1 e + a: Language Specification

Valeria Starkova

Spring 2022

1 Introduction

This language allows drummers to notate drums efficiently, without using inconvenient intermediary software. As it can be tedious to write drum tabs in UIs that are not designed for this purpose, this language lets users write drum tabs in a simple and intuitive way: by utilizing the good old ”1 e and a” counting system.

A drum part’s structure is often constant throughout a song, with only slight variations across its parts (intro, verse, pre-chorus, chorus, bridge, etc.). This language also helps to avoid repetition by allowing the user to write one structure only once, then reuse it in other parts with slight modifications if necessary.

2 Design Principles

Writing sheet music for drums in this language is as easy as 1 e + a! The most important design principle in this language is that users can create and re-use patterns, bars, and beats that can be easily modified when writing the drum tabs. Further, the language has an online editor in which users can quickly write, test, and listen to the beats they wrote. Perhaps in the future, users will be able to share the source code for each song written in this programming language on a platform similar to GitHub, where they can keep track of their commits, collaborators, etc.

3 Examples

3.1 Example 1

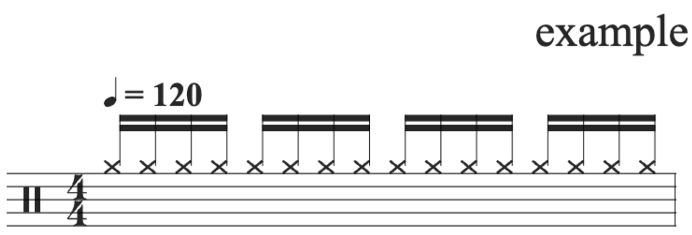
It is very easy to create a pattern, just by using the ”1 e and a” counting system. We put separators `|` to disambiguate the division of the beats. Since patterns alone cannot be rendered, we can put it inside a bar expression like this:

```
time: 4/4
division: 1/16
tempo: 120
title: example 1

pattern sixteenth_notes: [1 e + a | 2  e + a  | 3 e + a  | 4  e + a  |]

bar mybar:
  hh: [sixteenth_notes]

render: { mybar }
```



3.2 Example 2

Simple example of creating a two measure beat by providing two bar expressions in the render instruction.

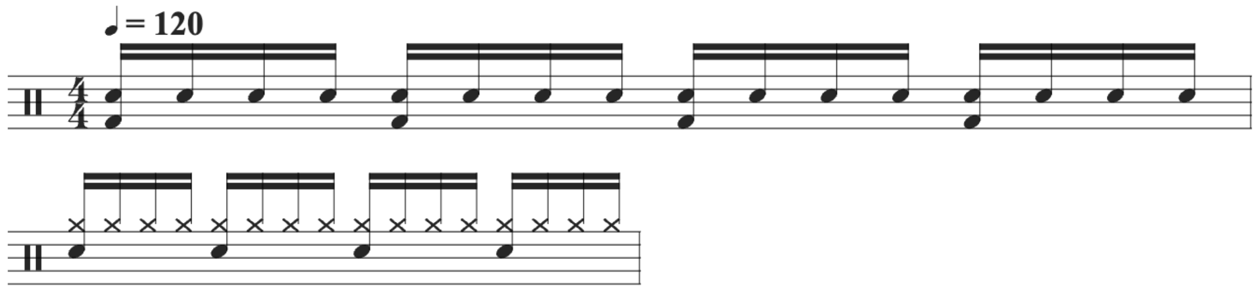
```
time: 4/4
division: 1/16
tempo: 120
title: example 2

pattern sixteenth_notes: [1 e + a | 2  e + a  | 3 e + a  | 4  e + a  |]

bar intro:
  sn: [sixteenth_notes]
  bd: [1 | 2 | 3 | 4|]

bar main:
  hh: [sixteenth_notes]
  sn: [1 | 2 | 3 | 4|]


render: { intro, main }
```



3.3 Example 3

Create a snippet by repeating the pre-defined measure and changing some of the drums and patterns on the last measure.

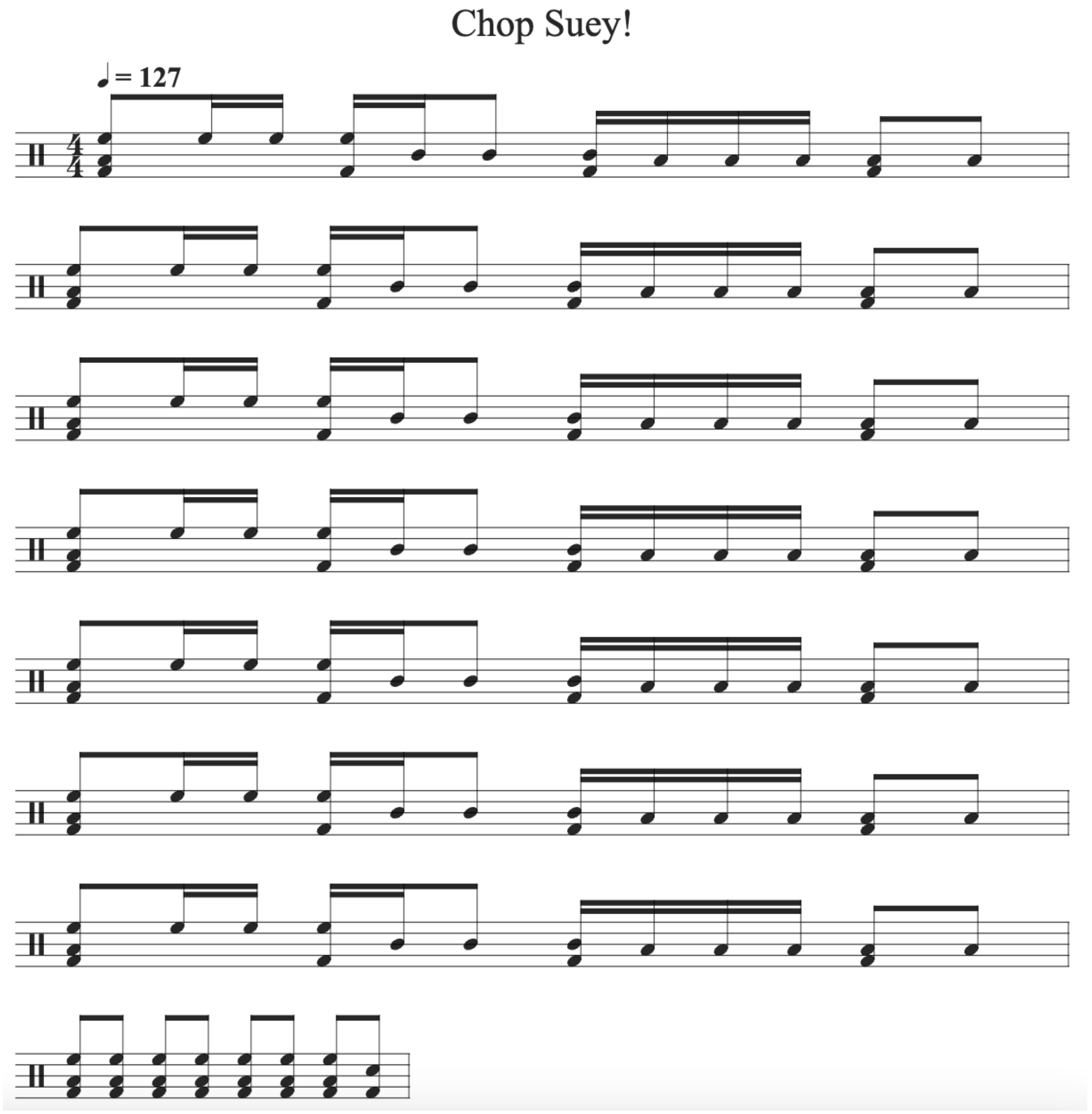
```
time: 4/4
division: 1/16
tempo: 127
title: Chop Suey!

pattern eigthNotes: [1 + | 2 + | 3 + | 4 + |]
pattern fourthNotes: [ 1 | 2 | 3 | 4 |]


bar intro_bar:
  bd: [ fourthNotes ]
  t1: [ 1 + a | 2 | | |]
  t2: [ |  e + | 3 | |]
  ft: [ 1 | | e + a | 4 + |]

snippet intro:
  change 8: intro_bar(8) {
    bd: [ eigthNotes ]
    sn: [  |  |  | + |]
    t1: [ 1 + | 2 + | 3 + | 4 |]
    t2: [ | | | |]
    ft: [ 1 + | 2 + | 3 + | 4 |]
  }

render: { intro }
```

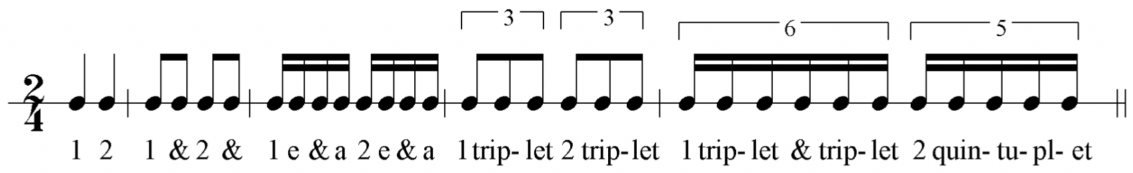


4 Language Concepts

There are two core concepts a user should understand in order to write programs in this language. First, how to notate different parts of the drumkit. Below is an example of a basic drumkit and the corresponding abbreviations we will use to write programs.

```
CC = Crash cymbal
HH = Hi-hat
RD = Ride cymbal
SN = Snare drum
T1 = High tom
T2 = Low tom
FT = Floor tom
BD = Bass drum
```

Secondly, a user should be familiar with the "1 e and a" counting system:



Now knowing this, we can assign patterns to each part of the drum kit. For example, in 4/4 time and 1/16 division, we can assign to SN (the snare drum) the pattern `1 e | 2 e | 3 e | 4 a |`, where | just symbolises division of the beat.

5 Syntax

A pattern is made up of notes. Patterns can be assigned to drums in a bar. Bars can be combined to create a snippet. Bars inside a snippet can be repeated or modified inside a repeat or change instruction.

```
    <num>           ::= x ∈ Z+
    <string>        ::= y ∈  a . . . z
    <varname>       ::= <string> | _

    <time>          ::= time: <num>/<num>
    <division>      ::= division: <num>/<num>
    <tempo>         ::= tempo: <num>
    <title>         ::= title: <string>
    <settings>      ::= <time> <division> <tempo> <title>

    <sep>               ::= |
    <note>              ::= <num> | e | + | a
    <notes>             ::= <note>+
    <notes_with_sep>    ::= <notes><sep>
    <pattern>           ::= <notes_with_sep>+
    <pattern_expr>      ::= pattern <varname>: [<pattern>]

    <drums>                 := cc | hh | rd | sn | t1 | t2 | ft | bd | r
    <drum_pattern_var>      ::= <drums>: [<varname>]
    <drum_pattern_notes>    ::= <drums>: [<pattern>]
    <drum_pattern>          ::= <drum_pattern_var> | <drum_pattern_notes>
    <many_drum_patterns>    ::= <drum_pattern>+
    <bar_expr>              ::= bar <varname>: <many_drum_patterns>
```

```
<repeat>                ::= repeat <num>: {<varname>,+}
<change_every>          ::= every <num>
<change_many>           ::= <num>,+
<change_option>         ::= <change_every> | <change_many>
<change>                ::= change <num>:
                            <varname> (<change_option>){
                                <drum_pattern>+
                            }
<snippet_data>          ::= <repeat> | <change>
<snippet_expr>          ::= snippet <varname>: <snippet_data>+

<render>                ::= render: {<varname>,+}

<data_expr>             ::= <pattern_expr> | <bar_expr> | <snippet_expr>
<many_data_expr>        ::= <data_expr>+
<expr>                  ::= <settings>
                            <many_data_expr>
                            <render>
```

6 Semantics

Syntax	Abstract Syntax	Type	Meaning
time: 4/4	Time of uint8 * uint8	Settings	Top number tells how many beats should be in one measure, and the bottom number tells what value of note should get the beat.
division: 1/16	Division of uint8 * uint8	Settings	The minimum value that notes can divide into
tempo: 120	Tempo of uint8	Settings	The speed of the music to be played
title: My Title	Title of string	Settings	Title of the music
1 e + a |	Num of int | E | And | A | Sep	Note	A way of writing notes: 1 specifies a strong beat, e is a weak beat, + is a strong beat, and a is a weak beat. Any of these notes (or even no notes at all), make up one beat. As defined by the time signature above, if we combine 4 beats then we get one full measure (ex: 1 e + a 2 e + a 3 e + a 4 e + a). We put a separator | after each beat to disambiguate the division of the beats: 1 e + a | 2 e + a | 3 e + a | 4 e + a |
sn	Drum of | CC | RD | HH | SN | T1 | T2 | FT | BD | Rest	Drum	Denotes a drum
pattern varname: [1 | 2 e + a | 3 | 4 |]	Pattern of PatternName * (Note list)	Pattern	A pattern is a data type that contains one measure of notes as defined by the time signature.
hh: [string]	Drum * PatternName	DrumPatternVar	Indirect way of assigning a pattern referenced by it's variable name to a drum
hh: [1 | 2 | 3 | 4]	Drum * (Note list)	DrumPatternNotes	Direct way of assigning a pattern to a drum
bar barName: hh: [1 | 2 | 3 | 4 |] sn: [somePattern]	Bar of BarName * (DrumPatternVar list * DrumPatternNotes list)	Bar	A bar is a data type that contains one or more pattern variable names or patterns assigned to drum instruments.

Syntax	Abstract Syntax	Type	Meaning
repeat 2: oneBar, twoBar	Repeat of int * (BarName list)	SnippetData	repeat instruction will repeat one or more bars a given number of times.
change 5: barName (1,2) { cc: [1 | 2 | 3 | 4 |] } or change 4: barName (every 2) { cc: [patternName]}	RepeatChange of int * BarName * RepeatOption * (DrumPattern list)	SnippetData	repeat instructions repeat and also change a given bar as specified by the option inside parens (every N-th bar or a given list of integers). The data to change the bar is written inside curly brackets, which are just drum-¿pattern assignments.
snippet mysnippet: change ... repeat ...	Snippet of SnippetName * (SnippetData list)	Snippet	Snippet is a data type that contains one or more combinations of the two possible instructions: repeat instruction, and change instruction.
render: { someBar, someSnippet, someBar, ...}	string list	Render	The render keyword will render (or in other words, evaluate) the variables specified inside curly brackets after the semicolon :. The render keyword must always come the very last in the program.

i. A primitive value is a note defined by a counting value such as `1`, `e`, `+`, `a`. We put separators `|` after each beat.

ii. Notes `1`, `e`, `+`, `a` are combined to create a `pattern`:

```
pattern: [1 + | 2 e | 3 | 4 e + a|]
```

A `bar` can be created by applying various `pattern`s to different parts of the drumset (`hh`, `sn`, `bd`, etc).

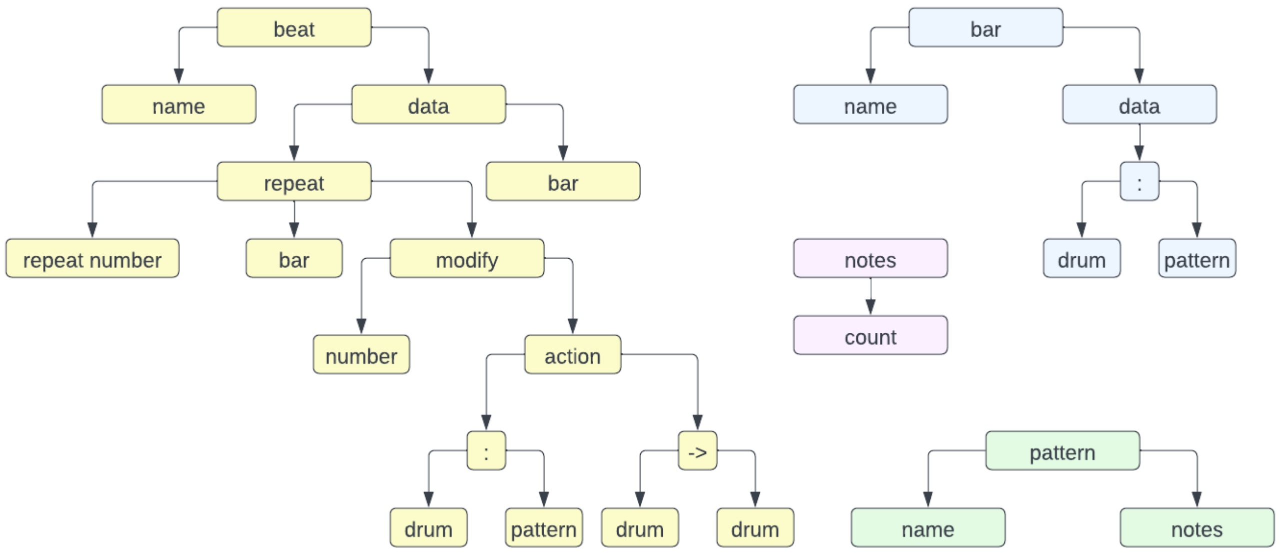
```
bar myBar:
    sn: [1 + | 2 e | 3 | 4 e + a|]
    bd: [patternVar]
```

And a `snippet` can be created by combining and/or repeating multiple `bar`s. There are 2 instructions which repeat a given bar(s): repeat without change and repeat with change. Repeat without change just repeats one or more bars N times. Repeat with change (or simply `change`) instruction changes one given bar on top of repeating it.

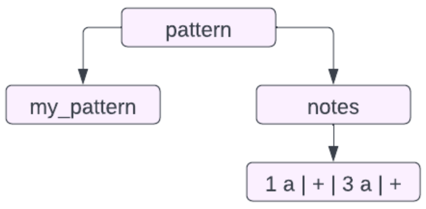
```
snippet mySnippet:
    repeat 2: {myBarOne, MyBarTwo}
    change 6: MyBarTwo (every 2) {
        bd: [different_pattern]
    }
```

```
change 16: MyBarTwo (1,6,8,9) {
    cc: [different_pattern]
    cc: [1|2|3|4|]
}
```

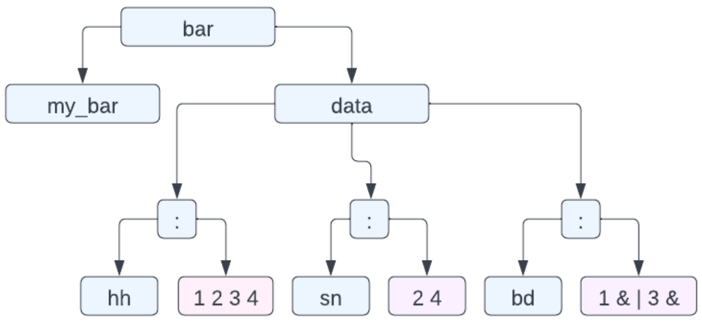
iii. Diagram representation of my program:



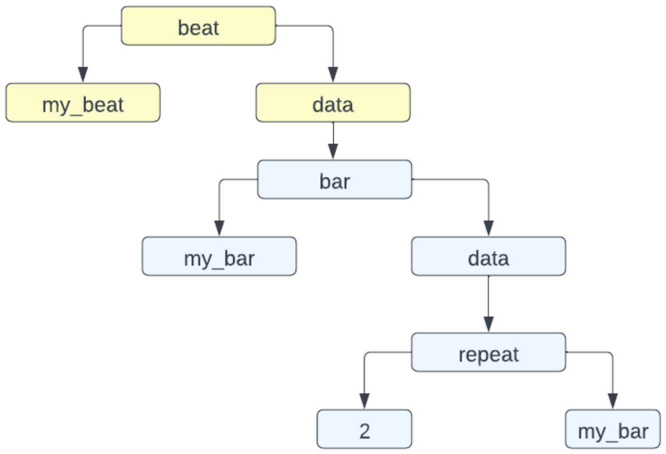
iv. AST for example 1:



AST for example 2:



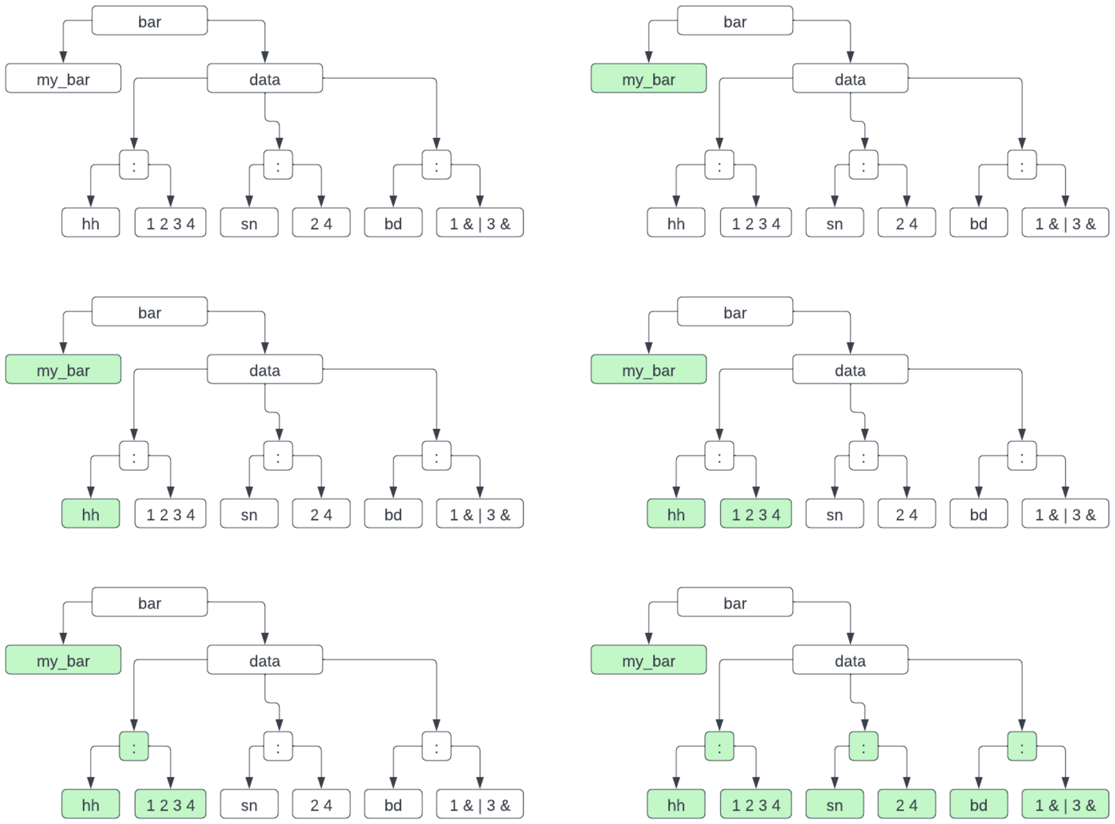
AST for example 3: (note that I've omitted the implementation of "my bar" as it is similar to example 2)

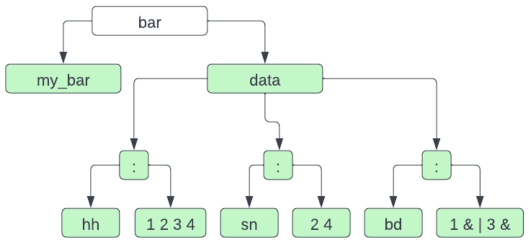


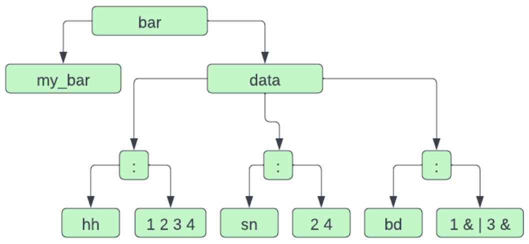
v. A. The programs in 1 e + a do not read any input.

B. The output is a txt file which contains abc notation code.

C. Evalution of the program illustrated by example 2. First we evaluate the left branch, which is name of the bar. The we evaluate the right branch which contains actions as children. Each action has a drum name as the left node and a pattern as the right node. We first evaluate the drum, and then the pattern and assign the pattern to the drum. We continue doing so until we evaluate every action.







7 Remaining Work

1. Add a metronome feature to the webapp
2. Add a count-in feature to the webapp
3. Add a
4. Add support for time signatures other than 4/4
5. Add support for division of beats other that 1/16
6. Add constant variables