Prolog Realization of a Poetry Generator

Vassia Atanassova, Stephan Pulov

***Abstract*—The paper presents a poetry generator realization in SWI Prolog. Based on an original approach for composing computer poetry, the generator produces rhymed verses on the basis of user-defined rhythm templates preserving the specific melody of the verse in grammatically correct sentences. Some results of the program tests are given.**

***Index Terms*—Artificial intelligence, Computer poetry**

I. Introduction

Loosely speaking, the artificial intelligence collects together various techniques aiming to understand and simulate the mental activity of humans. The main, but still unachieved target of the artificial intelligence is to find the key, which will allow constructing the most versatile computer, such that combines skills for perceiving, reasoning, planning, performing art, etc. Our ability to explain or express thoughts and feelings is a challenging perspective for the artificial intelligent systems. In this paper we focus on one implementation of the Prolog programming language for creating a generator of poems. To the authors' best knowledge, there is not clear definition of what precisely poetry is. The lack of understanding of the mental processes that endow the verse with enduring fascination is the major obstacle to approaching "the universal poetry generator." There is not even a generally held view of which text is a verse, and which is not a verse. The very frequent situation is when someone accepts a piece of writing as a poem while others may not comprehend its meaning at all. It is quite usual that individual opinions about poetry differ very much from one to another. To some extreme extent, a free verse, which means a verse without rhythm and rhyme, is widely regarded as a true verse, although it may be taken for a plain text as well.

We came to the idea of poetry generator after we read the articles [1]-[3], deciding this to be our project for the "Prolog programming" course we attended at Sofia University as second year students of computer science.

II. What Makes Any Text a Poem?

In order to tackle with the problem, we have to accept an adequate conception of the verse. To our best understandings of the matter, we agree with the following working definition taken from [4].

"Poem is a piece of writing in which the words are chosen for their sound and the images they suggest, not just for their obvious meanings. The words are arranged in separate lines, usually with a repeated rhythm, and sometimes the lines rhyme at the end."

According to this definition, we distinguish rhythm and rhyme to be the two main features that differentiate verse from prose. We consider rhythm and rhyme as the basic constructive elements, which in a closely interwoven pattern form the unique consonance of the verse. From phonetic point of view rhythm is a periodic repetition of sounds of human speech determined by an alternation of syllables being stressed and those being not. Rhyme is when the words have the same sound, especially at the ends of lines in a verse, thus lending the specific harmony of the poem. We wondered whether computer is able to determine the verse rhyme and rhythm by itself. Obviously, rhyme is not a problem to be realized as long as there are firm rules for word rhyming, while rhythm generating has all the difficulties being observed with the algorithms for music generating. However, since we have a formal definition of what rhythm is, we are able to program the computer to compose rhythmic verse samples, but it is only a matter of statistics how many and which of the poems in result would sound good enough. In order to avoid generating verses with "bad rhythm" we decided to use rhythm templates based on already existing poetry samples, so that our poetry generator would produce rhymed verses with user-defined rhythmic sound.

III. Description of the Algorithm

We set several rhythm templates (allowing new user-defined ones to be added) with a single parameter consisting of a sequence of the digits 0 and 1. One special atom "**b**" serves to indicate the break of the verse line as it is seen from the following example

```
rhythm_template([0,1,0,1,0,0,0,1,0,b,
                 0,0,0,1,0,0,0,1,b,
                 0,0,1,0,0,0,1,0,1,0,b,
                 0,1,0,1,0,0,0,1,b]).
```

The digit 1 corresponds to a syllable, on which the stress falls, while the digit 0 corresponds to an unstressed syllable. We only have to fill in the template with words from a static database. The database consists of words classified in categories depending on the part of the speech the words belong to, as nouns, verbs, adjectives, adverbs, pronouns, interjections, etc. Here is an example how the noun "sample" is represented

```
noun(sample, [1,0]).
```

Vassia K. Atanassova is a second year student in the Faculty of Mathematics and Computer Science, Sofia University, Bulgaria (e-mail: vassia_atanassova@abv.bg)

Stephan V. Pulov is a second year student in the Faculty of Mathematics and Computer Science, Sofia University, Bulgaria (e-mail: s_pulov@abv.bg)

These categories exist in all human languages but their interrelations are specific and depend on the choice of the language. We started our work over the Bulgarian version of the generator. The problems we met were related mainly to the order-free structure of the sentence – one and same thing can be expressed in the same words but in different order. Defining many syntactically correct sentence constructions – like this one

```
sentence_template([adjective,noun,
verb]).
```

particularly solves this problem. So, we have two templates– one for the rhythm and the other for the word order corresponding to the morphological classification. The declarative nature of logic programming allows us to make the computer choose the proper word matching both templates, of course, paying the time price. Since the templates do not flow synchronously, we set a *plait-like algorithm* to compose the verse. The general idea how to make compatible the two threads of "rhythm and rhyme" and "word order and grammatical correctness" is letting them complement one another rather than opposing them. We realized this principle in a way that when a verse line (a line from the rhythm template) finishes before the end of the sentence (i.e. the sentence template), it continues from the beginning of the next verse line. Vice versa, if the sentence finishes at the middle of the line, a new sentence starts, thus keeping the melody and the rhythm of the poem. The only restriction we imposed was the end of the verse to coincide with the end of the present sentence.

Unfortunately, in Bulgarian language the morphological word order is never enough to make grammatically correct sentences. Each word is related to grammar categories like *gender* (masculine, feminine and neuter), *number* (singular and plural), *tense* (present, past, future, and their various forms), and *voice* (active and passive). Making agree of words one with another according to all of these categories is impossible since there are no formal rules for adding the proper suffix.

Being unable to solve this problem, we decided to run the program in English. We had in mind that English language does not depend that much on the actual conjugation of the verbs. After the change in language we had to change also the structure of the word database, since we discovered that in English we need one more parameter referring to the phonetic transcription – note the third parameter in the following example

```
noun(example,[0,1,0],[e,g,z,a,m,p,y,l]).
```

We needed this third parameter, since the rhyme realization was one of the differences between the Bulgarian and the English versions, we had to deal with. Analogously to the principle of "What You See Is What You Get", for the Bulgarian language it holds the principle "What You Spell Is What You Pronounce". Rhyme in Bulgarian is easier to realize, since a simple morphological analysis – letter by letter back to the accentuated syllable – ensures rhyming of the words. Not the same is the matter with the English language where a certain accumulation of letters rarely promises a common pronunciation. For instance, three words may end in "ough" but pronounce different – enough, through, plough. In English language apart of the monophthongs there are also diphthongs and trifthongs, which appear to be atoms in Prolog, and having the form of list they are easier to analyze. The generator uses the transcription parameter to rhyme the words, and when the proper words are chosen, the generator outputs the first parameters indicating their spelling form.

IV. Testing the Program

Each time we run the program it generates different rhythmic and rhymed poetic samples. For testing we used words from well-known classic poems. For instance, we tested the English version using the rhythm of the Beateles' song "Yellow Submarine". The generated verse is based on the original words of the song and some additional ones, which we added for the case.

In the town where I was born
Lived a man who sailed the sea
And he told us of his life
In the land of submarines

In the horse where I was born
Jump a chick who play the horn
And he rides us on his bean
In the land of submarine

V. Conclusions and Future Work

A Prolog realization of a poetry generator has been developed. Based on a database of words, categorized as to what part of the speech they belong, the poetry generator composes rhymed verses according to the user defined rhythm and sentence templates. The database is open to addition of new words. The sentence template guarantees constructing of grammatically correct sentences. The program uses an original approach for *conveying* the melody of the poem carried by a given rhythm template. As a result various verse samples inheriting the rhythmic sound of the template are produced. The user has the advantage to select those best corresponding to the human sense of poetry.

As a subject of future work we should mention the need of some additional efforts to be made in order to achieve sentences with semantically related words.

VI. Acknowledgment

The authors are indebted to Prof. Magdalina Todorova for her support and inspiring enthusiasm on the subject.

VII. References

[1] H. Tanev, "Computer poetry", *Computer*, vol. 11, 1997 (in Bulgarian).
[2] E. Sendova, "How to make the computer gossip, write poetry and silly jokes", *Mathematics and Information*, vol. 5-6, 2000 (in Bulgarian).
[3] A. Andrew, *Artificial Intelligence*, Abacus Press, UK, 1981.
[4] *Oxford Advanced Learner's Dictionary of Current English*, Sixth edition, Edited by Sally Wehmeier, 2000.