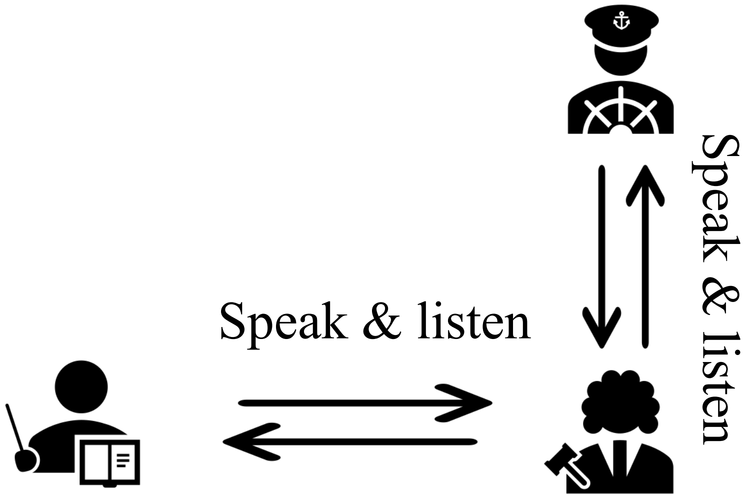
Multiple Languages Competition and Agent-Based Modeling

Kehang Li

Languages competition model

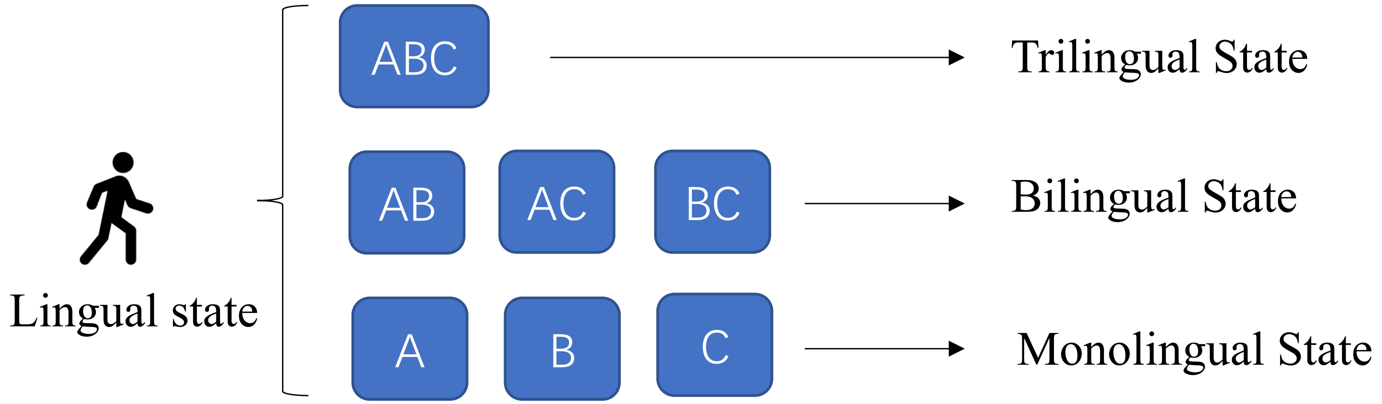
- Concept: Language change can be viewed as a diffusion process of some new linguistic elements in a language community (Nettle 1999)
- Elements: imagine that in a given society, three independent grammars G_A, G_B, G_C coexist and are available for all the language users in the social network to learn.
- Interaction: simplify the realistic complex language interaction into the communications between agents through utterances only.



Assumptions

1. Grammar is a set of production rules that generate language strings. Assume the competition between languages is the systematic redistribution of grammatical trajectories.

2. Agents' multilingual states are stable and has long-term existence in networks.

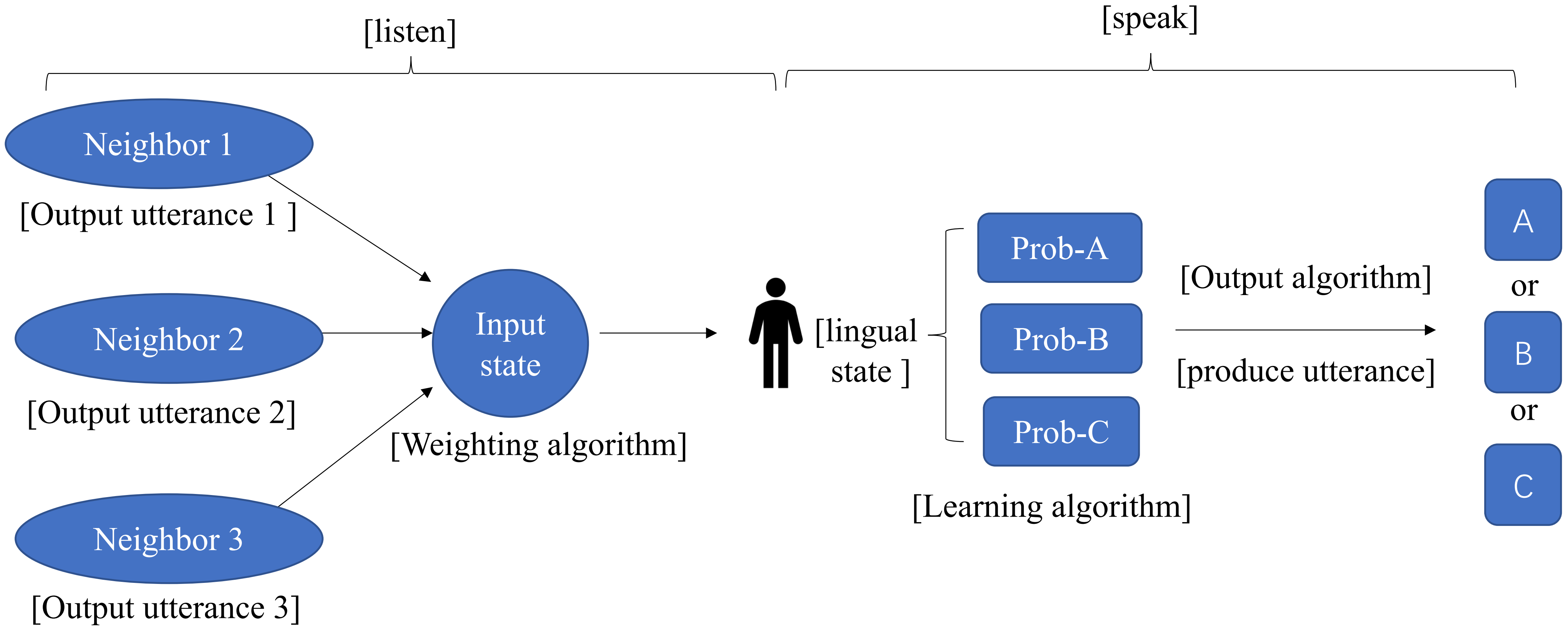


3. In each iteration, agent speaks utterances that generated by exclusive unique grammar, no mixed or multilingual output state in the model.

4. The population size remains constant.

Agent Behavior

- Each agent is treated as a generalized data processor with two different modes of immature and mature determined by agent age.



Weighting algorithm

- Consider the social distance between two agents in the network, agents have closer relationship and larger influence when distance getting shorter. Define the impact of grammar G_P by the following formula:

$$\hat{l}_p = {N_p}^a\left(\sum (s_i/{d_i}^2)\,/N_p\right)$$

$$s_i = \begin{cases} 1 & if\ G_P \rightarrow\ s_i \\ 0 & if\ G_P \nrightarrow\ s_i \end{cases}$$

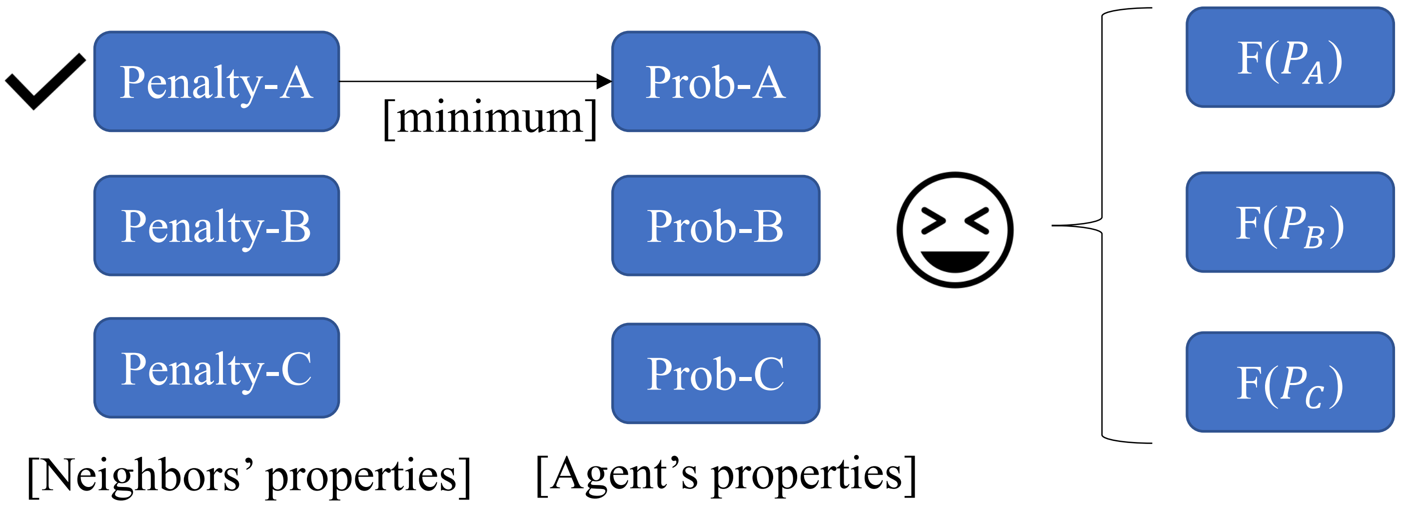
- Where s_i is the status of utterances that agent i using grammar G_P to generate in one iteration, $s_i/{d_i}^2$ is the net impact of agent i of grammar G_P on the learner, distance is applied in squared form, which is similar with the laws of gravitation. The summation $\sum(s_i/{d_i}^2)$ is the sum of the impacts of grammar G_P of all the connected neighbors, N_p is the number of the neighbors using utterances that generated by grammar G_P in one iteration.

Learning algorithm

- Suppose learning is the adaptive changes in the weight of grammars in response to the utterances successively presented to the learner (Charles D. Yang, 2001). In our learning algorithm, split individual learning process in each iteration into successive queuing utterances, learner select one of its inherited grammars to analyze one utterance at one time, instead of receiving and analyze utterances simultaneously.
- Write $G \rightarrow s_i$ if grammar G can analyze utterance s_i; write $G \nrightarrow s_i$ if grammar G fails to analyze utterance s_i. Assume γ is the learning rate. Therefore, given an input utterance s_i, the agent select grammar G_p with probability P_p to analyze:
- when $G_p \rightarrow s_i$, $\begin{cases} P_p^{'} = P_p + \gamma(1 - P_p) \\ P_j^{'} = (1 - \gamma)P_j \end{cases}$ if $j \neq p$
- when $G_p \nrightarrow s_i$, $\begin{cases} P_p^{'} = (1 - \gamma)P_p \\ P_j^{'} = \frac{\gamma}{N-1} + (1 - \gamma)P_j \end{cases}$ if $j \neq p$

Output algorithm

- Agent has preference of using the grammar with smallest penalty probability to produce utterances, the probability of using the selected grammar is consistent with filtering function F.
- $F(p) = \frac{1}{1+e^{-\alpha * p_p}}$
- p_p is the probability that an agent chooses grammar p to analyze the received utterance, it also represents the proportion of grammar p in agent's lingual state.



Code implement

- Initialization of agent properties and social networks

```
extensions [ nw rnd]

turtles-own [
  age
  prob-a
  prob-b
  prob-c
  lingual
  Sa
  Sb
  Sc
  Na
  Nb
  Nc
  la
  lb
  lc
  penalty-a
  penalty-b
  penalty-c
  select-state
  choose-state
  spoken-state
]
```

```
to setup
  clear-all
  set-default-shape turtles "circle"
  ask patches [set pcolor gray]
  if network-type = "Preferential Attachment"
  [ repeat N [ make-node ]
    create-network
    repeat 1000 [layout] ]

  if network-type = "Small world"
  [ repeat N [make-node]
    layout-circle (sort turtles) max-pxcor - 1
    wire-them ]
  distribute
  ask turtles [
  initialization
  lingual-state ]
  plot-degree
  reset-ticks
end

to make-node
  create-turtles 1 [
    rt random-float 360
    fd max-pxcor
    set size 2
    set prob-a 0
    set prob-b 0
    set prob-c 0
    set age 1
  ]
end
```

Code implement

- Initialization of age distribution and grammar distribution

Age stage	Age	Percantage
Infant	0 – 9	12%
Adolescent	10 -19	12%
Younger adult	20 – 34	20%
Older adult	35 – 65	40%
Elder people	65+	16%

```
to distribute
  ask n-of ( 0.12 * N ) turtles
    [ set age 2 ]
  ask n-of ( 0.2 * N ) turtles
    [ set age 3 ]
  ask n-of ( 0.4 * N ) turtles
    [ set age 4 ]
  ask n-of ( 0.16 * N ) turtles
    [ set age 5 ]
  let adults turtles with [age != 1]
  let N-adults count adults
  ask n-of ((percent-grammar-1 / 100) * N-adults ) adults
    [ set prob-a 1.0 ]
  ask n-of ((percent-grammar-2 / 100) * N-adults ) adults
    [ set prob-b 1.0 ]
  ask n-of ((1 - ((percent-grammar-2 + percent-grammar-1)/ 100)) * N-adults) adults
    [ set prob-c 1.0 ]
  ask turtles [
    set spoken-state ""
    update-color
  ]
end
```

Reference: take age structure in US society as reference

Code implement

- Weighting algorithm

- $\hat{l}_p = {N_p}^a(\sum(s_i/{d_i}^2)\,/N_p)$
- $s_i = \begin{cases} 1 & if\ G_P \rightarrow\ s_i \\ 0 & if\ G_P \nrightarrow\ s_i \end{cases}$

```
to initialization
    let nearby sort nw:turtles-in-radius 2
    set Sa 0
    set Sb 0
    set Sc 0
    let i 0
    set Na count (nw:turtles-in-radius 2) with [ prob-a > 0 ]
    set Nb count (nw:turtles-in-radius 2) with [ prob-b > 0 ]
    set Nc count (nw:turtles-in-radius 2) with [ prob-c > 0 ]
    while [ i < count (nw:turtles-in-radius 2) ] [
      let turtle-x item i nearby
      let d-x nw:distance-to turtle-x
      if d-x != 0 [
      set Sa Sa + (([prob-a] of turtle-x) / (( d-x ) ^ 2))
      set Sb Sb + (([prob-b] of turtle-x) / (( d-x ) ^ 2))
      set Sc Sc + (([prob-c] of turtle-x) / (( d-x ) ^ 2))]
      set i (i + 1) ]

    ifelse age != 1 [
      let numlist (list Na Nb Nc )
      let sumlist (list Sa Sb Sc )
      if Sa + Sb + Sb != 0 [
      set penalty-a 1 - ( Sa / (Sa + Sb + Sb))
      set penalty-b 1 - ( Sb / (Sa + Sb + Sb))
      set penalty-c 1 - ( Sc / (Sa + Sb + Sb))]
  ] [
      if Na != 0 [set la Sa / (Na ^ 0.5) ]
      if Nb != 0 [set lb Sb / (Nb ^ 0.5) ]
      if Nc != 0 [set lc Sc / (Nc ^ 0.5) ]
      if la + lb + lc != 0 [
      set prob-a la / (la + lb + lc)
      set prob-b lb / (la + lb + lc)
      set prob-c lc / (la + lb + lc)]]
    update-color
end
```

Code implement

- Learning algorithm
- when $G_p \to s_i$, $\begin{cases} P_p' = P_p + \gamma(1 - P_p) \\ P_j' = (1-\gamma)P_j \end{cases}$ if $j \neq p$
- when $G_p \nrightarrow s_i$, $\begin{cases} P_p' = (1-\gamma)P_p \\ P_j' = \frac{\gamma}{N-1} + (1-\gamma)P_j \end{cases}$ if $j \neq p$

```
to listen [heard-state]
  let gamma 0.02 * (1 - (1 / (1 + exp (-0.1 * ( age - 1 )))))
  let state [ "a" "b" "c" ]
  let prob (list prob-a prob-b prob-c )
  let ind-1 [ 0 0 0 ]
  let ind-2 [ 0.5 0.5 0.5 ]
  let pairs (map list state prob)
  set choose-state first rnd:weighted-one-of-list pairs [ [p] -> last p ]
  ifelse heard-state = choose-state [
    let ind_1 replace-item (position choose-state state) ind-1 1
    set prob (map [ [ P ind ] -> (1 - gamma) * P + (ind * gamma) ] prob ind_1) ]
  [ let ind_2 replace-item (position choose-state state) ind-2 0
    set prob (map [ [ P ind ] -> (1 - gamma) * P + (ind * gamma) ] prob ind_2) ]
  set prob-a item 0 prob
  set prob-b item 1 prob
  set prob-c item 2 prob
end
```

Code implement

- Output algorithm
- $F(p) = \frac{1}{1+e^{-\alpha * p_p}}$

```
to speak
  let state [ "a" "b" "c" ]
  let penalty (list (1 - penalty-a) (1 - penalty-b) (1 - penalty-c) )
  let prob (list prob-a prob-b prob-c )
  let pairs (map list state penalty)
  set select-state first rnd:weighted-one-of-list pairs [ [p] -> last p ]
  let filter-val item (position select-state state) prob
  let filter-prob  1 / (1 + exp(-(10 * filter-val - 5)))
  ifelse random-float 1.0 <= filter-prob [
    set spoken-state select-state ]
  [ set spoken-state one-of state ]
end
```

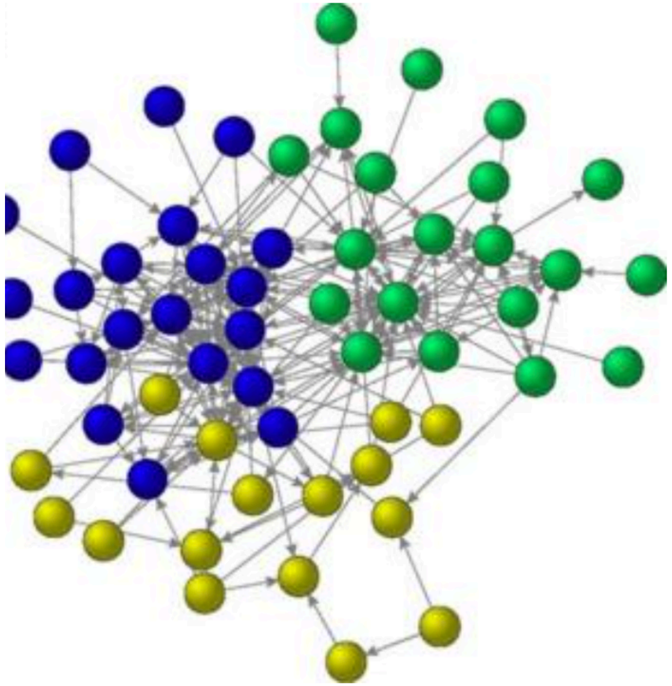
Code implement

- Update agents’ ages, when agent died out and replace it with a new agent

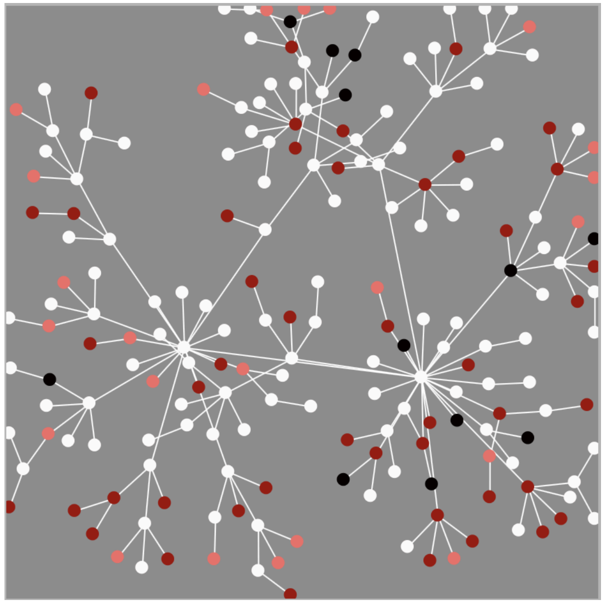
```
to go
  ask turtles [
  communicate-via update-algorithm
  initialization
  lingual-state]
  ask turtles [update-color]
  if ticks > 0 and ticks mod 100 = 0 [
  ask turtles [
  set age age + 1
  while [ age > 5 ]
  [ set age 1 ]]]
  plot-degree
  tick
end

to communicate-via [ algorithm ]
  if (algorithm = "reward")
  [ speak
    ask link-neighbors
    [ listen [spoken-state] of myself ]
]
end
```

Problems encountered

- How to make spatial distribution when initialize the grammars
- Currently, the grammars in initialization is randomly assigned according to the percentage, different monolingual state agents are mixed with each other in social networks, however, it doesn't match with the real world situation, that agents with similar grammars will cluster and gather around.



Clustering in real world



My simulation in Netlogo

Problems encountered

- Unable to simulate in large scale nodes (about 10000 nodes)
- Very time consuming to initialize and run the code in Netlogo, cause kernel interruption
- Tried to run sbatch file in HPC cluster, but outputs include 4 spreadsheet and have error to save output files. Moreover, the Netlogo version in HPC is 6.0.4, which is older than the version I used to construct nlogo file.

```
#!/bin/bash
# Request 3 Gigabytes of RAM per core
#$ -l rmem=3G
#Combine stderr and stdout into one file
#$ -j y
#Make sure that the value below matches the number of threads
#$ -pe openmp 4
#Make sure this matches the number of openmp slots requested
threads=4

module load netlogo/6.0.4

netlogo-headless.sh --model /home/kl3751/language_2.nlogo --table /home/kl3751/output_$thre
ads.csv --experiment trilingual_1

echo "$threads threads requested"

#You have to run netlogo from its install directory in order for extensions to work
cd $NETLOGO_DIR
java -Xmx1024m -Dfile.encoding=UTF-8 -cp $NETLOGO_DIR/NetLogo.jar org.nlogo.headless.Main -
-model $SGE_O_WORKDIR/$model --experiment $experiment --table $SGE_O_WORKDIR/$output_table
--threads $threads
```

Problems encountered

- Can I verify the results and accuracy of improved agent-based model by self-modified multilingual macroscopic Abrams & Strogatz differential equation?
- Bilingual Abrams & Strogatz differential equations have been widely proved that they can be used to describe the dynamic process of language competition, based on the practical data.
- Because I don’t have real world data on language competition, it is a difficult problem to test the accuracy of model simulation results.

Self-modified multilingual macroscopic Abrams & Strogatz differential equation

- $\begin{cases} P_i(A \to A) = 1 \\ P_i(B \to B) = 1 \\ P_i(C \to C) = 1 \end{cases}$
- $\begin{cases} P_i(AB \to A) = S_A \cdot (1 - \delta_i^B) \\ P_i(AB \to B) = S_B \cdot (1 - \delta_i^A) \\ P_i(AB \to AB) = 1 - S_A \cdot (1 - \delta_i^B) - S_B \cdot (1 - \delta_i^A) \end{cases}$
- $\begin{cases} P_i(AC \to A) = S_A \cdot (1 - \delta_i^C) \\ P_i(AC \to C) = S_C \cdot (1 - \delta_i^A) \\ P_i(AC \to AC) = 1 - S_A \cdot (1 - \delta_i^C) - S_C \cdot (1 - \delta_i^A) \end{cases}$
- $\begin{cases} P_i(BC \to B) = S_B \cdot (1 - \delta_i^C) \\ P_i(BC \to C) = S_C \cdot (1 - \delta_i^B) \\ P_i(BC \to BC) = 1 - S_B \cdot (1 - \delta_i^C) - S_C \cdot (1 - \delta_i^B) \end{cases}$
-
- $\begin{cases} P_i(ABC \to AB) = (1 - S_C) \cdot (1 - \delta_i^C) \\ P_i(ABC \to AC) = (1 - S_B) \cdot (1 - \delta_i^B) \\ P_i(ABC \to BC) = (1 - S_A) \cdot (1 - \delta_i^A) \end{cases}$
- $P_i(ABC \to ABC) = 1 - (1 - S_C) \cdot (1 - \delta_i^C) - (1 - S_B) \cdot (1 - \delta_i^B) - (1 - S_A) \cdot (1 - \delta_i^A)$