eBCSgen: Tutorial

Matej Troják, David Šafránek, Lukrécia Mertová, and Luboš Brim
## Contents

1 Introduction ........................................... 2

2 Getting Started ........................................ 3
   2.1 Galaxy UI basics .................................. 3
   2.2 BioChemical Space Language models ............... 6
      2.2.1 Language syntax .............................. 6
      2.2.2 Editor ........................................ 7
      2.2.3 Example model ............................... 8

3 Analytical Tools ........................................ 10
   3.1 Transition System generator ...................... 10
      3.1.1 Input specification ......................... 10
      3.1.2 Output specification ....................... 10
      3.1.3 Results visualisation ...................... 11
   3.2 Simulation ........................................ 12
      3.2.1 Input specification ......................... 13
      3.2.2 Output specification ....................... 13
      3.2.3 Results visualisation ...................... 13
   3.3 CTL Model Checking ............................... 13
      3.3.1 Input specification ......................... 13
      3.3.2 Output specification ....................... 14
   3.4 PCTL Model Checking .............................. 14
      3.4.1 Input specification ......................... 14
      3.4.2 Output specification ....................... 14
   3.5 PCTL Parameter Synthesis ......................... 14
      3.5.1 Input specification ......................... 14
      3.5.2 Output specification ....................... 15
      3.5.3 Results visualisation ...................... 15
   3.6 Probability Sampling ............................. 15
      3.6.1 Input specification ......................... 15
      3.6.2 Output specification ....................... 16
   3.7 Static Analysis .................................... 16
      3.7.1 Input specification ......................... 16
      3.7.2 Output specification ....................... 17
Chapter 1

Introduction

This tutorial is dedicated to assist users in learning the basic usage of the tool eBCSgen. eBCSgen is used to develop and maintain models written in BioChemical Space language (BCSL) and to analyse them. The eBCSgen is implemented as a part of the Galaxy tool and is available online on the address https://biodivine-vm.fi.muni.cz/galaxy/.

In order to be able to use the tool, it is necessary to learn basics of the Galaxy user interface and syntax of BCSL. With this knowledge, it is then possible to proceed to Analytical tools provided by eBCSgen accompanied by specific visualisations.
Chapter 2

Getting Started

To learn how to use the Galaxy UI for the purposes of this tutorial, it should be sufficient to read and follow the steps described in the following section. Besides this text, Galaxy interactive tours are available to walk the user through Galaxy features and individual tools. However, if you need more information, we recommend the official Galaxy project homepage, where a constantly growing set of tutorials is available.

2.1 Galaxy UI basics

![Figure 2.1: A screenshot of the main page of Galaxy tool. The key components of UI are highlighted by red frames: (1) available eBCSgen Analytical tools; (2) uploading utility; (3) history of your computations; (4) create new BCSL model; (5) available shared data; (6) userspace.](image)

The Galaxy tool is composed of several components. Figure 2.1 shows the homepage of Galaxy tool with highlighted key components:

1. *eBCSgen Analytical tools* – the main functionality of eBCSgen can be found here. Individual tools and how to use them is described in section Analytical tools.
2. **uploading utility** – upload local files and download data from the web by entering URLs. To learn how to use uploading utility, we recommend Galaxy UI tour.

3. **history of your computations** – data space where all files and analysis results are available. This helps to keep track of analysis results origin, allows to easily chain tools, and share reproducible data. To learn how to work with history, we recommend History guide tour.

4. **create new BCSL model** – open an empty file in BCSL editor which allows to create a new model from scratch.

5. **shared data** – published data libraries and histories available as reference data. There are also available example files for this tutorial.

6. **userspace** – it is recommended to register and stay logged in for all times. This enables some features such as analysis of results and saves history for other sessions.

---

**Figure 2.2:** A screenshot of a tool in Galaxy UI. The highlighted components in red frames are: (1) input file to the tool; (2) argument of the tool; (3) hidden advanced options; (4) execution button.

Every tool in the Galaxy UI has its interface, which allows the user to comfortably fill all required arguments for the tool in order to execute it successfully. In Figure 2.2, there are highlighted some of the key components:

1. **tool input files** – Galaxy is based on file transformation principle, i.e. each tool takes an input a file and produces an output file (can be multiple on both ends).

2. **additional arguments** – along with files, the tool might require some additional arguments, e.g. textual field, numerical value, a variant choice.
3. **hidden options** – tool can have some optional or repetitive arguments which can be collapsed/hidden.

4. **execute button** – once all arguments have been filled, the tool can be executed and its results will appear in the history.

![Figure 2.3: A screenshot of a history item in Galaxy UI. The highlighted components in red frames are: (1) ID of the data; (2) ID of the source data; (3) complete information about the data; (4) run the job again; (5) visualise the data; (6) view file content; (7) peek of the file content.](image)

The history is a key component of Galaxy UI. It is a timeline of tool results and allows the user to reproduce the tool execution completely. The background colour of the file indicates its state: green – execution is finished successfully, orange – execution in progress, and red – execution failed (check the bug report). In figure 2.3, there are highlighted some of the key components of an item in the history:

1. **ID of the data** – unique identification in the history.

2. **ID of the source data** – the ID of file used as an input file for the tool.

3. **information** – complete information about the file (e.g. size, format) and job it was created by (e.g. input arguments, execution time).

4. **re-run the job** – possibility to run the job again with the same or modified arguments.

5. **visualise** – displays visualisation options for the data. All displayed visualisations can be used to produce a graphical output for the data. However, we recommend particular visualisation tailored for the data format for each individual tool in section Analytical tools.

6. **view file content** – show entire file content.

7. **peek of the file content** – just a peek of the file content or file statistics.
2.2 BioChemical Space Language models

2.2.1 Language syntax

The BCSL model is composed of several sections: rules contains the set of rules defining the behaviour of the model, inits defines the initial state of the model, definitions allows to assign parameter values, and (optional) complexes defines aliases for complexes.

In Figure 2.4, the context-free grammar of the BCSL is provided. It is defined in extended Backus–Naur form (EBNF), recognised by context-free grammar parser Lark, a parsing library for Python. An example of the model is available in section Example model.

```
model: rules inits definitions (complexes)?

rules: "#! rules" (rule|COMMENT)+
inits: "#! inits" (init|COMMENT)+
definitions: "#! definitions" (definition|COMMENT)+
complexes: "#! complexes" (complex_def|COMMENT)+

init: const? rate_complex (COMMENT)?
definition: param "=" const (COMMENT)?
rule: side "=>" side ("@" rate)? (";" variable)? (COMMENT)?
complex_def: complex_name "=" sequence (COMMENT)?

side: (const? complex "+")* (const? complex)?
complex: (abstract_sequence|sequence) ":=" compartment

rate_complex: sequence ":=" compartment
sequence: (agent ".")* agent
agent: atomic | structure
structure: s_name (" composition ")
composition: (atomic ",")* atomic?
atomic: a_name (" state ")

!state: (digit|letter|"+"|"-"|"*"|"")+
!rate: fun "/" fun | fun
!fun: const | param | rate_agent | fun "+" fun | fun "-" fun | fun "*" fun | fun "**" const | "(" fun ")"
!rate_agent: "[" rate_complex "]"

abstract_sequence: atomic_complex | atomic_structure_complex | structure_complex
atomic_complex: atomic ":" (complex_name|"")
atomic_structure_complex: atomic ":" structure ":" (complex_name|"")
structure_complex: structure ":" (complex_name|"")
variable: ("?" "=": (" complex_name ("," complex_name)+ "))

COMMENT: "//" /[-\n]/
```

Figure 2.4: The context-free grammar of BCSL in EBNF notation. The grammar is simplified – obvious terminals such as names, constants and numbers are omitted.
2.2.2 Editor

BCSL editor is technically an interactive visualisation of a BCSL model. Therefore it can be accessed the same way as all the other visualisations (see Galaxy UI basics) by visualising a model. The key features of the editor are highlighted in Figure 2.5:

1. a rule containing syntax error – a single rule containing syntax error, which is indicated by the red cross on the left to the line number.

2. error message – details of the syntax error detected by syntax checker. The message identifies an unexpected symbol, its exact position in the model and list of expected symbols.

3. position of cursor – information about the current position of the cursor in the model file.

4. action buttons – there are three buttons available: Save modifications to save the current content as a new BCSL model, Undo and Redo to reverse your last modification (resp. to reverse your last undo).

Figure 2.5: A screenshot of a BCSL editor in Galaxy UI. The highlighted components in a red frame are: (1) a rule containing syntax error; (2) console containing error message; (3) position of cursor; (4) action buttons.
2.2.3 Example model

As example model in this tutorial, we use an artificial signalling pathway. This model is available in Shared Data Library or as Shared History including all analysis results demonstrated in Analytical Tools section. The data from both sources can be imported to user workspace. There are actually two separate models – `model.bcs` and `model_param.bcs`, both representing the same system with the exception of parameter values – the latter model has two unknown parameter values. There is actually a third model called `repressilator.bcs`, but this model is only used for demonstration by Simulation tool for its interesting qualitative properties. For details about the syntax of the model, see Language syntax section.

![Diagram of the signalling pathway](image)

Figure 2.6: A simplified scheme depicting the signalling pathway represented by `model.bcs`. The signal transduction starts in extracellular space `ext` and continues inside the cell. There are two signals – one with a positive effect (`sig`) on the pathway and the other one with a negative (`block`).

The example model is also written in Figure 2.7, with its simplified scheme shown in Figure 2.6. It represents a signalling pathway where two external signals, `sig` and `block`, can transfer inside the cell. There, positive signal `sig` can react with protein `P1()` and activate it (denoted by changing its feature `active` from state `off` to `on`). On the other hand, the negative signal `block` can disrupt the activation of `P1()` by blocking its interaction with signal `sig`. Activated protein `P1()` can consequently activate key target protein `P2()`. Its activity is modelled by the same mechanism as in the case of protein `P1()`. 
#! rules
// positive signal
sig {i}::ext ⇒ sig {a}::cell @ (k_sig_2*[sig {a}::ext])/(1+[block {a}::cell])

// blocking signal
block {i}::ext ⇒ block {a}::cell @ k_block_1
block {a}::ext ⇒ block {a}::cell @ (k_block_2*[block {a}::ext])/(1+[sig {a}::cell])

// signal and protein P1 interaction
sig {a}::cell + P1()::cell ⇒ sig {a}.P1()::cell @ param_sig*[sig {a}::cell]*[P1()::cell]
sig {._}.P1()::cell ⇒ sig {._}::cell + P1()::cell @ k_deg*[sig {._}.P1()::cell]
sig {a}.P1( active {off})::cell ⇒ sig {a}.P1( active {on})::cell @ 0.5*[sig {a}.P1( active {off})::cell]
sig {a}.P1()::cell ⇒ sig {a}::cell + P1()::cell @ k_deg*[sig {a}.P1()::cell]

// block and protein P1 interaction
block {a}::cell + P1()::cell ⇒ block {a}.P1()::cell @ param_block*[block {a}::cell]*[P1()::cell]

// proteins P1 and P2 interaction
P1( active {on})::cell + P2()::cell ⇒ P1( active {on}).P2()::cell @ 0.4*[P1( active {on}).P2()::cell]*[P2()::cell]
P1().P2()::cell ⇒ P1()::cell + P2()::cell @ k_deg*[P1().P2()::cell]
P1().P2( active {off})::cell ⇒ P1().P2( active {on})::cell @ k_prod*[P1().P2( active {off})::cell]

#! inits
sig {i}::ext
block {i}::ext
P1( active {off})::cell
P2( active {off})::cell

#! definitions
k_sig_1 = 0.8
k_sig_2 = 0.9
k_block_1 = 0.9
k_block_2 = 0.3
k_deg = 0.3
k_prod = 0.6
param_sig = 0.3
param_block = 0.4

Figure 2.7: The example model named model.bcs used in this tutorial. It consists of 12 rules forming a signalling pathway, initial state, and definition of parameter constants. Note that example model model_param.bcs is identical except the values for the last two parameters param_sig and param_block are not given.
Chapter 3

Analytical Tools

eBCSgen is equipped by several analytical tools: Transition system generator, Simulation, CTL Model checking, PCTL Model checking, PCTL Parameter synthesis, several Static analysis, and Probability Sampling. This chapter describes all of them in details and how to use them in Galaxy platform. Moreover, the Example model is used for their demonstration.

3.1 Transition System generator

In this section, we describe how to generate a transition system from given BCSL model. Following the idea of using approximate models with discrete-time semantics, the obtained transition system is a Discrete Time Markov Chain (DTMC) or a parametric Markov Chain (pMC) [7, 14], depending on whether there are some parameters used in rule rates which do not have defined value in definitions. A step by step Galaxy tour showing how to use the tool can be found here.

3.1.1 Input specification

- **Model file**: Selected model in BCSL language (see BCSL model syntax for details).
- **Bound** (optional): Bound represents the maximal multiplicity of any agent in any state. If not given, an optional bound is computed automatically for potentially infinite systems. It implies the construction of special “hell” state aggregating all states beyond the bound.
- **Advanced Options** (all optional)
  - **Maximal computation time**: Possibility to specify the time (in seconds) boundary for the computation.
  - **Maximal TS size**: Represents approximately the maximal number of nodes in the final transition system.
  - **Precomputed TS file**: Possibility to enter pre-generated transition system for the same model and continue in its generating. Usually connected with the usage of two previous arguments.

*Example*. To demonstrate the usage of the tool, we select `model.bcs` (see Example model) as Model file. All the other optional arguments are unspecified.

3.1.2 Output specification

Transition system generator tool generates a file format `.bcs.ts`, which is JSON with the following structure (order not given):

---

```json
```
• **ordering**: Ordered list of all distinct agents.

• **nodes**: Set of nodes representing states of the transition system. Each of them contains a unique ID and a tuple of numbers, representing the multiplicity of the agent on the respective position from the ordering.

• **edges**: Set of edges representing transitions. Each edge contains ID of the source node \((s)\), and ID of the target node \((t)\), and the probability of the transition \((p)\). In the case of the parametrised model, the probability is replaced by a probability function of parameters.

• **initial**: ID of the initial state.

### 3.1.3 Results visualisation

The transition system can be visualised as a network in an interactive chart called *Transition system visualisation* (see Figure 2.3 point 5 to learn how to visualise data). The nodes represent the states of the transition system, while the edges represent transitions between the particular states. It is possible to adjust the position of nodes by dragging them, highlight their content by clicking on them, and also display a particular transition by clicking on edge. The initial state from which was the transition system generated is shown in the orange colour.

**Figure 3.1**: Visualisation of the transition system computed for **model.bcs**. The key features are highlighted in red frames: (1) highlighted edge; (2) contents of the highlighted object; (3) options for graph details; (4) control panel for zooming options; (5) navigation panel.

Figure 3.1 shows the visualisation for **Example**. The highlighted features are:

1. **highlighted edge** – edges and nodes can be highlighted by left-clicking on them.

2. **contents of the highlighted object** – the contents of the clicked node or edge are displayed. An edge contains particular interaction responsible for the transition with evaluated probability (or a probability function of parameters). A node contains agent counts present in the state.

3. **options for graph details** – it is possible to adjust three features of the graph by switch buttons: top – nodes with a transition to special state “hell” (if any) are highlighted by a squared border, middle – self-loops are hidden, bottom – special state “hell” (if any) is hidden.
4. **zooming options** – zooming and centring buttons. Zooming can also be achieved by mouse wheel.

5. **navigation panel** – vertical and horizontal navigation in the chart. It can also be achieved by left-clicking of mouse followed its dragging on an empty space in the chart.

![Figure 3.2](image)

**Figure 3.2:** Visualisation of time series data as a result of simulation tool. The top picture shows result of stochastic simulation from Example 1 and the bottom picture shows result of deterministic simulation from Example 2.

### 3.2 Simulation

In this section, we describe how to simulate given BCSL model. Simulation tool provides a stochastic simulation of the model using an adapted variant of the standard Gillespie algorithm. Despite the natural probabilistic semantics, the deterministic simulation of the model is also possible. There are available step by step Galaxy tours showing how to use stochastic and deterministic variants of the tool.
3.2.1 Input specification

- **Model file**: Selected model in BCSL language (see BCSL model syntax for details).

- **Choose simulation method**:
  - For the stochastic simulation of the system, the **Number of runs** can be specified (default 1) to average individual runs (due to the stochastic nature of the method).
  - For the deterministic simulation of the system, a set of constructed ODEs is simulated. In this case, the **Volume** argument has to be filled in to deal with molecular counts in the initial state and **Time step** argument defining distance between two time points.

- **Maximum simulation time**: specification of simulation time.

**Example 1.** To demonstrate the usage of the tool for the case of stochastic simulation, we select `repressilator.bcs` (see Example model) as Model file. Then choose 2 runs and 200 for the simulation time.

**Example 2.** To demonstrate the usage of the tool for the case of deterministic simulation, we select `repressilator.bcs` (see Example model) as Model file. Then choose value 1 for the volume, 0.01 for the time step, and 200 for the simulation time.

3.2.2 Output specification

Simulation tool generates time series with respect to each agent. The result is stored in a `.csv` file, where the first column represents time, and the rest of the columns store values of for individual agents. Note that specified simulation time might not be achieved in the case of stochastic simulation when there are multiple runs due to averaging the runs.

3.2.3 Results visualisation

The simulation time series can be visualised in an interactive chart called *Simulation Plot* (see Galaxy UI basics to learn how to visualise data). Please note there are many available visualisations for this type of output (due to general `.csv` format) sorted alphabetically. The plot provides basic functionality such as zooming, curves filtering, and exporting `png` picture. The visualisation of both examples is available in Figure 3.2.

3.3 CTL Model Checking

CTL model Checking can be performed on non-parametric models with respect to the given CTL formula. The Model Checking procedure is performed using a package called `pyModelChecking`.

3.3.1 Input specification

- **TS file**: Generated transition system for the desired model (computed using Transition system generator).

- **CTL formula**: Given formula expressing property to be checked on the transition system.

**Example.** To demonstrate the usage of the tool, we select transition system computed for `model.bcs` (see Example model) as TS file. For CTL formula we choose `E(F([ P2(active{on})::cell > 0]))`. 
3.3.2 Output specification

The textual result is stored in a .ctl.check file. The text contains number of states satisfying the formula and the boolean value for the initial state.

For our Example we obtain boolean result true, as the formula is satisfied in the initial state.

3.4 PCTL Model Checking

PCTL model Checking can be performed on non-parametric models with respect to the given PCTL formula expressing a property regarding the probability of an event to occur. The tool allows checking whether a given probability threshold is satisfied or find the probability of satisfaction for the given path formula. The Model Checking procedure is performed using an external tool called Storm. A step by step Galaxy tour showing how to use the tool can be found here.

3.4.1 Input specification

- **TS file**: Generated transition system for the desired model (computed using Transition system generator).

- **PCTL formula**: Given formula expressing property to be checked on the transition system.

  *Example 1.* To demonstrate the usage of the tool when the probability threshold of PCTL formula is given, we select transition system computed for model.bcs (see Example model) as TS file. For PCTL formula we choose P <= 0.2 [ F P2(active{on})::cell > 0 ].

  *Example 2.* To demonstrate the usage of the tool when the probability threshold of PCTL formula is not given, we select transition system computed for model.bcs (see Example model) as TS file. For PCTL formula we choose P =? [ F P2(active{on})::cell > 0 ].

3.4.2 Output specification

The textual result is stored in a .storm.check file. The text contains some details about Storm model checker performance. Finally, the boolean or numerical result can be found as Result (for initial states). Additionally, any errors Storm encountered are also shown in this file.

For our examples, we obtain boolean result true for Example 1 and probability 0.1304038484 for Example 2.

3.5 PCTL Parameter Synthesis

PCTL parameter Synthesis can be performed on parametric models with respect to the given PCTL formula expressing a property regarding the probability of an event to occur. If the formula has defined probability threshold, then the partitioning of the given parameter space (defined by the user) to regions which satisfy (resp. violate) the property is computed. If the threshold is not given, a probability function of parameters is computed instead, which evaluates to the probability of satisfaction for particular parametrisation. PCTL Parameter Synthesis procedure is performed using an external tool called Storm. A step by step Galaxy tour showing how to use the tool can be found here.
3.5.1 Input specification

- **TS file**: Generated transition system for the desired model (computed using Transition system generator).

- **PCTL formula**: Given formula expressing property to be used to perform parameter synthesis on the transition system.

- **Intervals**: Relevant only in the case when PCTL formula has defined probability threshold. Then, an interval of allowed values has to be specified for each unknown parameter in the model, together forming parameter space. If PCTL formula does not have defined probability threshold (i.e. starts with $P = ?$), the intervals should not be defined.

**Example 1.** To demonstrate the usage of the tool when the probability threshold of PCTL formula is given, we select transition system computed for model_param.bcs (see Example model) as TS file. For PCTL formula we choose $P \leq 0.2 \left[ F P2(\text{active(on)}):\text{cell} > 0 \right]$ and enter the following intervals – param_sig: $[0.1, 0.6]$ and param_block: $[0.05, 1.0]$.

**Example 2.** To demonstrate the usage of the tool when the probability threshold of PCTL formula is not given, we select transition system computed for model_param.bcs (see Example model) as TS file. For PCTL formula we choose $P = ? \left[ F P2(\text{active(on)}):\text{cell} > 0 \right]$, and intervals are not specified.

3.5.2 Output specification

The textual result contains some details about Storm model checker performance. It is stored in a .storm.sample or .storm.regions file, for with and without threshold respectively.

- .storm.sample file contains computed probability function of parameters, which can be evaluated to the probability of satisfaction for particular parametrisation. This result can be further analysed in Probability Sampling tool (demonstrated using Example 2).

- .storm.regions file contains segmentation of specified parameter space to regions where the given property is satisfied and violated. For some of the regions, the satisfiability might not be decided due to efficiency reasons.

3.5.3 Results visualisation

The Parameter Synthesis results stored in .storm.regions file can be visualised in Parameter synthesis result visualisation (see Galaxy UI basics to learn how to visualise data). The visualisation shows green, red, and grey regions where the satisfiability of PCTL formula is True, False, and unknown, respectively. The visualisation allows user to change chosen parameters on X and Y axis, which is particularly useful when there are more than two parameters. In that case, slices of parameter space in other dimensions can be chosen. The visualisation of Example 1 is available in Figure 3.3 left.

3.6 Probability Sampling

Probability Sampling is a tool dedicated for the sampling of computed probability function of parameters as a result of Parameter Synthesis. A step by step Galaxy tour showing how to use the tool can be found here.
3.6.1 Input specification

- **Storm results file**: Selected PCTL Parameter Synthesis result in `.storm.sample` format.
- **Intervals**: An interval of allowed values have to be specified for each unknown parameter in the function, together forming parameter space to be sampled. Moreover, for each interval, it has to be specified how many samples should be taken.

*Example*: To demonstrate the usage of the tool, we select the output file of [Example 2](Parameter Synthesis) as Storm results file. We enter the following intervals – `param_sig`: `[0.1, 0.6]` with 101 samples and `param_block`: `[0.05, 1.0]` with 101 samples.

3.6.2 Output specification

The result of this tool is an `html` file containing a graphical output of the sampling. The output is similar to visualisation mentioned in Parameter Synthesis, but the legend of the visualisation is different – there is a scale assigning a colour to probability values. It can be viewed directly (see Figure 2.3, item 6). The visualisation of [Example](Parameter Synthesis) is available in Figure 3.3 right.

3.7 Static Analysis

Static analysis techniques are performed directly on the model syntax and are therefore extremely fast by definition.
In particular, we have developed unreachability static analysis method which can be used to check whether a single agent is unreachable before enumerating the entire transition system of the model. This analysis is based on the idea that in order to reach an agent, there has to be a rule which creates it or a lesser specified agent; in other words, the particular states inside of the agent are changed at some point. A step by step Galaxy tour showing how to use the tool can be found here.

The second method aims to detect (potentially) redundant rules in the model. This can be useful in large models to detect potentially concurrent rules and to make the model more compact. The redundant rules are denoted by adding comments at the end of such potentially redundant rules and connecting them by an integer identifier. A step by step Galaxy tour showing how to use the tool can be found here.

The third static analysis technique is used to reduce the context of the model to a minimal level in order to produce a smaller and more abstract model. The resulting model still preserves some properties while making the analysis of the model computationally simpler. A step by step Galaxy tour showing how to use the tool can be found here.

3.7.1 Input specification

- **Model file**: Selected model in BCSL language (see BCSL model syntax for details).
- **Choose static analysis method**:
  - **Static unreachability**: This option computes unreachability for the particular agent given in Complex agent.
  - **Rule redundancy elimination**: This option marks pairs of rules which might be redundant in the model.
  - **Context based reduction**: This option removes all states from all agents in the model, reducing context to a minimum. Rule with no sense (e.g. rules where left- and right-hand sides are equal) are removed.

  **Example 1**. To demonstrate the Static unreachability method, we select `model.bcs` (see Example model) as Model file and enter the expression `sig{i}.P1()::cell` as Complex agent.

  **Example 2**. To demonstrate the Rule redundancy elimination method, we select `model.bcs` (see Example model) as Model file.

  **Example 3**. To demonstrate the Context based reduction method, we select `model.bcs` (see Example model) as Model file.

3.7.2 Output specification

The results of the tool depend on the particular method which was chosen.

For Static unreachability method, the output is textual and states whether the agent of interest cannot be reached and can possibly be reached in the model. For **Example 1**, the output states:

The given agent
\[ \text{sig}\{i\}.P1()::\text{cell} \]

cannot be reached in the model.
For Rule redundancy elimination method, the output is a .bcs model. In the first case, just some comments are added to the file indicating potential (if any) redundant rules. For Example 2, the output model contains two rules which are potentially redundant:

\[
\begin{align*}
\text{sig}(a).P1() &\rightarrow \text{sig}(a)\cdot\text{cell} + P1()\cdot\text{cell} \otimes k_{\text{deg}} \times \{\text{sig}(a).P1()\cdot\text{cell}\} \quad \text{// redundant #{1}} \\
\text{sig}(_).P1() &\rightarrow \text{sig}(a)\cdot\text{cell} + P1()\cdot\text{cell} \otimes k_{\text{deg}} \times \{\text{sig}(_).P1()\cdot\text{cell}\} \quad \text{// redundant #{1}}
\end{align*}
\]

For Context based reduction method, the output is a .bcs model. In the output model, all states are absent, leaving only rules for complex formation and dissociation, production, and degradation. For Example 3, we recommend to run Transition system generator and compare visualisations for the original and reduced model. The transition system of the reduced model is significantly smaller but also preserves fewer details.