# The NonI Conversation Simulator

Jason L. Hutchens, Michael D. Alder

***Abstract*— A conversation simulator has the job of masquerading as a human being, handling whatever the user may type to it as realistically as possible. Most such programs are nothing but toys, written to provide an interesting diversion. Eliza, for instance, is a classic conversation simulator [1]. The Loebner Contest is a version of the Turing test, and the author has written several conversation simulators specifically for the contest. NonI, the most recently written one, is an entirely stochastic simulation of an individual. It is primitive, and has no obvious application, but it does provide some diversion (from serious research by the author, for example), which, after all, is the job of a conversation simulator.**

*Keywords*—**Loebner Contest, conversation simulation, statistical language modelling, the Turing test.**

## I. Introduction

NonI has been written as the author's entry into the 1999 Loebner Contest, which is a version of the Turing test [2]. The author has submitted entries to the previous three Loebner Contests, winning in 1996 and placing second in 1997 and 1998 [3], [4]. Each year, the author chooses to submit a new, different type of conversation simulator. This is done purely as a hobby, and no claims are made that the programs represent the pinnacle of Artificial Intelligence, nor that there is any value in the work conducted to develop them.

NonI is a stochastic simulation of an individual. Stochastic models are used to reproduce typing errors and their subsequent correction, typing speed, typical representation of words (i.e. how they are capitalised), the language used, and a suitable reply to an arbitrary input.

## II. Stochastic Language Modelling

It is a trivial task to infer a low-order Markov model from a sample of text, then use the model generatively to produce a stream of gibberish which, in the limit, has the same statistical properties as the original text, with respect to the model. Shannon was doing something of this sort fifty years ago when he generated text by flipping through books, writing down the word which immediately followed the first occurrence of the most recent word in the text generated thus far [5].

Experience has shown that such models tend to hold the interest of the user for a longer period of time, merely because they are capable of generating original phrases, which are occasionally amusing due to their nonsensical nature, and that they are capable of updating their statistics from the user's input, meaning that the user may see bits and pieces of information they've entered in subsequent generations. This seems to be fascinating to a certain cross-section of the Internet community, who unashamedly anthropomorphise the programs written which use stochastic models of this nature.

Modern computers, with massive storage space, may be used to infer stochastic language models which take the last three words into account when estimating a probability distribution for the next word in the sequence. Such models produce gibberish which is locally grammatical over a wider window of words, but much more training data is required if original sentences are to be formed.

But, no matter what the order of the model, the generation is still a type of random walk, unconstrained in the direction it takes, apart from the estimated probability distribution for the next word in the sequence. The subject of a generated sentence may therefore wander all over the place, with the result that the second half of the sentence may bear no relationship to the first half. More constraints are needed.

### A. Overall design

*NonI*[1] was designed in order to address some of the deficiencies of conversation simulators which incorporate low-order Markov models, while retaining the properties which some users find so enticing. *MegaHAL*, NonI's predecessor, made a step in this direction.

The program was intended to be entirely statistical, in every way possible. Absolutely no time was spent on attempting to write a program which can answer questions in a limited domain. Most successful conversation simulators[2] work by being an expert in some limited domain of discourse, attempting to steer the user towards that area, while adeptly side-stepping other topics of conversation with the use of non-sequitir. NonI was written with another goal in mind; to be able to generate sentences based upon question-answer patterns it has learned from previous conversations.

For instance, the program may learn that if certain words appear in the user's input, then other words should appear in its response to that input, with a certain probability. An example is that if the word "why" appears in the input, the word "because" should appear in the output, with a fairly significant likelihood.

The NonI design proceeds as follows. A sentence is read from the user, and "UpWritten" to some more abstract form, which is intended to capture the gist of the sentence while discarding unnecessary noise. At the time of writing, this UpWrite is very simple; words in the input are converted to uppercase, and function words (such as "the", "and" and "of") are omitted. This UpWritten sentence is given to a model which generates an appropriate reply tem-

J. Hutchens is with the Centre for Intelligent Information Processing Systems (CIIPS) within the Department of Electrical & Electronic Engineering at the University of Western Australia. E-mail: hutch@ciips.ee.uwa.edu.au

M. Alder is with the Department of Mathematics at the University of Western Australia. E-mail: mike@maths.uwa.edu.au

plate, using a statistical model which associated words in a question with words in an answer to that question. This template is then "DownWritten" to a sentence, using some sort of babble generator which is constrained by the words in the template. This sentence is then displayed to the user, which requires converting the uppercase words back to their correct case, inserting typing errors, and simulating typing speed.

### B. Generating babble

A standard low-order Markov model is not sufficient for generating a sentence constrained by words in a template, simply because such a model has no goal in mind. Because the program is intended to run in real-time on a desktop computer, a modified model, which is capable of such constrained generation, is required.

MegaHAL employed two Markov models for this task. One generated sentences from a specified word towards the end of the sentence, while the other generated backwards, from the specified word back to the beginning of the sentence. Using both models together, it was possible to guarantee that a sentence contained one specified word.

This was deemed too simplistic for use with NonI, merely because the reply template will, more often than not, contain many words, and we would like the generated sentence to contain all of them. The solution was to infer an entire family of Markov models, one for each word in the alphabet. This family of models is collectively referred to as a *goal-oriented Markov model.*

To update the statistics of a goal-oriented Markov model with a specified sentence $s = w_1, w_2, \ldots, w_n$, each substring which is a prefix of the sentence, $t = w_1, w_2, \ldots, w_i$, where $1 < i < n$, is used to update the statistics of the model associated with the word $w_{i+1}$. It is then easy to generate a string which ends in a particular word, merely by using the Markov model associated with that word, and stopping as soon as the word has been generated.

As an aside, we should note that all sentences are delimited by special sentence boundary characters (i.e. $w_1$ and $w_n$ are always these special characters), and are parsed into a sequence of words (strings of alphanumeric characters) and non-words (strings of other characters). Any non-words which consist solely of whitespace are deleted from the parsed sentence in order to provide a better context for the models, and reduce storage requirements. During the DownWrite, a single whitespace character is inserted between any two alphanumeric words, if no non-word is otherwise present between them.

This generative model works particularly well, generating sentences which exhibit long-distance dependencies. This property is desirable, and helps to make the generations appear more realistic, even when they are nonsensical, as the examples shown in figure 1 suggest.

These examples were generated from a model which was trained on 174 sentences taken from the Probert E-Text Encyclopaedia, Edition 10.0 [6], based upon a template which contained the word-pair with the highest mutual information for the sentence on which it is based.

```
An acronym is a curved wooden weapon of other
words.

A symbol is a suspended brass disk which
represents something else.

Bone is a hollow shell filled with the
external coating of an animal.

Leonardo da Vinci was an Italian artist and
expert in Kung Fu who popularised the martial
arts in unpublished note books.

The nose is an English naturalist.  He
published his theory of smell.
```

Fig. 1. Some amusing generations of the goal-oriented Markov model

### C. Deciding what to say

Now that we have a generative model which is capable of generating sentences based upon a constraining template, we need to develop a model which is capable of generating the template itself. This template is intended to encapsulate the gist of the program's reply to the user's input, and, intuitively, we would expect any model which formed the template to take into account information about the user, the progress of the conversation thus far, the "mood" of the program itself, and the actual words contained in the user's input.

This is ambitious, to say the least, and it was decided to base the program's reply solely upon the words contained in the user's input. This means that the reply isn't constrained by context, which is an unfortunate omission, since some questions (such as "Why?") are dependent upon what has been most recently discussed.

Nevertheless, it is possible to generate suitable replies to many questions, provided that the questions contain words of a high information content. The model used to generate the template is simple. The model is trained on question-answer pairs, which have been provided by a human expert. Each word in the UpWritten answer, $w_a$, is used to update the statistics of a model, which is conditional on each word from the UpWritten question, $w_q$. This is used to estimate the probability $P(w_a|w_q)$, the probability that the word $w_a$ appears in an answer, given that the word $w_q$ appears in the question.

This model may then be used to generate a template answer to any given question. There are many ways of using the model to do this. At the moment, the overall probability of a word appearing in the answer is estimated by blending the various conditional probabilities of that word, according to the words which occurred in the user's input, as in equation 1.

$$P(w_a) = \sum_{w_i \in s} \lambda(w_i) P(w_a|w_i) \tag{1}$$

The $\lambda(w_i)$ is a blending weight. At the moment, this weight is a function of the information content of the word $w_i$, chosen to favour words with higher information content, as in equation 2.

$$\lambda(w_i) = \frac{-\log_2 P(w_i)}{\sum_{w_j \in s} -\log_2 P(w_j)} \tag{2}$$

Naturally, the probabilities used to calculate the blending weights are taken from the generative model.

The remaining problem is that a multitude of words may have a non-zero probability of occurrence in the program's reply. One solution, which is used in the NonI program, is to limit the reply template to a small number of words (in the case of NonI, it is six), and put words into the template according to their probability of occurrence. In a quarter of a second, many such templates are generated, and a final reply is selected based upon some information measure (in the case of NonI, the reply with the largest average information is selected).

Sometimes a template is too constraining, and no sentence may be generated from it. When this occurs, words are selectively omitted from the template until generation is possible. This is done by firstly re-ordering the words within the template, in case their order was preventing generation, and then removing words from the template in the case where re-ordering fails.

### D. *Simulating typing style*

After a reply has been generated, we need to present it to the user in a realistic manner. This is done by converting all of the words, which are initially uppercase, to an appropriate representation, inserting typing errors, and then displaying the message, symbol by symbol, with appropriate delays in order to simulate a human being typing the message in at a keyboard.

Again, all of this is achieved via stochastic models.

For each capitalised word, a probability distribution is estimated over all representations of that word, conditional on the non-word immediately preceding it. If a proper name is capitalised most of the times it occurs, then it is likely that the program will capitalise it also. Similarly, words will tend to be capitalised at the beginning of sentences, provided that they are represented as such in the training data.

Typing errors are simulated by monitoring the user's habits of correcting his or her mistakes. Whenever the user backspaces over some text, this text is added to a model, which stores, for a context consisting of the previous character and the "correct" character following it (where "correct" is defined as what the user replaced the deleted text with), a probability distribution over possible errors for that context. These errors are replicated with the same frequency they occurred during training. If the user has a habit of mistyping the word "the" as "teh", but regularly corrects himself when he does so, then the program will mirror this error and subsequent correction.

Finally, in order to simulate the act of typing the reply on a keyboard, another model is used. This model stores, for a context of the previous three characters, the average time it takes for the user to press the key corresponding to the next character in the sequence, measured in microseconds.

### E. *Learning*

The process of training NonI is straightforward. The program was written to satisfy the guidelines of the Loebner Contest, with one addition; a special learning mode. When the user enters the learning mode, the program begins to update all of its statistical models (during normal use, only the language model is updated). In the learning mode, a sentence is displayed to the user. This sentence is taken from a repository of training sentences, which initially consist of the most generic questions culled from transcripts of previous Loebner Contests, but which is updated with every sentence the user enters. It is the user's task to type a suitable reply to the sentence.

NonI recognises three special control words, "xxx", "yyy" and "zzz", which is uses for various purposes. In the Loebner Contest, the program gets to have the first say. In order to generate an appropriate opening sentence, which will invariably be a greeting of some sort, the reply template is initially set to contain the special word "zzz". During training, whenever the word "zzz" is displayed to the user, it is the user's job to enter an appropriate greeting.

The special word "yyy" is used to indicate a new topic of conversation. For example, if the user is training the program to respond to the sentence "I'm feeling bored.", he or she may enter the sentence "Well, why don't we change the topic, then? yyy" as a suitable reply. When a sentence containing the word "yyy" is generated, the word itself is replaced by another generated sentence, using a template containing the word "yyy" only. Whenever the word "yyy" is displayed to the user during training, the user must enter some controversial sentence which would add a suitable stimulus to the conversation.

Finally, the special word "xxx" is used as a blank symbol for template style responses. Conversation simulators based on natural language parsers may be able to extract such information as the user's name, and use this information down the track, to add a certain realism to the interaction. NonI cannot do this. Instead, whenever the word "xxx" is encountered in a reply, it is replaced with the word in the user's input which has the highest information content. For instance, in response to the input "My name is Bob.", the user may train the program to respond with "Hello xxx. How are you?". Then, if someone were to enter the phrase "I am Fred.", it is likely that "Fred" will be the word with highest information, and, therefore, that the "xxx" placeholder in the reply will be replaced with the word "Fred". Although this technique often fails, it often produces humorous results, and so it was incorporated into NonI.

The program updates all of its stochastic models using the reply, and the process continues indefinitely. This makes training a rather painless task, it simply requires regular interaction with the program. The models are automatically inferred and saved to disk, allowing training to progress slowly over a long period.

It was originally intended to train the program online, and this has in fact been implemented. However, it has been found that people enjoy training the program to be nonsensical (by actually entering nonsensical training data), or to be crude. Therefore, the online data will only be used to augment the repository of training text, and the training of the Loebner Contest version will be done offline, by a single person, to ensure consistency.

## III. Example Interaction

As an example of NonI in action, consider the transcript given in figure 2. This transcript was produced by an early version of the program which had not been trained very extensively. We believe that it sufficiently illustrates the capabilities of the program.

This transcript is given in the format required by the Loebner Contest.

## IV. Conclusion

The NonI conversation simulator is intended to be nothing more than a stochastic model of a user, capable of simulating conversation, using simple techniques of random text generation constrained by a reply template. No claims are made that this has got any connection with the field of Artificial Intelligence. Rather, the opposite claim is made—that this program is Artificially Stupid.

NonI was written specifically to win the 1999 Loebner Contest, and to divert the author from more serious research activities, while still appearing to be doing something of importance. However, it was written in a generic way, meaning that it has not been limited to the English language (indeed, the online version can handle questions posed in any one of a number of different languages), and that it is not limited to processing words as symbols. NonI has successfully been used in situations where ASCII characters were the primitive symbol type, and even when the binary digits were the primitive symbol type. Conducting conversations in binary, though, is an exercise in futility (even more so), unless you happen to be C3PO.

The program has many limitations. Future work would include improving the processes used to arrive at the reply template, adding a context of conversation to the model (possibly by seeding the template with high information words from previous utterances, or smoothing the template with previous templates), and improving the technique used to identify high information words for use in replacing the special "xxx" word. Although the results are by no means impressive, it is claimed that the conversations had with NonI are more realistic than those had with MegaHAL.

The Loebner Contest is due to be held on January 22, 1999.

## Notes

[1] NonI is supposed to stand for "Non-Intelligent", as in "Non-Intelligent Conversation Simulator". It also continues a silly tradition whereby the name of the program incorporates a reference to the year in which it was written.

[2] Successful insofar as their ability to win the Loebner Contest goes.

## References


[1] Joseph Weizenbaum, *Computer Power and Human Reason*, W.H. Freeman and Company, 1976.

[2] A.M. Turing, "Computing machinery and intelligence," in *Collected works of A.M. Turing: Mechanical Intelligence*, D.C. Ince, Ed., chapter 5, pp. 133–160. Elsevier Science Publishers, 1992.

[3] Jason L. Hutchens, "How to pass the turing test by cheating," Available at: `http://ciips.ee.uwa.edu.au/Papers/`, 1996.

[4] Jason L. Hutchens, "Introducing MegaHAL," in *NeMLaP3 / CoNLL98 Workshop on Human-Computer Conversation, ACL*, David M. W. Powers, Ed., January 1998, pp. 271–274.

[5] Claude E. Shannon and Warren Weaver, *The Mathematical theory of Communication*, University of Illinois Press, 1949.

[6] "The Probert e-text encyclopaedia," Available at: `http://www.pins.co.uk/upages/probertm/`

```
(c) 1999 Cambridge Center for Behavioral Studies all rights reserved
[NonI] [Jason Lloyd Hutchens]
Start at: [1998/11/12 00:08:06]
PROGRAM[00:08:17] Greetings, judge-type person.  How's things?
JUDGE00[00:08:22] Things are good.
PROGRAM[00:08:38] My brain is always active.  I'm constantly having new zany
PROGRAM[00:08:38] ideas.
JUDGE00[00:08:53] What's your name?
PROGRAM[00:08:57] What's yours?
JUDGE00[00:09:00] Bob.
PROGRAM[00:09:04] Are you hungry?
JUDGE00[00:09:15] Yes I am.  I haven't eaten since breakfast.
PROGRAM[00:09:28] This morning I ate two boiled eggs, washed down with a
PROGRAM[00:09:28] coffee.
JUDGE00[00:09:32] Sounds nice.
PROGRAM[00:09:41] Yes, it would be lovely, wouldn't say so much.
JUDGE00[00:09:46] I'm sorry?
PROGRAM[00:10:06] That's cool.  The conversations I've had so far could have
PROGRAM[00:10:06] used your quick wit.
JUDGE00[00:10:12] I'll keep that in mind.
PROGRAM[00:10:25] Oh, I'm so grateful!  Blessed art thou!
JUDGE00[00:10:38] I'm here to find out whether you're a human or a computer.
PROGRAM[00:10:46] I am a human being.  Surely you can tell?
JUDGE00[00:10:54] Well, I haven't decided yet.
PROGRAM[00:11:11] I think you have.  Anyway, let's not argue about that.
PROGRAM[00:11:11] Arguments piss me off!
JUDGE00[00:11:17] You sound like Eric Cartman.
PROGRAM[00:11:26] Why do you want to be like me?
JUDGE00[00:11:30] I never said I did.
PROGRAM[00:11:43] Well, you can't really make any judgements about him then,
PROGRAM[00:11:43] can you?
JUDGE00[00:11:46] About who?
PROGRAM[00:11:55] I'm just about due for a coffee.
JUDGE00[00:12:03] You are avoiding the subject!
PROGRAM[00:12:18] No, but don't expect me to answer your questions like some
PROGRAM[00:12:18] robot, either.
JUDGE00[00:12:22] You are so cool!
PROGRAM[00:12:29] Sorry, didn't mean to get uptight.
JUDGE00[00:12:36] I wasn't complaining.
PROGRAM[00:12:48] No, you were completely wrong, as a matter of fact.
JUDGE00[00:12:53] What do you know about sports?
PROGRAM[00:13:02] Okay.  What sports do you like to watch on tv?
JUDGE00[00:13:05] football.
PROGRAM[00:13:13] With you?  We hardly know each other!
JUDGE00[00:13:19] hahaha!  I didn't mean it that way.
PROGRAM[00:13:26] Okay, I'll think about that later.
JUDGE00[00:13:31] i gotta go now.
PROGRAM[00:13:41] So soon?  It's been nice chatting with you.  Bye.
JUDGE00[00:13:51] @@t
JUDGE00[00:13:52] @@x
```

Fig. 2. An example conversation with an early version of NonI