Supplementary Information: Spiking neural P systems with mute rules

1 The P-Lingua syntax for SNPMR systems

The syntax for SNPMR systems is introduced in the following subsections. In order to provide a self-contained description for readers not familiar with P-Lingua language, some aspects that are expressed in the same way as with classical SNP systems will also be recalled.

Model specification

In order to specify SNPMR systems in P-Lingua files, we should use the generic model type `spiking_psystems` at the beginning of the file:

```
@model<spiking_systems>.
```

The rest of the file must define the main elements describing the SNPMR system $\Pi = \{O, \sigma_1, \sigma_2, \ldots, \sigma_m, syn, in, out\}$. As the singleton alphabet $O = \{a\}$ is fixed, so it does not need to be made explicit. The elements $\sigma_1, \sigma_2, \ldots, \sigma_m$, syn, in, and out will be explained as follows.

Neural network structure

The structural element consisting of the network of neurons for an SNPMR system is given analogously to the process in classical SNP systems. That is, the list of neurons σ_1, σ_2, ..., σ_m is defined by the reserved word `@mu`, and the set of synapses between neurons is defined by the reserved word `@marcs`. Additionally, the two reserved words `@min` and `@mout` define the input neuron in and output neuron out of the SNPMR system, respectively. Indeed, the output neuron out can also be inferred from `@marcs`, considering the neurons that have an edge sending to the environment.

```
@mu = r{0}, r{1}, r{2}, state;
@marcs = (r{1},state), (state,r{1}), (r{2},state),
         (state,r{2}), (state,r{0}), (r{0},environment);
@min = r{1};
@out = r{0}.
```

Initial multisets

When defining SNPMR systems, we need to specify the spikes initially placed in each neuron. Given a neuron i containing n_i spikes, it can be specified by using the reserved word `@ms`.

```
@ms(i) = a*n_i.
```

Input spike trains

As there are some SNPMR systems not having any spikes in the initial configuration, but instead receiving the spikes from a so-called input spike train. In those cases, we can use the reserved word `@minst` to assign the number of spikes input into the system in certain steps:

```
@minst = (1, 1), (24, 1), (25, 1).
```

Thus, this sentence assigns one spike to be introduced by the spike train in steps 1, 24 and 25, respectively.

Definition of rules

As in classical SNP systems, the spiking rules in the newly defined SNPMR systems can be defined using one of the following two forms (the second one is preferred for simplicity):

```
[a*c]'h --> [a*p]'h "e";
[a*c    -->  a*p]'h "e";
```

where the label h indicates the neuron firing the spikes, i.e., where the rule is applied, c and p are integer expressions (for the consumed and produced spikes, respectively), and e is a regular expression over $\{a\}$.

Moreover, a new type of rule has been defined specifically for SNPMR systems, i.e., mute rules. This type of rules follows the schema $E'/\texttt{op}\ q$, with $\texttt{op} \in \{+, -\}$ and q a natural number. Within this work, some changes have been introduced in P-Lingua language and its corresponding parser for SNPMR systems, in order to recognize this type of rules. The syntax of such rules is defined as follows:

```
["e" / +q]'h;
["e" / -q]'h;
```

where h indicates the label of the neuron, e is a regular expression, and q is the natural number of spikes to add or remove from the target neuron. Some examples of these rules could be expressed in the following way:

```
["(a{59})*a{2}" / +59]'r{2};
["(a{22})+a{5}" / -22]'r{0}.
```

Regular Expressions

The regular expressions have been preserved in P-Lingua as they were previously defined, as shown in the previous sections. However, the regular expressions could not include auxiliary elements of P-Lingua as variables or parameters, and were literally sent to `regex`. Consequently, to make it easier for the designers to translate their models into the P-Lingua language, an alternative syntax has

been included to express simple regular expressions but using P-Lingua parameters in their definition. In order to differentiate between the pre-existing regular expressions and the new simpler but parameterized ones, the latter expression is included using single quotes instead of double quotes. Such expressions can be applied to classical spiking rules and to the new mute rules. Let us illustrate the difference with some examples:

```
/* Classical and new parameterized regular
   expressions in spiking rules */
[a*43 --> a*5]'1 "(a{36})+a{7}";
[a*(T+li-lj) --> a*171]'state 'a{T+li}'

/* Classical and new parameterized regular
   expressions in mute rules */
["(a{22})*a{283}" / -283]'r{0}
['a{lj}' / +T]'state;
```

2 The simulation algorithm for SNPMR systems

A new simulator has been developed within P-Lingua framework, making use of the general simulator available for SNP systems, but including the syntax and dynamics of SNPMR systems. Therefore, once an SNPMR system is translated into a file of P-Lingua language, the simulator performs a possible computation from the initial configuration given by the structure of neurons and spikes specified, producing the corresponding no-deterministic transitions until a halting configuration is reached or until the desired number of computation steps run. The simulation algorithm follows the general schema found in most of the simulators in the platform:

1. Initialization
2. For each computation step, while there are applicable rules:
   (a) Selection of rules
   (b) Execution of rules

The **Initialization stage** sets the initial structures needed by the algorithm. Then, the main loop runs until a halting configuration is reached, i.e., until no applicable rule is present at a given computation step.

The **Selection phase** checks for the applicability of the rules for each neuron. The applicability of a rule is determined by whether the contents of the neuron where the rule is present match the regular expression of the rule, and by the presence of at least the number of spikes in the left hand side of the rule. As the languages accepted by the regular expressions for spiking and mute rules are disjoint, so no conflict among different types of rules will happen at a step of the computation in the same neuron. However, more than one spiking rule could be applicable at a computation step, and the same could apply if different mute

rules are applicable. Thus, there is a non-deterministic choice of rules whenever more than one can be applied at the same step in the same neuron.

The **Execution phase** executes the selected rules from the current configuration to the next one, in all the neurons where one rule was selected. In the case of spiking rules, it consumes the corresponding spikes and sends the corresponding produced spikes to all the neurons connected to the firing one through an outgoing synapse. In the case of mute rules, it will imply either adding or removing the corresponding spikes to the neuron where the rule is present, not affecting any neighboring ones.

Apart from executing rules, a given configuration can get altered by the input of a certain number of spikes through an input spike train. In the simulator, this phase occurs after the selection and execution of rules, interpreting that those processes take place from a given static configuration. However, either after them or in absence of rules executed, if an input train caused the appearance of new spikes in some neuron, this will also alter the configuration.

For any computation step, in other types of P systems as cell-like or tissue-like P systems, if no rule is applied to a certain configuration, the system is considered to have reached a halting configuration. However, in the case of SNP systems, and especially in SNPMR systems, it may happen that no rules are applied but some spikes have been received from an input spike train. Besides, it may be the case that for a number of steps no rules were applied nor spikes were received from an input spike train, but some spikes could be ready to be received in later steps. For this reason, a new condition has been incorporated in the simulator for SNPMR systems: if some spikes are expected to be received from an input spike train at a later step, then the configuration is not a halting one, even if no change has been observed in the contents of any rule (that is the case when no rule has been applied nor spikes have been received from any input spike train in the current step).

Availability of the Software Tools Developed: The tools described in the previous sections, concerning the P-Lingua language, parsing and simulation have made publicly available in the current version of MeCoSim (more specifically, within the P-Lingua version forked from P-Lingua 4.0 and available with MeCoSim software), and can be downloaded from MeCoSim website: http://www.p-lingua.org/mecosim/.

3 The instructions of the universal register machine U_{32}

The 32 instructions of the universal register machine U_{32} comprise three types of instructions:

- ADD instructions: $l_i : (\texttt{ADD}(r), l_j)$: add 1 to the value of register r and then the machine executes the instruction l_j;
- TEST instructions: $l_i : (\texttt{TEST}(r), l_j, l_k)$: test whether the value of register r is positive or not, and if it is positive, the machine executes the instruction l_j, otherwise the machine executes the instruction l_k;

$q_1 : (\texttt{TEST}(1), q_2, q_6)$, $q_2 : (\texttt{SUB}(1), q_3)$,
$q_3 : (\texttt{ADD}(7), q_1)$, $q_4 : (\texttt{TEST}(5), q_5, q_7)$,
$q_5 : (\texttt{SUB}(5), q_6)$, $q_6 : (\texttt{ADD}(6), q_4)$,
$q_7 : (\texttt{TEST}(6), q_8, q_4)$, $q_8 : (\texttt{SUB}(6), q_9)$,
$q_9 : (\texttt{ADD}(5), q_{10})$, $q_{10} : (\texttt{TEST}(7), q_{11}, q_{13})$,
$q_{11} : (\texttt{SUB}(7), q_{12})$, $q_{12} : (\texttt{ADD}(1), q_7)$,
$q_{13} : (\texttt{TEST}(6), q_{14}, q_1)$, $q_{14} : (\texttt{TEST}(4), q_{15}, q_{16})$,
$q_{15} : (\texttt{SUB}(4), q_1)$, $q_{16} : (\texttt{TEST}(5), q_{17}, q_{23})$,
$q_{17} : (\texttt{SUB}(5), q_{18})$, $q_{18} : (\texttt{TEST}(5), q_{19}, q_{27})$,
$q_{19} : (\texttt{SUB}(5), q_{20})$, $q_{20} : (\texttt{TEST}(5), q_{21}, q_{30})$,
$q_{21} : (\texttt{SUB}(5), q_{22})$, $q_{22} : (\texttt{ADD}(4), q_{16})$,
$q_{23} : (\texttt{TEST}(2), q_{24}, q_{25})$, $q_{24} : (\texttt{SUB}(2), q_{32})$,
$q_{25} : (\texttt{TEST}(0), q_{26}, q_{32})$, $q_{26} : (\texttt{SUB}(0), q_1)$,
$q_{27} : (\texttt{TEST}(4), q_{28}, q_{29})$, $q_{28} : (\texttt{SUB}(3), q_{32})$,
$q_{29} : (\texttt{ADD}(0), q_1)$, $q_{30} : (\texttt{ADD}(2), q_{31})$,
$q_{31} : (\texttt{ADD}(3), q_{32})$, $q_{32} : (\texttt{TEST}(4), q_{15}, q_0)$,

– SUB instructions: l_i : (SUB$(r), l_j$): subtract 1 from the value of register r, and then the machine executes the instruction l_j. If the value of register r is zero, it remains unchanged.

4 The P-Lingua translation for the SNPMR system Π_u simulating the universal register machine U_{32}

According to the P-Lingua syntax for SNPMR systems defined above, the details of the P-Lingua translation for the SNPMR system Π_u to simulate the universal register machine U_{32} is shown as follows.

```
@model<spiking_psystems>

/* Auxiliary parameters */
m  = 22;
T  = 7*(m+110);
l0 = 7*(0+2);

/* Main method */
def main()
{
  /* Neuron definition */
  @mu = state;
  {
    @mu += r{i};
    @marcs += (r{i},state), (state, r{i});
  } : 0 <= i <= 7;

 /* Synapse graph definition: edges (arcs) */
  @marcs += (r{7}, r{1}), (r{7}, r{2}), (r{0}, out), (out, environment);
 /* The input and output neurons */
  @min = r{7};
  @mout = out;

  /* Initial spikes in neuron state to simulate the instruction l0 */
  /* Initial spikes in neuron 1 to encode the number 23 in register 1 */
  /* Initial spikes in neuron 2 to encode the number 1 in register 2 */
  @ms(state) = a*938;
  @ms(r{1})  = a*828;
  @ms(r{2})  = a*59;


  /* Instructions for simulating the universal register machine U_{32}*/
  /* The corresponding calls to modules */
  call SUB(0, 1, 1, 2); /*Initiate the simulation of the instruction l0*/
  call ADD(1, 7, 0);
  call ADD(2, 6, 3);
  call SUB(3, 5, 2, 4);
  call SUB(4, 6, 5, 3);
  call ADD(5, 5, 6);
  call SUB(6, 7, 7, 8);
  call ADD(7, 1, 4);
  call SUB(8, 6, 9, 0);
  call ADD(9, 6, 10);
  call SUB(10, 4, 0, 11);
  call SUB(11, 5, 12, 13);
  call SUB(12, 5, 14, 15);
  call SUB(13, 2, 18, 19);
  call SUB(14, 5, 16, 17);
  call SUB(15, 3, 18, 20);
  call ADD(16, 4, 11);
  call ADD(17, 2, 21);
  call SUB(18, 4, 0, 22);
```

```
  call SUB(19, 0, 0, 18);
  call ADD(20, 0, 0);
  call ADD(21, 3, 18);
  call HALT(22);
}

def INPUT()
{
  /* RState_INPUT */
  ["a{3}" / +(T+l0-3)]'state;

  /* R7_INPUT */
  [a --> a]'r{7};

  /* R1_INPUT */
  ["(a{36})*a" / +36]'r{1};
  ["(a{36})*a{3}" / -3]'r{1};

  /* R2_INPUT */
  ["(a{59})*a{2}" / +59]'r{2};
  ["(a{59})*a{3}" / -3]'r{2};
}

def ADD(i, n, j)
{
  let li = 7*(i+2);
  let lj = 7*(j+2);

  /* Rstate_ADD */
  [a*(T+li-lj) --> a*22]'state 'a{T+li}' : n==0;
  [a*(T+li-lj) --> a*36]'state 'a{T+li}' : n==1;
  [a*(T+li-lj) --> a*59]'state 'a{T+li}' : n==2;
  [a*(T+li-lj) --> a*104]'state 'a{T+li}' : n==3;
  [a*(T+li-lj) --> a*171]'state 'a{T+li}' : n==4;
  [a*(T+li-lj) --> a*194]'state 'a{T+li}' : n==5;
  [a*(T+li-lj) --> a*283]'state 'a{T+li}' : n==6;
  [a*(T+li-lj) --> a*746]'state 'a{T+li}' : n==7;

  ['a{lj}' / +T]'state;

  /* R0_ADD */
  ["(a{22})*a{36}" / -36]'r{0} : n==1;
  ["(a{22})*a{59}" / -59]'r{0} : n==2;
  ["(a{22})*a{104}" / -104]'r{0} : n==3;
  ["(a{22})*a{171}" / -171]'r{0} : n==4;
  ["(a{22})*a{194}" / -194]'r{0} : n==5;
  ["(a{22})*a{283}" / -283]'r{0} : n==6;
  ["(a{22})*a{746}" / -746]'r{0} : n==7;

  /* R1_ADD */
  ["(a{36})*a{22}" / -22]'r{1} : n==0;
  ["(a{36})*a{59}" / -59]'r{1} : n==2;
  ["(a{36})*a{104}" / -104]'r{1} : n==3;
  ["(a{36})*a{171}" / -171]'r{1} : n==4;
  ["(a{36})*a{194}" / -194]'r{1} : n==5;
  ["(a{36})*a{283}" / -283]'r{1} : n==6;
  ["(a{36})*a{746}" / -746]'r{1} : n==7;

  /* R2_ADD */
  ["(a{59})*a{22}" / -22]'r{2} : n==0;
  ["(a{59})*a{36}" / -36]'r{2} : n==1;
  ["(a{59})*a{104}" / -104]'r{2} : n==3;
  ["(a{59})*a{171}" / -171]'r{2} : n==4;
  ["(a{59})*a{194}" / -194]'r{2} : n==5;
  ["(a{59})*a{283}" / -283]'r{2} : n==6;
  ["(a{59})*a{746}" / -746]'r{2} : n==7;

  /* R3_ADD */
```

```
  ["(a{104})*a{22}" / -22]'r{3} : n==0;
  ["(a{104})*a{36}" / -36]'r{3} : n==1;
  ["(a{104})*a{59}" / -59]'r{3} : n==2;
  ["(a{104})*a{171}" / -171]'r{3} : n==4;
  ["(a{104})*a{194}" / -194]'r{3} : n==5;
  ["(a{104})*a{283}" / -283]'r{3} : n==6;
  ["(a{104})*a{746}" / -746]'r{3} : n==7;

  /* R4_ADD */
  ["(a{171})*a{22}" / -22]'r{4} : n==0;
  ["(a{171})*a{36}" / -36]'r{4} : n==1;
  ["(a{171})*a{59}" / -59]'r{4} : n==2;
  ["(a{171})*a{104}" / -104]'r{4} : n==3;
  ["(a{171})*a{194}" / -194]'r{4} : n==5;
  ["(a{171})*a{283}" / -283]'r{4} : n==6;
  ["(a{171})*a{746}" / -746]'r{4} : n==7;

  /* R5_ADD */
  ["(a{194})*a{22}" / -22]'r{5} : n==0;
  ["(a{194})*a{36}" / -36]'r{5} : n==1;
  ["(a{194})*a{59}" / -59]'r{5} : n==2;
  ["(a{194})*a{104}" / -104]'r{5} : n==3;
  ["(a{194})*a{171}" / -171]'r{5} : n==4;
  ["(a{194})*a{283}" / -283]'r{5} : n==6;
  ["(a{194})*a{746}" / -746]'r{5} : n==7;

  /* R6_ADD */
  ["(a{283})*a{22}" / -22]'r{6} : n==0;
  ["(a{283})*a{36}" / -36]'r{6} : n==1;
  ["(a{283})*a{59}" / -59]'r{6} : n==2;
  ["(a{283})*a{104}" / -104]'r{6} : n==3;
  ["(a{283})*a{171}" / -171]'r{6} : n==4;
  ["(a{283})*a{194}" / -194]'r{6} : n==5;
  ["(a{283})*a{746}" / -746]'r{6} : n==7;

  /* R7_ADD */
  ["(a{746})*a{22}" / -22]'r{7} : n==0;
  ["(a{746})*a{36}" / -36]'r{7} : n==1;
  ["(a{746})*a{59}" / -59]'r{7} : n==2;
  ["(a{746})*a{104}" / -104]'r{7} : n==3;
  ["(a{746})*a{171}" / -171]'r{7} : n==4;
  ["(a{746})*a{194}" / -194]'r{7} : n==5;
  ["(a{746})*a{283}" / -283]'r{7} : n==6;
}

def SUB(i, n, j, k)
{
  let li = 7*(i+2);
  let lj = 7*(j+2);
  let lk = 7*(k+2);

  /* Rstate_SUB */
  [a*(T+6) --> a*(6+n)]'state 'a{T+li}';
  ['a{li-1}' / +(T+lj-li+1)]'state;
  ['a{li-2}' / +(T+lk-li+2)]'state;

  /* R0_SUB */
  {
    [a*28 --> a*5]'r{0} "(a{22})+a{6}";
    [a*6 --> a*4]'r{0};
  } : n==0;

  ["(a{22})*a{7}" / -7]'r{0} : n == 1;
  ["(a{22})*a{8}" / -8]'r{0} : n == 2;
  ["(a{22})*a{9}" / -9]'r{0} : n == 3;
  ["(a{22})*a{10}" / -10]'r{0} : n == 4;
  ["(a{22})*a{11}" / -11]'r{0} : n == 5;
  ["(a{22})*a{12}" / -12]'r{0} : n == 6;
```

```
["(a{22})*a{13}" / -13]'r{0} : n == 7;

/* Rout_SUB */
{
  ["a{4}" / -4]'out;
  ["a{5}" / -5]'out;
} : n==0;


/* R1_SUB */
["(a{36})*a{6}" / -6]'r{1} : n == 0;

{
  [a*43 --> a*5]'r{1} "(a{36})+a{7}";
  [a*7 --> a*4]'r{1};
} : n==1;

["(a{36})*a{8}" / -8]'r{1} : n == 2;
["(a{36})*a{9}" / -9]'r{1} : n == 3;
["(a{36})*a{10}" / -10]'r{1} : n == 4;
["(a{36})*a{11}" / -11]'r{1} : n == 5;
["(a{36})*a{12}" / -12]'r{1} : n == 6;
["(a{36})*a{13}" / -13]'r{1} : n == 7;

["(a{36})*a{4}" / -4]'r{1};
["(a{36})*a{5}" / -5]'r{1};

/* R2_SUB */
["(a{59})*a{6}" / -6]'r{2} : n == 0;
["(a{59})*a{7}" / -7]'r{2} : n == 1;

{
  [a*67 --> a*5]'r{2} "(a{59})+a{8}";
  [a*8 --> a*4]'r{2};
} : n==2;

["(a{59})*a{9}" / -9]'r{2} : n == 3;
["(a{59})*a{10}" / -10]'r{2} : n == 4;
["(a{59})*a{11}" / -11]'r{2} : n == 5;
["(a{59})*a{12}" / -12]'r{2} : n == 6;
["(a{59})*a{13}" / -13]'r{2} : n == 7;

["(a{59})*a{4}" / -4]'r{2};
["(a{59})*a{5}" / -5]'r{2};

/* R3_SUB */
["(a{104})*a{6}" / -6]'r{3} : n == 0;
["(a{104})*a{7}" / -7]'r{3} : n == 1;
["(a{104})*a{8}" / -8]'r{3} : n == 2;

{
  [a*113 --> a*5]'r{3} "(a{104})+a{9}";
  [a*9 --> a*4]'r{3};
} : n==3;

["(a{104})*a{10}" / -10]'r{3} : n == 4;
["(a{104})*a{11}" / -11]'r{3} : n == 5;
["(a{104})*a{12}" / -12]'r{3} : n == 6;
["(a{104})*a{13}" / -13]'r{3} : n == 7;

/* R4_SUB */
["(a{171})*a{6}" / -6]'r{4} : n == 0;
["(a{171})*a{7}" / -7]'r{4} : n == 1;
["(a{171})*a{8}" / -8]'r{4} : n == 2;
["(a{171})*a{9}" / -9]'r{4} : n == 3;

{
  [a*181 --> a*5]'r{4} "(a{171})+a{10}";
```

```
    [a*10 --> a*4]'r{4};
  } : n==4;

  ["(a{171})*a{11}" / -11]'r{4} : n == 5;
  ["(a{171})*a{12}" / -12]'r{4} : n == 6;
  ["(a{171})*a{13}" / -13]'r{4} : n == 7;

  /* R5_SUB */
  ["(a{194})*a{6}" / -6]'r{5} : n == 0;
  ["(a{194})*a{7}" / -7]'r{5} : n == 1;
  ["(a{194})*a{8}" / -8]'r{5} : n == 2;
  ["(a{194})*a{9}" / -9]'r{5} : n == 3;
  ["(a{194})*a{10}" / -10]'r{5} : n == 4;

  {
    [a*205 --> a*5]'r{5} "(a{194})+a{11}";
    [a*11 --> a*4]'r{5};
  } : n==5;

  ["(a{194})*a{12}" / -12]'r{5} : n == 6;
  ["(a{194})*a{13}" / -13]'r{5}: n == 7;

  /* R6_SUB */
  ["(a{283})*a{6}" / -6]'r{6} : n == 0;
  ["(a{283})*a{7}" / -7]'r{6} : n == 1;
  ["(a{283})*a{8}" / -8]'r{6} : n == 2;
  ["(a{283})*a{9}" / -9]'r{6} : n == 3;
  ["(a{283})*a{10}" / -10]'r{6} : n == 4;
  ["(a{283})*a{11}" / -11]'r{6} : n == 5;

  {
    [a*295 --> a*5]'r{6} "(a{283})+a{12}";
    [a*12 --> a*4]'r{6};
  } : n==6;

  ["(a{283})*a{13}" / -13]'r{6} : n == 7;

  /* R7_SUB */
  ["(a{746})*a{6}" / -6]'r{7} : n == 0;
  ["(a{746})*a{7}" / -7]'r{7} : n == 1;
  ["(a{746})*a{8}" / -8]'r{7} : n == 2;
  ["(a{746})*a{9}" / -9]'r{7} : n == 3;
  ["(a{746})*a{10}" / -10]'r{7} : n == 4;
  ["(a{746})*a{11}" / -11]'r{7} : n == 5;
  ["(a{746})*a{12}" / -12]'r{7} : n == 6;

  {
    [a*759 --> a*5]'r{7} "(a{746})+a{13}";
    [a*13 --> a*4]'r{7};
  } : n==7;

}

def HALT(h)
{
  let lh = 7*(h+2);

  /* Rstate_HALT */
  [a*(T+lh) --> a*4]'state 'a{T+lh}';
  ["a" / -1]'state;

  /* R0_HALT */
  [a*21 --> a]'r{0} "(a{22})+a{4}";
  ["(a{22})+a{5}" / -22]'r{0};
  ["a{5}" / -4]'r{0};
  [a*5 --> a]'r{0};

  /* Rout_HALT */
```

```
  [a --> a]'out;

  /* R1_HALT */
  ["(a{36})*a{4}" / -4]'r{1};

  /* R2_HALT */
  ["(a{59})*a{4}" / -4]'r{2};

  /* R3_HALT */
  ["(a{104})*a{4}" / -4]'r{3};

  /* R4_HALT */
  ["(a{171})*a{4}" / -4]'r{4};

  /* R5_HALT */
  ["(a{194})*a{4}" / -4]'r{5};

  /* R6_HALT */
  ["(a{283})*a{4}" / -4]'r{6};

  /* R7_HALT */
  ["(a{746})*a{4}" / -4]'r{7};
}
```

5 Simulation results of the universal SNPMR system Π_u for simulating the universal register machine U_{32}

We also provided a detailed trace of a computation within the system Π_u for simulating a computation of the machine U_{32} with the input $g(x) = 1$ and $y = 1$. The computation of the machine U_{32} is finally halting at step 74, and the computation of the system Π_u lasts 151 steps.

Table 1: The computation trace of the universal SNPMR system Π_u for simulating the computation of the universal register machine U_{32} for $g(x) = 1$ and $y = 1$

Computation step		**Active Instruction**		**Register's content**								**Neuron's content**								
U_{32}	Π_u	U_{32}	Π_u	r_0	r_1	r_2	r_3	r_4	r_5	r_6	r_7	σ_{state}	σ_0	σ_1	σ_2	σ_3	σ_4	σ_5	σ_6	σ_7	σ_{out}
0	0	q_1	l_0	0	1	1	0	0	0	0	0	938	0	36	59	0	0	0	0	0	0
1	0	q_2	l_0	0	1	1	0	0	0	0	0	938	0	36	59	0	0	0	0	0	0
2	3	q_3	l_1	0	0	1	0	0	0	0	0	945	0	0	59	0	0	0	0	0	0
3	5	q_1	l_0	0	0	1	0	0	0	0	1	938	0	0	59	0	0	0	0	746	0
4	8	q_6	l_2	0	0	1	0	0	0	0	1	952	0	0	59	0	0	0	0	746	0
5	10	q_4	l_3	0	0	1	0	0	0	1	1	959	0	0	59	0	0	0	283	746	0
6	13	q_7	l_4	0	0	1	0	0	0	1	1	966	0	0	59	0	0	0	283	746	0
7	13	q_8	l_4	0	0	1	0	0	0	1	1	966	0	0	59	0	0	0	283	746	0
8	16	q_9	l_5	0	0	1	0	0	0	0	1	973	0	0	59	0	0	0	0	746	0
9	18	q_{10}	l_6	0	0	1	0	0	1	0	1	980	0	0	59	0	0	194	0	746	0
10	18	q_{11}	l_6	0	0	1	0	0	1	0	1	980	0	0	59	0	0	194	0	746	0
11	21	q_{12}	l_7	0	0	1	0	0	1	0	0	987	0	0	59	0	0	194	0	0	0
12	23	q_7	l_4	0	1	1	0	0	1	0	0	966	0	36	59	0	0	194	0	0	0
13	26	q_4	l_3	0	1	1	0	0	1	0	0	959	0	36	59	0	0	194	0	0	0
14	26	q_5	l_3	0	1	1	0	0	1	0	0	959	0	36	59	0	0	194	0	0	0
15	29	q_6	l_2	0	1	1	0	0	0	0	0	952	0	36	59	0	0	0	0	0	0
16	31	q_4	l_3	0	1	1	0	0	0	1	0	959	0	36	59	0	0	0	283	0	0
17	34	q_7	l_4	0	1	1	0	0	0	1	0	966	0	36	59	0	0	0	283	0	0
18	34	q_8	l_4	0	1	1	0	0	0	1	0	966	0	36	59	0	0	0	283	0	0
19	37	q_9	l_5	0	1	1	0	0	0	0	0	973	0	36	59	0	0	0	0	0	0
20	39	q_{10}	l_6	0	1	1	0	0	1	0	0	980	0	36	59	0	0	194	0	0	0
21	42	q_{13}	l_8	0	1	1	0	0	1	0	0	994	0	36	59	0	0	194	0	0	0

Continued on next page

– **continued from Table 1.**

Computation step		**Active Instruction**		**Register's content**								**Neuron's content**								
U_{32}	Π_u	U_{32}	Π_u	r_0	r_1	r_2	r_3	r_4	r_5	r_6	r_7	σ_{state}	σ_0	σ_1	σ_2	σ_3	σ_4	σ_5	σ_6	σ_7	σ_{out}
22	45	q_1	l_0	0	1	1	0	0	1	0	0	938	0	36	59	0	0	194	0	0	0
23	45	q_2	l_0	0	1	1	0	0	1	0	0	938	0	36	59	0	0	194	0	0	0
24	48	q_3	l_1	0	0	1	0	0	1	0	0	945	0	0	59	0	0	194	0	0	0
25	50	q_1	l_0	0	0	1	0	0	1	0	1	938	0	0	59	0	0	194	0	746	0
26	53	q_6	l_2	0	0	1	0	0	1	0	1	952	0	0	59	0	0	194	0	746	0
27	55	q_4	l_3	0	0	1	0	0	1	1	1	959	0	0	59	0	0	194	283	746	0
28	55	q_5	l_3	0	0	1	0	0	1	1	1	959	0	0	59	0	0	194	283	746	0
29	58	q_6	l_2	0	0	1	0	0	0	1	1	952	0	0	59	0	0	0	283	746	0
30	60	q_4	l_3	0	0	1	0	0	0	2	1	959	0	0	59	0	0	0	566	746	0
31	63	q_7	l_4	0	0	1	0	0	0	2	1	966	0	0	59	0	0	0	566	746	0
32	63	q_8	l_4	0	0	1	0	0	0	2	1	966	0	0	59	0	0	0	566	746	0
33	66	q_9	l_5	0	0	1	0	0	0	1	1	973	0	0	59	0	0	0	283	746	0
34	68	q_{10}	l_6	0	0	1	0	0	1	1	1	980	0	0	59	0	0	194	283	746	0
35	68	q_{11}	l_6	0	0	1	0	0	1	1	1	980	0	0	59	0	0	194	283	746	0
36	71	q_{12}	l_7	0	0	1	0	0	1	1	0	987	0	0	59	0	0	194	283	0	0
37	73	q_7	l_4	0	1	1	0	0	1	1	0	966	0	36	59	0	0	194	283	0	0
38	73	q_8	l_4	0	1	1	0	0	1	1	0	966	0	36	59	0	0	194	283	0	0
39	76	q_9	l_5	0	1	1	0	0	1	0	0	973	0	36	59	0	0	194	0	0	0
40	78	q_{10}	l_6	0	1	1	0	0	2	0	0	980	0	36	59	0	0	388	0	0	0
41	81	q_{13}	l_8	0	1	1	0	0	2	0	0	994	0	36	59	0	0	388	0	0	0
42	84	q_1	l_0	0	1	1	0	0	2	0	0	938	0	36	59	0	0	388	0	0	0
43	84	q_2	l_0	0	1	1	0	0	2	0	0	938	0	36	59	0	0	388	0	0	0
44	87	q_3	l_1	0	0	1	0	0	2	0	0	945	0	0	59	0	0	388	0	0	0
45	89	q_1	l_0	0	0	1	0	0	2	0	1	938	0	0	59	0	0	388	0	746	0
46	92	q_6	l_2	0	0	1	0	0	2	0	1	952	0	0	59	0	0	388	0	746	0
47	94	q_4	l_3	0	0	1	0	0	2	1	1	959	0	0	59	0	0	388	283	746	0
48	94	q_5	l_3	0	0	1	0	0	2	1	1	959	0	0	59	0	0	388	283	746	0
49	97	q_6	l_2	0	0	1	0	0	1	1	1	952	0	0	59	0	0	194	283	746	0
50	99	q_4	l_3	0	0	1	0	0	1	2	1	959	0	0	59	0	0	194	566	746	0
51	99	q_5	l_3	0	0	1	0	0	1	2	1	959	0	0	59	0	0	194	566	746	0
52	102	q_6	l_2	0	0	1	0	0	0	2	1	952	0	0	59	0	0	0	566	746	0
53	104	q_4	l_3	0	0	1	0	0	0	3	1	959	0	0	59	0	0	0	849	746	0
54	107	q_7	l_4	0	0	1	0	0	0	3	1	966	0	0	59	0	0	0	849	746	0
55	107	q_8	l_4	0	0	1	0	0	0	3	1	966	0	0	59	0	0	0	849	746	0
56	110	q_9	l_5	0	0	1	0	0	0	2	1	973	0	0	59	0	0	0	566	746	0
57	112	q_{10}	l_6	0	0	1	0	0	1	2	1	980	0	0	59	0	0	194	566	746	0
58	112	q_{11}	l_6	0	0	1	0	0	1	2	1	980	0	0	59	0	0	194	566	746	0
59	115	q_{12}	l_7	0	0	1	0	0	1	2	0	987	0	0	59	0	0	194	566	0	0
60	117	q_7	l_4	0	1	1	0	0	1	2	0	966	0	36	59	0	0	194	566	0	0
61	117	q_8	l_4	0	1	1	0	0	1	2	0	966	0	36	59	0	0	194	566	0	0
62	120	q_9	l_5	0	1	1	0	0	1	1	0	973	0	36	59	0	0	194	283	0	0
63	122	q_{10}	l_6	0	1	1	0	0	2	1	0	980	0	36	59	0	0	388	283	0	0
64	125	q_{13}	l_8	0	1	1	0	0	2	1	0	994	0	36	59	0	0	388	283	0	0
65	130	q_{14}	l_{10}	0	1	1	0	0	2	1	0	1008	0	36	59	0	0	388	283	0	0
66	133	q_{16}	l_{11}	0	1	1	0	0	2	1	0	1015	0	36	59	0	0	388	283	0	0
67	133	q_{17}	l_{11}	0	1	1	0	0	2	1	0	1015	0	36	59	0	0	388	283	0	0
68	136	q_{18}	l_{12}	0	1	1	0	0	1	1	0	1022	0	36	59	0	0	194	283	0	0
69	136	q_{19}	l_{12}	0	1	1	0	0	1	1	0	1022	0	36	59	0	0	194	283	0	0
70	139	q_{20}	l_{14}	0	1	1	0	0	0	1	0	1036	0	36	59	0	0	0	283	0	0
71	142	q_{30}	l_{17}	0	1	1	0	0	0	1	0	1057	0	36	59	0	0	0	283	0	0
72	144	q_{31}	l_{21}	0	1	2	0	0	0	1	0	1085	0	36	118	0	0	0	283	0	0
73	146	q_{32}	l_{18}	0	1	2	1	0	0	1	0	1064	0	36	118	104	0	0	283	0	0
74	149	q_0	l_{22}	0	1	2	1	0	0	1	0	1092	0	36	118	104	0	0	283	0	0
	150											0	4	40	122	108	4	4	287	4	0
	151											0	0	36	118	104	0	0	283	0	0