CMSB 2019
Spike – appendix

Jacek Chodak*, Monika Heiner

Computer Science Institute, Brandenburg Technical University
Postbox 10 13 44, 03013 Cottbus, Germany
jacek.chodak@b-tu.de, monika.heiner@b-tu.de
https://www-dssz.informatik.tu-cottbus.de

A Spike Configuration Script (SPC) specification

SPC is a structured data-oriented configuration script. The SPC syntax is concise and human readable, requiring as little effort as possible from the user. This specification is based on RFC 8259 [15].

A.1 Data model

SPC describes a data model, which is a labelled, directed graph. The nodes of the graph are data such as: string, number, object, range, reference, or result of string concatenation.

A.2 Grammar

SPC consists of a set of tokens which include strings, numbers, keywords and structural characters. The structural characters are:

```
'[' - left square bracket - beginOfArray,
']' - right square bracket - endOfArray,
'{' - left curly bracket - beginOfObject,
'}' - rght curly bracket - endOfObject,
':' - colon - nameSeparator,
';' - semicolon - valueSeparator,
```

```
SPC-text = ws value ws
```

White space (ws) is allowed before or after structural characters.

```
ws = *(space / horizontal tab
       / line feed or new line
       / carriage return)
```

* Corresponding author

Values The value is an array, string, number, boolean, range, reference, result of string concatenation or one of two literal names: true, false.

Objects An object is a structure represented by a pair of curly brackets. It can contain zero or more members. A member is a pair of name and value. A name is an identifier, which can contain alias, followed by a colon, which separate the name from the value. An alias is an alternative member name.

```
object =  beginOfObject
            [member *(valueSeparator member)]
          endOfObject

member = identifier nameSeparator [alias] value

alias = identifier nameSeparator

identifier = (A–Za–z\_)*(A–Za–z0–9\_)
```

Arrays An array is a structure represented by a pair of square brackets. It can contain zero or more values of the same type separated by comma.

```
array =  beginOfArray
           [value *(arrayValueSeparator value)]
         endOfArray

arrayValueSeparator = ,
```

Booleans A boolean is represented by one of two literal names: true, false.

```
boolean = true / false
```

Numbers A number is represented by decimal digits, with the base of 10. It starts with an integer component, which can be followed by a fraction or exponent part. A fraction part is represented by a decimal point followed by digits.

```
number = int [frac] [exp]

int = (0) / (1–9)*(0–9)

decimalPoint = .
```

```
frac = decimalPoint +(0-9)

exp = (e / E) (plus / minus ) +(0-9)

plus = +

minus = -

+ = 1* = at least once
```

Strings A string is represented be a pair of quotation marks, which mark the beginning and end of the string.

```
string = quotationMark *char quotationMark

quotationMark = "
```

The concatenation operator allows to join multiple values to a single string. Before joining, a value is evaluated to string data.

```
string concatenation +(value)

concatenation = <<
```

Ranges A range is represented by set of three numbers separated by colon. The first number represents the beginning of the range, the second the step of incrementation, and the third the end of range.

```
range = number 2*(rangeSeparator number)

rangeSeparator = :
```

References A reference is represented by a set of identifiers separated by a dot. It allows for access to an object member.

```
reference = identifier *(dot identifier)

dot = .
```

Keywords / reserved member names The name of the following keywords are reserved member names and can't be used to assign new values.

```
as
configuration
constants
csv
export
from
import
interval
model
name
observers
places
range
sbml
simulation
transitions
type
```

Comments A comment is a text, which is not evaluated. It can be defined in one line up to the end of the line, as well as a block.

```
oneLineComment = startOneLineComment *(char) endLine

startOneLineComment = //

endLine = end of line = new line | end of file

blockComment = startBlockComment *(char) endBlockComment

startBlockComment = /*

endBlockComment = */
```

B Example

The following example presents an experiment configuration exploiting the scanning options of Spike. To check how a model behaves under different hybrid simulation solvers, the set of solvers needs to be defined. Here, the set contains two solvers: static and HRSSAacc. The set will trigger the process of configuration branching. In this case, two configuration branches will be created. Additionally,

the experiment requires to scan one option per solver, e.g. the relative tolerance (relTol) of static solver and the fluctuation ratio (fluctRatio) of HRSSAacc. It can be seen from the example below, the option relTol will be scanned over two arguments, which gives two branches for the static solver. The option fluctRatio will be scanned over a range of arguments, giving three branches for the HRSSAacc solver. This gives in total five configuration branches.

```
// Configuration of a simulation
// Line comment
/*
   Block comment
 */

// Import - exactly one model
import: {
  /* relative path to the place(current/working directory)
   * of spike execution
   */
  from: "model/EucaryoticCell12_1.andl";
}

configuration: {

  simulation: {
    name: "exampleEucaryoticCell"; // Name of a simulation
    type: hybrid; // continuous, stochastic, hybrid
    solver: [
      static: {
        threads: 1;
        runs: 1;
        // "ARK", "BDF", "ADAMS"
        odeSolver: "ADAMS";
        iniStep: 0.1;
        /*
         * "CVDense", "CVSpgmr", "CVDiag", "CVSpbcg",
         * "CVSptfqmr"
         */
        linSolver: "CVDense";
        relTol: [1e-5, 2e-3];// Relative tolerance
        absTol: 1.0e-10;
        autoStepSize: true;
        reductResultingODE: true;
        checkNegativeVal: false;
        outputNoiseVal: false;

        fluctRatio: "none";
      },
      HRSSAacc: {
        threads: 1;
        runs: 1;
```

```
44          // "ARK", "BDF", "ADAMS"
45          odeSolver: "ADAMS";
46          iniStep: 0.1;
47          /*
48           * "CVDense", "CVSpgmr", "CVDiag", "CVSpbcg",
49           * "CVSptfqmr"
50           */
51          linSolver: "CVDense";
52          relTol: 1e-5;
53          absTol: 1.0e-10;
54          autoStepSize: true;
55          reductResultingODE: true;
56          checkNegativeVal: false;
57          outputNoiseVal: false;
58          //
59          // Fluctuation ratio
60          fluctRatio: [0.2:0.1:0.4];
61          // Apply interface places
62          applyInterfacePlaces: true;
63        }
64      ];
65
66      //range: 0:2:100;// start:step:end
67      // or interval
68      /*
69       * range.start = interval.start;
70       * range.end = interval.end;
71       * range.step = (interval.end - interval.start)
72       *              / interval.splitting;
73       */
74      interval: 0:100:100;// start:spliting:end
75
76      export: {
77        // Array of places to save (if empty export all)
78        places: [];// all places
79        //places: c[];// coloured places
80        //places: u[];// uncoloured places
81        //transitions: [];// all transitions
82        //transitions: c[];// all coloured transitions
83        //transitions: u[];// all uncoloured transitions
84        csv: {
85          sep: ";";// Separator
86          // File name
87          file: name << "_" << type << "_"
88              << solver << "_" << solver.relTol
89              << "_" << solver.fluctRatio << ".csv";
90        }
91      }
92    }
93  }
```

C Supported simulation algorithms

Spike supports three simulation types: stochastic, continuous and hybrid [4]; each comes with several algorithms:

Stochastic simulation follows basically the standard Gillespie algorithm; some algorithms apply approximation ideas for reasons of efficiency. All implementations are part of Snoopy's library of simulation algorithms.

- direct - Gillespie's stochastic simulation algorithm [8],
- tauLeaping - τ-leaping [9],
- deltaLeaping - δ-leaping [2], [1],
- fau - fast adaptive uniformization [1, 3, 11] .

Continuous simulation supports stiff/unstiff solvers ranging from simple fixed-step-size unstiff solvers (e.g. Euler) to more sophisticated variable-order, variable-step, multi-step stiff solvers (e.g. Backward Differentiation Formulas (BDF)). The ODE solvers BDF and ADAMS use the external library SUNDIAL CVODE [7]; all others are part of Snoopy's library of simulation algorithms.

- BDF - Backward Differentiation Formulas [7],
- ADAMS - Adams-Moulton [7],
- Classic - classical Runge-Kutta method (RK4) [16],
- RosenBrock - Rosenbrock method [16],
- Euler - Euler method [16],
- ModEuler - two-steps Euler method.

Hybrid simulation allows for static or dynamic partitioning. In both cases, continuous transitions are simulated using an ODE solver, while stochastic transitions are simulated by the direct method of the Gillespie algorithm [13]. Static partitioning can be combined with static, staticAcc, HRSSA, or HRSSAacc. Dynamic partitioning always applies the exact method. The ODE solvers BDF and ADAMS use the external library SUNDIAL CVODE [7]; all others are part of Snoopy's library of simulation algorithms.

- static - exact method [10, 14],
- staticAcc - accelerated exact method [5],
- HRSSA - Hybrid Rejection-based Stochastic Simulation Algorithm [12],
- HRSSAacc - accelerated HRSSA [6],
- dynamic - dynamic partitioning [14].

References

1. C. Rohr : Simulative analysis of coloured extended stochastic Petri nets. Ph.D. thesis, BTU Cottbus, Dep. of CS (January 2017), `https://opus4.kobv.de/opus4-btu/files/4147/Rohr_Christian.pdf`
2. C. Rohr : Discrete-Time Leap Method For Stochastic Simulation. Fundamenta Informaticae **160**(1-2), 181–198 (2018). https://doi.org/10.3233/FI-2018-1680, accepted for publication: May, 16 2017
3. M. Heiner, C. Rohr, M. Schwarick, S. Streif : A Comparative Study of Stochastic Analysis Techniques . In: Proc. 8th International Conference on Computational Methods in Systems Biology (CMSB 2010). pp. 96–106. ACM digital library (September 2010). https://doi.org/10.1145/1839764.1839776
4. M. Herajy, F. Liu, C. Rohr, M. Heiner : Snoopy's Hybrid Simulator: a Tool to Construct and Simulate Hybrid Biological Models. BMC Systems Biology (2017). https://doi.org/10.1186/s12918-017-0449-6, published: July 28, 2017
5. M. Herajy, M. Heiner : Accelerated Simulation of Hybrid Biological Models with Quasi-disjoint Deterministic and Stochastic Subnets. In: E Cinquemani, A.D. (ed.) Proc. 5th Int. Workshop on Hybrid Systems Biology (HSB 2016). LNBI, vol. 9957, pp. 20–38. Springer (October 2016). https://doi.org/10.1007/978-3-319-47151-8_2
6. M. Herajy, M. Heiner : An improved simulation of hybrid biological models with many stochastic events and quasi-disjoint subnets. In: M. Rabe, AA. Juan, N. Mustafee, A. Skoogh, S. Jain, B. Johansson (ed.) Proceedings of the 2018 Winter Simulation Conference (WSC 2018), Gothenburg, Sweden. pp. 1346–1357. 978-1-5386-6572-5/18, IEEE (December 2018). https://doi.org/10.1109/WSC.2018.8632514, wSC 2018, December 9-12, 2018
7. A. Hindmarsh, P. Brown, K. Grant, S. Lee, R. Serban, D. Shumaker, C. Woodward, C.: SUNDIALS: Suite of Nonlinear and Differential/Algebraic Equation Solvers. ACM Trans. Math. Softw. **31**, 363–396 (2005). https://doi.org/10.1145/1089014.1089020
8. D. T. Gillespie: Exact Stochastic Simulation of Coupled Chemical Reactions. Journal of Physical Chemistry **81**(25), 2340–2361 (1977). https://doi.org/10.1021/j100540a008
9. D. T. Gillespie: Approximate accelerated stochastic simulation of chemically reacting systems. The Journal of Chemical Physics **115**(4), 1716–1733 (2001). https://doi.org/10.1063/1.1378322
10. E. Haseltine, J. Rawlings : Approximate simulation of coupled fast and slow reactions for stochastic chemical kinetics. J. Chem. Phys. **117**(15), 6959–6969 (2002). https://doi.org/10.1063/1.1505860
11. F. Didier, T. A. Henzinger, M. Mateescu, V. Wolf: Fast adaptive uniformization of the chemical master equation. In: 2009 International Workshop on High Performance Computational Systems Biology. pp. 118–127. IEEE (2009). https://doi.org/10.1109/HiBi.2009.23
12. L. Marchetti, C. Priami, V.H. Thanh: HRSSA–Efficient hybrid stochastic simulation for spatially homogeneous biochemical reaction networks. Journal of Computational Physics **317**, 301–317 (2016). https://doi.org/10.1016/j.jcp.2016.04.056
13. M. Heiner, M. Herajy, F. Liu, C. Rohr, M. Schwarick: Snoopy – A Unifying Petri Net Tool. In: ATPN 2012. pp. 398–407. Springer, LNCS 7347 (2012). https://doi.org/10.1007/978-3-642-31131-4_22
14. M. Herajy and M. Heiner: Hybrid Representation and Simulation of Stiff Biochemical Networks . J. Nonlinear Analysis: Hybrid Systems **6**(4), 942959 (November 2012). https://doi.org/10.1016/j.nahs.2012.05.004

15. T. Bray: The JavaScript Object Notation (JSON) Data Interchange Format (December 2017). https://doi.org/10.17487/RFC8259
16. T. W. Vetterling, W. H. Press, S. A. Teukolsky, B. P. Flannery, P. Brian: Numerical Recipes Example Book (C++): The Art of Scientific Computing. Cambridge University Press (2002)