

# Computing New Behavior from Learned Patterns

Tobey Thorn  
( tobeythorn.com )

March 2011

## Abstract

This article presents a simple but powerful technique for computing new behavior from learned patterns of behavior. This technique was developed for a name generator application I wrote for iOS, but could be applied to a variety of other artificial intelligence problems. In practice, this approach is effective, flexible, and computationally efficient.

## Approach

Many artificial intelligence problems require a program to learn patterns of behavior and then to act accordingly. In the case of name generators, the goal is construct novel names based on the patterns by which syllables are arranged in an existing lexicon (a lexicon is the set of names/words belonging to a language or culture). Determining these patterns is difficult for most real-world lexicons. For example, in the English language, the complete set of possible syllables is not known (there likely many thousands). Moreover, the number of valid arrangements of those syllables is nearly infinite and the rules of arrangement are not well understood. Subsequently, hard-coded rules cannot adequately capture the patterns within most lexicons.

The solution presented in this paper largely overcomes the problem of determining the set of syllables and the rules for arranging them. Many real names are analyze in order to find which sequences of syllables occur and how often. Novel names are then constructed based on these statistics. This method works well with most lexicons. The resulting names sound realistic and have the same "feel" as the real names found in the original lexicon.

## Finding Patterns

We start with a large list of real names belonging to a given lexicon where two space characters separate each name. Stepping through the list one character at a time, we record each unique three-character sequence and the number of times it occurs, as is shown in Figure 1. Spaces and case are taken into account because they encode the patterns for how names begin and end. Each sequence is represented by four variables, which I have suggestively called PAST, PRESENT, FUTURE, and OCCURANCE. This can add up to a lot of sequence data, but it can be stored, updated, and traversed efficiently using a tree structure.

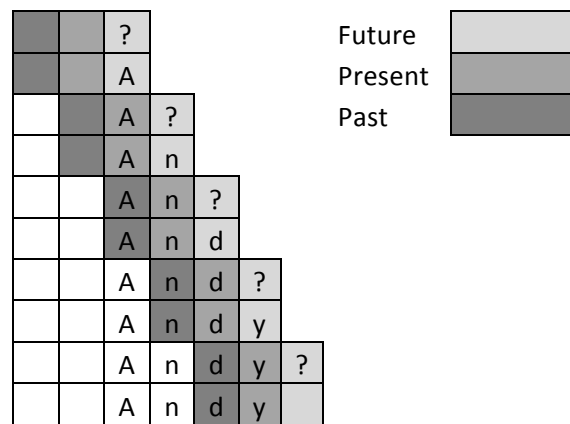
Past			A	n	d	r	e			S	a	n	d	y			C
Present		A	n	d	r	e			S	a	n	d	y			C	a
Future	A	n	d	r	e			S	a	n	d	y			C	a	n
Occurrence	1	1	1	1	1	1	1	1	1	1	2	2	2	2	1	1	1

Figure 1.

Each column represents a sequence and the number of times it occurred. This example shows the sequences that would be found in the list of names " Andre Sandy Candy ".

## Generating Novel Behavior

Now that we have found patterns for how syllables can be arranged, we can use them to generate novel names. We start with two space characters, since names or words always start with spaces in Romanized lexicons (PAST=" ", and PRESENT=" "). The first non-space character of the name is not yet known. We look at all the stored sequences that start with two spaces. We then select a FUTURE character at random based on the probability of it completing the partial sequence. We now have the first non-space character of our name. We then shift forward. The PAST character takes the value of the PRESENT character, the PRESENT character takes the value of the newly determined FUTURE character, and the FUTURE character once again becomes unknown. Given this partial sequence, the next letter is then found in the same way as the first letter. This process continues until the FUTURE variable happens to become a space character, marking the end of the name.



**Figure 2.**

Here, the process of creating one possible name using the lexicon used in Figure 1 is shown. Note that "Andy" is a novel name given the lexicon it derives from.

Note that this algorithm automatically makes names of reasonable lengths based on the probability of choosing a terminating space character. At any time, pattern data can be updated or added to without interrupting the generation process. This technique could be modified to learn sequential patterns longer than three variables or where each data point consisted of multiple variables.

## Case Study

This technique described in this paper was featured in a name generator application I wrote for iOS called *essa*. This technique works well to generate realistic and novel names with great reliability for a variety of lexicons.

## Limitations

This technique assumes that each variable is definite and that there are relatively few possible values. This works well for name generator applications, since there are few characters in the Roman alphabet. However, this technique is not suited for problems where data may take on many different values or where those values may be ambiguous. For example, this technique could work well for navigating an AI bot around a grid filled with obstacles, but would work poorly for navigating an AI bot around an environment with a floating-point coordinate system.