ReNoam

A Grammar Sequencing Tool for Renoise 2.8

Contents

1. Introduction

ReNoam is a Renoise 2.8 tool to generate pattern sequences by rules you specify as a formal grammar.

To all trained linguists reading this: I'm not planning to give formally correct definitions here, as this is addressing musicians and/or composers. I studied computational linguistics, computer science and psychology at Universität Tübingen, so I should be able to do the formal stuff, but this is no scientific paper.

This is fun stuff I do in my spare time -- instead of finishing some tracks. ;)

To all other readers: Please don't be afraid, most of ReNoam is really easy to understand and use. Have fun and don't hesitate to ask in the renoise forums, IRC, etc.

Please note: As ReNoam is in its beta phase, some more documentation comes in a *readme.txt* file. There is also a short introduction in the tool's text area as you start it up for the first time. Furthermore, there are some simple example grammars you could use as a starting point.

2. Short Workflow Example

- Create some nice unique patterns and sort them.
- Start ReNoam
- Write or load a grammar
- Hit the **Generate** button
- If you don't like the outcome, focus renoise, undo, and try again. Adjust your grammar if appropriate.

3. Accessing ReNoam

ReNoam can be activated from Renoise by

- menu entry in the Tools menu
- context menu in the Pattern Matrix area
- preferences keybinding Pattern Matrix

4. The GUI

The GUI is pretty simplistic: Just a text area and a few knobs:

- The **text area** is where you will edit your grammar. When you start the tool for the first time, there is a small example grammar (along with a short blurp by me).
- The **Generate** button will parse your input, constructing the grammar object for the tool. Next it will generate a sequence of numbers by using the given grammar. Those numbers will be used to append corresponding patterns in the pattern sequencer/pattern matrix.

- The **Save** button saves your input from the text area as a text file
- The **Load** button will load a text file to the text area
- If you use the **Close** button to close the tool window, the text in the text area is stored in the preferences, and will be restored at next start.

5. Formal Grammars and How to Use Them in ReNoam

Grammars supported in ReNoam 0.8 consist of

- a grammar type declaration
- a start symbol
- a set of rules

- comment lines (traditionally starting with a '#')
- empty lines (use them for readability)

If this is all gibberish to you, please have a look at the example grammars coming with ReNoam. I promise you it's really easy to understand what this is all about.

How does this work?
Starting out from your start symbol, the generator will use the rules to expand it to a sequence of pattern numbers. Whenever the generator finds a symbol matching the left hand side (LHS) of a rule, it will replace the symbol with the right hand side (RHS) symbols of the rule. If there are more than one matching rules, the generator will pick one of them. The choice will depend on the type of grammar you use.

6. Grammars Supported in ReNoam 0.8

ReNoam 0.8 supports two types of grammars (more to come):

- Context Free Grammar (CFG)
- Probabilistic Context Free Grammar (PCFG)

Read more about them below.

6.1 Context Free Grammar (CFG)

In a CFG, if you supply more than one matching rule, the generator will pick one of them at random, all with the same probability.

6.1.1 Grammar Declaration

Please note: The line for declaring a CFG is optional.
The line to declare this type of grammar is

```
ReNoamGrammarType = cfg
```

6.1.2 Start Symbol

Simply enter the start symbol on one line, like:

```
Song
```

6.1.3 Rules

Rules in a CFG look like this for example:

```
Song -> Intro Main End

Main -> PartA PartB Bridge PartA
Main -> PartA PartB C
Main -> X Main C

Bridge -> 6 9 10
```

6.1.4 Notes:

- If you use numbers in a rule, they correspond to pattern numbers. So obviously you mustn't use them on the left hand side of a rule. And they should be integrals, just your plain old pattern numbers, right?!
- For any non-number symbol on the right hand side, make sure there is a rule with this symbol on the left hand side, too. If you don't do this, the generator won't be able to expand the symbol.
- All non-number symbols may be used left and right of the arrow symbol. Recursiveness for your songs!
- If you use recursion, beware of infinite loops! Supply alternative non-recursive rules.

6.2 Probabilistic Context Free Grammar (PCFG)

In a PCFG, if you supply more than one matching rule, the generator will pick one of them at random with the probability you assign to the rules.

6.2.1 Grammar Declaration

The line to declare this type of grammar is

```
ReNoamGrammarType = pcfg
```

Please note: If you want to use a PCFG (or any grammar to be supported in the future), this declaration is mandatory!

6.2.2 Start Symbol

Simply enter the start symbol on one line, like:

```
Song
```

6.2.3 Rules

Rules in a PCFG look like this for example:

```
Song -> Intro Main End <1.0>

Main -> PartA PartB Bridge PartA <0.5>
Main -> PartA PartB C <0.4>
Main -> X Main C <0.1>

Bridge -> 6 9 10 <1.0>
```

6.2.4 Notes:

- All of the notes for CFGs apply.
- Please set probabilities for alternative rules! If you don't define a probability for a rule, the tool assumes a value just slightly bigger than zero.
- Probabilities for rules with the same left hand side are normalized to 1.0 If you have two rules - *both* with no probability specified – they both will get a probability of 0.5.

7. Questions

I don't know. If you have questions, just go ahead and ask me at the renoise forums. I prefer to answer questions in public, but if you're shy, PM me.

8. What's next?

- Of course the whole grammar writing has to become more user friendly. So grammar checking is a top prio for the next version.
- Jonas from Renoise forums asked for L-systems, and this seems to be

doable and interesting. Also high prio.

- I definitely want to go beyond context free stuff. This will come for sure.
- I like HPSG (head driven phrase structure grammar, google Carl Pollard and Ivan Sag) with AVMs (attribute value matrices).

- Right now the so-called lexicon consists of pattern numbers. The tool could be easily expanded to other levels, like events in tracks, DSPs etc.

9. Why ReNoam?

As you might have guessed, the 'Re' is in honour of Renoise, a damn good piece of software.

'Noam' is in honour of Noam Chomsky. Google him if you don't know him.

Peace, and don't stop creating, whatever field you're in.

Ralf Kibiger | f+d+k (fdk@kibiger.com)
2012-01-18