Bailey, T. M. & Pothos, E. M. (2008). AGL StimSelect: Software for automated selection of stimuli for artificial grammar learning. Behavior Research Methods, 40(1), pp. 164-176. doi: 10.3758/BRM.40.1.164

AGL StimSelect: Software for automated selection of stimuli for Artificial Grammar Learning

Todd M. Bailey

School of Psychology

Cardiff University

Emmanuel M. Pothos

Department of Psychology

Swansea University

Please address correspondence to Todd Bailey, School of Psychology, Cardiff University, Cardiff CF10 3AT, UK, or Emmanuel Pothos, Department of Psychology, Swansea University, Swansea SA2 8PP, UK. Electronic mail may be sent at baileytm1@cardiff.ac.uk or e.m.pothos@swansea.ac.uk

Running head: AGL StimSelect; **Word count**: 7,865 (including abstract)

Abstract

Artificial Grammar Learning (AGL) is an experimental paradigm that has been used extensively in cognitive research for many years to study implicit learning, associative learning, and generalization based either on similarity or rules. Without computer assistance it is virtually impossible to generate appropriate grammatical training stimuli along with grammatical or non-grammatical test stimuli that control relevant psychological variables. We present the first flexible, fully automated software for selecting AGL stimuli. The software allows users to specify a grammar of interest, and to manipulate characteristics of training and test sequences, and their relationship to each other. The user thus has direct control over stimulus features that may influence learning and generalization in AGL tasks. The software enables researchers to develop AGL designs that would not be feasible without automatic stimulus selection. It is implemented in Matlab.

Artificial Grammar Learning (AGL) experiments test people's sensitivity to sequential dependencies. In its most common form, an AGL study involves letter strings (e.g. XVSX) which do or do not conform to some simple set of rules (a finite state grammar). Strings that conform to the rules are grammatical (G strings), and strings that do not conform are ungrammatical (NG strings). Without knowing anything about rules, participants study a sample of strings that conform to the rules. After being told that the studied stimuli all conformed to an unspecified set of rules, participants are then asked to observe a set of novel strings and decide which are G and which are NG. The first AGL study was presented in 1967 (A. Reber, 1967) and since then there have been more than 125 studies, spanning research themes as diverse as implicit cognition (Berry & Dienes, 1993; Pothos, 2007; A. Reber, 1993; Shanks, 2005), associative learning (Boucher & Dienes, 2003; Perruchet et al., 2002; Servan-Schreiber & Anderson, 1990), rules vs. similarity (Ashby et al., 1988; Pothos, 2005), and cognitive neuropsychology (including amnesia, Alzheimer's disease, and Parkinson's disease; Knowlton & Squire, 1996; Poldrack et al., 2001; P. Reber & Squire, 1999; Witt, Nuehsman, & Deuschl, 2002). The AGL paradigm has also been used recently to study the psychopathology associated with dyslexia (Pothos & Kirk, 2004), and to examine cognitive processes supporting the maladaptive behavior of alcohol abuse (Pothos & Cox, 2002).

In principle, the structural properties of AGL stimuli can be controlled to a high degree of specificity to allow rigorous examination of different theories of learning. For example, Vokey and Brooks (1992; Brooks & Vokey, 1991) manipulated the similarity of test strings to training strings, across both G and NG test items. Knowlton and Squire (1996) factorially combined test string grammaticality with "chunk strength" to test effects of the relative familiarity of subsequences within the test strings (pairs or triplets of letters, called chunks or fragments). Others have combined grammaticality with both similarity and chunk

strength (e.g. Meulemans & van der Linden, 2003). Another possibility is to experimentally control some properties (particularly grammaticality), and then analyze the effects of other properties through mathematical modeling (Johnstone & Shanks, 1999; Pothos & Bailey, 2000).

The research above took great care to control important stimulus properties. It is far from trivial to construct AGL stimulus sets that simultaneously control multiple psychologically relevant factors. At present, AGL researchers employ trial-and-error methods to identify appropriate stimuli, with more or less assistance from whatever software tools they develop themselves on an ad hoc basis. These informal methods are perhaps sufficient to control two or three key stimulus properties across small stimulus sets with fewer than about 50 test items. However, these methods do not scale up well to more complex designs. Also, although AGL studies typically aim to draw general conclusions, they usually rely on a single set of stimuli on which all participants are tested. Although it would often be preferable to test each participant on a different set of stimuli (Dienes & Altmann, 2003; R. Reber & Perruchet, 2003; Redington & Chater, 1996), it is not usually feasible to identify more than one suitable set of stimuli for a given study using the present trial-and-error methods. Thus, the lack of an automated procedure to generate AGL stimuli is a major limitation in AGL research.

In this paper we present AGL StimSelect, a software package that automatically generates training and test strings that embody structural properties that can be selected by the user from a flexible and extendable set of parameterized constraints. With StimSelect, a user can quickly generate AGL stimuli, while controlling multiple variables that are likely to influence performance on an AGL task.

Program Design

The StimSelect software is written in the Matlab programming language, which is available for Windows, Macintosh, and Unix computing environments. Matlab is particularly suitable for mathematical and computational modeling in a variety of disciplines, including psychology and cognitive science. The base Matlab system is required, as well as the Statistics toolbox. Matlab and all its toolboxes are frequently available in many academic departments and, where they are not, individual licenses can be purchased from the MathWorks (www.mathworks.com).

StimSelect consists of a set of parameterized functions that may be called manually from Matlab's command window, or from within scripts or other functions written by the user. The software is designed to be easy to use, and requires only an elementary familiarity with Matlab programming. A great deal of flexibility is built into StimSelect in the form of optional arguments for the various functions to specify constraints on a wide range of stimulus properties. More advanced users can extend the range of stimulus properties controlled by StimSelect, by writing additional functions that interface with the lower level StimSelect functions and data structures. However, the present paper focuses on the basic aspects of StimSelect that will be of interest to most users.

Finite state grammars.

A user-specified finite state grammar is central to the operation of StimSelect. In AGL studies the training strings conform to some set of rules, and these rules are most often specified as finite state grammars. An example grammar, from Knowlton and Squire (1996, Exp. 1), is shown in Figure 1. In general, a finite state grammar identifies beginning and end states (indicated as IN and OUT in Figure 1), and defines continuation relations among elementary symbols, allowing certain sequences of symbols to be constructed (Chomsky & Miller, 1958). Letter sequences are either G or NG, according to whether or not they can be constructed by the finite state grammar of interest.

----------FIGURE 1 ABOUT HERE----------

StimSelect allows the user to specify a finite state grammar of their choice, or to simply select one of two widely used grammars that are built in (Knowlton and Squire, 1996, Exp. 1; A. Reber & Allen, 1978). StimSelect requires a deterministic grammar, so that no state of the grammar can have two outward transitions labeled with the same letter. Because any non-deterministic finite state grammar can readily be converted into an equivalent deterministic one, this is a restriction on form but not substance.

The grammar specifies which letters are relevant to a particular study, and determines which sequences of those letters are G and which are NG. If the number of either G or NG strings is large, StimSelect will operate on a random subset of those strings (by default, the limit is 10,000 G and NG strings combined). Training strings are always chosen from among the G items. Test strings may be either G or NG, unless restricted to one or the other by user-specified constraints. The sets of training and test strings are strictly non-overlapping, so any given string can be chosen as a training string, a test string, or neither, but not both.

Properties of test strings that can be controlled in StimSelect include grammaticality, similarity, chunk strength, chunk novelty, and rule strength. The following sections introduce these properties in relation to the AGL literature, and then outline the selection strategy used by StimSelect to identify training and test strings that have the combinations of properties desired by the user.

Grammaticality.

Most AGL studies manipulate the grammaticality of test strings relative to the rules of the finite state grammar employed, so that some test strings are G and some are NG. This makes it possible to assess how well participants discriminate between test strings that do or do not

follow the rules exemplified by the training strings. Originally, the ability to distinguish G from NG strings, even to a limited degree, was argued to indicate knowledge of the underlying finite state grammar (e.g., A. Reber, 1976). However, later investigators have rejected this interpretation, including the original proponent himself (e.g., Dulany et al., 1984; Pothos & Bailey, 2000; A. Reber, 1993). There is currently some controversy about the appropriate psychological interpretation of participants' partial ability to discriminate between G and NG strings (Pothos, 2007). Nevertheless, the grammaticality of test strings is easy to establish and provides a highly intuitive relation between test strings and training strings. Perhaps for these reasons, as well as its historical importance in the origins of AGL, grammaticality continues to feature prominently in AGL studies.

Similarity.

The similarity of test strings to training strings is one property of AGL stimuli that might be psychologically relevant (Vokey & Brooks, 1992). According to exemplar theories of categorization (e.g., Nosofsky, 1988), classification of a new instance as a member of this or that category depends on the overall perceived similarity between the new instance and familiar members of each category. In AGL, Vokey and Brooks (1992) used edit distance to determine relative similarities between any two test and training strings. In its simplest form, the edit distance between two letter strings is the number of letter substitutions, insertions and deletions required to convert one string into the other. In addition to its use in some AGL studies, edit distance has also been used widely in psycholinguistics (e.g., Luce & Pisoni, 1998), where there is some evidence that people are more sensitive to substitutions than to insertions or deletions of phonemes within words (Bailey & Hahn, 2001; Hahn & Bailey, 2005).

The default similarity function in StimSelect is based on edit distance, normalized by the maximum possible distance for strings of the same length. This normalized distance

ranges from 0 to 1, and is subtracted from 1 to arrive at a measure of similarity between 0 (maximally dissimilar, given the lengths of each string) and 1 (identical). By default, StimSelect assigns somewhat greater importance to substitutions than to insertions and deletions in its calculation of edit distance, so that a cost of 1 is assigned to each substitution and a cost of 0.7 to each insertion or deletion (cf. Bailey & Hahn, 2001). Different costs for these operations can be specified by the user. StimSelect also allows the user to specify an alternative similarity function altogether.

Chunk strength.

After participants study the training strings in an AGL task, they may be more or less familiar with subsequences of those strings, depending on how many times each subsequence occurred within the training set. Then, when participants classify test strings, their responses may be influenced by the perceived familiarity of the various subsequences within each test string (Servan-Schreiber & Anderson, 1990; Perruchet & Pacteau, 1990; Knowlton & Squire, 1994). The subsequences usually considered in AGL research are pairs or triplets of letters, called fragments or chunks. This theory of AGL performance derives from general principles of associative learning theory, according to which the cognitive system learns by gradually combining elementary units that co-occur frequently into a single representational unit (e.g. Wasserman and Miller, 1997).

In StimSelect, chunk familiarity is operationalized by defining the strength of a chunk as the ratio $\frac{F}{F+E}$, where F is the frequency of the chunk across all training strings, and E is the expected frequency for a chunk of that size (basically, the average frequency across all chunks of that size in training). Chunk strength can range from 0 to 1; a chunk that occurs with average frequency will have a chunk strength of 0.5, and if F>>E then $\frac{F}{F+E} \rightarrow 1$. The

chunk strength of a test string is the average strength of all its chunks from bigrams up to any user-specified chunk size.

The definition of chunk strength in StimSelect is obviously related to, but slightly different from, the measure of global associative chunk strength used by Knowlton and Squire (1996). They defined chunk strength in terms of the absolute frequency of a chunk, that is, the number of times that chunk appeared in the training strings. In general, Knowlton and Squire's measure of chunk strength is sensitive to the number of training strings, and also the number of times the same training strings are shown to the participant. Because StimSelect's measure of chunk strength takes into account the expected frequency as well as the raw frequency of a chunk, it does not depend on the number of times the same training strings are shown, and is generally independent of the number of training strings. This allows users to change the number of training strings independently of the target chunk strength values. Also, for a given number of training strings, Knowlton and Squire's measure of chunk strength varies across different grammars depending on the number of different symbols employed. StimSelect's measure of chunk strength does not. This should facilitate comparisons across different grammars. Finally, it would be difficult, if not impossible, to do incremental selection of training and test items using Knowlton and Squire's measure of chunk strength, since the final chunk frequencies would depend on both the number and length of training items yet to be chosen. For these reasons, StimSelect adopts a new measure of chunk strength rather than that of Knowlton and Squire. Nevertheless, the two measures would typically correlate very highly with each other.

Chunk novelty.

There is some evidence that participants in AGL studies are sensitive to the presence of novel chunks within test strings, that is, chunks that did not appear in any of the training strings (e.g., Johnstone & Shanks, 1999; Meulemans & van der Linden, 1997, 2003). Sensitivity to

chunk novelty may be independent of sensitivity to relative familiarity (which is measured by chunk strength). In StimSelect, a test string's level of chunk novelty is determined by counting how many novel bigrams (or chunks of some user-specified size) are contained within the string. In effect, chunk novelty is a measure of how surprising a given test item is, in light of the training strings.

Rule strength.

Sensitivity to novel subsequences in an AGL study relies on knowledge about which chunks did or did not appear in the training strings. Pothos (2007) has recently argued that such frequency-independent knowledge amounts to knowledge of certain kinds of rules. This proposal derives from earlier discussions of what might constitute knowledge of rules in AGL (e.g., Dulany et al., 1984), as well as considerations of when it is meaningful to distinguish between rules and similarity (Pothos, 2005; Sloman & Rips, 1998; Smith, Langston, & Nisbett, 1992; Smith, Patalano, & Jonides, 1998).

In accordance with the above, StimSelect defines the rule strength of a test item as the proportion of chunks it contains that have been observed in training. Rule strength is computed across all chunks from bigrams up to a user-specified maximum chunk size. In some ways, rule strength is a conceptual inverse of chunk novelty, but is based on the proportion of chunks within a test item that are or are not novel rather than the absolute number of novel chunks.

Selection of training and test strings.

Initially, the training and test sets are empty. A few strings are added to the training set, choosing them carefully to control the overall frequency with which each letter, bigram, and trigram occurs in the training set. Only G strings are eligible for inclusion in the training set, as is the case with the vast majority of AGL studies. After making a start on the training set, stimulus selection alternately adds strings to the various test sets and further strings to the

training set. Strings are chosen for various test sets in order to achieve different combinations of various attributes, like G versus NG, high versus low average similarity to the training strings, high versus low chunk strength, and so on. Further training strings are chosen according to the effect they have on the various test sets, so that the final combination of training and test sets has the desired combination of attributes. The program aims to fully counterbalance the attributes of interest (however, if attributes are highly correlated within the strings generated by a particular grammar, there will be a limited range of values over which it is possible to vary them independently). Stimulus selection stops when the training and test sets contain the desired number of strings. At that point, StimSelect displays a list of the selected training and test strings. Test strings are reported with categorical information for each attribute (e.g. high vs. low similarity), along with the actual numerical measure of each attribute.

Program Use

As mentioned above, StimSelect consists of a set of functions, distributed as Matlab source code. We describe here the highest-level functions, which support the stimulus constraints most commonly of interest to AGL researchers. If necessary, users with advanced Matlab programming skills can readily extend the set of constraints by analogy with those described here. In that event, a fuller understanding of the program should be pursued through the examples distributed with the documentation, as well as the comments provided within the various Matlab files.

A list of the high level StimSelect functions is given in Table 1, with an example function call for each. The text below describes how to use these functions to identify stimuli

for an AGL study, and covers the most common variations available through the function arguments. A complete list of arguments for each function is given in the Appendix.

----------TABLE 1 ABOUT HERE----------

The basic idea of StimSelect is to identify a large number of potential stimulus strings, and then identify appropriate strings to include in a training set as well as in various test sets. There are six basic steps in using StimSelect. First, the user selects one of the pre-defined grammars, or specifies the finite state grammar of interest. Second, the grammar is embedded within an AGLSS data object to identify potential stimulus strings as G or NG. Third, the user specifies the factors required to control various properties of the desired test strings, including the number of levels of each factor. Fourth, the user specifies how the different levels of the selected factors combine to define different sets of test strings. Fifth, a constraint satisfaction function is run to choose the desired number of training and test strings. Finally, the training and test strings are displayed, together with statistics that indicate how well the selected strings achieve the desired properties. These six steps are described and illustrated below.

1. Selecting the finite state grammar

We have provided functions specifying two frequently used grammars (Knowlton & Squire, 1996, Exp. 1., and A. Reber & Allen, 1978). Alternatively, users wishing to specify their own finite state grammar can easily do so. StimSelect includes a ‘grammars’ folder that contains several functions defining finite state grammars. The grammar first used by Knowlton and Squire (1996, Exp. 1) is defined by the function `knowlton_squire_grammar`. The grammar of A. Reber and Allen (1978) is defined by the

function `reber_grammar`. There are also some trivial ‘toy’ grammars, as examples of how finite state grammars are defined.

In StimSelect, information defining a finite state grammar is stored in an FSG data object. Two functions are provided to allow the user to easily define a finite state grammar from scratch: `FSG` and `LINK`. The `FSG` function creates a new FSG data object corresponding to a degenerate finite state grammar with a specified number of states but no transitions between them. It also specifies the set of letters to be used by the grammar. Transitions between states are added using the `LINK` function. For example, the grammar shown in Figure 1 uses the letters ‘J’, ‘T’, ‘V’, and ‘X’, and requires five states. To create an appropriate FSG data object, execute the expression:

```
g = fsg([], 'JTVX', 5);
```

The first argument to `FSG` (given as `[]` above) potentially allows the user to provide a matrix defining transitions between states, but it is usually simpler to specify transitions afterwards using the `LINK` function. The grammar in Figure 1 has two transitions out of state 1, one of which returns to state 1 and one of which goes to state 2. To add these transitions to the FSG data object created above:

```
g = link(g, 1, 'XV', [1 2]);
```

The arguments to `LINK` respectively specify the FSG data object to be modified, the state from which the specified links originate, the letters labeling the specified links, and the states to which the links lead, given in the same order as the letters. The beginning state of a grammar is always state 1. The end state is -1. As shown in Figure 1, transitions to the end state are not labeled with letters of the grammar, but instead the symbol ‘.’ is used to refer to

these transitions in calls to `LINK`. The remaining links of Figure 1 can be added using the following expressions:

```
g = link(g, 2, 'JXT', [3 4 5]);
g = link(g, 3, 'T.', [1 -1]);
g = link(g, 4, 'J.', [4 -1]);
g = link(g, 5, 'V.', [4 -1]);
```

For convenience, the user would usually create a Matlab function containing the expressions that define the grammar of interest.

2. Creating an AGLSS data object

The many working variables of StimSelect are organized within a specialized data structure that has been defined specifically for this purpose. We refer to this data structure here as an AGLSS data object, and most of the high level StimSelect functions operate on this object. The function `AGLSS` creates a new AGLSS data object, and embeds the user's finite state grammar within it (in Matlab terminology, the AGLSS function is a class constructor method). In addition to the user's grammar, the AGLSS data object also contains a large set of potential stimulus strings, based on the letters used in the specified grammar. Each potential stimulus string is classified as G or NG by the grammar. Eventually, training and test strings will be chosen from the potential stimulus strings. An argument to the AGLSS function specifies the range of string lengths to be considered. By way of example, suppose an AGL study is going to employ the grammar of A. Reber and Allen (1978), henceforth simply the Reber grammar. To embed this pre-defined grammar in an AGLSS data object and consider strings of lengths 3 to 7, execute the Matlab expression:

```
s = aglss(reber_grammar, [3 7]);
```

This expression creates a new AGLSS data object, assigns it to the variable ‘s’, and produces the output below:

```
Potential items:
     Grammar involves 5 symbols (MRSVX)
     97625 possible strings of length 3-7
     68 grammatical strings ( 0.07%)
     97557 ungrammatical strings (99.93%)
     Using all 68 grammatical strings
     Using sample of 9932 ungrammatical strings
```

The user must give some thought to the relevant range of string lengths. In general, the range of string lengths must be determined primarily by the psychological hypotheses of interest to the researcher, but the ability of StimSelect to efficiently identify appropriate stimuli depends both on the particular grammar specified and on the range of string lengths of interest. For example, with the Reber grammar, if the minimum length of strings is 3 and the maximum is 7, then there are just 68 possible G strings and 97,557 NG ones. The small number of G strings here will make it difficult to find a stimulus set for an AGL study that simultaneously controls more than two or three key properties of the test items. By contrast, if the maximum length of strings is increased to 11 then there are 388 possible G strings (and 61,034,737 NG ones). The greater number of possible G strings will make it easier to control more properties of the stimuli, but at the same time there is probably a limit to the extent to which participants in AGL studies are sensitive to sequential dependencies in very long strings.

By default, out of all the possible G and NG strings, the AGLSS function ordinarily selects at most 10,000 of these for subsequent processing as potential stimulus strings. The

user can specify a different limit with an argument to the AGLSS function. To consider 30,000 potential stimulus strings the expression below could be used:

```
s = aglss(reber_grammar, [3 7], 30000);
```

In terms of obtaining the best possible stimuli, the more potential strings, the better. However, with very large numbers Matlab may run out of memory, or program completion times may be prohibitively long. In our simulations, searching in a set of 20,000 strings was reasonably fast and typically led to the identification of highly acceptable training and test sets.

If the number of possible G strings is less than half the specified limit, then all of them will be potential stimulus strings. Otherwise, a random sample of them will be selected. Similarly, a random sample of NG strings will be selected if necessary. The selection process sometimes results in the number of potential strings being slightly less than the specified limit, but never more.

The AGLSS data object variable (‘s’ in the examples above) will typically be successively modified by subsequent calls to various StimSelect functions. Generally, before starting another search for a new stimulus set users should be careful either to clear the variables in Matlab memory (using the ‘Clear all’ command; see also later), or use different variables for AGLSS objects corresponding to different searches.

3. Factors

The highly modular structure of StimSelect means that the user specifies as many factors as required, in a straightforward ‘pick and mix’ manner. We describe here the basic options for the available factors, only briefly mentioning additional advanced options where available

(also see the Appendix). We anticipate that many users will not require these advanced options, which are described in detail in the program documentation.

3.1 gram_factor

Grammaticality is the simplest factor, because it just distinguishes between the G and NG strings. This distinction is made when the potential stimulus strings are originally selected by the AGLSS function, so the grammaticality factor contributes very little to the time required to run the final stimulus selection constraint satisfaction process. To specify the grammaticality factor and assign it the name gram, execute the expression:

```
[s, levels] = gram_factor(s, 'gram');
```

This expression modifies the AGLSS data object ‘s’, and also creates the variable ‘levels’ which contains the names of the two levels of grammaticality, ‘G’ and ‘NG’. These level names will be used later when combinations of factors are defined, and they will also appear eventually in the lists of stimuli chosen. Here and elsewhere, the ‘levels’ variable is an optional value returned by the function, and may be omitted if not needed, as in some examples discussed below in the section on combining factors. Thus, the expression

```
s = gram_factor(s, 'gram');
```

has exactly the same effect as the previous one, but does not create the ‘levels’ variable.

An advanced option for the grammaticality factor allows the user to specify other names for the grammaticality levels (e.g., ‘Good’ and ‘Bad’ instead of ‘G’ and ‘NG’; here and elsewhere, for any advanced options not fully covered in text, please refer to the Appendix). A similar option is available for all the factors, but will not be explicitly mentioned for each.

3.2 sim_factor

The similarity factor controls the average similarity of test strings to training strings. To specify the default similarity factor and assign it the name Sim:

```
[s, levels, targets] = sim_factor(s, 'Sim');
```

In addition to modifying the AGLSS data object ‘s’, this expression creates the variables ‘levels’ and ‘targets’. The ‘levels’ variable contains names for the different levels of similarity. Here, those will be ‘LowSim’ and ‘HighSim’, but in general the number of factor levels and their names depend on the arguments specified for the `sim_factor` function. The ‘targets’ variable contains the actual values of similarity desired for test strings assigned to the various levels of the similarity factor.

By default, two levels of similarity are specified based on the distribution of pairwise similarities of the potential G items to each other. The 25th and 75th percentile values of these similarities are the respective default target similarities for low similarity and high similarity test strings. Different percentile values can be specified by an additional argument. The expression

```
[s, levels, targets] = sim_factor(s, 'Sim', [20 80]);
```

assigns low similarity and high similarity values based on the 20th and 80th percentile values of similarities between potential G strings. The user can specify more than two percentile values to obtain any number of different levels of similarity. Alternatively, the user can specify absolute similarity values rather than percentile values.

If levels of similarity are specified in terms of percentiles, the user can be reasonably certain that the specified levels will be feasible, regardless of the grammar that is being used. However, the same percentile for two different grammars might represent very different levels of absolute similarity. If results are to be compared across different grammars, it may be more appropriate to use absolute levels of similarity than percentiles. Note that a given

level of absolute similarity may be feasible with some grammars and impossible to achieve with others.

The 100th percentile represents the similarity between the two most similar G items. Usually this would not be a useful or attainable level of similarity to aim for among a set of test strings. Indeed, the reported average high versus low values of similarity in the literature are generally only marginally different from each other (e.g., Knowlton & Squire, 1994, 1996; Johnstone & Shanks, 1999; Pothos & Bailey, 2000; Vokey & Brooks, 1992).

Other advanced options allow the user to specify the relative cost of insertions, deletions and substitutions in computing edit distances between strings, or to specify a different similarity function altogether.

3.3 chstr_factor

The chunk strength factor controls degrees of familiarity of the subsequences in test strings. To specify the default chunk strength factor and assign it the name ChStr:

```
[s, levels, targets] = chstr_factor(s, 'ChStr');
```

The return values on the left hand side of this expression, and others below, are just like those of sim_factor, discussed above.

By default, two levels of chunk strength are specified, based on the frequency of bigrams and trigrams in training strings (see the discussion of chunk strength in the program logic section above). High chunk strength test strings will contain chunks occurring about twice as often in training strings as the chunks within low chunk strength strings. This is a somewhat more ambitious target than that typically employed in the literature. For example, in Knowlton and Squire's (1996) study the chunk strength manipulation varied the average frequency of chunks only by a ratio of 1.54 to 1. The average global associative chunk strength of the 'low chunk strength' items in their study was 5.6. Whereas the default

criterion employed in StimSelect would produce high chunk strength items whose average global associative chunk strength was twice as high (2 x 5.6 = 11.2), the average in Knowlton and Squire's stimuli was only 8.6.

Different relative frequencies can be specified by an additional argument. The expression

```
[s, levels, targets] = chstr_factor(s, 'MegaChStr', [1 3]);
```

assigns more extreme low and high chunk strength levels, where high chunk strength test strings contain chunks occurring, on average, three times as often in training as the chunks in low chunk strength strings. The user can specify any number of relative frequency values, to obtain as many levels of chunk strength as are required. Alternatively, the user can directly specify the absolute target chunk strength for the chunks in the strings in each category.

An advanced option allows the user to control the variability in frequency allowed among chunks within a single test string. Ideally, every chunk within a test string would have the desired overall level of chunk strength for that string. However, it is not feasible to achieve this level of zero variability if chunk strength is to be combined with other factors. By default, frequencies are controlled so that strings can achieve their target chunk strength by containing a mixture of 80% of chunks of the correct frequency and 20% of chunks at the opposite end of the frequency spectrum. For example, a low chunk strength string of length six could contain four low frequency bigrams and one high frequency bigram. The default values of 80% and 20% can be altered, if necessary, to obtain the best feasible stimulus set.

An additional advanced option allows the user some control over the size of chunks used to compute chunk strength. To date, all AGL research has used chunks of length two and/or three (bigrams and trigrams). There are some theoretical justifications for doing so (e.g., Cabrera, 1995; Dirlam, 1972; Servan-Schreiber, 1991), but the main reason for restricting investigations to bigrams and trigrams has been ease of computation. The chunk

strength factor allows consideration of chunk strength when the maximum chunk size is arbitrarily large. For example, if a maximum chunk length of four is chosen, then chunk strength computations will involve bigrams, trigrams, and 4-grams.

3.4 chnov_factor

The chunk novelty factor controls the number of novel chunks within test strings. Eligible potential stimulus strings are identified ahead of time, so chunk novelty is computed very efficiently and adds relatively little to the amount of time required by the constraint satisfaction process used to select training and test strings. To specify the default chunk novelty factor and assign it the name ChNov:

```
[s, levels, targets] = chnov_factor(s, 'ChNov');
```

By default, two levels of chunk novelty are specified, based on the number of bigrams within a test string that appear in no training strings. Low novelty test strings will contain only bigrams that also appear somewhere in the training strings. Each high novelty test string will contain exactly one bigram that does not appear in any of the training strings. Note that `chnov_factor` operates on chunks of just one size, unlike `chstr_factor` which by default takes into account both bigrams and trigrams.

Any number of different novelty levels can be specified by an additional argument. There is also an argument that controls the size of chunks on which chunk novelty operates, allowing the user to control trigram novelty rather than bigram novelty, for example. A further optional argument allows the user to control how many different chunks are to be avoided in training strings to act as novel chunks in test strings. By default, two chunks that occur in the fewest possible G strings are reserved in this way. Potential stimulus strings containing these particular chunks will not be selected as training strings. Usually, any test

string that contains a novel chunk will either contain one of these reserved chunks or an ungrammatical chunk that does not appear in any G strings.

3.5 rulestr_factor

The rule strength factor controls the proportions of chunks in a test string that are familiar from training. To specify the default rule strength factor and assign it the name RuleStr:

```
[s, levels, targets] = rulestr_factor(s, 'RuleStr');
```

By default, two levels of rule strength are specified, based on proportions of familiar bigrams and trigrams. High rule strength test strings will contain 100% familiar bigrams and trigrams. Low rule strength test strings will contain 75% familiar bigrams and trigrams (that is, 75% of the chunks in each string will be familiar, and 25% will be novel). Different proportions can be specified by an additional argument. The expression

```
[s, levels, targets] = rulestr_factor(s, 'RuleStr', [.5 .8]);
```

assigns low and high rule strength levels based on 50% and 80% familiar chunks. The user can specify more than two proportions to obtain any number of different levels of chunk strength. Note, however, that if chunk strength were to be used in combination with rule strength, it would be difficult to find strings that have both high chunk strength and very low rule strength.

An advanced option controls the maximum size of chunks on which rule strength should operate. For example, if a maximum chunk length of four is chosen, then rule strength will be computed over bigrams, trigrams, and 4-grams.

A further argument allows the user to control how many different chunks are to be avoided in training strings to act as novel chunks in test strings. This option is just like the corresponding one discussed above for chunk novelty.

4. Combining factors

After specifying the individual factors of interest, the user must specify which levels of which factors are to be combined with each other to create sets of test strings with particular combinations of properties. The user is cautioned that, except where noted, the examples below are not intended to be run one after the other. In general, each factor of interest should be specified just once. To change the specification of a factor, the user should start again from the first step, with a call to the AGLSS function, and proceed from there. The use of the Matlab command 'Clear all' (which eliminates all variables in the Matlab workspace) is advised before starting the process.

Suppose that grammaticality and chunk strength factors are specified as:

```
[s, glevs] = gram_factor(s, 'gram');
[s, chlevs] = chstr_factor(s, 'ChStr');
```

Then, to factorially combine grammaticality and chunk strength:

```
s = factorial_testsets(s, {'gram', glevs{:}},
   {'ChStr', chlevs{:}});
```

The arguments to `factorial_testsets` specify an AGLSS data object plus one or more bracketed lists (cell arrays, in Matlab terminology), each of which names a factor and whichever of its levels are to be combined factorially with specified levels of other factors. The order of factors in the arguments to `factorial_testsets` is irrelevant. In the expression above, `glevs{:}` specifies all levels of the grammaticality factor ('G' and 'NG'). Similarly, `chlevs{:}` specifies all levels of chunk strength ('LowChStr' and 'HighChStr'). Alternatively, the user could spell out the factor levels explicitly:

```
s = factorial_testsets(s, {'gram', 'G', 'NG'},
   {'ChStr', 'LowChStr', 'HighChStr'});
```

To factorially combine similarity as well as grammaticality and chunk strength, the expressions below could be used:

```
[s, glevs] = gram_factor(s, 'gram');
[s, chlevs] = chstr_factor(s, 'ChStr');
[s, slevs] = sim_factor(s, 'Sim');
s = factorial_testsets(s, {'gram', glevs{:}},
   {'ChStr', chlevs{:}}, {'Sim', slevs{:}});
```

As a degenerate case, to control a single factor on its own, `factorial_testsets` can be passed a single bracketed list naming the factor and levels of interest. For example, to manipulate the grammaticality of test items:

```
s = gram_factor(s, 'gram');
s = factorial_testsets(s, {'gram', 'G', 'NG'});
```

An advanced option also allows non-factorial combinations of factors to be specified. This would allow a user to, e.g., specify just three of the four possible combinations of two binary variables, for example, omitting contradictory or infeasible combinations like low rule strength – high chunk strength. In general, the feasibility of simultaneously controlling multiple factors will depend on the length and number of potential strings to be considered, the particular options specified for the factors of interest, and the finite state grammar.

5. Choosing the training and test strings

Once the desired combinations of factors are specified, the AGLSS data object is ready for stimulus selection. The choose_items function runs a constraint satisfaction process to identify the desired number of training and test strings. For example, to identify 10 training strings plus five strings for each set of test strings:

```
s = choose_items(s, 10, 5);
```

The number of test sets corresponds to the number of different combinations of factors and factor levels. Thus, if only grammaticality is specified via the `factorial_testsets` function, then `choose_items` will find five test strings for the 'G' category and five for the 'NG' one. If binary categories of grammaticality, chunk strength, and similarity are combined factorially, then there will be eight test sets of five strings each, for a total of 40 test strings.

The choose_items function dynamically displays a summary of its progress to give the user an indication of how long the search for stimulus strings will take. On a Pentium Duo IBM compatible computer, creating a stimulus set with 10 training strings and five strings in each test set, balancing grammaticality and chunk strength, requires just a few seconds (using the Reber grammar, with the default option of searching in a set of 10,000 potential items). By default, `choose_items` identifies an initial set of three training strings, based on how well they represent the target distribution of chunk frequencies. Thereafter, each selection round identifies an additional training string, then one test string for each set, until the desired number of strings are selected. An advanced option allows the user to specify how many training strings should be chosen initially, before selection of test strings begins.

6. Displaying the training and test strings

After stimulus strings have been selected, the function `format_train_items` lists the training strings in a formatted text string. This can be displayed in Matlab's output window, and copied and pasted into other applications as desired. Table 2 shows 10 training strings that were selected based on the Reber grammar, and displayed using the expression below (with no trailing semicolon):

```
format_train_items(s)
```

----------TABLE 2 ABOUT HERE----------

The first column of output from `format_train_items` enumerates the strings, and the second column lists the strings themselves.

The function `format_test_items` lists test strings in a formatted text string, as illustrated in Table 3. These test strings were selected to factorially control grammaticality and chunk strength, and were displayed using the expression:

```
format_test_items(s)
```

----------TABLE 3, 4 ABOUT HERE----------

Table 3 shows four test sets (two levels of grammaticality by two levels of chunk strength), with five strings in each set. The first column, Tset_num, enumerates the different test sets. The next two columns (gram_cat and ChStr_cat) list the names of the factor levels that define each set of test strings. The Itm_num column enumerates the training strings within each set, and the Itm_name column lists the test strings themselves. The last two columns list the actual chunk strength for each string, and then the target chunk strength corresponding to whichever test set the string is in.

Consider the first test string, VXRRM, which is in the set of G and LowChStr strings. Its chunk strength is 0.367, which seems reasonably close to the target value for LowChStr strings, 0.400. There is no such information for the grammaticality factor, because the grammaticality of each potential stimulus string is defined ahead of time, and the selection of

strings appropriate to each level of grammaticality does not require dynamic checking of string properties against target factor levels.

For a quick, informal assessment of the extent to which the test strings are suitable or not, summary information for each test set can be obtained by giving `'summary'` as an additional argument to the function `format_test_items`:

```
format_test_items(s, 'summary')
```

This expression lists average factor values for each test set, as illustrated in Table 4. The columns are the same as in Table 3, except that item numbers and item strings themselves are omitted. In evaluating the extent to which selected strings have the desired properties, the user may wish to check whether the various properties of individual strings are closer to their target values than to the target values of other levels of the same factor. Also, various authors have used statistical tests to evaluate the collective appropriateness of AGL test sets (e.g., Brooks & Vokey, 1991; Knowlton & Squire, 1996; Vokey & Brooks, 1992). It is straightforward to carry out such tests from the output produced by StimSelect.

Note that if the finite state grammar defines only a small number of G items, it may be difficult or impossible to find suitable test strings that combine several other factors with grammaticality. In that case, the properties of some test strings may be far away from the specified factor levels. If this does occur, the user may wish to increase the maximum length of the items in the first instance, and possibly experiment with some of the advanced options.

Program installation

The software is distributed as a .zip file. Users running Mac OS-X can unpack this file simply by opening it with the Finder. Users running Windows XP can unpack this file by selecting it in Windows Explorer and then choosing File > Extract All from the command menu. Users

running Unix can unpack the .zip file using the unzip command. It does not matter where the contents of the .zip file are extracted, but a Matlab path must be specified to that location so that Matlab knows where to look for the StimSelect functions. The Matlab path can be modified using the File > Set Path menu option in the Matlab command window. The 'Set Path with Subfolder' button will allow the user to browse to the directory containing the unpacked StimSelect files.

Summary

StimSelect allows the user to create AGL stimulus sets based on a particular finite state grammar, so that the test strings are balanced across any combination of the following factors: Grammaticality, similarity, chunk strength, chunk novelty, and rule strength. These factors represent by far the most common stimulus properties in AGL studies (Pothos, 2007). A number of advanced options allow the user to alter the specification of all these factors in theoretically interesting ways (including, in particular, the use of larger chunk sizes for chunk-based factors). Additional factors can be defined by analogy with the specification of the existing ones.

To achieve flexible and efficient stimulus selection, the mathematical specifications of some factors have been slightly refined relative to traditional definitions in the AGL literature. We have altered the definition of chunk strength to obtain a more robust measure in which chunk strength is scaled relative to expected chunk frequencies, so that chunk strength values do not depend on training item repetitions or the number of training items. Also, we have generalized the definition of similarity to allow differential weighting of insertions, deletions and substitutions (such flexibility will be desirable for some research projects). Finally, we have implemented for the first time a measure of rule strength,

consistent with a recent proposal of what rules are (Pothos, 2005). Associating rules knowledge with grammaticality has been highly problematic (e.g., A. Reber, 1993), therefore providing an alternative conception of what would constitute rules knowledge in AGL would enable new research possibilities.

The appeal of AGL has been recognized in that AGL designs allow the concurrent examination of several hypotheses about learning. Researchers in the past have relied on informal methods to balance the postulated influence of different stimulus attributes. With such methods it is difficult, if not impossible, to balance arbitrarily many factors for stimulus sets of arbitrary size. StimSelect allows AGL designs that are much more sophisticated than was possible previously, including factorial manipulations of four or five stimulus properties simultaneously. By greatly expanding the range of feasible AGL designs, StimSelect has the potential to significantly aid further progress in AGL and learning research more generally.

Appendix: Parameters for high level StimSelect functions.

```
fsg([], 'L', N)
          |   |
          |   Number of states in grammar
          |
         Letters used in grammar
```

```
link(F, S0, 'L', [S…])
     |   |    |     |
     |   |    |     States to which transitions go
     |   |    |
     |   |    Letters labeling links from S0 to S
     |   |
     |   State from which links originate
     |
     FSG data object; required
```

```
gram_factor(S, 'F', {'L'…})
            |   |      |
            |   |      Level names, one per factor level; default {'G', 'NG'}
            |   |
            |   Factor name; required
            |
            AGLSS data object; required
```

```
chstr_factor(S, 'F', [T…], {'L'…}, [p P], Z)
             |   |    |       |       | |   |
             |   |    |       |       | |   Maximum chunk size to control; default 3
             |   |    |       |       | |
             |   |    |       |       | Fraction of high chunk strength string's chunks at highest frequency level; default p
             |   |    |       |       |
             |   |    |       |       Fraction of low chunk strength string's chunks at lowest frequency level > 0; default 0.8
             |   |    |       |
             |   |    |       Level names, one per factor level; defaults depend on number of levels
```

| | | |
| | | Chunk strength targets (relative or absolute), one per level; default `[1 2]`
| | |
| | Factor name; required
| |
AGLSS data object; required

rulestr_factor`(S, 'F', [T…], {'L'…}, [X G], Z)`

| | | | | | | |
| | | | | | | Maximum chunk size to control; default `3`
| | | | | | |
| | | | | | Minimum number of grammatical chunks of size *`Z`* to exclude from training strings; default `2`
| | | | | |
| | | | | Minimum number of chunks (grammatical or not) to exclude from training strings; default *`G`*
| | | | |
| | | | Level names, one per factor level; defaults depend on number of levels
| | | |
| | | Rule strength targets, one per level; default `[1 2]`
| | |
| | Factor name; required
| |
AGLSS data object; required

chnov_factor`(S, 'F', [T…], {'L'…}, [X G], Z)`

| | | | | | | |
| | | | | | | Chunk size to control; default `2`
| | | | | | |
| | | | | | Minimum number of grammatical chunks of size *`Z`* to exclude from training strings; default `2`
| | | | | |
| | | | | Minimum number of chunks (grammatical or not) to exclude from training strings; default *`G`*
| | | | |
| | | | Level names, one per factor level; defaults depend on number of levels
| | | |
| | | Chunk novelty targets, one per level; default `[0 1]`
| | |
| | Factor name; required

```
                |
                AGLSS data object; required
```

```
sim_factor(S, 'F', [T…], {'L'…}, @M, {P…}, @N)
           |    |     |       |       |    |      |
           |    |     |       |       |    |      Scaling function; default XX
           |    |     |       |       |    |
           |    |     |       |       |    Parameters to similarity function; default {0.7,0.7,1} for @EditSim, else {}
           |    |     |       |       |
           |    |     |       |       Similarity function; default @EditSim
           |    |     |       |
           |    |     |       Level names, one per factor level; defaults depend on number of levels
           |    |     |
           |    |     Similarity targets (percentile or absolute), one per level; default [25 75]
           |    |
           |    Factor name; required
           |
           AGLSS data object; required
```

```
factorial_testsets(S, {'F', 'L'…}, …)
                   |     |     |
                   |     |     Names of levels of F to be included; required
                   |     |
                   |     Name of factor to be combined factorially with others; required
                   |
                   AGLSS data object; required
```

```
factorial_testsets(S, {{'F', 'L'…}, {…}}, …)
                   |      |     |
                   |      |     Names of levels of F to combine pairwise with corresponding levels of other factors; required
                   |      |
                   |      Name of factor to combine pairwise with other factors; required
                   |
                   AGLSS data object; required
```

choose_items(S, N, M, N0)
- S: AGLSS data object; required
- N: Number of training strings desired; required
- M: Number of test strings desired per combination of factor levels; required (XX?)
- N0: Number of head start training items; default XX

format_train_items(S)
- S: AGLSS data object; required

format_test_items(S, 'P')
- S: AGLSS data object; required
- 'P': 'detail' or 'summary'; default 'detail'

Acknowledgments

This research was partly supported by ESRC grant R000222655 and EC Framework 6 grant contract 516542 (NEST).

References

Ashby, G. F., Alfonso-Reese, L. A., Turken, A. U., & Waldron, E. M. (1998). A Neuropsychological Theory of Multiple Systems in Category Learning. Psychological Review, 105, 442-481.

Bailey, T. M., & Hahn, U. (2001). Determinants of wordlikeness: Phonotactics or lexical neighborhoods? Journal of Memory and Language, 44, 568-591.

Berry, D. C., & Dienes, Z. (1993). Implicit learning: Theoretical and empirical issues. Hove, England: Lawrence Erlbaum Associates.

Boucher, L., & Dienes, Z. (2003). Two ways of learning associations. Cognitive Science, 27, 807-842.

Brooks R. L., & Vokey, R. J. (1991). Abstract Analogies and Abstracted Grammars: Comments on Reber (1989) and Mathews et al. (1989). Journal of Experimental Psychology: Learning, Memory and Cognition, 120, 316-323.

Cabrera, A (1995). The "Rational" Number e: A Functional Analysis of Categorization. In Proceedings of the 17th Annual Conference of the Cognitive Science Society, Mahwal, NJ: Lawrence Erlbaum Associates.

Chomsky, N., & Miller, G. A. (1958). Finite State Languages. Information and Control, 1, 91-112.

Dienes, Z., & Altmann, G. (2003). Measuring learning using an untrained control group: Comment on R. Reber and Perruchet. The Quarterly Journal of Experimental Psychology, 56A, 117-123.

Dirlam, D. K. (1972). Most Efficient Chunk Sizes. Cognitive Psychology, 3, 355-359.

Dulany, D. E., Carlson, R. A., & Dewey, G. I. (1984). A case of syntactical learning and judgment: How conscious and how abstract? Journal of Experimental Psychology: General, 113, 541-555.

Hahn, U., & Bailey, T. M. (2005). What makes words sound similar? Cognition, 97(3), 227-267.

Johnstone, T., & Shanks, D. R. (1999). Two Mechanisms in Implicit Grammar Learning? Comment on Meulemans and Van der Linden (1997). Journal of Experimental Psychology: Leaning, Memory, and Cognition, 25, 524-531.

Knowlton, B. J., & Squire, L. R. (1994). The Information Acquired During Artificial Grammar Learning. Journal of Experimental Psychology: Learning, Memory and Cognition, 20, 79-91.

Knowlton, B. J., & Squire, L. R. (1996). Artificial Grammar Learning Depends on Implicit Acquisition of Both Abstract and Exemplar-Specific Information. Journal of Experimental Psychology: Learning, Memory and Cognition, 22, 169-181.

Luce, P. A., & Pisoni, D. B. (1998). Recognizing spoken words: The neighborhood activation model. Ear and Hearing, 19, 1-36.

Meulemans, T., & van der Linden, M. (1997). Associative Chunk Strength in Artificial Grammar Learning. Journal of Experimental Psychology: Learning, Memory, and Cognition, 23, 1007-1028.

Meulemans, T., & Van der Linden, M. (2003). Implicit learning of complex information in amnesia. Brain and Cognition, 52, 250-257.

Nosofsky, R. M. (1988). Similarity, Frequency, and Category Representation. Journal of Experimental Psychology: Learning, Memory and Cognition, 14, 54-65.

Perruchet, P., & Pacteau, C. (1990). Synthetic Grammar Learning: Implicit Rule Abstraction or Explicit Fragmentary Knowledge? Journal of Experimental Psychology: General, 119, 264-275.

Perruchet, P., Vinter, A., Pacteau, C., & Gallego, J. (2002). The formation of structurally relevant units in artificial grammar learning. Quarterly Journal of Experimental Psychology, 55A, 485-503.

Poldrack, R. A., Clark, J., Pare-Blagoev, E. J., Shohamy, D., Creso Moyana, J., Myers, C., & Gluck, M. A. (2001). Interactive memory systems in the human brain. Nature, 414, 546-550.

Pothos, E. M. (2007). Theories of Artificial Grammar Learning. Psychological Bulletin, 133, 227-244.

Pothos, E. M. (2005). The rules versus similarity distinction. Behavioral & Brain Sciences, 28, 1-49.

Pothos, E. M., & Bailey, T. M. (2000). The Importance of Similarity in Artificial Grammar Learning. Journal of Experimental Psychology: Learning, Memory, and Cognition, 26, 847-862.

Pothos, E. M., & Cox, W. M. (2002). Cognitive bias for alcohol-related information in inferential processes. Drug and Alcohol Dependence, 66, 235-241.

Pothos, E. M. & Kirk, J. (2004). Investigating learning deficits associated with dyslexia. Dyslexia, 10, 61-76.

Reber, A. S. (1967). Implicit Learning of Artificial Grammars. Journal of Verbal Learning and Verbal Behavior, 6, 855-863.

Reber, A. S. (1976). Implicit learning of synthetic language. Journal of Experimental Psychology: Human Learning and Memory, 2, 88-94.

Reber, A. S. (1993). Implicit learning and tacit knowledge. New York: Oxford University Press.

Reber, A. S., Allen, R. (1978). Analogic and abstraction strategies in synthetic grammar learning: A functional interpretation. Cognition, 6, 189-221.

Reber, P. J., & Squire, L. R. (1999). Intact Learning of Artificial Grammars and Intact Category Learning by Patients with Parkinson's Disease. Behavioral Neuroscience, 113, 235-242.

Reber, R., & Perruchet, P. (2003). The use of control groups in artificial grammar learning. The Quarterly Journal of Experimental Psychology, 56A, 97-115.

Redington, F. M., & Chater, N. (1996). Transfer in Artificial Grammar Learning: Methodological Issues and Theoretical Implications. Journal of Experimental Psychology: General, 125, 123-138.

Servan-Schreiber, E. (1991). The Competitive Chunking Theory: Models of Perception, Learning, and Memory. Doctoral Dissertation, Department of Psychology, Carnegie- Mellon University.

Servan-Schreiber, E., & Anderson, J. R. (1990). Learning Artificial Grammars With Competitive Chunking. Journal of Experimental Psychology: Learning Memory and Cognition, 16, 592-608.

Shanks, D. R. (2005). Implicit learning. In K. Lamberts and R. Goldstone (Eds.), Handbook of Cognition (pp. 202-220). London: Sage.

Sloman, S. A. & Rips, L. J. (1998). Similarity as an explanatory construct. Cognition, 65, 87-101.cc

Smith, E. E., Langston, C., & Nisbett, R. E. (1992). The case for rules in reasoning. Cognitive Science, 16, 1-40.

Smith, E. E., Patalano, A. L., & Jonides, J. (1998). Alternative strategies of categorization. Cognition, 65, 167-196.

Vokey, J. R., & Brooks, L. R. (1992). Salience of Item Knowledge in Learning Artificial Grammar. Journal of Experimental Psychology: Learning, Memory & Cognition, 20, 328-344.

Wasserman, E. A., & Miller, R. R. (1997). What's elementary about associative learning? Annual Review of Psychology, 48, 573-607.

Witt, K., Nuehsman, A., & Deuschl, G. (2002). Intact artificial grammar learning in patients with cerebellar degeneration and advanced Parkinson's disease. Neuropsychologia, 40, 1534-1540.

Table 1. High level StimSelect functions, showing output arguments (returned values). The function Groups are: 1) defining a grammar; 2) creating an AGLSS data object; 3) specifying factors and factor levels; 4) combining factors; 5) constraint satisfaction to choose stimulus strings; 6) displaying chosen strings.

Group	Function and example

1	**fsg** – Create an FSG data object
	`F = fsg([], 'xy', 3)`
	link – Add links to an FSG data object
	`F = link(F, 1, 'xy.', [2 1 -1])`

2	**aglss** – Create an AGLSS data object `S = aglss(G, [4 8])`

3	**gram_factor** – Add a grammaticality factor to an AGLSS data object
	`[S, levNames] = gram_factor(S, 'Gram')`
	chstr_factor – Add a chunk strength factor to an AGLSS data object
	`[S, levNames, tgts] = chstr_factor(S, 'MyChStr')`
	rulestr_factor – Add a rule strength factor to an AGLSS data object
	`[S, levNames, tgts] = rulestr_factor(S, 'MyRuleStr')`
	chnov_factor – Add a chunk novelty factor to an AGLSS data object
	`[S, levNames, tgts] = chnov_factor(S, 'MyChNov')`
	sim_factor – Add a similarity factor to an AGLSS data object
	`[S, levNames, tgts] = sim_factor(S, 'MySimStr')`

4	**factorial_testsets** – Specify combinations of factors for an AGLSS data object `S = factorial_testsets(S, {'Gram', 'G', 'NG'})`
5	**choose_items** – Identify training and test strings that satisfy constraints in an AGLSS data object `S = choose_items(S, 20, 6)`
6	**format_train_items** – Format training strings in an AGLSS data object, ready for display `format_train_items(S)`

`format_test_items` – Format test strings in an AGLSS data object, ready for display

```
format_train_items(S)
```

Table 2. Ten training strings selected based on the Reber and Allen (1978) finite state grammar.

Itm_num	Itm_name
01	VXM
02	MSVRXM
03	MSV
04	MVRXRRM
05	VXVRXSV
06	VXSSV
07	VXSSSVS
08	VXV
09	MSSVRXR
10	MSSVRXV

Table 3. Displaying the test strings identified for a particular stimulus set, with the Reber and Allen (1978) finite state grammar. This test set has been created with a view to balance grammaticality and chunk strength.

Tset_num	gram_cat	ChStr_cat	Itm_num	Itm_name	ChStr	ChStr_tgt
01	G	LowChStr	01	VXRRM	0.367	0.400
01	G	LowChStr	02	VXRR	0.358	0.400
01	G	LowChStr	03	VXVS	0.450	0.400
01	G	LowChStr	04	VXRRRRM	0.293	0.400
01	G	LowChStr	05	VXRRRM	0.323	0.400
02	NG	LowChStr	01	RRXRXSR	0.377	0.400
02	NG	LowChStr	02	XXSVVXS	0.391	0.400
02	NG	LowChStr	03	SVRMVSS	0.405	0.400
02	NG	LowChStr	04	SVSM	0.358	0.400
02	NG	LowChStr	05	VMMRVRX	0.379	0.400
03	G	HighChStr	01	VXVRXRR	0.566	0.571
03	G	HighChStr	02	VXVRXRM	0.555	0.571
03	G	HighChStr	03	MSSSSVS	0.573	0.571
03	G	HighChStr	04	VXVRXV	0.591	0.571
03	G	HighChStr	05	MSSSSSV	0.589	0.571
04	NG	HighChStr	01	MVXVRX	0.565	0.571
04	NG	HighChStr	02	MSVXSVR	0.575	0.571
04	NG	HighChStr	03	RXSSVRR	0.581	0.571
04	NG	HighChStr	04	SVRXVRV	0.576	0.571
04	NG	HighChStr	05	SVXVRXS	0.577	0.571

Table 4. A summary of the properties of the stimulus set identified.

Test item summary:

Tset_num	gram_cat	ChStr_cat	ChStr	ChStr_tgt
01	G	LowChStr	0.358	0.400
02	NG	LowChStr	0.382	0.400
03	G	HighChStr	0.575	0.571
04	NG	HighChStr	0.575	0.571

Figures

Figure 1. The finite state grammar used by Knowlton and Squire (1996, Exp. 1).