Verifying and Analyzing Systems under scheduling policies with SSpinJa

The input of our tool includes 3 files:

- *process program* (*.pml),
- *process attribute* (*.proc), and
- *scheduling policy*(*.sch)

The corresponding property can be specified in the *process program* and/or in the *scheduling policy*. To perform the verification and analysis with SSpinJa tool, we do these steps.

- *First*, we convert the description of the scheduling strategy (including the *process attributes* and the *scheduler description*) into the information necessary for performing the scheduling tasks using the following command.

  ```
  java -jar schedulerDSL.jar -proc pri.proc -sch pri.sch
  ```

  where `schedulerDSL.jar` is the SSpinJa library file, with this example, `pri.proc` and `pri.sch` is the *priority* policy in the DSL. As a result, the following files are generated.

```
schedulerinfo.dat
CTLFormula.java
ProcessCollection.java
ProcessCollection_priorityOrder.java
ProcessCollectionBase.java
ProcessSet.java
RunningSet.java
SchedulerObject.java
SchedulerObject_Priority.java
SchedulerProcess.java
SchedulerProcess_Priority.java
SchedulerProcessBase.java
SchedulerState.java
SortedProcessCollectionBase.java
StaticProperty.java
StaticProperty_Priority.java
```

  These *java* files implement the following things:

  - the processes of the system (files: *SchedulerProcess*, *SchedulerProcess_Priority*, *SchedulerProcessBase*, *StaticProperty* and *StaticProperty_Priority*) to specify the information of the processes (e.g. the attributes of the processes),
  - the process collections (files: *ProcessCollection*, *ProcessCollection_priorityOrder*, *ProcessCollectionBase*, *ProcessSet*, *RunningSet*, and *SortedProcessCollectionBase*) to define the collections with the ordering methods used by these collections,

- the scheduler (files: *SchedulerObject* and *SchedulerObject_Priority*) to implement the behaviors of the scheduler (i.e. handling the scheduling tasks),
- the analyzing information (files: *CTLFormula* and *SchedulerState*) for the analysis behaviors of the system ; these files express the property to be checked and the information of the system state.

The behaviors of the scheduler (i.e. handling the *scheduling event*s) are implemented in *SchedulerObject* file.

The main functionalities of the scheduler are defined and represented by the following the data structures and a set of methods defined in the *SchedulerObject* as follows.

- The variables (e.g. clock variables) and data structures (e.g. process collections);
- The methods to handle the scheduling tasks (e.g. the events `select_process`, `new_process`, `preTake`, `postTake`, etc.);
- Interface methods (e.g. `sch_api`, `sch_get`) for the interaction between the processes and the scheduler;
- The methods for handling the time managed by the scheduler (e.g. `inc_time`, `dec_time`, `time_out`, etc.);
- Encoding methods (e.g. `encode`, `decode`) for storing the states of the scheduler;
- Analysis methods (e.g. `schedulerCheck`, `stateCheck`, etc.) for analysis the behaviors of the system ;
- Other utilities methods (e.g. `getInstance`, `isTime`, etc.) used by the scheduler.

We note that these methods (functions) are used by the search algorithm to search the system states. The states of the system to be visited are determined by this algorithm based on the scheduling strategy specified in the DSL.

- *Second*, the *process program* (*.pml) in Promela is compiled into the process model in Java using the following command. The result of this step is the model of the processes in Java (`SchedulerPanModel.java`).

```
java -cp sspinja.jar sspinja.Compile example.pml
```

where `sspinja.jar` is the library file and `example.pml` is a *process program* in Promela.

- *Third*, we compile the process model in Java into Java bytecode. This step automatically compiles and links the `SchedulerPanModel.java` with the files generated from the scheduling strategy description in step 1.

```
javac -cp sspinja.jar sspinja/SchedulerPanModel.java
```

- *Fourth*, we perform this command to do the verification and the analysis

```
java -cp sspinja.jar;. sspinja.SchedulerPanModel
```

Our tool will indicate the **verification result**, which contains the information: **the error** (if exist, with the counterexample), **number of states**, **memory usage**, and **the time for verifying** as shown in Figure 1. *For verifying the corresponding property, our tool will indicate the satisfaction and the witness* (as depicted in Figure 2).

```
...
assertion violated (balance >= 0)
0.(proc 0 trans 59): run VerificationCase2_0()
1.(proc 1 trans 55): printf("config3");; balance = 10; amount = 15; run login_deposit_0();
run login_withdraw_0(); run logout_0(); run deposit_0(); run withdraw_0()
2.(proc 3 trans 62): (status == 0); status = 2
3.(proc 3 trans 62): (status == 0); status = 2
4.(proc 3 trans 62): (status == 0); status = 2
5.(proc 3 trans 62): (status == 0); status = 2
6.(proc 3 trans 62): (status == 0); status = 2
7.(proc 3 trans 62): (status == 0); status = 2
8.(proc 3 trans 62): (status == 0); status = 2
9.(proc 3 trans 62): (status == 0); status = 2
10.(proc 3 trans 62): (status == 0); status = 2
11.(proc 6 trans 65): (status == 2); _bwithdraw0_0 = false; if; balance = (balance -
amount); status = 0; if
-----------------------------------------------

State-vector 49 byte, depth reached 11, errors: 21
      89 states, stored
      68 states, matched
     157 transitions (= stored+matched)
       0 atomic steps
8.00584 memory usage (Mbyte)

sspinja: elapsed time 22.00 miliseconds
sspinja: rate     4045 states/second
8.75444 real memory usage (Mbyte)
```

Figure 1: Result of the verification (counterexamples)

```
...
+ Check system (start state: 705970) with: AG<=2 (balance >= 5): Satisfied
-----------------------------------------------
AG<=2 (balance >= 5)
0. (proc 0 trans 43): run VerificationCase0_0()
1. (proc 1 trans 41): printf("config1");; balance = 10; amount = 5; run login_deposit_0();
run login_withdraw_0(); run logout_0(); run deposit_0(); run withdraw_0()
2. (proc 3 trans 46): (status == 0); status = 2
3. (proc 6 trans 49): (status == 2); _bwithdraw0_0 = false; if; balance = (balance -
amount); status = 0; if
-----------------------------------------------
-----------------------------------------------
AG<=2 (balance >= 5)
0. (proc 0 trans 43): run VerificationCase0_0()
1. (proc 1 trans 41): printf("config1");; balance = 10; amount = 5; run login_deposit_0();
run login_withdraw_0(); run logout_0(); run deposit_0(); run withdraw_0()
2. (proc 3 trans 46): (status == 0); status = 2
3. (proc 4 trans 47): (status != 0); status = 0
-----------------------------------------------
-----------------------------------------------
AG<=2 (balance >= 5)
0. (proc 0 trans 43): run VerificationCase0_0()
1. (proc 1 trans 41): printf("config1");; balance = 10; amount = 5; run login_deposit_0();
run login_withdraw_0(); run logout_0(); run deposit_0(); run withdraw_0()
2. (proc 2 trans 45): (status == 0); status = 1
3. (proc 5 trans 48): (status == 1); balance = (balance + amount); status = 0
-----------------------------------------------
-----------------------------------------------
AG<=2 (balance >= 5)
0. (proc 0 trans 43): run VerificationCase0_0()
1. (proc 1 trans 41): printf("config1");; balance = 10; amount = 5; run login_deposit_0();
run login_withdraw_0(); run logout_0(); run deposit_0(); run withdraw_0()
2. (proc 2 trans 45): (status == 0); status = 1
3. (proc 4 trans 47): (status != 0); status = 0
-----------------------------------------------

State-vector 49 byte, depth reached 11, errors: 5
      39 states, stored
      30 states, matched
      69 transitions (= stored+matched)
       0 atomic steps
8.00270 memory usage (Mbyte)

sspinja: elapsed time 54.00 miliseconds
sspinja: rate      722 states/second
8.74844 real memory usage (Mbyte)
```

Figure 2: Result of the analysis (witnesses)