2:2 P3:inh OUT P1
        T3:rate:15 IN P3:5:pcn P1 OUT P2
        R:cyclic:7:1 IN P2 RESET P0:P1:P3
        V:beta:1:2:0.25 IN P0 P1 P3 OUT P2 VOTE 2
```

Figure 13: The example Petri Net (as seen in figure 2) in `*.mpn` format.

ulation will end, having crossed the **maxClock** threshold. If simulations of an exact maximum duration are required, use a fixed delay transition set to terminate a simulation when it has fired once.

maxSteps Greatest number of steps permitted in any one simulation (Default is 10^{12}). If **runMode** is set to schedule, this is synonymous with the maximum number of transition firings permitted.

simsFactor Parameterises the total number of simulations performed (Default is 1.5×10^3). Repetition of simulations ends once the total simulated time surpasses the product of **maxClock** and **simsFactor**. If a set number of simulations is specified at the command line, **simsFactor** is overruled.

4.1.4 Places

The places section of the file begins following the line **Places**. To add a place, simple add a new line with the desired name (spaces are not permitted). By default, a place's initial token count is zero, but a value may be specified after a

9

space on the same line, e.g. `P1 2` will add a place of name *P1* with two tokens at the start of a simulation.

4.1.5 Transitions

The transitions section of the file begins following the line **`Transitions`**. A transition and its properties are specified with a line with the following format (do not include the brackets):

```
{Name}:{Timing}:{Parameters} IN {places} OUT {places} VOTE
   {threshold} RESET {places}
```

`{Timing}` specifies the duration from the enabling of a transition to its firing, which may be instantaneous, of a fixed length, or generated from a stochastic distribution. Most options require one or multiple parameters, subsequently delimited by `:` and expressed in terms of the parameter **`units`** where relevant. Refer to §4.1.7 *Additional Transitions Features* for setting **`RESET`** relations. The following timing options are available:

`instant` A instant transition will fire on the next simulation step with zero advancement of the system clock. If multiple instant transitions are simultaneously enabled, one will be chosen at random with uniform weight.

`delay:a` A transition with a fixed delay fired after a set duration `a` once enabled.

`uniform:u` A transition with a uniform distribution will fire at some time, t, in the interval $0 < t <$ `u`.

`cyclic:c:ω` A cyclic transition fires at the next instance at with the simulation clock is a non-zero integer multiple of `c`. The second parameter `ω` allows one to apply an offset. For instance, if two transitions with the parameters **`cyclic:1:0`** and **`cyclic:1:0.5`** are persistently enabled, they will respectively fire at the system times, 1, 2, 3 units etc, and 1.5, 2.5, 3.5 **`units`** etc.

`weibull:<t>:β:σ` A transition with this option will be characterised by a Weibull distribution [8] with mean `<t>` and shape parameter `β`. Its firing time, *t*, is given by $t = \eta \left[\ln(X)\right]^{-\beta}$ where η is the scale parameter, such that $\eta = \langle t \rangle \left[\Gamma\left(\beta^{-1} + 1\right)\right]^{-1}$, and X is a random variable uniformly distributed in the range $0 < X < 1$, with Γ being the Gamma Function [9]. The parameter `σ` is optional and is used when the mean time has an associated uncertainty, such that the scale parameter used for each firing delay calculated is produced from a normal distribution [10] with mean equal to the default η and a standard deviation of `σ`.

`lognorm:μ:σ` A transition with log-normal distribution [11] timing fires after time, *t*, where $t = \exp\left(\mu + \sigma X\right)$, with X being a standard normal variable, and `μ` and `σ` respectively being the mean and standard deviation of the natural logarithm of the firing delay.

rate:r A transition of this type fires with a constant rate parameterised by `r`. Firing time, t, is exponentially distributed [10], such that $t = -r^{-1}\ln(X)$, where X is a random uniform variable in the range $0 < X < 1$.

beta:p:q:k A transition with the Beta Distribution [12] produces a firing delay, t, in the interval $0 < t < 1$, parameterised by `p` and `q`, which weight the probability density towards the extreme or central regions of the available outcome space. An optional parameter `k` can be added to scale the distribution, such that the range of possible values becomes $0 < t <$ `k`.

Note that a transition must be continuously enabled for the duration from firing time generation until it fires. If its enabled status is interrupted, its scheduled firing time will be discarded.

4.1.6 Arc Properties

Any places listed after `IN` and `OUT` will be connected to the transition by incoming and outgoing arcs respectively and places listed should be separated by a single space. The weight of an arc is 1 by default and can be given some other value by appending it, separated by `:`, to the name of the relevant place in the list. for example, `IN P1 P2:3 P3 OUT P4 P5:2` specifies three incoming arcs, the second of which has a weight of 3, and two outgoing arcs, the latter of which has a weight of 2.

An incoming arc may be designated as a place conditional or inhibit arc with the code `:pcn` or `:inh` respectively, placed after the arc weight. The action of an inhibit arc is simple to disable its target transition when its weight is met, regardless of the status of the other arcs. A place conditional arc does not enable or disable its target transition but instead modifies its firing time parameters. The modification factor for a transition with C place conditional arcs is a function of the arc weights (which can be non-integer for place conditionals) and the number of tokens on the connected places, such that, $P = 1 + \sum_{i=1}^{C} W_i N_i$. The alterations to parameters are then, $a \to {}^a/P$, $u \to {}^u/P$, $c \to {}^c/P$, $\langle t \rangle \to {}^{\langle t \rangle}/P$, $\mu \to {}^\mu/P$, $k \to {}^k/P$, and $r \to rP$.

4.1.7 Additional Transitions Features

Transitions may also be given the properties `VOTE` and `RESET`. A voting transition does not require all of its incoming arc weights to be met to become enabled. Instead, only a given threshold need be met, placed after `VOTE`, separated by a single space. For example, a transition `T1` with the place relationship given by `T1:instant IN P1 P2 P3 OUT P4 VOTE 2` would only require two of `P1`, `P2`, and `P3` to hold a token in order to fire. Note that all incoming arcs whose weight is satisfied are treated normally for the purposes of removing tokens when the transition fires. A reset transition has an associated list of places, delimited by `:` and following `RESET`, separated by a single space, e.g. `RESET P1:P2:P3` [3]. When the transition fires, the places marked for reset are restored to the token count held at the beginning of the simulation.

---

[3]Please note that `:` must also be used to separate place tags in the shape data *Reset* field when working with the graphical editor, see §4.2 *Graphical Petri Net Construction with Microsoft Visio*.

4.2 Graphical Petri Net Construction with Microsoft Visio

As an alternative to transcribing a Petri Net structure by hand, it is possible to graphically construct a model in Microsoft Visio and export as an `*.mpn` file. This is general considerably simpler and more expedient, and has the secondary benefit of concurrently producing high quality figures for reports and publications.

4.2.1 Usage

In the directory PetriNetDrawingTools, one will find a Microsoft Visio drawing and a stencil file. The drawing file contains an example and the stencil file contains the description of the Petri Net elements that Microsoft Visio needs to work with Macchiato.

Do not edit the stencil file unless you know what you are doing.

To begin, make a copy of the the example drawing file, which contains the macro that exports the Petri Net to `*.mpn`. The macro is a small embedded program that is executed from within Microsoft Visio to convert the structure you have drawn into a `*.mpn` that Macchiato can read. A copy of the stencil file must be saved in the same directory or imported from the repository itself. Only objects from this stencil should be used.

When the example file is opened, a small banner will appear, asking the user whether macros should be enabled, as seen in figure 14. Click *"Enable Content"* as otherwise it will not be possible to run the macro to export the Petri Net as an `*.mpn`.
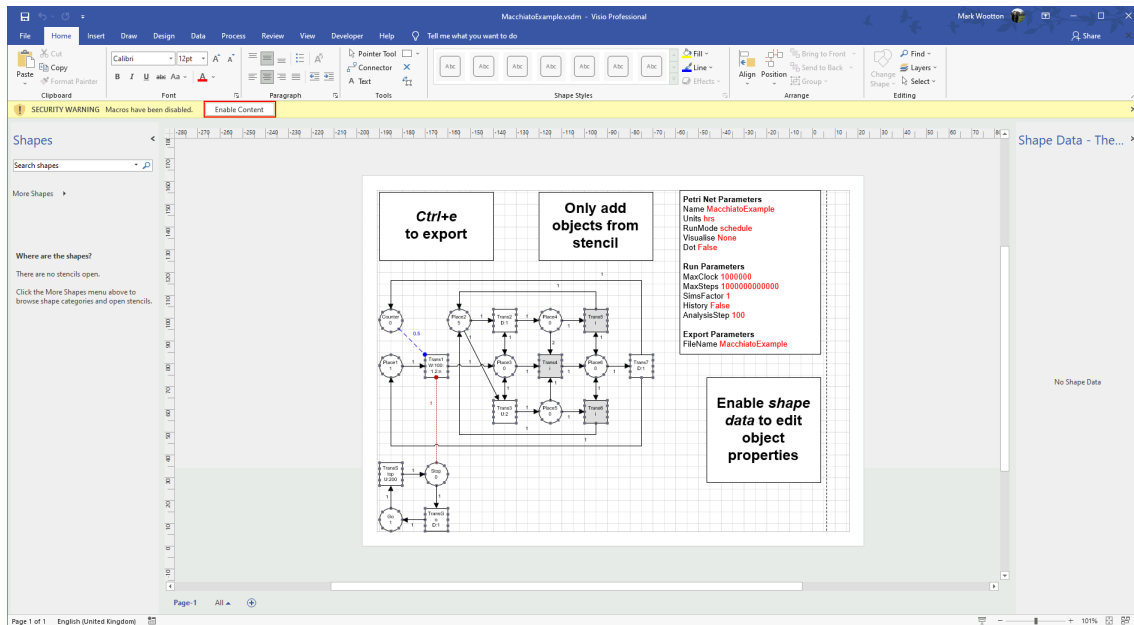

Figure 14: The example Petri Net as viewed in Microsoft Visio 2019. Remember to allow macros by clicking *"Enable Content"* on the yellow warning message bar.

If the Macchiato stencil is not already visible in the *"Shapes"* panel, import `MacchiatoStencil.vssx` via *"More Shapes"* → *"Open Stencil"*, see figure 15.
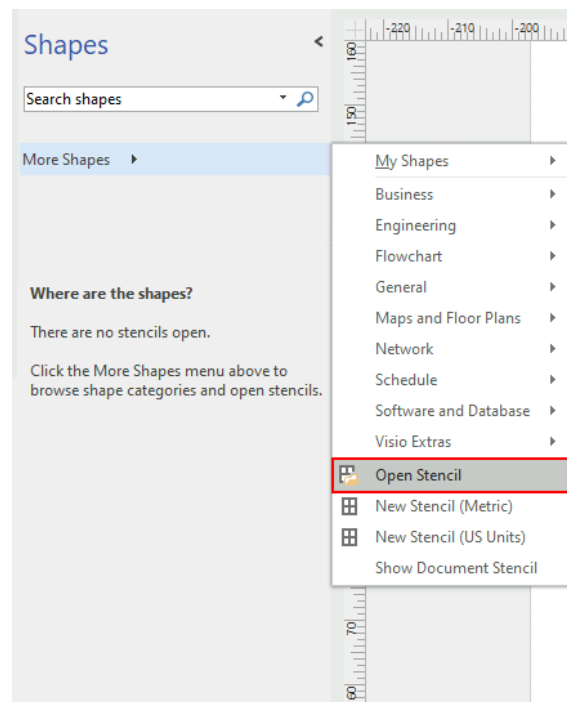
Figure 15: If not already visible, open the Macchiato stencil via the *"More Shapes"* menu, found on the left of the screen by default.

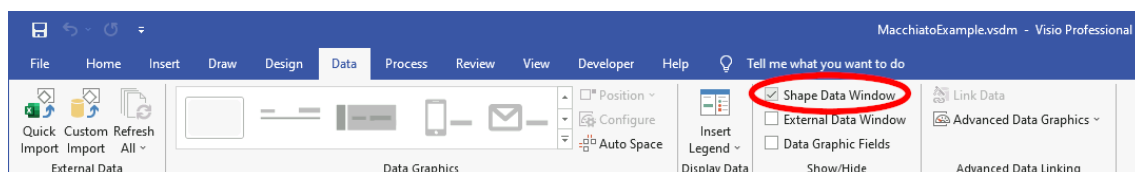Make sure that *"Shape Data Window"* is enabled in the *"Data"* tab, see figure 16.

Figure 16: *"Shape Data Window"* must be enabled to edit Macchiato object parameters.

The *"Shape Data"* panel, shown in figure 17, is used to set the parameters for each object in the model, including the system parameters block.

New objects are added, either by dragging shapes from the stencil, or by copying existing ones. Particularly in the latter case, make sure that every place and transition has a unique name. The formatting of the objects can be changed freely. Likewise, the text of their labels can be edited as required or desired, but be aware that if the content pertaining to its properties is altered, the field in question will no longer auto-update with respect to later changes.

Connections must be made using the arc objects and not the default Visio connectors. The arcs are connected by dragging their end points onto the places and transitions, either linked to the centre or to an attachment point, as shown in figure 18.

Additional points can be added from the *"Home"* tab using the button circled on figure 19. However, be certain that the correct object is selected as Visio will allow the user to attach a connection point associated with the currently selected object
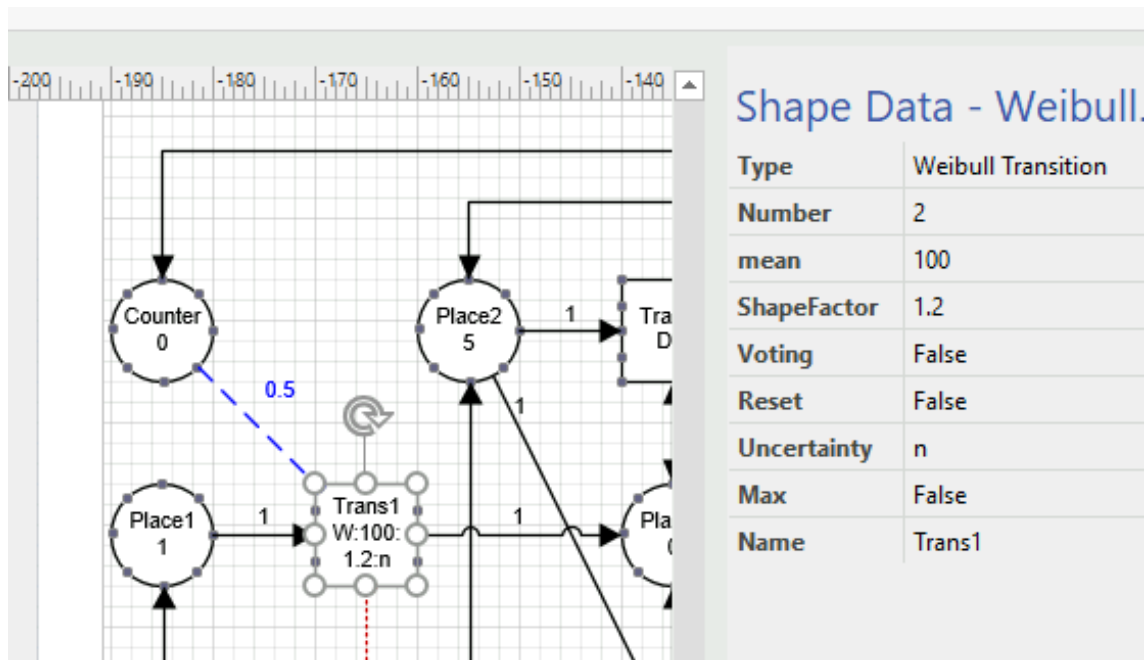
Figure 17: Object parameters are edited via the *"Shape Data"* window.

(a) Gluing an arc to an object (b) Affixing an arc to a connection point
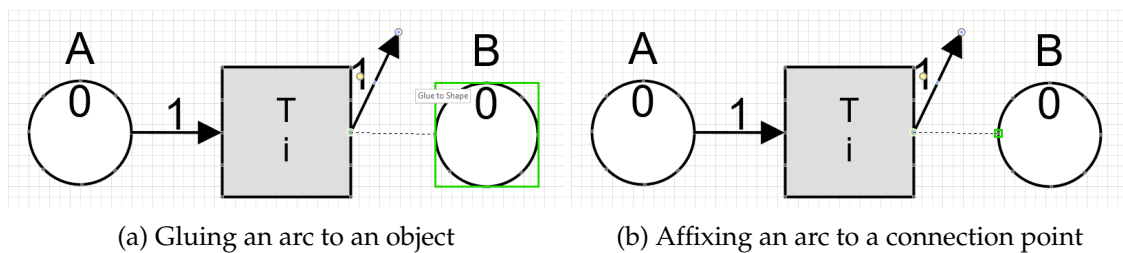
Figure 18: An arc can be either glued to an object, which case it will align to its centre, or be affixed to a connection point on its edge.
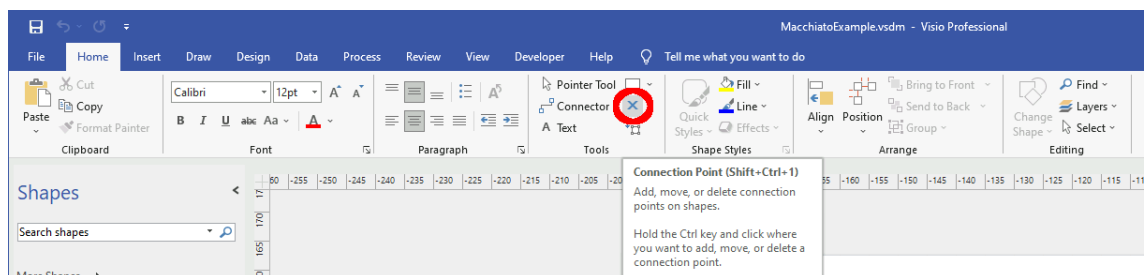

Figure 19: Connection points can be added via the circled button, or by pressing *"Ctrl+Shift+1"*.

to any shape in the file, potentially causing failed or erroneous `*.mpn` export. The *"Shape Data"* panel, shown in figure 19, is used to set the parameters for each object in the model, including the system parameters block.

The model is exported to an `*.mpn` file when the key combination *"ctrl+e"* is pressed[4]. The output is saved in the same directory as the source file, with the

---

[4]If this shortcut fails, the macro can be activated manually from the top menu via *"View"* →
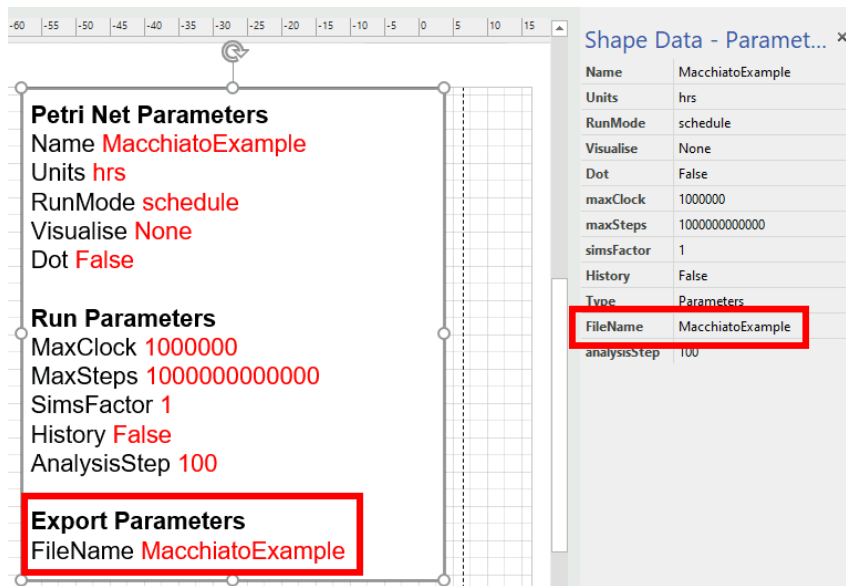
14

Figure 20: The file name used when the Petri Net is exported is set in the Parameters object.

name specified in the system parameters object, see figure 20. Existing files will be overwritten, with no warning issued, so be careful not to unintentionally destroy work.

4.2.2 Troubleshooting

Some users have reported that they have been unable to dismiss the security protections disabling the use of the macro. In this event, one can create a new Visio file and add the code provided in `MacchiatoExport.Module1.ParsePetriNet` as a new macro with the same name via the *"View"* → *"Macros"* menu.

5 Analysis of Simulation Results

Four Python scripts are available in the `Analysis` directory to extract information from the results of Macchiato simulations, and a fifth script provides some additional graphing tools. Open these files in Spyder and for all of them, the working directory needs to be set as the same folder where the results are found, seen in figure 21 – this is the folder where `Text_Summary.txt` was produced, in our case, `C:\MacchiatoSimulations`. Output files will be placed in this folder.

5.1 `OutcomesData.py`

This script will produce statistics describing the final states of key places and produces the following files:

- `{NAME}_TimingData.csv` – This file contains the number and proportion of simulations terminating with non-zero tokens in the target places.
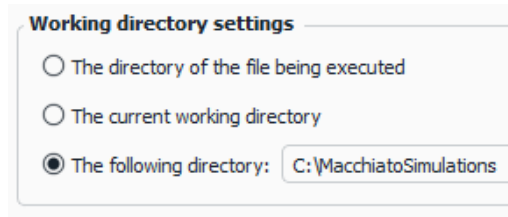

"Macros" → *"Run"*.

Figure 21: Working directory for the analysis scripts.

Confidence bounds are calculated via standard error [13]. The 10th and 90th percentiles are also given, as well as the mean and median of the entire population.

- `{NAME}_OutcomeTimes.csv` – This file contains all finishing times of the simulations organised according to their outcome(s).

- `{NAME}_{PLACE}_end_histogram.png` – A histogram for each outcome is produced displaying the distribution of its final times (outcomes which were not achieved by any simulation are skipped). The axis labelled *"Duration"* taking the same units as those specified in the simulated Petri Net.

To use it, set the command line options as the name of the simulation output folder and the list of places of interest, see figure 22.
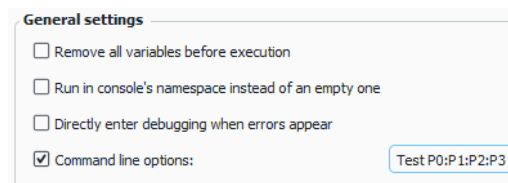

Figure 22: Command line options for `OutcomesData.py`.

5.2 `TransFireData.py`

This script will produce a file, `{NAME}_TransFireData.csv`, containing the average number of times each transition in a Petri Net model fired with the standard error [13]. To use the it, set the command line options to be the name of simulation output folder, i.e. `Test` for our example, see figure 23.
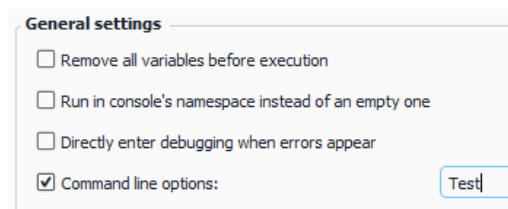

Figure 23: Command line options for `TransFireData.py`.

16

5.3 `ExtractPlacesEndings.py`

This script will output the the final state of each of a given set of places in a batch of simulations, with each place having its own file, named `Endings_MacchiatoSimulations_{PLACE}.csv`. As seen in figure 24, it takes two command line options – the simulation output folder and the list of the places to be inspected. The latter is delineated by colons e.g. `P0:P1:P2:P3`.
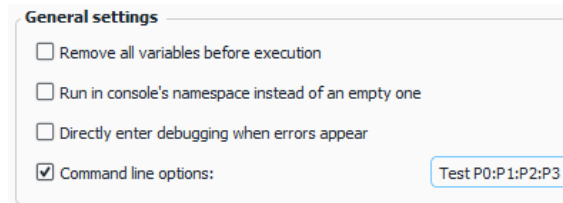

Figure 24: Command line options for `ExtractPlaceEndings.py`.

5.4 `Places_wrt_Time.py`

This script will give the average number of tokens in each of a given set of places, plus standard error [13], sampling the simulation results at a user specified time interval. The total number of simulations continuing to run up to that point is also given. Image files containing graphs to illustrate these results are produced. The output files for each place are a `*.csv` and a `*.png`, both labelled `MacchiatoSimulations_{PLACE}_averages`. Additionally, a graph showing the number of simulations continuing to run with respect to time is produced with the name `MacchiatoSimulations_simulations_running.png` (this data is also found in the aforementioned `*.csv` files). As seen in figure 25, four command line options are needed, these being the simulation ouput folder, the upper bound of time to be covered[5], the sampling interval, and the list of places, respectively. As before, the place list is delineated by colons.
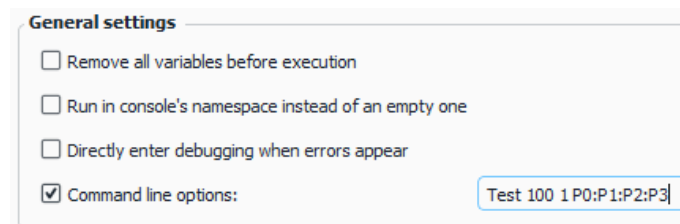

Figure 25: Command line options for `Places_wrt_Time.py`.

5.5 `HistogramTime.py`

This script will create histograms using the output file `{NAME}_OutcomeTimes.csv` from `OutcomesData.py`. It takes the name of this file as its command line option, see figure 26. The first entry of each line will be used as the title for the resulting graph, allowing the user to change it from the default (i.e. the label of the place) to something more human readable.

---

[5]E.g. if a value of 100 was given for this value, the time axis would have a range of 0 to 100

However, take care when doing this if you are using a spreadsheet application, as the number of columns may exceed its maximum. Therefore, it is often safer to rename the entries using a text editor program.

If a number is provided as an additional option, as seen in figure 27, this serves as a cut-off value for the x-axis, i.e. simulations ending past this point will be excluded. Be aware that this value should be given in years and an assumption is made that the simulation units are in hours.

If a percentage is given instead, see figure 28, this will be used as an upper cut-off for the proportion of simulations, i.e. the y-axis.
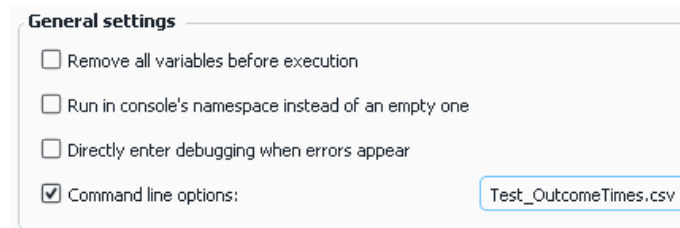

Figure 26: Basic command line options for `HistogramTime.py`.
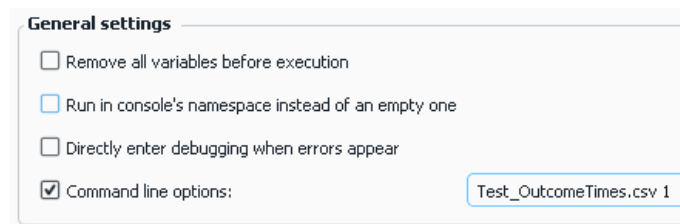

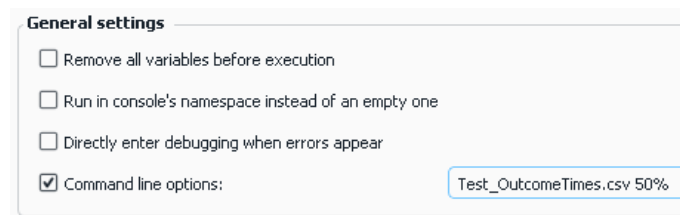Figure 27: Command line options for `HistogramTime.py` with duration (x-axis) cut-off.

Figure 28: Command line options for `HistogramTime.py` with percentage (y-axis) cut-off.

Acknowledgements

References

[1] Carl Adam Petri. *Kommunikation mit Automaten* (In German). PhD thesis, Technical University Darmstadt, 1962.

[2] Winfrid G. Schneeweiss. *Petri Net Picture Book (An Elementary Introduction to the Best Pictorial Description of Temporal Changes)*. LiLoLe – Verlag GmbH (Publ. Co. Ltd.), 2004.

[3] Python. *www.python.org*.

[4] NumPy. *www.numpy.org*.

[5] Matplotlib. *www.matplotlib.org*.

[6] Microsoft Visio. *www.microsoft.com/en/microsoft-365/visio/flowchart-software*.

[7] Spyder. *www.spyder-ide.org*.

[8] Athanasios Papoulis and S. Unnikrishna Pillai. *Random Variables and Stochastic Processes*. McGraw Hill, 4th edition, 2002.

[9] Ionuţ Florescu and Ciprian A. Tudor. *Handbook of Probability*. John Wiley & Sons, 2013.

[10] G. M. Clarke and D. Cooke. *A Basic Cource in Statistics*. John Wiley & Sons Ltd., 2004, 5th edition, 1978.

[11] Brian Dennis and G. P. Patil. *Lognormal Distributions, Theory and Applications*. Marcel Dekker New York, 1987.

[12] Jack R. Benjamin and C. Allin Cornell. *Probability, Statistics, and Decision for Civil Engineers*. McGraw-Hill Book Company, 1970.

[13] John R. Taylor. *An Introduction to Error Analysis – The Study of Uncertainties in Physical Measurements*. University Science Books, 1997, 2nd edition, 1982.

[14] Unsplash Licence. *www.unsplash.com/license*.

[15] Mark James Wootton, Ying Zhou, John D. Andrews, Roger Smith, John Arul, Gopika Vinod, Hari Prasad Muruva, and Vipul Garg. Optimisation of nuclear reactor primary coolant design and maintenance parameters. *Applied Stochastic Models in Business and Industry*, 40(3):725–745, October 2022.

6   Petri Net Integration Algorithm

The flowchart in figure 29 depicts the process followed by Macchiato to execute an individual Petri Net simulation.

Figure 29: Algorithm for the integration of Petri Nets by Macchiato. Image from Wootton *et al.* 2022 [15], (CC BY 4.0).