NESTML Tutorial

I. Blundell, M.J. Eppler, D. Plotnikov

Software Engineering
RWTH Aachen

http://www.se-rwth.de/
Usage of the NESTML Infrastructure

- Starting eclipse:
  cd /home/nest/eclipse_nestml
  ./eclipse

- Working folder for the code generation:
  /home/nest/nestml_workshop/nestml_workshop_project

- Console-tool for the codegeneration
  java -jar nestml-core-0.0.3-SNAPSHOT-jar-with-depdependencies.jar pathToFile.nestml
  - Optional parameters:
    - --target generationPath (current directory if omitted)

- Change to the generated folder
  - cd codegeneration\neuron_level_1 (or _2, _3 for particular task)

- Execute the following 3 commands (enter them individually)
  sh bootstrap.sh
  ./configure --with-nest=${NEST_INSTALL_DIR}/bin/nest-config
  make && make install
Task 1: Simple Case
Integrate neuron 1/2

- Implement a simple integrate neuron
  - The neuron doesn’t spike, but integrates over the time
Task 1: Simple Case
Integrate neuron 2/2

- Use the template neuron_level_1.nestml
  - Fix errors showed by the editor
  - Fill/extend TODO

- The dynamics is described as:

\[
G := \frac{E}{\text{tau}_{\text{syn}}} \cdot t \cdot \exp\left(-\frac{1}{\text{tau}_{\text{syn}}} \cdot t\right)
\]

\[
\frac{d}{dt}V := -\frac{1}{\text{Tau}} \cdot V + \frac{1}{C_m} \cdot G + I_e + \text{cur}
\]

- Use tester_workshop_neuron_level_1.py to test
Task 2: Threshold
Integrate and fire neuron 1/2

- Add the threshold test in the dynamics
- Increase the refractory time to 20 ms
Task 2: Threshold
Integrate and fire neuron 2/2

- Use the template neuron_level_2.nestml
  - Fill/extend TODOs

- Implement threshold crossing using the variable `thresholdTheta`

- Use python `tester_workshop_neuron_level_2.py` to test
Task 3: Adaptive Threshold 1/2

- Make an adaption of the threshold after each spiking
Task 3: Adaptive Threshold 2/2

- Use the template neuron_level_3.nestml
  - Fill/extend TODO

- Use a threshold adaption, e.g. Theta = Theta + 3 after spiking

- Use tester_workshop_neuron_level_3.py to test
Language Concepts
Procedural Language: Declarations

Multiple variables in the same declaration

Type

Optional initial value

a, b, c \texttt{real} = 0

x \texttt{real} = 3; y \texttt{real} = 4; z \texttt{real}

f \texttt{real} = -2e12

Possible types: integer, real, string, ms, mV, ...
Simple Programming Language

Function Calls

\[
\text{base}, \ \text{power} \ \text{real} = 0
\]

\[
\text{pow} (\text{base}, \ \text{power})
\]

Function name \rightarrow Parameters

Important pre-defined functions:

emitSpike(): emits spike

\exp(x): Returns the base-e exponential function of \( x \), which is \( e \) raised to the power \( x \): \( e^x \)

pow(base, power): raises base to the power exponent.

Constants:

\( E \): Euler's number
Simple Programming Language: Control flow 1/2

```python
if 2 < 3:
    ...
end

if 2 < 3:
    ...
else:
    ...
end

elif 4 > 6:
    ...
else:
    ...
end

for x in 1 ... 5:
    ...
end

for x in 1 ... -5.6 step 0.1:
    ...
end

x, y real
x = 1
y = 2
while x <= 10:
    y = x*2
    x = x + 1
end
```
NESTML

Model structure

Package name. Relevant for model crossreferences. package testing:

neuron WorkingNeuron:
    state:
        i_0 mV
    end

input:
    spikeBuffer <- inhibitory excitatory spike
end

output: spike

Dynamics definition
dynamics timestep(t ms):
end

Entire SPL code is possible
Buffer Blocks

```
input:
  bufferName <- inhibitory excitatory spike
end

"spike" and "current" are possible

"inhibitory", "excitatory", both or none are possible

output: spike

"spike" and "current" are possible
```
Simple Programming Language
Differential Equations

\[
\text{dynamics \ timestep(t ms):} \\
\text{ODE:} \\
G := E/\tau_{\text{syn}} \times t \times \exp(-1/\tau_{\text{syn}}*t) \\
\frac{d}{dt} V := -1/\text{\tau} \times V + 1/C_m \times G + I_e + \text{cur} \\
\text{end} \\
\text{end}
\]