Getting Started With Sibilla

Michele Loreti

June 30, 2021

Contents

1 Introduction

SIBILLA is a Java framework designed to support analysis of Collective Adaptive Systems. These are systems composed by a large set of interactive agents that cooperate and compete to reach local and global goals. The framework is structured in three main packages:

- a *simulation package*, supporting system simulation;
- an *analysis package*, implementing analysis techniques based on statistical model checking;
- a *modelling package*, supporting different specification languages.

SIBILLA is not based on a specific formalism. Indeed, thanks to the use of recurrent patterns, all the packages can be easily extended with new features.

1.1 How to install

SIBILLA is available at the GitHub repository accessible via the following link:

https://github.com/quasylab/sibilla

The simplest way to download SIBILLA is to clone the repository locally by executing in a shell

```
git clone https://github.com/quasylab/sibilla.git
```

After that a directory named `sibilla`, that contains all the source files, is created. You can run the available gradle script to build the tool (replace `./gradlew` with `.\gradlew.bat` in Windows):

```
cd sibilla
./gradlew build
./gradlew installDist
```

The compiled tool is then available in the folder `shell/build/install/sshell`.

2 Population Models

In this section we will show how population models can be defined in SIBILLA.

2.1 Formal definition

We briefly recall here that a Population Continuous Time Markov Chain (PCTMC) model is a tuple $M = (\mathbf{X}, \mathcal{D}, \mathcal{T}, \mathbf{d}_0)$ where:

- $\mathbf{X} = (X_1, \ldots, X_n)$ is a vector of variables;
- each X_i takes values in a *finite* or *countable* domain $\mathcal{D}_i \subset \mathbb{R}$;
- $\mathcal{D} = \mathcal{D}_0 \times \cdots \times \mathcal{D}_n = \Pi_i \mathcal{D}_i$;
- $\mathbf{d}_0 \in \mathcal{D}$ is the *initial state* of the model;
- $\mathcal{T} = \{\tau_1, \ldots, \tau_m\}$ is the set of *transitions* $\tau_i = (\ell, \mathbf{s}, \mathbf{t}, r)$ where:
  - ℓ is the *label* of the transition;
  - $\mathbf{s} \in \mathbb{R}^n_{\geq 0}$, is the *pre-vector*;
  - $\mathbf{t} \in \mathbb{R}^n_{\geq 0}$, is the *post-vector*;
  - $r : \mathcal{D} \to \mathbb{R}_{\geq 0}$ is a *rate function* such that for any $\mathbf{d} \in \mathcal{D}$, if $\mathbf{d} - \mathbf{s} + \mathbf{t} \notin \mathcal{D}$ then $r(\mathbf{d}) = \mathbf{0}$.

Let $M = (\mathbf{X}, \mathcal{D}, \mathcal{T}, \mathbf{d}_0)$, $\mathbf{d}_1, \mathbf{d}_2 \in \mathcal{D}$ and $\tau_i = (\ell, \mathbf{s}, \mathbf{t}, r) \in \mathcal{D}$. We let $\rightarrow_{\tau_i} \subseteq \mathcal{D} \times \mathbb{R}_{>0} \times \mathcal{D}$ denote the transition relation induced by transition τ_i:

$$\frac{r(\mathbf{d}_1) = \lambda \neq 0 \quad \mathbf{d}_2 = \mathbf{d}_1 - \mathbf{s} + \mathbf{t}}{\mathbf{d}_1 \xrightarrow{\lambda}_{\tau_i} \mathbf{d}_2}$$

We say that τ_i is *enabled* in $\mathbf{d}_1$ if and only if $r(\mathbf{d}_1) > 0$. Finally, function $\rho_{\tau_i} : \mathcal{D} \times \mathcal{D} \rightarrow \mathbb{R}_{\geq 0}$ is used to denote the rate of a transition τ_i from $\mathbf{d}_1$ to $\mathbf{d}_2$:

$$\rho_{\tau_i}(\mathbf{d}_1, \mathbf{d}_2) = \begin{cases} r(\mathbf{d}_1) & \mathbf{d}_2 = \mathbf{d}_1 - \mathbf{s} + \mathbf{t} \\ 0 & \text{otherwise} \end{cases}$$

Let $M = (\mathbf{X}, \mathcal{D}, \mathcal{T}, \mathbf{d}_0)$, $\mathbf{d}_1, \mathbf{d}_2 \in \mathcal{D}$ and $\tau_i = (\ell, \mathbf{s}, \mathbf{t}, r) \in \mathcal{D}$.

Function $\rho_{\mathcal{T}} : \mathcal{D} \times \mathcal{D} \rightarrow \mathbb{R}_{\geq 0}$ is used to denote the rate of a transition in M from $\mathbf{d}_1$ to $\mathbf{d}_2$:

$$\rho_{\mathcal{T}}(\mathbf{d}_1, \mathbf{d}_2) = \sum_{\tau_i \in \mathcal{T}} \rho_{\tau_i}(\mathbf{d}_1, \mathbf{d}_2)$$

We let $\rightarrow_{\mathcal{T}} \subseteq \mathcal{D} \times \mathbb{R}_{>0} \times \mathcal{D}$ denote the transition relation induced by transitions $\mathcal{T}$:

$$\frac{\rho_{\mathcal{T}}(\mathbf{d}_1, \mathbf{d}_2) = \lambda \neq 0}{\mathbf{d}_1 \xrightarrow{\lambda}_{\mathcal{T}} \mathbf{d}_2}$$

2.2 Population Models in Sibilla

SIBILLA provides a module that simplifies the specification of population models. In this section we will use a simple scenario, based on a classical epidemic model, to introduce the basic features of our specification language.

The *SIR model* is one of the simplest compartmental models. The goal of the model is to represent the spread of an infection disease. The model consists of three species (or compartments):

S The number of susceptible individuals. When a susceptible and an infectious individual come into "infectious contact", the susceptible individual contracts the disease and transitions to the infectious compartment.

I The number of infectious individuals. These are individuals who have been infected and are capable of infecting susceptible individuals.

R The number of removed (and immune) or deceased individuals. These are individuals who have been infected and have either recovered from the disease and entered the removed compartment, or died. It is assumed that the number of deaths is negligible with respect to the total population. This compartment may also be called "recovered" or "resistant".

The SIR model can be described in SIBILLA as follows:

```
param lambdaMeet = 1.0;      /* Meeting rate */
param probInfection = 0.25;  /* Probability of Infection */
param recoverRate = 0.05;    /* Recovering rate */

const startS = 90;           /* Initial number of S agents */
const startI = 10;           /* Initial number of I agents */
```

```
species S;
species I;
species R;

rule infection {
    S|I -[ #S*%I*lambdaMeet*probInfection ]-> I|I
}

rule recovered {
    I -[ #I*recoverRate ]-> R
}

system init = S<startS>|I<startI>;
```

Model parameters. A specification can contain a set of *parameters*. This are *real values* that can be changed after the model is loaded. In the SIR model above, we have three parameters:

```
param lambdaMeet = 1.0;      /* Meeting rate */
param probInfection = 0.25;  /* Probability of Infection */
param recoverRate = 0.05;    /* Recovering rate */
```

`lambdaMeet` is the *meeting rate* while `probInfection` is the probability that an agent `S` get infected when it meets an agent `I`. Finally, `recoverRate` is the recovering rate.

Constants. Constants are used to associate specific values to names. In our example, for instance, we use `startS` and `startI` to indicate the initial number of agents susceptibles and infected (90 and 10, respectively).

Rules. A set of rules are used to describe system dynamics. The simper form of the rule is the following:

```
rule <rulename> {
  <species>(|<species>)* -[ <exp> ]->  <species>(|<species>)*
}
```

where `<rulename>` is the name of the rule, `<species>` is a species name and `<exp>` is the expression used to compute the rule rates. Expressions are build by using standard mathematical operators and the two special operators `%X` and `#X`: the former returns the *fraction of agents of species* `X` while the latter amounts to the *number of agents of species* `X`. In our SIR scenario, such operators are used in the rule `infection`:

```
rule infection {
    S|I -[ #S*%I*lambdaMeet*probInfection ]-> I|I
}
```

Systems. The initial configurations of a system are described in the form

```
system <name> = <species>(|<species>)*;
```

When multiple copies of the same species `X` are in the system, the form `X<n>` can be used, where n is a numerical value.

3 Sibilla Shell

sshell is an interactive shell that can be used to interact with the tool. To run the shell one needs to execute the script `sshell` (or `sshell.bat`) in the folder `shell/build/install/sshell/bin`. Below a session that

can be used to analyse the SIR model described in the previous section:

```
module "population"
load "examples/sir/sir.pm"
init "init"
add all measures
deadline 100
dt 1.0
replica 100
simulate
save output "./results" prefix "sir" postfix "__"
```

In the following a detailed list of commands that are available in the shell:

`modules`: show the list of available modules.

`module <name>`: load the module language. Currently only `population` is available.

`load <filename>`: load the specification from the given file name.

`env` (or `environment`): shows the current assigment of paranmeters.

`set "<name>" <real>`: set the given parameter to the given value.

`clear`: reset the shell content.

`reset "<name>"`: reset the given parameter to the default value.

`states`: show the list of initial configurations.

`init "<name>"`: set initial state to the given value.

`replica <num>`: set the number of simulation replicas.

`deadline <real>`: set the simulation deadline.

`dt <real>`: set the sampling time.

`measures`: show the list of available measure.

`add measure "<name>"`: add the given measure to the ones collected during the simulation.

`add all measures`: add all the available measures to the simulation.

`remove measure "<name>"`: remove the given measure from the ones collected during the simulation.

`remove all measures`: remove all measures from the simulation.

`save output "<output_folder>" prefix "<prefix>" postfix =<"postfix">`: save the collected results in the fiven folder. Each measure will be saved in a `csv` file named `<prefix><measurename><postfix>.csv`. The generate file will contain three columns with: time of sampled value, mean, variance and standard deviation.

`run "<script_file>"`: execute the script in the given file name.

`quit`: terminates the execution.