Rough guide to use THSim v4.0

Paul Vogt
Language Evolution and Computation Research Unit, University of Edinburgh
40 George Square, Edinburgh EH8 9LL, UK.
Induction of Linguistic Knowledge / Computational Linguistics, Tilburg University
P.O. Box 90153, 5000 LE Tilburg, The Netherlands.
paulv@ling.ed.ac.uk, http://www.ling.ed.ac.uk/~paulv/

August 13, 2004

1 Introduction

THSim v4.0 is a simulation toolkit for studying aspects of language evolution. The main functionality of the toolkit is described in [2] and the user is referred to this paper for a functional description of THSim. If you intend to use the toolkit for research, please acknowledge the author of the toolkit by citing that paper in any publication that comes out of your research.

This manual is served to help the THSim user to install the package and to start investigating the system. The manual is written such, that the reader is assumed to be familiar with the general functionality of the system, so it is advisable to read [2] first. New additions of version v4.0 with respect to versions v3.2 and v3.2.1 include two different types of agents. These agents are described in [3, 1]. If you use either of these agents in published research, please cite the relevant paper too.

The remainder of this manual discusses the installation of THSim and how to get started. Then it briefly summarises the different agent types (Section 4) and game types (Section 5). Various parameters are discussed in Section 6. Section 7 describes the various output modes and presents some hints and tips how to analyse the data. Finally, Section 8 presents some additional notes on command line processing.

2 Installation

To install THSim, you first have to download the package `thsim.tar.gz` from `http://www.ling.ed.ac.uk/~paulv/thsim.html`. Once you have dowloaded `thsim.tar.gz`, you unpack the package in any directory you like, e.g. `~/thsim/` (Linux) or `c:\thsim` in windows.

Be sure to set Java's classpath to the current directory. In Linux this can be done by adding the line

```
export CLASSPATH=.
```

to your `.bashrc` file. In windows you can either adjust the `autoexec.bat` or provide the argument `-classpath=...` when running java. In some cases, you might need to recompile the package (especially if you have a different version of java, or if java is installed at a different location than with which it was compiled). This can be done using `make` or by `javac THSim.java`. The version of java that should work is Sun's Java 2 Platform, Standard Edition v 1.4.1 or higher (http://java.sun.com).

There is a limited documentation of the source code, which can be found in the `docs` directory of the package, or the online version at `http://www.ling.ed.ac.uk/~paulv/thsim/docs/index.html`.

THSim should work well under Windows and Mac, but it has only been tested recently under the Linux operating system. Some of the features, especially with respect to some data-output, are specifically designed for use under Linux. Regarding these aspects, you are on your own I'm afraid...

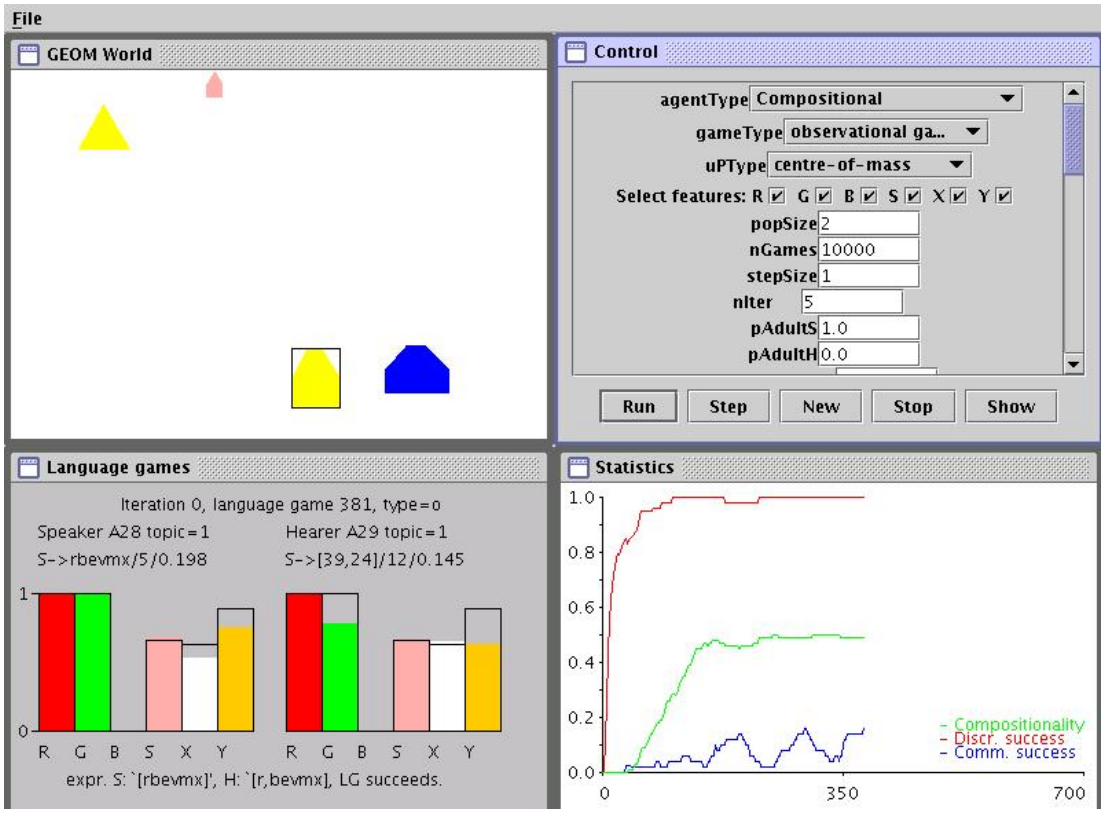

Figure 1: A screen shot of THSim.

3 Getting started

There are basically two ways to start running THSim: with or without the user-interface. In this section, THSim is explained in case you start it with the interface. To start up the interface, you simply type from the dos or linux/unix prompt `java THSim`.

If all goes well, THSim is started displaying the screen shown in Fig. 1. In the window, there are four subpanels. The GEOM world, Language games, Statistics and Control panels. This first three panels are only output panels and display various information to the screen (see Section 7); the Control panel allows you to start the game and change various parameter settings. To start a simulation with the default setting, simple push the **Run** or **Step** button.

When the **Run** button is pressed, THSim will – in the default setting – simulate 1000 guessing games played by a population of 2 compositional agents under a number of conditions set in the control panel. Using the **Step** button, the system will run for the number of games indicated with the parameter `stepSize`, which at default is set to 1. For both buttons, the simulation will only proceed to play language games if the current language game number is lower than `nIter`×`nGames`.

When you want to start a new game, press the **New** button. This reinitialises the simulation. It is often necessary to press this button when you want to run the simulation after you have changed a parameter, because I do not guarantee that all settings will be changed properly when the **New** button is not pressed and sometimes the simulator may even crash in these circumstances.

When you want to stop the simulation when running, you can press **Stop**. When it has stopped (either because you have pressed **Stop** or because the simulation has ended), you can press the **Show** button to show you the lexicon or grammar (depending on the type of agent). In order to proceed running the simulation, the lexicon window needs to be closed before you press **Run**, **Step** or **New** again.

Table 2 presents an overview of the parameters that can be set in the control panel, together with their default setting. These will be discussed in Section 6. Two tips on running the simulation: 1) If you simulate a large population, you might need to increase the heap size of the java interpreter, e.g.,

by specifying the maximum heap size using the argument `-Xmx256MB`. 2) Java is faster when you let the interpreter compile the simulation first as a server by specifying the argument `-server`. However, this will take longer for the simulator to start running, so it only useful for longer simulations. Using these options, you will have to start the program using:

```
java -Xmx256MB -server THSim
```

4 Three agents

In THSim, three different classes of agents have been implemented: 2 holistic agents (HA & HA2) and 1 compositional agent (CA). The holistic agents only develop lexicons (or holistic communication systems) and the compositional agent can develop a language with compositional structures. The two holistic agents differ in the way their semantics are constructed.

HA The agent defined in HolisticAgent.java has one conceptual space into which meanings are constructed as prototypes, see [2] for details. HA is in the UI referred to as 'Holistic - flat'.

HA2 The agent defined in HolisticAgent2.java has a hierarchically layered organisation of conceptual spaces. Each layer allows a different number of meanings (represented as prototypes), thus having different taxonomies on categories. The layer with the least number of meanings (ie. lowest density) contains general categories and the most dense layer contains specialised categories. The details of this agent are explained in [3].

CA The compositional agent is defined in CompositionalAgent2.java and is described in detail in [1].

5 Three games

THSim has three different types of language games implemented: *observational games* (OG), *guessing games* (GG) and *selfish games* (SG), see [2] for details. These three games differ in the way hearers have to infer the speakers' intentions and in the way associations between the semantics and signals are adapted. In the current implementation, all games can be played by HA and HA2; CA can only play OG and GG (the SG will be implemented in due time). Below follows a brief summary of the three games.

Observational game: In the observational game, the speaker draws the attention of the hearer to the topic the speaker selected to communicate about, thus establishing joint attention prior to the verbal interaction. This way, the hearer observes both the signal and the reference, allowing association between signal and meaning to be adapted following Hebbian learning.

Guessing game: In the guessing game, the speaker does not draw the hearer's attention to the topic, so the hearer has to guess the speaker's intention from its utterance. After the hearer guessed something, the agents verify whether the guess was right or not. If it was not right, the speaker provides *corrective feedback* so the hearer can adopt new knowledge and adapt associations using reinforcement learning.

Selfish game: In the selfish game, the hearer also has to guess the speaker's topic, but the agents do not verify the success of the hearer's guess. Associations between signals and meanings present in a context (or situation) are increased. Learning then proceeds following cross-situational statistical learning.

Table 1 summarises the three different language games.

type	Joint Attention	Corrective Feedback	Learning Rule
OG	+	-	Hebbian
GG	-	+	Reinforcement
SG	-	-	Cross-situational statistical

Table 1: A summary of the three different game types

6 Parameters of the Control Panel

Table 2 presents the parameters that can be set in the control panel. This section explains these parameters in more detail, see also [2, 3, 1] for more details on some of the parameters.

agentType specifies the type of agent. `Holistic - flat` relates to HA, `Holistic - hierarchical` relates to HA2 and `Compositional` (default) to CA.

gameType specifies the type of language game that is played during a simulation, unless `varGames=true` because that overrides `gameType`. The games that can be set are `observational games`, `guessing games` (default) and `selfish games`.

uPType allows the experimenter to set the rule with which prototypes are moved in the direction of the observation. Four rules can be selected from: `centre-of-mass` (default), `simulated annealing`, `walk` and `none`. In the latter case, the prototypes are static. For the other rules, consult [2].

features allows the user to select those features that the agents use for categorising objects. The features that can be selected from are `R`ed, `G`reen and `B`lue colour components, a `S`hape feature, `X`-axis position and `Y`-axis position (see [2] for a description).

popSize is the total population size at each instance. If the ILM is activated (ie. `nIter>1`, the number of adults are `popSize/2`, which is rounded to the floor: if `popSize` is an odd number (e.g., if `popSize=3`, then `nAdults=1`). The number of learners is `nLearners=popSize-nAdults`.

nGames lets you specify the number of games that are played each iteration. I.e. the simulation runs until the game number `lg` equals `nIter`×`nGames`. When the simulation has been stopped somewhere and (`lg mod nIter`)≥`nGames`, the simulation does not continue to run when you press **Run** or **Step**.

stepSize lets you specify the number of games played when the Step button is pressed, as long as (`lg mod nIter)<nGames`.

nIter If `nIter=1`, there is no population turnover in the experiment and all agents are considered equal. If `nIter>1`, there is a population turnover according to the iterated learning model, and half of the population are adult agents and the other half are learner agents (TIP: let `popSize` be an even value)

pAdultSpeaker is the probability with which the speaker is selected from the adult population. The speaker is selected from the learner population with probability `1-pAdultSpeaker`. This only applies when `nIter>1`.

pAdultHearer is the probability with which the hearer is selected from the adult population. The hearer is selected from the learner population with probability `1-pAdultHearer`. This only applies when `nIter>1`.

trainingSetSize is the parameter with which bottlenecks can be regulated. If `trainingSetSize>0`, a bottleneck is imposed as follows: At the start of each iteration, `trainingSetSize` different objects are

Parameter	Value	Description	Possible values
agentType	Compositional	The type of agent that is initialised.	Holistic - flat, Holistic - hierarchical, Compositional
gameType	guessing game	Indication which type of language game is played during the simulation.	Observational game, guessing game, selfish game
uPType	centre-of-mass	Indication with which adaptation scheme the prototypes are moved toward the topic's feature vector at the end of a successful disrimination game.	centre-of-mass, simulated annealing, walk, none
features	RGBAXY	Indication which features are used by the agents.	RGBAXY
popSize	2	Number of agents in the population.	$N > 2$
nGames	1000	Number of games played (during one iteration).	$N > 0$
stepSize	1	Number of game played when 'Step' button is pressed.	$N > 0$
nIter	1	Number of iterations.	$N > 0$
pAdultSpeaker	1.0	Probability that speaker is selected from adult group.	$0 \leq X \leq 1$
pAdultHearer	0.0	Probability that hearer is selected from adult group.	$0 \leq X \leq 1$
trainingSetSize	0	Bottleneck size	$0 \leq N \leq 120$
pOG	1.0	Probability that an OG is played (varGames).	$0 \leq X \leq 1$
pGG	0.0	Probability that a GG is played (varGames).	$0 \leq X \leq 1$
pWC	1.0	Word-creation probability.	$0 \leq X \leq 1$
alphabetSize	26	Size of the alphabet (Compositional Agent only)	$1 \leq N \leq 26$
etaN	0.9	Learning parameter for OG and GG.	$0 \leq X \leq 1$
etaS	0.9	Learning parameter for SG.	$0 \leq X \leq 1$
preLing	0	Nr. of games played before learner starts inducing grammar (CompositionalAgent only)	$0 \leq N \leq$ nGames
memMeanings	1000	Size of memory for meanings (HolisticAgent(2))	$N > 0$
memWords	1000	Size of memory for words/symbols (HolisticAgent(2))	$N > 0$
cxtSize	4	The context size.	$2 \leq N \leq 8$
foa	4	Size of the focus of attention	$1 \leq$ cxtSize.
pNoise	0.0	Perceptual noise.	$0 \leq X \leq 1$
popGrowth	2	Growth of the population.	$0 \leq N \leq$ maxAgents
maxAgents	20	Maximum population size (incrPop).	$N > 0$
workDir	null	Working directory for print scores	any valid directory.
testPop	true	Indication whether the population is tested or not	true/false
printScores	false	Indication whether the scores are traced or not	true/false
S2S	true	Indication whether in the first iteration all agents are considered equal (true) or not (false).	true/false
varGames	false	Indication whether different types of games are played during a simulation run.	true/false
uScore	true	Indication whether or not the association scores are updated according to the score-based rules.	true/false
adaptSG	true	Indication whether the hearer adapts its lexicon in SG.	true/false
fCol	true	Indication whether the colours are selected from a fixed set of 12 different colours.	true/false
incrPop	false	Indication whether the population grows incrementally.	true/false
log	false	Indication whether the simulation is logged.	true/false
lex	false	Indication whether the lexicon is saved at the end of a simulation run.	true/false
printGame	false	Indication whether to write the output of a language game to the screen.	true/false

Table 2: An overview of the parameters that can be set in the control panel. The parameters (1st column) with default values (2nd column) are described (3rd column) and have possible values (4th column). For the possible values N means it has to be an integer and X means it has to be a double value. See the text for further details.

selected randomly from the 120 different objects (10 shapes $\times$ 12 colours) to form a training set. During each iteration, the population only communicate about the objects from the training set.

varGames pOG pGG can be used to vary the types of language games played within one simulation. If `varGames=true`, the simulation selects each time one type of game from the OG, GG and SG. The probability with which an OG is selected can be set by `pOG`, the probability with which a GG is selected can be specified by `pGG` and the probability for selecting a SG is `pSG=1-pOG-pGG`.

pWC is the word-creation probability. The probability with which the speaker creates a new word when it is unable to produce an utterance.

alphabetSize is the number of letters that are in the alphabet of a CA. At default, this is set to the maximum of 26, which comprises the entire English alphabet. If lower, the alphabet contains the first alphabetSize letters of the alphabet. So, when `alphabetSize=3`, the alphabet is {a,b,c}.

etaN is the learning parameter η used for the OG and the GG.

etaS is the learning parameter η used for the SG.

preLing sets the number of language games before a learner CA starts inducing language. Thus the first `preLing` games are a 'pre-linguistic' phase used to develop a part of the ontology.

memMeanings sets the memory size of HA and HA2 for the number of meanings the agent can store. If the limit is reached, the agents start a forgetting mechanism that prunes all elements (both words and meanings) of which the scores (or probabilities) are lower than a certain threshold.

memWords sets the memory size of HA and HA2 for the number of words (or symbols) the agent can store. If the limit is reached, the agents start a forgetting mechanism that prunes all elements (both words and meanings) of which the scores (or probabilities) are lower than a certain threshold.

cxtSize The size of the context specifies the number of objects that are constructed for each language game. It has been found that the maximum context size is 8, which is due to the limited spatial size of the display of the GEOM-world. If `cxtSize` is larger, then the simulation might end up in a infinite loop. This should be fixed in future releases.

foa specifies the focus-of-attention size. The focus-of-attention is a subset of the context from which the topic is selected and used by the hearers to guess the reference of a topic from. As it is a subset of the context, `foa`$\leq$`cxtSize`.

pNoise determines the noise that is applied to each feature perceived by each individual agent. The noise is calculated based on the Gaussian function:

$$G(x,\sigma) = \begin{cases} e^{-(\frac{x}{2\sigma})^2} & \text{if } x < 0 \\ 2 - e^{-(\frac{x}{2\sigma})^2} & \text{if } x \geq 0 \end{cases} \tag{1}$$

where $\sigma =$`pNoise` and x is a random value. Now, each feature f_i is perceived by an individual agents as f_i' according to

$$f_i' = f_i * G(x,\sigma) \tag{2}$$

where $x \in [-0.5, 0.5]$ is a randomly generated value and i is the feature's dimension (e.g. the red component of the RGB colour space). Each feature is always bounded $f_i' \in [0,1]$. In order to have noise, the value `pNoise` must be larger than 0. However, the smaller `pNoise` is, the bigger the actual noise. If `pNoise=0`, no noise is calculated. Figure 2 shows the $G(x,\sigma)$ for different values of σ.

incrPop popGrowth maxAgents allow the experimenter to investigate how the the population can be scaled up. If the population is large from the beginning, this is difficult. However, if the population grows incrementally, larger populations might develop a stable communication system. If `incrPop=true`,

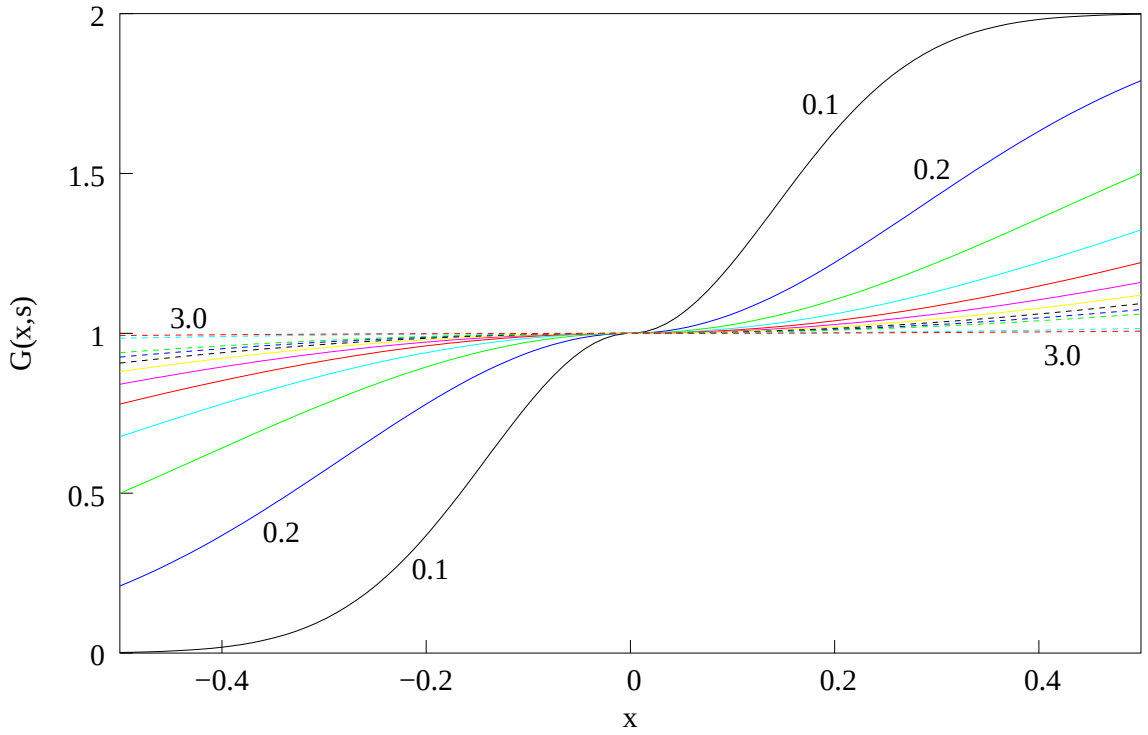

Figure 2: The Gaussian-based function $G(x, \sigma)$ that regulates the noise for different values of σ = pNoise = $\{0.1, 0.2, \ldots, 1.0, 2.0, 3.0\}$. In the production of noise, a random value $-\frac{1}{2} \leq x \leq \frac{1}{2}$ is produced and the result of $G(x, \sigma)$ is multiplied by the original feature values. Here, σ = pNoise is specified by the experimenter. If pNoise=0, the feature values remain undisturbed.

the population increases each iteration with the number specified with `popGrowth`, until a maximum population size is reached of `maxAgents`. `incrPop` only has effect when `nIter>1`, while `popGrowth` and `maxAgents` only apply to `incrPop`.

workDir sets the work directory in which the files are stored for tracing the word-meaning scores (see Section 7.9). If you specify a directory that does not exist, THSim will try to create the directory. If that fails, THSim will produce an error message and quits.

testPop If `testPop=true`, the population is tested at the end of each iteration/simulation. During this test, 200 different situations (context and topic) are constructed. Then each agent plays the role of a speaker, and each agent tries to interpret (or decode) the utterances produced by the other agents. From these tests, a couple of measures are calculated and written to the standard output. During the test period, agents do not adapt their knowledge. (See Section 7 for more information.)

printScore indicates whether or not the word-meaning scores of some words are traced and put into a file (see Section 7.9). These files are stored in the working directory specified with `workDir`. This works only in combination with HA and HA2.

S2S regulates wether or not in the first iteration all agents are considered equal or not. When they are (default), the first iteration makes no distinction between adults and learners and each agent is equally likely to be selected as speaker or hearer. If `S2S=false`, the population in the first iteration distinguishes between adults and learners and the selection of speakers and hearers are regulated according to `pAdultSpeaker` and `pAdultHearer`. In all cases, the system will work as a 'normal' iterated learner from the second iteration onwards. This only applies when `nIter>1`.

uScore allows the user to specify whether the association scores are adapted based on the score functions, e.g., $\sigma = \eta \cdot \sigma + 1 - \eta$, or based on the usage. The default value (true) selects the score-based update.

name	code	RGB-vector
black	000	[0.00,0.00,0.00]
blue	010	[0.00,0.00,1.00]
cyan	020	[0.00,1.00,1.00]
green	030	[0.00,1.00,0.00]
magenta	040	[1.00,0.00,1.00]
orange	050	[1.00,0.78,0.00]
pink	060	[1.00,0.69,0.69]
red	070	[1.00,0.00,0.00]
yellow	080	[1.00,1.00,0.00]
gray	090	[0.50,0.50,0.50]
dark gray	100	[0.25,0.25,0.25]
light gray	110	[0.75,0.75,0.75]

Table 3: The colours of the objects with names, description code (for output) and the RGB values.

adaptSG can be used to turn off the adaptation of scores in the SG. This can be useful to investigate the effect of score adaptation in the SG relative to the OG and GG when `varGames=true`.

fCol indicates whether or not the colours are selected (randomly) from a fixed set of predefined colours. If this is not selected, the colours are generated as random RGB vectors, ie. each component R, G and B is assigned a random value between 0 and 1.

log indicates whether the language game data is saved in a file. When `log=true`, the system will ask for a filename.

lex indicates whether the lexicon is saved in a file at the end of each iteration. The system will ask for a filename when `lex=true`.

printGame tells the system to write the output of a game to the standard output of the program.

7 Output

THSim has various ways of digesting information that is relevant for a user. In the user interface there are three windows that display information to the user, see Fig. 1. By pressing the **Show** button, you can additionally display the lexicon or grammar on the screen as will be discussed shortly. Additionally, there are a number of different ways to output information in a file or to the standard output. The remainder of this section describes the output presented by THSim.

7.1 GEOM world

The left top window of THSim displays the GEOM world. GEOM world is the environment where the agents 'look' at during a language game. This environment is drawn for every language and displays the figures that are generated by the simulator. The figures displayed form the context of the language game, consult [2] for more details.

The objects can have 12 different colours (see Table 3) and 10 different shapes (see Table 4). Each object is positioned at a x- and y-coordinate in the display. The agents detect the feature values of the objects as values between 0 and 1. These values depend on the aspects of the objects. The colour features the agents can detect are the components of the RGB vector of the colour. The shape feature (f_S) is calculated based on the area A_s of the shape and the area A_b of the smallest rectangle that can be drawn around the shape according to:

$$f_S = 2(\frac{A_s}{A_b} - \frac{1}{2}) \tag{3}$$

name	code	feature
circle	0	0.57
triangle	1	0.00
rectangle	2	1.00
square	3	1.00
pentagon	4	0.50
regular hexagon	5	0.33
irregular hexagon 1	6	0.67
irregular hexagon 2	7	0.77
irregular pentagon	8	0.88
cross	9	0.11

Table 4: The shapes of objects with names, description code (for output) and their feature values, calculated following Eq. (3).

7.2 Language games

The left bottom window of THSim shows more information about the language game that is played. On the top line the window provides information concerning the iteration number, the language game number and the type of language game (o=OG, g=GG, s=SG).

The second row presents which agents are speaker and hearer and what topic they identified. If no topic was identified, the hearer's topic is indicated with -1. (This also happens if the speaker does not produce an utterance.)

The third row indicates for holistic agents (HA and HA2) which meaning the agents used to distinguish the topic. If the discrimination game failed, this is indicated by the message "Discr. game fails". This message also appears if the speaker did not produce an utterance, because then the hearer did not receive an utterance.

For compositional agents (CA), the third row shows the rule that is used, where the rule is of the form `S->word/F/S` or of the form `S->[3,60]/F/S`, where F is the frequency with which the rule has been used and S is the rule weight. The 'word' is presented when the rule is holistic. Compositional rules are specified by the *covers* of the conceptual space (or linguistic category) they rewrite to. A cover is given by the integer representation of a bit-string, which indicates which dimensions belong to the conceptual space. This is calculated as $\sum_{i=1}^{n} \delta_i 2^{i-1}$, where n is the dimension of all features used in the simulation (depends on which features you select), and

$$\delta_i = \begin{cases} 1 & \text{if } i \text{ is a dimension of the the conceptual space.} \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

For instance, if the agents use all features RGBAXY to perceive their world, the value 63 represents the bitstring 111111 relating to all features, the value 3 represents 110000 relating to the space covering dimensions R and G (RG for short), and value 60 represents 001111 relating to conceptual space BAXY. A prefix `new` or `new part` informs us that the rule is new or a part of the rule is new.

Then there are two diagrams indicating the feature vector of the identified topic (open boxes) and the prototype of the meaning (coloured boxes). In Fig. 1 there are 6 features for each agents, when less features have been selected, less will be displayed. The labels on the bottom of the diagram indicate to which feature the value belongs.

Below the diagram, a line appears with the utterance the speaker produced and, if the hearer interpreted it, the line repeats the hearer's interpretation. This is followed by the information 'LG fails' or 'LG succeeds'. In case no expression is made, this is informed as well. For a holistic agent (HA and HA2), the expression is the string given; for the compositional agent, the expression is given in a list between square brackets. This list can have 1 or 2 words depending on whether the rule is holistic or not. (Fig. 1 shows an example of both a holistic and compositional rule.)

7.3 Statistics

The statistics diagram shows the running values of discriminative success and communicative success. These values are calculated over the past 50 language games and is an average of the number of successful discrimination or language games during this period. When the boundary of the diagram is reached, the scale on the x-axis (number of games) is doubled. For the compositional agents, an additional measure 'compositionality' is presented. This is – in contrast to the compositionality presented in [1] – a measure that gives us the average number of compositional utterances used in a period of 50 language games.

At the end of each iteration or at the end of the simulation, the display also shows the test results. For the holistic agents (HA and HA2) these are `pCoherence` and `iCoherence`. `pCoherence` stands for production coherence and is the average number of agents producing the same words to express an object averaged over 200 test games. `iCoherence` is the average number of agents that successfully interpreted the mostly used expressions for the objects over the same 200 test games.

For the compositional agent, the `pCoherence` given at the end of each iteration is the same as above. The `iAccuracy` is the proportion of agents that could successfully interpret the utterances produces by the other other agents, averaged over the 200 test games. The `compositionality` given at the end of the population test, differs from the one shown in the graph. Here it is the proportion of compositional rules with respect to the number of expressions made, averaged over the number of agents, as presented in [1].

7.4 Lexicons and Grammars

When pressing the **Show** button after a simulation has stopped, the lexicon window pops up. This window shows a representation of the lexicon (HA and HA2) or the grammar (CA) for all agents in the population at that moment.

Figure 3 shows the lexicon window for a HA. The lexicon is shown as a table of dimension `popSize` x `nWords`. Each column of the table represents the lexicon of an agent, and each row a word in the agents' lexicons. The cells of the table are filled with grids of dimension NxM, where N=10 is the horizontal dimension in which each grid indicates a value between 0 and 1 (the leftmost cells have values in the range $[0.0, 0.1\rangle$, the second cells have values in the range $[0.1, 0.2\rangle$ and so forth), and the M equals the number of features an agent detects (dimension of the conceptual space). Inside some of the grid cells, black squares of different sizes appear. These squares indicate that the word has a meaning with a value that is represented by the horizontal position of the grid-cell in the dimension specified in the grid cell's row. The size of the square indicates the weight of that part of the meaning with the word.

Following the largest blocks in the grid for word "decihe" for agent A6 (Fig. 3), we see that the agent has a preferred meaning for "decihe" with $f_R \in [0.0, 0.1\rangle$, $f_G \in [0.9, 1.0]$, $f_B \in [0.0, 0.1\rangle$, $f_S \in [0.9, 1.0]$, $f_X \in [0.7, 0.8\rangle$, $f_Y \in [0.8, 0.9\rangle$. Other meanings of A6 for "decihe" use more or less the same colour (green) for different shapes on different positions in the world. Looking at the meanings for the same word of agent A7, we see that it has similar meanings.

Figure 4 shows the grammar window for a population of two compositional agents. This display is very similar to the lexicon shown in Fig. 3 but differs in a number of ways. The first few lines show the compositional rules of the agents grammars, together with their rule weights, displayed as black squares of different size (as before, the size is proportional to the value). The rules rewrite from a sentence `S`[1] to compositions of linguistic categories that are indicated by a string containing the labels of each dimension that is part of the conceptual space of this linguistic category. For instance, the strongest rule (ie. the one with highest rule weight) is of the form `S->R ; GBS`, so where the first constituent has the 1-dimensional conceptual space relating to the red component, and the second constituent has a linguistic category of which the conceptual space is spanned by the dimensions G (green componenent), B (blue component) and S (shape).

The cells of this table contain the terminal rules, which are preceded by a string indicating the dimensions of the conceptual space that determines the linguistic category of the rule. Each cells now prints the word of the rule above the grid, followed by the rule weight (again this is a black square). The grid, again, shows the meaning for this rule, which are now occupied by the categorical features (ie. the

[1] Apologies for the unfortunate coincidence with the letter used to refer to the feature shape.

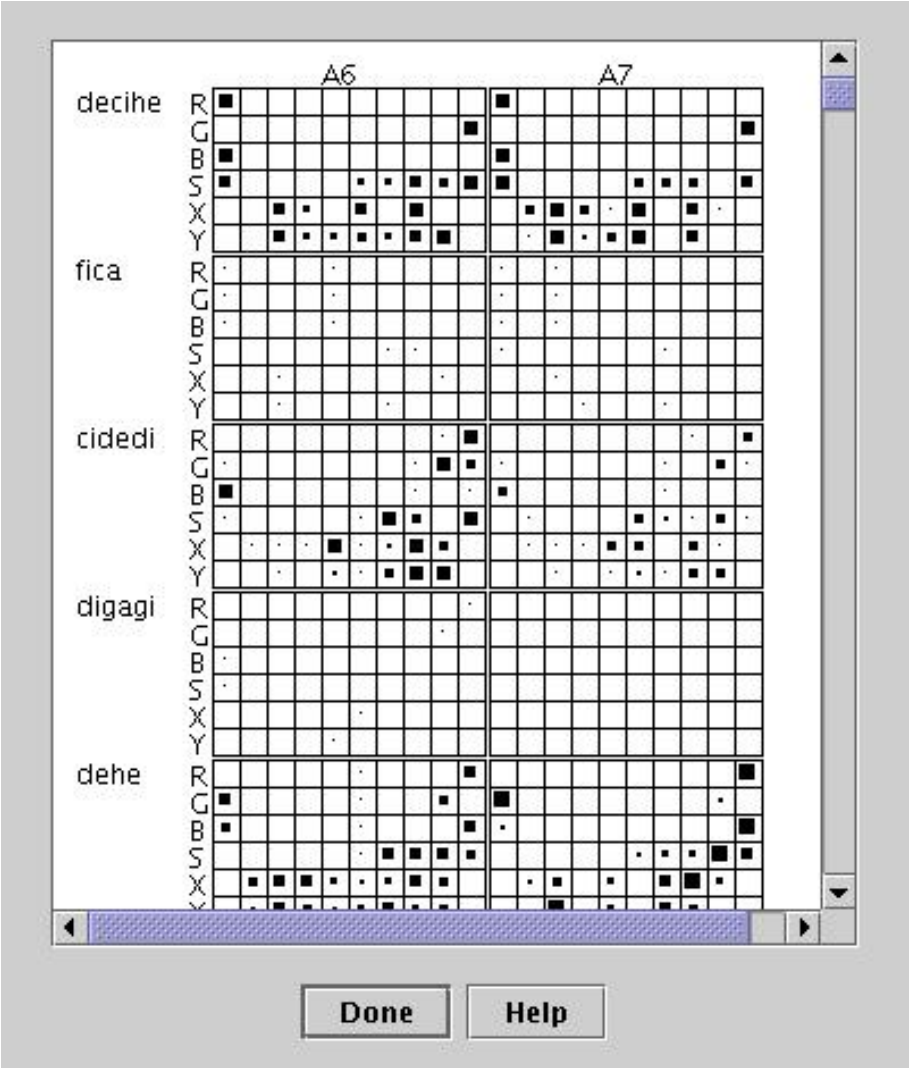

Figure 3: The lexicon window for a holistic agent.

meaning blocks of the CA). At the bottom of Fig. 4 three terminal slots are shown that have a lower dimension than the holistic space. For instance, the lowest terminal rule has the linguistic category that covers dimensions B and S, has a word "muu" and for both agents, the rule weight is low (it looks like 1x1 pixel, so the rule weight is in the range $[0.0, 0.1\rangle$). The categorical features are at the same position in the grids, so they word has – at least – similar meanings for both agents.

The lexicon (and grammar) window is scrollable, so you can scroll through the whole language for all agents. Elements of the that have extremely small weights are excluded for display for reasons of clarity. Before you continue the simulation, you *MUST* press the button **Done** to close the window, otherwise you cannot continue. This is done to prevent THSim to update the language after each game, which would take a lot of overhead on the computations.

7.5 Logfile

When `log=true` the log file outputs for each language game data that can be used for analysing the simulations. For the holistic agents HA and HA2, the data is printed in 17 columns as indicated in Table 5.

Topic's are described (column 3 and 9) to indicate in which range the object's feature fit:

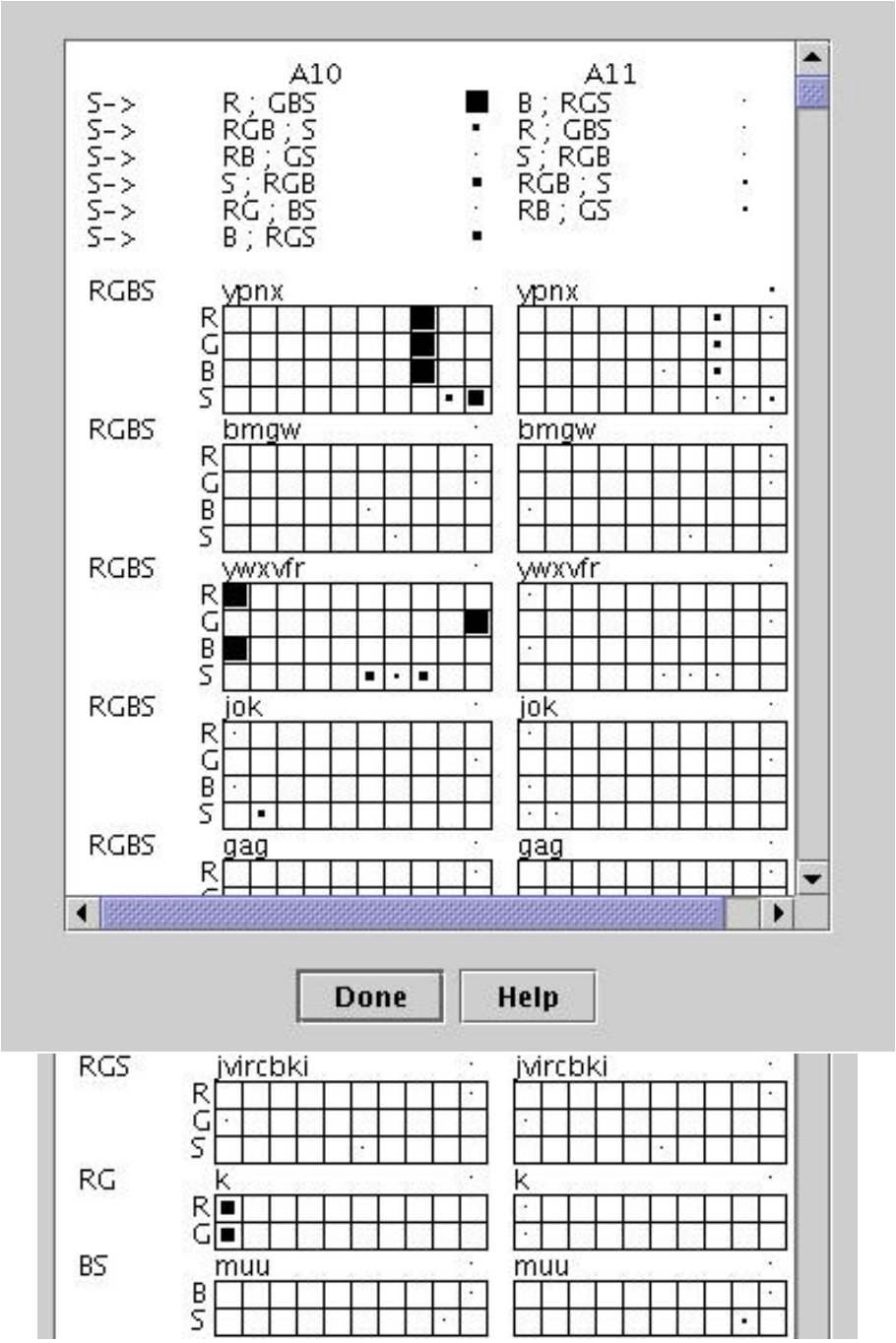

Figure 4: Top: the lexicon window showing the grammar for a compositional agent. Bottom: Another fragment of the grammar.

Colors If the features include the colour features, there are two ways these are descrided

1. if the colours are fixed, the first two places indicate the colour number, see Table 3.
2. otherwise, each value is classified as the X and Y co-ordinates.

Shapes the shape feature (S) is assigned a value indicating one of the 10 objects, see Table 4

XY coordinates the X and Y features are represented with an integer between 0 and 9 indicating their

Col	Abbr.	Description
1	lg	Language game number
2	S	Speaker ID
3	topic	A description of the speaker's topic.
4	mS-id	Meaning ID of the speaker's topic, -1 means no meaning.
5	mS-layer	The hierarchical layer of the meaning.
6	mS-ptype	The meaning's prototype in array notation [-1] means that no meaning is found.
7	uS-form	The form that is uttered, * means no utterance is made.
8	H	The hearer's ID.
9	topic	A description of the hearer's topic.
10	mH-id	Meaning ID of the hearer's topic.
11	mS-layer	The hierarchical layer of the meaning.
12	mH-ptype	The meaning's prototype.
13	uH-form	The utterance recognised by the hearer.
14	$< DS >$	Discriminative success
15	DS	Boolean indicating iscrim. success
16	$< CS >$	Communincative success
17	CS	Boolean indicating commun.success

Table 5: The output of the log file for holistic agents HA and HA2.

place along the x- or y-axis. These features f are classified accroding to the following classification (these may also be used to classify the RGB features when they are constructed randomly):

0: $0.0 \leq f < 0.1$

1: $0.1 \leq f < 0.2$

2: $0.2 \leq f < 0.3$

3: $0.3 \leq f < 0.4$

4: $0.4 \leq f < 0.5$

5: $0.5 \leq f < 0.6$

6: $0.6 \leq f < 0.7$

7: $0.7 \leq f < 0.8$

8: $0.8 \leq f < 0.9$

9: $0.9 \leq f \leq 1.0$

For instance, the object with feature vector (1.0,0.0,0.0,0.57,0.81,0.89) will be classified as 070088 when the color was selected as a fixed color; 070 is the colour code, the final 0 is the shape code, the first 8 encodes the X-coordinate and the second 8 the Y-coordinate. If the object was formed with a randomly constructed RGB vector and has feature vector (0.10,0.22,0.76,0.77,0.35,0.55) is described by 127735. You can use these codes to have an own notion of the topic when analysing the data.

The logfile for the compositional agents has 11 columns as outlined in Table 6.

7.6 Lexicon file

The lexicon file prints the lexicon of holistic agents (HA and HA2) as it is at the end of each iteration, first in a readable (global) format, followed by a table. The readable lexicon lists for each word for each agent the meaning that has the highest association score at the end of an iteration, followed by its association score. Below is a fragment of the output. The first column indicates the agent's ID, in the second column the word is presented, the third to eighth columns indicate the values of the meaning's prototype in each dimension (if less than six features are selected, there are less columns), the final column presents the association score.

Col	Abbr.	Description
1	lg	Language game number
2	$< DS >$	Discriminative success
3	DS	Boolean indicating iscrim. success
4	$< CS >$	Communincative success
5	CS	Boolean indicating commun.success
6	$< COMP >$	Compositionality (as in graph)
7	COMP	Boolean indicating compositional rule use
8	sGramm	Grammar size speaker
9	sHol	Number of holistic rules in grammar speaker
10	hGramm	Grammar size hearer
11	hHol	Number of holistic rules in grammar hearer

Table 6: The output of the log file for the compositional agents.

```
Global lexicon, iteration 9999:

A2 vuto 0.20 0.83 0.28 0.93 0.64 0.62 : 0.40
A3 vuto 0.22 0.52 0.22 0.94 0.78 0.28 : 0.47
A2 taga 0.64 0.58 0.31 0.94 0.72 0.27 : 0.04
A3 taga 0.16 0.25 0.23 0.89 0.31 0.73 : 0.00
A2 dajahe 0.38 0.38 0.83 0.95 0.25 0.41 : 0.39
A3 dajahe 0.59 0.20 0.84 0.93 0.35 0.35 : 0.28
```

After the global lexicon is printed, the full lexicon of each individual agent is printed in a table form. A fragment of this lexicon is given below.

```
A3: nMeanings=65 nSymbols=51
M F dummy vuto taga dajahe ...
0 (-100) 0.0 0.0 0.0 0.0 ...
1 (0.59,0.75,0.81,0.82,0.31,0.20) 0.0 0.0 0.0 0.07880 ...
2 (0.27,0.23,0.31,0.90,0.42,0.30) 0.0 0.14980 0.00254 0.00205 ...
.
.
.
```

The first line gives the agent ID, the number of meanings and symbols (words) the agent has. The second line is the legend of the table: It says that the meanings (M) are in the rows and forms/words (F) are in the columns, followed by a dummy word and a series of words. In the first row, a dummy meaning is given. (The table is a direct copy from the table stored by the agents. The dummies are used for the usage-based approach to reserve space where we can keep track of the sum of the values in the row or column without needing to calculate these every time we need them.) In the first column of the table the meaning ID is given, followed by its prototype in vector notation in the second column. The third column is a dummy, and all other columns give the associations scores relating to the words of the corresponding column. Note that the meaning with `ID=0` is also a dummy meaning.

The above is only printed for HA and HA2. If the simulation concerns a CA, its grammar is stored as outlined below.

7.7 Lexicon II file

The second lexicon file can only be specified from the command line using option `-L` (see Section 8) and prints out the global readable lexicon at the end of each iteration for the holistic agents (HA and HA2) or the individual grammars for the compositional agents. The grammars look like the fragment printed below. For each iteration and each agent, the number of `nSymbols`, `nMeanings` and `nRules` are given (note `nSymbols` is redundant). This information is followed by the actual grammar, which contains rules

as `15->jmqkur/[5,14,15,8]/1/0.01`. This rule rewrites the sentence with cover 15 (this relates to the conceptual space of RGBS) to the signal `jmqkur` with a meaning that can be constructed from categorical features with index 5, 14, 15 and 8. These indices relate to the categorical features stored in the ontology as printed below the grammar. In this example, we only see categorical feature 5, which is written as: `5: 0: [0.300]` and is of dimension 0 (Red-component) with value 0.300. The integer behind the second slash in the rule indicates the frequency with which this rule has been used, and the final value is the rule's rule weight.

Compositional rules are given in the form: `15->[8,7]/[12,13,10,3,...,28,30]/174/0.45604`, where the sentence rewrites to two components with covers 8 and 7 (shape S and colour RGB). The meanings that can be used in relation to this rule are listed after the first slash, which is followed by the rule's frequency and weight. The components with cover 8 and 7 rewrite to rules such as shown below this compositional rule.

```
Grammar, iteration 1:

A2: nSymbols=0 nMeanings=35 nRules2=804

grammar:
15->jmqkur/[5,14,15,8]/1/0.01
15->flrm/[9,2,11,16]/2/0.01
15->nnn/[13,14,7,12,1,18,15,11]/7/0.10899
15->ppyfhea/[13,10,15,12]/2/0.10899
.
.
15->[8,7]/[12,13,10,3,...,28,30]/174/0.45604
8->p/[12,20,4,27,16]/95/0.28133
7->tbt/[13,10,3]/1/0.01
7->pyfhea/[13,10,15]/1/0.01
.
.

ontology:
1: 0 : [0.579]
2: 1 : [0.116]
3: 2 : [0.637]
4: 3 : [0.111]
5: 0 : [0.300]
.
.
```

Currently, the lexicon II file can only be specified from the command-line using the '-L' option, see below.

7.8 Standard output

In addition to the writing to files and displays, THSim also outputs some information to the standard output (ie. the terminal from which we started the simulator). Each line of output is preceded by a word or abbreviation indicating the type of output. This is handy with respect to finding the information back and to 'grep' the information. In linux it is possible to redirect the standard output into a file, e.g., by starting THSim using `java THSim > myfile.data`. Later on, you can extract the information by using the commands `cat` and `grep`. For instance, if I redirect the output into the file `myfile.data` and I want to extract the results of the population test (preceded by TP), I give the following command:

```
cat myfile.data | grep TP
```

and I get the information I want.

There are different types of output, which are discussed here and organised by their prefixes:

param Outputs the parameters that you have set from the command line (see next section).

info Contains currently two types of information:

1. Always at the start of an iteration (or after the simulation was stopped and restarted) there appears a line giving information about the number of iterations (`nIter`) and languages (`nGames`) that have to be played, the current iteration number (`iter`) and language game number (`lg`), the population size (`popSize`), the number of adults (`nAdults`) and the number of learners (`nLearners`).
2. If `testPop=true`, the population is tested of which the start is marked by the line "info testing population".

TP Outputs the test population results. The following information is digested in the given order (in between brackets I indicate for which agents the output appears):

lg (HA,HA2,CA) the language game number.

pCoh (HA,HA2,CA) the production coherence.

iCoh (HA,HA2) the interpretation coherence.

iAcc (CA) the interpretation accuracy.

comp (CA) the compositionality measure

sim (CA) the proportion of the learners' grammars that coincide with the adults' grammars.

var (HA,HA2) the average variance between the meanings of speakers and hearers.

pCohAdults (HA,HA2,CA) the production coherence within the group of adults.

iAccAdults (HA,HA2,CA) the interpretation accuracy of the adult group with respect to the group of learners (i.e. the proportion of utterances from the learners group that has successfully been interpreted by the adults).

pCohLearners (HA,HA2,CA) the production coherence within the group of learners.

iAccLearners (HA,HA2,CA) the interpretation accuracy of the learner group with respect to the group of adults (i.e. the proportion of utterances from the adults group that has successfully been interpreted by the learners).

GR information regarding the agents' grammars. This is only output for CA and contains the following:

id the identity of the agent.

gSize the grammar size.

nComp the number of compositional rules

nHol the number of holistic rules

domRule the rule that has been used most frequently in the past period, written in short hand together with the frequency.

lx information regarding the lexicons. It prints the language game number followed by, for each agent, the lexicon size. (HA and HA2 only.)

on information regarding the ontologies. It prints the language game number followed by, for each agent, the ontology size. (HA and HA2 only.)

game This information is given when printGame is true and digests information regarding each game. For the HA and HA2, the output is the same as in the logfile (see above). For the CA, it outputs the game number, the speaker's rule, the hearer's rule, the game's success and whether one of the used rules was compositional.

Here is an example output for a holistic agent (HA) that is generated by the command line startup using `java THSim -T 0 -R true -s true -v false -k RGB`, see Section 8

```
param agentType=1
param printGame=true
param start=true
param uInterface=false
param features=RGB
info nIter=1 nRuns=1000 iter=0 lg=0 popSize=2 nAdults=0 nLearners=0
game 1 3 050 1 1 [1.0,0.784,0.0] becegi 2 -1 -1 -1 -1 [-1] * 0.75 true 0.0 false
lx 0 1 2
on 0 5 2
game 2 3 030 2 1 [0.0,1.0,0.0] hagaha 2 -1 -1 -1 -1 [-1] * 0.71 true 0.0 false
.
.
game 1000 3 080 1 1 [1.0,0.611,0.0] becegi 2 080 10 2 [1.0,0.887,0.0] becegi 0.41 true 0.66 true
info testing population
TP 1 0.575 0.55 0.008 0.0 0.545 1.0 0.555
```

The final fame of this simulation (`game 1000` was played by agent A3 as speaker about colour with code `080` (ie. yellow, cf. Table 3). The meaning used by the speaker had `ID` 1, at layer 1 (note this is an agent with hierarchical layering of conceptual spaces) and with meaning of prototype [1.0,0.611,0.0] and the expressed form was `becegi`. (Note that this form was also used in `game 1`, showing how the meaning has shifted from [1.0,0.784,0.0] to [1.0,0.611,0.0].) The hearer was agent A2, whose topic had code `080` too with meaning `ID` 10 at layer 2 with prototype [1.0,0.887,0.0]. It successfully interpreted the utterance `becegi`. The discriminative success at game 1000 is 0.41, this discrimination game was a success (according to the first `true`), the communicative success at game 1000 is 0.66 and so was this guessing game.

The test results given after the `TP` indicate that at the end of the 1st iteration (only one was played), the production coherence yielded 0.575, the interpretation coherence was 0.55, the average variance between the meaning used was calculated to be 0.008. Within the adult-group (only agent A2) production coherence (0.0) has no meaning as there is only one agent, the group of adults (ie. A2) has an interpretation coherence for understanding the group of learners (ie. A3) of 0.545. The production coherence within the learner group (1.0) has, again, no meaning. This group has an interpretation coherence with respect to the adult group of 0.555.

The next fragment, shows the output obtained by a similar run of compositional agents.

```
param printGame=true
param start=true
param uInterface=false
param features=RGBS
info nIter=1 nRuns=1000 iter=0 lg=0 popSize=2 nAdults=0 nLearners=0
game 1 0->nil/[]/0/0.01 0->nil/[]/0/0.01 false false
.
.
game 1000 15->nyr/[5,6,..,26]/79/0.037 15->nyr/[5,6,..,27]/94/0.364 true false
info testing population
TP 1 0.55 0.207 0.396 0.823 0.55 0.21 0.55 0.205
GR a2 283 13 54 15->[1,14]/354
GR a3 254 10 70 15->[7,8]/641
```

At `game 1` we see that the agents did not produce an expression. What you see is just a dummy rule. In `game 1000` the rule used by the speaker was `15->nyr/[5,6,..,26]/79/0.037`, where the dots were filled with a large number of other categorical features. The hearer used a very similar rule. The game was a success (occurrence of `true`), but it was not a compositional rule (given the `false`).

The test results after iteration 1 yielded a production coherence of 0.55, an interpretation accuracy of 0.207, a compositionality of 0.396, a similarity of 0.823, a production coherence between adults (does not mean a lot with 1 adult) or 0.55, an interpretation accurracy of the adult (A2) with respect to the learner (A3) was 0.21, the production coherence within learners (A3) is 0.55 and the interpretation accuracy of the learner with respect to the adult was 0.205.

The grammar size of agent A2 was 283 rules, of which 13 were compositional and 54 were holistic. The most frequently used rule was `15->[1,14]/354`, which was used 354 times.

7.9 Printed scores

If you set the parameter `printScore=true`, the simulator selects from the holistic agents (HA and HA2) that make up the population a list of 10 words that first become successful. These words are then traced for the remainder of the simulation (even over the subsequent generations). Everytime an agent uses this word, it outputs the positive scores associated with these words (i.e. the scores > 0). The output is stored in files in the working directory specified by `workDir`. De filenaam wordt dan opgebouwd uit de string "scoreA"`+id+word+`".txt", where `id` stands for the identity of the agent and `word` for the word. So, if word "hadeda" is traced for agent 2, the scores are stored in the file `scoreA2hadeda.txt`.

Below is a fragment of such a file. The first column of the file refers to the language game number and the subsequent columns indicate the `ID` of the meaning (followed by a colon) and its score. If you are unlucky, one of the meanings was merged with another meaning, in which case the `ID` might be the `ID` of the other meaning. The prototypes of the meanings are not stored here. If you are interested in these, you need to store the lexicon as well.

```
22 1: 0.107
78 1: 0.107 2: 0.108
81 1: 0.196 2: 0.108
85 1: 0.196 2: 0.198
89 1: 0.196 2: 0.178 22: 0.01
102 1: 0.176 2: 0.178 22: 0.01
121 1: 0.259 2: 0.178 6: 0.01 22: 0.01
```

Now that you've been given all kinds of output from THSim, you should be able to analyse the data further yourself. For doing this, you are kind of on your own, though in linux there are many handy tools, such as `cat`, `grep` and – especially – `awk`.

8 Command line processing

In order to speed up processing, it is possible to start THSim without a user interface. In addition, parameter settings can be given from the command-line, so the user does not have to set these in the control panel. Basically, one has to start THSim from the command-line by typing: `java THSim -option1 value1 -option2 value2 ...` When you type `java THSim -h`, a list of possible options with their default values and parameters they affect is displayed. The options are as given in Table 7. If you do not specify an option, THSim will use its default value. One remark is required: If you wish to process THSim without user interface, you need to set option `-v false` and you need to start the simulation using option `-s true` (or you might evaluate a test function in which case you use the option `-t true`).

References

[1] P. Vogt. The emergence of compositional structures in perceptually grounded language games. Submitted for publication.

[2] P. Vogt. THSim v3.2: The Talking Heads simulation tool. In W. Banzhaf, T. Christaller, P. Dittrich, J. T. Kim, and J. Ziegler, editors, *Advances in Artificial Life - Proceedings of the 7th European Conference on Artificial Life (ECAL)*. Springer Verlag Berlin, Heidelberg, 2003.

option	value	Parameter	Possible values	Dependencies
-a	2	popSize	$N > 1$	$N \leq maxAgents$
-A	1.0	pAdultSpeaker	$0 \leq X \leq 1$	nIter> 1
-b	true	adaptSG	true/false	SG only
-B	0	trainingSetSize	$0 \leq N \leq 120$	0 means whole set
-c	4	cxtSize	$1 < N \leq 8$	
-C	true	fCol	true/false	
-d	null	workDir	valid directory	HA & HA2 only
-D	true	testPop	true/false	
-e	0.9	etaN	$0 \leq X \leq 1$	
-E	0.9	etaS	$0 \leq X \leq 1$	SG only
-f	log.txt	logFile	filename	
-F	26	alphabetSize	$1 < N \leq 26$	CA only
-g	g	gameType	o: OG, g: GG, s: SG	s is HA & HA2 only
-G	1.0	pGG	$0 \leq X \leq 1$	
-H	0.0	pAdultHearer	$0 \leq X \leq 1$	nIter> 1
-i	1	nIter	$N > 1$	
-I	false	iHolistic	true/false	nIter> 1
-j	false	incrPop	true/false	nIter> 1
-J	2	popGrowth	$0 < N \leq (maxAgents - popSize)$	nIter> 1
-k	RGBAXY	features	strings containing RGBAXY	
-K	0	preLing	$1 < N < nGames$	CA only
-l	lex.txt	lexFile	filename	
-L	lexII.txt	lexIIFile	filename	
-m	4	foa	$1 < N < cxtGame$	HA & HA2 only
-M	1000	memMeanings	$N > 1$	HA & HA2 only
-n	0.0	pNoise	$0 \leq X$	
-N	2000	maxAgents	$N \geq popSize$	
-O	0.0	pOG	$0 \leq X \leq 1$	
-p	1.0	pWC	$0 \leq X \leq 1$	
-P	false	printScore	true/false	HA & HA2 only
-r	1000	nGames	$N > 1$	
-R	false	printGame	true/false	
-s	false	start	true/false	required if no UI
-S	true	S2S	true/false	nIter> 1
-t	false	test	true/false	
-T	2	agentType	0: HA, 1: HA2, 2: CA	
-u	c	uPType	c: centre-of-mass, s: simulated annealing, w: walking/running average, n: none	HA & HA2 only
-U	s	uScore	s: score-based, u: usage based	HA & HA2 only
-v	true	uInterface	true/false	
-V	false	varGames	true/false	
-W	1000	memWords	$N > 1$	HA & HA2 only
-h		this text		

Table 7: The options and values that can be given as arguments to THSim as `java THSim -option value [-option value] ...`. The fourth column indicates the possible values and the final columns any preconditions or other dependencies.

[3] P. Vogt. Minimum cost and the emergence of the Zipf-Mandelbrot law. In *Proceedings of ALife 9*. The MIT Press, 2004.