Drum Machine Language

ZFAC230

April 2021

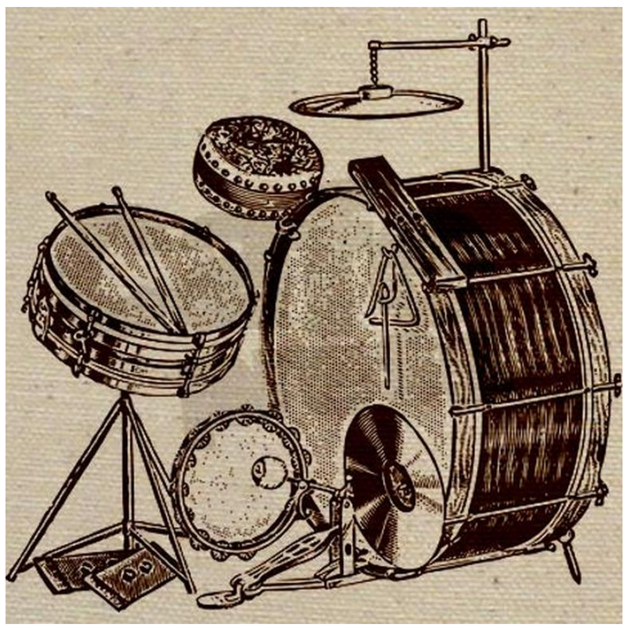

Figure 1: Sketch of a 1920's drum kit from https://forums.ledzeppelin.com/topic/25321-the-evolution-of-the-drum-set/page/6/

1 Introduction

This report discusses the domain specific language, Drum Machine Language(DML). DML is a domain specific language for creating drum beats and patterns, inspired by simple drum machines such as the pocket operator series. A series of drum sounds are provided with functionality to produce and loop patterns, combine patterns into songs, set the bpm, and select the volume and note length of different drums. The goal of this language is to be easily used regardless of the user's knowledge of music theory and allow for the production of interesting and varied drum tracks using simple code that is intuitive to write.

A combination of 10 different drums can be used in patterns of varying length. Patterns are created inside tracks, which can be looped or played in succession. The tracks are dynamic structures mapping positions in the sequence to sounds. Multiple tracks can be combined into songs. Included in this report is a brief description of the language features, complete lists of internal constructors and ESOS rules, and the parser used to translate external syntax to internal syntax.

2 Informal Language Specification

This section provides an informal overview of the language features included in DML. DML includes a variety of data structures and fundamental language features for control flow, arithmetic and logic. These fundamentals allow for more creative use of the domain specific features. The domain specific features are those related to the production drum machine style tracks.

DML can perform arithmetic and logic operations with operator priority and bracketing. Control flow is implemented with if, when and for statements. Songs behave like functions and could be used as a function if required. Drum patterns are constructed by implementing and running tracks. These are internal data structures to handle the construction of tracks. The combination of domain specific features that execute code written using Java's Midi libraries and control flow elements are used to produce drum patterns.

2.1 Data Types and Structures

DML uses the following data structures:

- Integer
- Real
- Boolean
- String

- Array. Arrays are implemented as a __flexArray. This can be used to store Integers. Arrays can be appended by adding an element or another array, written as:

  ```
  element = 5;
  myArray = [1,2,3,4];
  myArray = myArray + element;
  ```

  Array elements can be retrieved by index and the length of an array can be returned.

  ```
  element = myArray get index;
  ```

- Drum

  The available sounds are given as the following ‘drum’ keywords

  ```
  snare, bass, tom1, tom2, floor_tom, crash, ride,
  hi_hat_open, hi_hat_closed, hi_hat_pedal
  ```

- Track, A dynamic collection of sounds represented by a hashmap.

2.2 Arithmetic and Logical Operators

All assignment statements use ‘;’ as the and also operator.

```
var1 = 1 * 6 +(5 - 2); var2 = 3 - 2; var3 = var2 + var1;
```

Large expressions are handles using the following operators:

Priority	Operator	Description
1	AND	boolean operators
2	OR XOR	
3	NOT	
4	<>==<=>=! =	logic operators
5	+−	addition and subtraction
6	∗/	multiplication and division
7	− len	negation and length
8	∗∗ get	exponential and get
9	− − ++	increment and decrement
10	real() int()	casting
11	( )	parenthesis

2.3 Casting

Mixed arithmetic between real numbers and integers is not supported. To allow users to work around this casting has been implemented.

```
a = 5; b= 12.0
c = 12.0 - real(5);
```

By casting integers into real numbers, variables defined as real can be used in expressions with integers.

2.4 Control Sequences

Selection and iteration are included in the form of if statements, while loops and food loops. If and while loops take an expression to determine whether the next instruction. The if loop uses the structure:

```
if (a > 5) {a = a -1} else {a = a + 1}
```

While loops have a similar structure:

```
while (a > 5) {a = a - 1}
```

For loops are similar to those in Java and C, taking an assigned variable, a condition and a statement executed each cycle at the end of the code block. The assigned variable can the be accessed inside the loop.

```
for (i = 1, i < 5, i++) {a = a ** 2}
```

2.5 Domain Specific Features

To support the creation of drum loops, various domain specific types and functions are included. The most important of which are tracks. Ten different percussion sounds are provided as keywords. The volume and duration of each sound can be set by the user, or has an existing default value. To play sounds, tracks must be constructed. The tracks in DML are not equivalent to midi tracks. Tracks are an internal data structure that are instantiated when a user generates a new track. Tracks consist of a limitless series of positions, which are 4 ticks apart. Each tick is equivalent to a fraction1/16 note, so each position denotes a crutchet. The length of beats of a particular drum can be altered to be a different length. Internally, tracks are data structures representing two mappings. Each track is mapped to a midi sequence and to a hashmap. The hashmap is a mapping between note positions and sounds. This means that when a track is run, the mapping of note positions is used to generate events inside the sequence mapped to the track ID.

Before generating tracks, users should be able to set the beats per minute of tracks, and the volume and duration of each sound.

By default the duration of each sound is 4 ticks(1 quavers), except for the crash cymbal which plays for 8 ticks. The volume of each sound can also be set by the user, this is equivalent to the velocity in midi.

```
drumVolume(bass, 120);
drumDuration(bass, 6);
```

To play sounds, they first need to be added to a track. This position is placed in square brackets.

```
maketrack track01{"trackID"};
setSound track01[0] snare;
setSound track01[4] bass;
```

The trackID is the ID referenced by the internal data structure, this is what the track variable dereferences to.

Songs, which behave like functions, can combine tracks.

```
song my_song {play_track track01; play_track track02;}
```

These methods allow the user to construct tracks, loop them, or join them with other tracks to produce drum tracks equivalent to those generated by drum machines.

3 Internal Syntax Constructors and Arities

This sections lists every internal constructor, it arity and what it does.

- assign 2 , binds _1 to _2 in variable map
- deref 1, retrieves value from _1 in variable map
- gt 2, computes _1 ¿ _2
- ge 2, computes _1 ¿= _2
- eq 2, computes _1 == _2
- ne 2, computes _1 != _2
- lt 2, computes _1 ¡ _2
- le 2, computes _1 ¡= _2
- sub 2, computes _1 - _2

- add 2, computes _1 + _2
- add 2, appends the _2 to the end of _1
- mult 2, computes _1 * _2
- div 2, computes $\frac{\_1}{\_2}$
- exp 2, computes _1 ** _2
- inc 1, computes _1 + 1
- dec 1, computes _1 - 1
- neg 1, negates _1
- get 2, retrieves value _1 from indesx _2 in a __flexArray
- len 1, retrieves number of elements contained in _1
- AND 2, computes truth value _1 AND _2
- OR 2, computes truth value _1 OR _2
- XOR 2, computes truth value _1 XOR _2
- NOT 1, computes truth value NOT _1
- castReal 1, takes value of _1 and retrieves a __Real64
- castInt 1, takes value of _1 and retrieves a __Int32
- seq 2, sequence _1 then _2
- if 3, if _1 then _2 else _3
- while 2, while _1 do _2
- for 4, assign _1, while _2 do _4 then _3
- func 2, binds _1 to _2 in variable map
- execute 1, retrieves statement from _1 in variable map
- setBpm 1, calls __user(bpm, _1)
- drumVolume 1, calls __user(drumVolume, _1)
- drumDuration 1, calls __user(drumDuration, _1)
- makeTrack 2, calls __user(track, _1, _2), binds _1 to _2 in variable map
- setSound 3, calls __user(setSound, _1, _2, _3)
- runTrack 1, calls __user(runTrack, _1)
- print 1, calls __user(printm _1)

4 eSOS Rules

This section contains all the eSOS rules used in my internal syntax.

4.1 Source

```
    -assignInt
_n |> __int32(_)
---
assign(_name, _n), _sig -> __done, __put(_sig, _name, _n)

-assignReal
_n |> __real64(_)
---
assign(_name, _n), _sig -> __done, __put(_sig, _name, _n)

-assignString
_s |> __string(_)
---
assign(_name, _s), _sig -> __done, __put(_sig, _name, _s)

-assignBool
_b |> __boolean(_)
---
assign(_name, _b), _sig -> __done, __put(_sig, _name, _b)

-assignArray
__termRoot(_a) |> __flexArray
---
assign(_name, _a), _sig -> __done, __put(_sig, _name, _a)

-assignResolve
_E, _sig -> _I, _sigP
---
assign(_name,_E), _sig -> assign(_name, _I), _sigP

-variable
__get(_sig, _R) |> _Z
---
deref(_R),_sig -> _Z, _sig


-appendArrayArray
__termRoot(_a1) |> __flexArray __termRoot(_a2) |> __flexArray
---
add(_a1, _a2), _sig -> __append(_a1, _a2), _sig
```

```
-appendArrayElement
__termRoot(_a) |> __flexArray
---
add(_a, _x), _sig -> add(_a, __flexArray(_x)), _sig

-getArrayElement
__termRoot(_a) |> __flexArray _i |> __int32(_)
---
get(_a, _i), _sig -> __get(_a, _i), _sig

-getArrayElementRight
__termRoot(_a) |> __flexArray _E, _sig -> _I, _sigP
---
get(_a, _E), _sig -> get(_a, _I), _sigP

-getArrayElementLeft
_E, _sig -> _I, _sigP
---
get(_E, _x), _sig -> get(_I, _x), _sigP

-getArraySize
__termRoot(_a) |> __flexArray
---
len(_a), _sig -> __size(_a), _sig

-getArraySizeResolve
_E, _sig -> _I, _sigP
---
len(_E), _sig -> len(_I), _sigP

-gtInt
_n1 |> __int32(_) _n2 |> __int32(_)
---
gt(_n1, _n2),_sig ->  __gt(_n1, _n2),_sig

-gtRightInt
_n |> __int32(_) _E2, _sig -> _I2,_sigP
---
gt(_n, _E2),_sig -> gt(_n, _I2), _sigP

-gtReal
_n1 |> __int32(_) _n2 |> __real64(_)
---
gt(_n1, _n2),_sig ->  __gt(_n1, _n2),_sig
```

```
-gtRightReal
_n |> __real64(_) _E2, _sig -> _I2,_sigP
---
gt(_n, _E2),_sig -> gt(_n, _I2), _sigP

-gtLeft
_E1, _sig -> _I1, _sigP
---
gt(_E1, _E2),_sig -> gt(_I1, _E2), _sigP

-geInt
_n1 |> __int32(_) _n2 |> __int32(_)
---
ge(_n1, _n2),_sig ->  __ge(_n1, _n2),_sig

-geRightInt
_n |> __int32(_)  _E2, _sig -> _I2,_sigP
---
ge(_n, _E2),_sig -> ge(_n, _I2), _sigP

-geReal
_n1 |> __real64(_) _n2 |> __real64(_)
---
ge(_n1, _n2),_sig ->  __ge(_n1, _n2),_sig

-geRightReal
_n |> __real64(_)  _E2, _sig -> _I2,_sigP
---
ge(_n, _E2),_sig -> ge(_n, _I2), _sigP

-eqInt
_n1 |> __int32(_) _n2 |> __int32(_)
---
eq(_n1, _n2),_sig ->  __eq(_n1, _n2),_sig

-eqRightInt
_n |> __int32(_)  _E2, _sig -> _I2,_sigP
---
eq(_n, _E2),_sig -> eq(_n, _I2), _sigP

-eqReal
_n1 |> __real64(_) _n2 |> __real64(_)
---
eq(_n1, _n2),_sig ->  __eq(_n1, _n2),_sig

-eqRightReal
```

```
_n |> __real64(_)  _E2, _sig -> _I2,_sigP
---
eq(_n, _E2),_sig -> eq(_n, _I2), _sigP

-eqLeft
_E1, _sig -> _I1, _sigP
---
eq(_E1, _E2),_sig -> eq(_I1, _E2), _sigP

-ne
_n1 |> __int32(_) _n2 |> __int32(_)
---
ne(_n1, _n2),_sig ->  __ne(_n1, _n2),_sig

-neRight
_n |> __int32(_)  _E2, _sig -> _I2,_sigP
---
ne(_n, _E2),_sig -> ne(_n, _I2), _sigP

-neLeft
_E1, _sig -> _I1, _sigP
---
ne(_E1, _E2),_sig -> ne(_I1, _E2), _sigP

-ltInt
_n1 |> __int32(_) _n2 |> __int32(_)
---
lt(_n1, _n2),_sig ->  __lt(_n1, _n2),_sig

-ltRightInt
_n |> __int32(_)  _E2, _sig -> _I2,_sigP
---
lt(_n, _E2),_sig -> lt(_n, _I2), _sigP

-ltReal
_n1 |> __real64(_) _n2 |> __real64(_)
---
lt(_n1, _n2),_sig ->  __lt(_n1, _n2),_sig

-ltRightReal
_n |> __real64(_)  _E2, _sig -> _I2,_sigP
---
lt(_n, _E2),_sig -> lt(_n, _I2), _sigP

-ltLeft
_E1, _sig -> _I1, _sigP
```

```
---
lt(_E1, _E2),_sig -> lt(_I1, _E2), _sigP

-leInt
_n1 |> __int32(_) _n2 |> __int32(_)
---
le(_n1, _n2),_sig ->  __le(_n1, _n2),_sig

-leRightInt
_n |> __int32(_)  _E2, _sig -> _I2,_sigP
---
le(_n, _E2),_sig -> le(_n, _I2), _sigP

-leReal
_n1 |> __real64(_) _n2 |> __real64(_)
---
le(_n1, _n2),_sig ->  __le(_n1, _n2),_sig

-leRightReal
_n |> __real64(_)  _E2, _sig -> _I2,_sigP
---
le(_n, _E2),_sig -> le(_n, _I2), _sigP

-subInt
_n1 |> __int32(_) _n2 |> __int32(_)
---
sub(_n1, _n2), _sig ->  __sub(_n1, _n2),_sig

-subRightInt
_n |> __int32(_) _E2,_sig -> _I2,_sigP
---
sub(_n, _E2),_sig -> sub(_n, _I2), _sigP

-subReal
_n1 |> __real64(_) _n2 |> __real64(_)
---
sub(_n1, _n2), _sig ->  __sub(_n1, _n2),_sig

-subRightReal
_n |> __real64(_) _E2,_sig -> _I2,_sigP
---
sub(_n, _E2), _sig -> sub(_n, _I2), _sigP

-subLeft
_E1,_sig -> _I1,_sigP
---
```

```
sub(_E1, _E2),_sig -> sub(_I1, _E2), _sigP

-addInt
_n1 |> __int32(_) _n2 |> __int32(_)
---
add(_n1, _n2),_sig -> __add(_n1, _n2), _sig

-addRightInt
_n |> __int32(_) _E2,_sig -> _I2,_sigP
---
add(_n, _E2),_sig -> add(_n, _I2), _sigP

-addReal
_n1 |> __real64(_) _n2 |> __real64(_)
---
add(_n1, _n2),_sig -> __add(_n1, _n2), _sig

-addRightReal
_n |> __real64(_) _E2,_sig -> _I2,_sigP
---
add(_n, _E2),_sig -> add(_n, _I2), _sigP

-addLeft
_E1,_sig -> _I1,_sigP
---
add(_E1, _E2), _sig -> add(_I1, _E2), _sigP

-multInt
_n1 |> __int32(_) _n2 |> __int32(_)
---
mult(_n1, _n2),_sig -> __mul(_n1, _n2), _sig

-multRightInt
_n |> __int32(_) _E2, _sig -> _I2, _sigP
---
mult(_n, _E2), _sig -> mult(_n, _I2), _sigP

-multReal
_n1 |> __real64(_) _n2 |> __real64(_)
---
mult(_n1, _n2),_sig -> __mul(_n1, _n2), _sig

-multRightReal
_n |> __real64(_) _E2, _sig -> _I2, _sigP
---
mult(_n, _E2), _sig -> mult(_n, _I2), _sigP
```

```
-multLeft
_E1, _sig -> _I1, _sigP
---
mult(_E1, _E2), _sig -> mult(_I1, _E2), _sigP

-divInt
_n1 |> __int32(_) _n2 |> __int32(_)
---
div(_n1, _n2), _sig -> __div(_n1, _n2), _sig

-divRightInt
_n |> __int32(_) _E2, _sig -> _I2, _sigP
---
div(_n, _E2), _sig -> div(_n, _I2), _sigP

-divReal
_n1 |> __real64(_) _n2 |> __real64(_)
---
div(_n1, _n2), _sig -> __div(_n1, _n2), _sig

-divRightInt
_n |> __real64(_) _E2, _sig -> _I2, _sigP
---
div(_n, _E2), _sig -> div(_n, _I2), _sigP

-divLeft
_E1, _sig -> _I1, _sigP
---
div(_E1, _E2), _sig -> div(_I1, _E2), _sigP

-expInt
_n |> __int32(_) _exponent |> __int32(_)
---
exp(_n, _exponent), _sig -> __exp(_n, _exponent), _sig

-expIntRight
_n |> __int32(_) _E, _sig -> _I, _sigP
---
exp(_n, _E), _sig -> exp(_n, _I), _sigP

-expIntLeft
_E, _sig -> _I, _sigP
---
exp(_E, _exponent), _sig -> exp(_I, _exponent), _sigP
```

```
-incrementInt
_n1 |> __int32(_)
---
inc(_n1), _sig -> __add(_n1, 1) , _sig

-incrementReal
_n1 |> __real64(_)
---
inc(_n1), _sig -> __add(_n1, 1.0), _sig

-incrementResolve
_E, _sig -> _I, _sigP
---
inc(_E), _sig -> inc(_I), _sigP

-decrementInt
_n1 |> __int32(_)
---
dec(_n1), _sig -> __sub(_n1, 1) , _sig

-decrementReal
_n1 |> __real64(_)
---
dec(_n1), _sig -> __sub(_n1, 1.0), _sig

-decrementResolve
_E, _sig -> _I, _sigP
---
dec(_E), _sig -> dec(_I), _sigP

-negateInt
_n |> __int32(_)
---
neg(_n), _sig -> __sub(0, _n), _sig

-negateReal
_n |> __real64(_)
---
neg(_n), _sig -> __sub(0.0, _n), _sig

-negateIntResolve
_E, _sig -> _I, _sigP
---
neg(_E), _sig -> neg(_I), _sigP

-and
```

```
_Bool1 |> __boolean(_) _Bool2 |> __boolean(_)
---
AND(_Bool1, _Bool2), _sig -> __and(_Bool1, _Bool2), _sig

-andRight
_Bool |> __boolean(_) _ERight, _sig -> _BoolRight, _sigP
---
AND(_Bool, _ERight), _sig -> AND(_Bool, _BoolRight), _sigP

-andLeft
_ERightLeft, _sig -> _ILeft, _sigP
---
AND(_ELeft, _ERight), _sig -> AND(_ILeft, _ERight), _sigP

-or
_Bool1 |> __boolean(_) _Bool2 |> __boolean(_)
---
OR(_Bool1, _Bool2), _sig -> __or(_Bool1, _Bool2), _sig

-orRight
_Bool |> __boolean(_) _ERight, _sig -> _BoolRight, _sigP
---
OR(_Bool, _ERight), _sig -> OR(_Bool, _BoolRight), _sigP

-orLeft
_ELeft, _sig -> _ILeft, _sigP
---
OR(_ELeft, _ERight), _sig -> OR(_ILeft, _ERight), _sigP

-not
_Bool |> __boolean(_)
---
NOT(_Bool), _sig -> __not(_Bool), _sig

-notResolve
_ERight, _sig -> _I, _sigP
---
NOT(_ERight), _sig -> NOT(_I), _sigP

-xor
_Bool1 |> __boolean(_) _Bool2 |> __boolean(_)
---
XOR(_Bool1, _Bool2), _sig -> __xor(_Bool1, _Bool2), _sig

-xorRight
_Bool |> __boolean(_) _ERight, _sig -> _BoolRight, _sigP
```

```
---
XOR(_Bool, _ERight), _sig -> XOR(_Bool, _BoolRight), _sigP

-xorLeft
_ELeft, _sig -> _ILeft, _sigP
---
XOR(_ELeft, _ERight), _sig -> XOR(_ILeft, _ERight), _sigP
-castReal
_n |> __int32(_)
---
castReal(_n), _sig -> __user(castReal, _n), _sig

-castRealResolve
_E1, _sig -> _I1, _sigP
---
castReal(_E1), _sig -> castReal(_I1), _sigP

-castInt
_n |> __real64(_)
---
castInt(_n), _sig -> __user(castInt, _n), _sig

-castIntResolve
_E1, _sig -> _I1, _sigP
---
castInt(_E1), _sig -> castInt(_I1), _sigP
-sequenceDone
---
seq(__done, _C), _sig -> _C, _sig

-sequence
_C1, _sig -> _C1P, _sigP
---
seq(_C1, _C2), _sig -> seq(_C1P, _C2), _sigP

-ifTrue
---
if(True, _C1, _C2),_sig -> _C1, _sig

-ifFalse
---
if(False, _C1, _C2),_sig -> _C2,_sig

-ifResolve
_E, _sig ->_EP, _sigP
---
```

```
if(_E,_C1,_C2),_sig -> if(_EP, _C1, _C2), _sigP

-while
---
while(_E, _C),_sig -> if(_E, seq(_C, while(_E,_C)), __done), _sig

-for
_v |> assign(_,__int32(_))
---
for(_v, _E, _i, _C), _sig -> seq(_v, while(_E, seq(_C, _i))), _sig

-subroutine
---
func(_name, _S), _sig -> __done, __put(_sig, _name, _S)

-runSubroutine
---
execute(_func), _sig -> deref(_func), _sig
-setBpm
_n |> __int32(_)
---
setBpm(_n), _sig -> __user(bpm, _n), _sig

-setBpmResolve
_E, _sig -> _I, _sigP
---
setBpm(_E), _sig -> setBpm(_I), _sigP

-drumVolume
_n |> __int32(_)
---
drumVolume(_drum, _n), _sig -> __user(drumVolume, _drum, _n), _sig

-drumVolumeResolve
_E, _sig -> _I, _sigP
---
drumVolume(_drum, _E), _sig -> drumVolume(_drum, _I), _sigP

-drumDuration
_n |> __int32(_)
---
drumDuration(_drum, _n), _sig -> __user(drumDuration, _n), _sig

-drumDurationResolve
_E, _sig -> _I, _sigP
---
```

```
drumDuration(_drum, _E), _sig -> drumDuration(_drum, _I), _sigP

-MakeTrack
_title |> __string(_)
---
makeTrack(_name, _title), _sig -> __user(track, _name, _title), __put(_sig, _name, _title)

-MakeTrackResolveTitle
_E, _sig -> _s, _sigP
---
makeTrack(_name, _E), _sig -> makeTrack(_name, _s), _sigP

-setSoundSnare
_track |>__string(_) _pos |> __int32(_) _sound |> snare
---
setSound(_track, _pos ,_sound), _sig -> __user(setSound, _track, _pos, _sound), _sig

-setSoundBass
_track |>__string(_) _pos |> __int32(_) _sound |> bass
---
setSound(_track, _pos ,_sound), _sig -> __user(setSound, _track, _pos, _sound), _sig

-setSoundTom1
_track |>__string(_) _pos |> __int32(_) _sound |> tom1
---
setSound(_track, _pos ,_sound), _sig -> __user(setSound, _track, _pos, _sound), _sig

-setSoundTom2
_track |>__string(_) _pos |> __int32(_) _sound |> tom2
---
setSound(_track, _pos ,_sound), _sig -> __user(setSound, _track, _pos, _sound), _sig

-setSoundFloorTom
_track |>__string(_) _pos |> __int32(_) _sound |> floorTom
---
setSound(_track, _pos ,_sound), _sig -> __user(setSound, _track, _pos, _sound), _sig

-setSoundCrash
_track |>__string(_) _pos |> __int32(_) _sound |> crash
---
setSound(_track, _pos ,_sound), _sig -> __user(setSound, _track, _pos, _sound), _sig

-setSoundRide
_track |>__string(_) _pos |> __int32(_) _sound |> ride
---
setSound(_track, _pos ,_sound), _sig -> __user(setSound, _track, _pos, _sound), _sig
```

```
-setSoundOpenHiHatOpen
_track |>__string(_) _pos |> __int32(_) _sound |> hiHatOpen
---
setSound(_track, _pos ,_sound), _sig -> __user(setSound, _track, _pos, _sound), _sig

-setSoundhiHatClosed
_track |>__string(_) _pos |> __int32(_) _sound |> hiHatClosed
---
setSound(_track, _pos ,_sound), _sig -> __user(setSound, _track, _pos, _sound), _sig

-setSoundhiHatPedal
_track |>__string(_) _pos |> __int32(_) _sound |> hiHatPedal
---
setSound(_track, _pos ,_sound), _sig -> __user(setSound, _track, _pos, _sound), _sig

-setSoundResolveSound
_track |> __string(_) _pos |> __int32(_) _E, _sig -> _I, _sigP
---
setSound(_track, _pos, _E), _sig -> setSound(_track, _pos, _I), _sigP

-setSoundResolvePos
_track |> __string(_) _E, _sig -> _I, _sigP
---
setSound(_track, _E, _x), _sig -> setSound(_track, _I, _x), _sigP

-setSoundResolveTrack
_E, _sig -> _I, _sigP
---
setSound(_E, _x, _y), _sig -> setSound(_I, _x, _y), _sigP

-setSoundDerefTrack
---
setSound(_track, _x, _y), _sig -> setSound(deref(_track), _x, _y), _sig

-runTrack
_track |> __string(_)
---
runTrack(_track), _sig -> __user(runTrack, _track), _sig

-runTrackResolve
_E, _sig -> _s, _sigP
---
runTrack(_E), _sig -> runTrack(_s), _sigP

-runTrackDeref
```

```
---
runTrack(_track), _sig -> runTrack(deref(_track)), _sig

-print
_s |> __string(_)
---
print(_s), _sig -> __user("print",_s), _sig
```

4.2 Typeset

[assignInt] $$\frac{n \triangleright \textit{\_\_int32}(\_)}{\langle \mathsf{assign}(name, n), \sigma\rangle \rightarrow \langle \textit{\_\_done}, \textit{\_\_put}(\sigma, name, n)\rangle}$$

[assignReal] $$\frac{n \triangleright \textit{\_\_real64}(\_)}{\langle \mathsf{assign}(name, n), \sigma\rangle \rightarrow \langle \textit{\_\_done}, \textit{\_\_put}(\sigma, name, n)\rangle}$$

[assignString] $$\frac{s \triangleright \textit{\_\_string}(\_)}{\langle \mathsf{assign}(name, s), \sigma\rangle \rightarrow \langle \textit{\_\_done}, \textit{\_\_put}(\sigma, name, s)\rangle}$$

[assignBool] $$\frac{b \triangleright \textit{\_\_boolean}(\_)}{\langle \mathsf{assign}(name, b), \sigma\rangle \rightarrow \langle \textit{\_\_done}, \textit{\_\_put}(\sigma, name, b)\rangle}$$

[assignArray] $$\frac{\textit{\_\_termRoot}(a) \triangleright \textit{\_\_flexArray}}{\langle \mathsf{assign}(name, a), \sigma\rangle \rightarrow \langle \textit{\_\_done}, \textit{\_\_put}(\sigma, name, a)\rangle}$$

[assignResolve] $$\frac{\langle E, \sigma\rangle \rightarrow \langle I, \sigma'\rangle}{\langle \mathsf{assign}(name, E), \sigma\rangle \rightarrow \langle \mathsf{assign}(name, I), \sigma'\rangle}$$

[leInt] $$\frac{n_1 \triangleright \textit{\_\_int32}(\_) \quad n_2 \triangleright \textit{\_\_int32}(\_)}{\langle \mathsf{le}(n_1, n_2), \sigma\rangle \rightarrow \langle \textit{\_\_le}(n_1, n_2), \sigma\rangle}$$

[leRightInt] $$\frac{n \triangleright \textit{\_\_int32}(\_) \quad \langle E_2, \sigma\rangle \rightarrow \langle I_2, \sigma'\rangle}{\langle \mathsf{le}(n, E_2), \sigma\rangle \rightarrow \langle \mathsf{le}(n, I_2), \sigma'\rangle}$$

[leReal] $$\frac{n_1 \triangleright \textit{\_\_real64}(\_) \quad n_2 \triangleright \textit{\_\_real64}(\_)}{\langle \mathsf{le}(n_1, n_2), \sigma\rangle \rightarrow \langle \textit{\_\_le}(n_1, n_2), \sigma\rangle}$$

[leRightReal] $$\frac{n \triangleright \textit{\_\_real64}(\_) \quad \langle E_2, \sigma\rangle \rightarrow \langle I_2, \sigma'\rangle}{\langle \mathsf{le}(n, E_2), \sigma\rangle \rightarrow \langle \mathsf{le}(n, I_2), \sigma'\rangle}$$

[not] $$\frac{Bool \triangleright \textit{\_\_boolean}(\_)}{\langle \mathsf{NOT}(Bool), \sigma\rangle \rightarrow \langle \textit{\_\_not}(Bool), \sigma\rangle}$$

[notResolve] $$\frac{\langle ERight, \sigma\rangle \rightarrow \langle I, \sigma'\rangle}{\langle \mathsf{NOT}(ERight), \sigma\rangle \rightarrow \langle \mathsf{NOT}(I), \sigma'\rangle}$$

[xor] $$\frac{Bool_1 \triangleright \textit{\_\_boolean}(\_) \quad Bool_2 \triangleright \textit{\_\_boolean}(\_)}{\langle \mathsf{XOR}(Bool_1, Bool_2), \sigma\rangle \rightarrow \langle \textit{\_\_xor}(Bool_1, Bool_2), \sigma\rangle}$$

[xorRight] $$\frac{Bool \triangleright \textit{\_\_boolean}(\_) \quad \langle ERight, \sigma\rangle \rightarrow \langle BoolRight, \sigma'\rangle}{\langle \mathsf{XOR}(Bool, ERight), \sigma\rangle \rightarrow \langle \mathsf{XOR}(Bool, BoolRight), \sigma'\rangle}$$

[xorLeft] $$\frac{\langle ELeft, \sigma\rangle \rightarrow \langle ILeft, \sigma'\rangle}{\langle \mathsf{XOR}(ELeft, ERight), \sigma\rangle \rightarrow \langle \mathsf{XOR}(ILeft, ERight), \sigma'\rangle}$$

[subInt] $$\frac{n_1 \triangleright \textit{\_\_int32}(\_) \quad n_2 \triangleright \textit{\_\_int32}(\_)}{\langle \mathsf{sub}(n_1, n_2), \sigma\rangle \rightarrow \langle \textit{\_\_sub}(n_1, n_2), \sigma\rangle}$$

[subRightInt] $$\frac{n \triangleright \textit{\_\_int32}(\_) \quad \langle E_2, \sigma\rangle \rightarrow \langle I_2, \sigma'\rangle}{\langle \mathsf{sub}(n, E_2), \sigma\rangle \rightarrow \langle \mathsf{sub}(n, I_2), \sigma'\rangle}$$

[subReal]
$$\frac{n_1 \triangleright \textit{\_\_real64}(\_) \quad n_2 \triangleright \textit{\_\_real64}(\_)}{\langle \mathsf{sub}(n_1, n_2), \sigma \rangle \rightarrow \langle \textit{\_\_sub}(n_1, n_2), \sigma \rangle}$$

[subRightReal]
$$\frac{n \triangleright \textit{\_\_real64}(\_) \quad \langle E_2, \sigma \rangle \rightarrow \langle I_2, \sigma' \rangle}{\langle \mathsf{sub}(n, E_2), \sigma \rangle \rightarrow \langle \mathsf{sub}(n, I_2), \sigma' \rangle}$$

[subLeft]
$$\frac{\langle E_1, \sigma \rangle \rightarrow \langle I_1, \sigma' \rangle}{\langle \mathsf{sub}(E_1, E_2), \sigma \rangle \rightarrow \langle \mathsf{sub}(I_1, E_2), \sigma' \rangle}$$

[castReal]
$$\frac{n \triangleright \textit{\_\_int32}(\_)}{\langle \mathsf{castReal}(n), \sigma \rangle \rightarrow \langle \textit{\_\_user}(\mathsf{castReal}, n), \sigma \rangle}$$

[castRealResolve]
$$\frac{\langle E_1, \sigma \rangle \rightarrow \langle I_1, \sigma' \rangle}{\langle \mathsf{castReal}(E_1), \sigma \rangle \rightarrow \langle \mathsf{castReal}(I_1), \sigma' \rangle}$$

[castInt]
$$\frac{n \triangleright \textit{\_\_real64}(\_)}{\langle \mathsf{castInt}(n), \sigma \rangle \rightarrow \langle \textit{\_\_user}(\mathsf{castInt}, n), \sigma \rangle}$$

[castIntResolve]
$$\frac{\langle E_1, \sigma \rangle \rightarrow \langle I_1, \sigma' \rangle}{\langle \mathsf{castInt}(E_1), \sigma \rangle \rightarrow \langle \mathsf{castInt}(I_1), \sigma' \rangle}$$

[runTrack]
$$\frac{track \triangleright \textit{\_\_string}(\_)}{\langle \mathsf{runTrack}(track), \sigma \rangle \rightarrow \langle \textit{\_\_user}(\mathsf{runTrack}, track), \sigma \rangle}$$

[runTrackResolve]
$$\frac{\langle E, \sigma \rangle \rightarrow \langle s, \sigma' \rangle}{\langle \mathsf{runTrack}(E), \sigma \rangle \rightarrow \langle \mathsf{runTrack}(s), \sigma' \rangle}$$

[runTrackDeref]
$$\frac{}{\langle \mathsf{runTrack}(track), \sigma \rangle \rightarrow \langle \mathsf{runTrack}(\mathsf{deref}(track)), \sigma \rangle}$$

[sequenceDone]
$$\frac{}{\langle \mathsf{seq}(\textit{\_\_done}, C), \sigma \rangle \rightarrow \langle C, \sigma \rangle}$$

[sequence]
$$\frac{\langle C_1, \sigma \rangle \rightarrow \langle C_1', \sigma' \rangle}{\langle \mathsf{seq}(C_1, C_2), \sigma \rangle \rightarrow \langle \mathsf{seq}(C_1', C_2), \sigma' \rangle}$$

[print]
$$\frac{s \triangleright \textit{\_\_string}(\_)}{\langle \mathsf{print}(s), \sigma \rangle \rightarrow \langle \textit{\_\_user}(\textit{\_\_string}(\text{"print"}), s), \sigma \rangle}$$

[variable]
$$\frac{\textit{\_\_get}(\sigma, R) \triangleright Z}{\langle \mathsf{deref}(R), \sigma \rangle \rightarrow \langle Z, \sigma \rangle}$$

[multInt]
$$\frac{n_1 \triangleright \textit{\_\_int32}(\_) \quad n_2 \triangleright \textit{\_\_int32}(\_)}{\langle \mathsf{mult}(n_1, n_2), \sigma \rangle \rightarrow \langle \textit{\_\_mul}(n_1, n_2), \sigma \rangle}$$

[multRightInt]
$$\frac{n \triangleright \textit{\_\_int32}(\_) \quad \langle E_2, \sigma \rangle \rightarrow \langle I_2, \sigma' \rangle}{\langle \mathsf{mult}(n, E_2), \sigma \rangle \rightarrow \langle \mathsf{mult}(n, I_2), \sigma' \rangle}$$

[multReal]
$$\frac{n_1 \triangleright \textit{\_\_real64}(\_) \quad n_2 \triangleright \textit{\_\_real64}(\_)}{\langle \mathsf{mult}(n_1, n_2), \sigma \rangle \rightarrow \langle \textit{\_\_mul}(n_1, n_2), \sigma \rangle}$$

[multRightReal]

$$\frac{n \triangleright \textit{\_\_real64}(\_) \quad \langle E_2, \sigma \rangle \to \langle I_2, \sigma' \rangle}{\langle \mathsf{mult}(n, E_2), \sigma \rangle \to \langle \mathsf{mult}(n, I_2), \sigma' \rangle}$$

[multLeft]

$$\frac{\langle E_1, \sigma \rangle \to \langle I_1, \sigma' \rangle}{\langle \mathsf{mult}(E_1, E_2), \sigma \rangle \to \langle \mathsf{mult}(I_1, E_2), \sigma' \rangle}$$

[ifTrue]

$$\frac{}{\langle \mathsf{if}(\textit{\_\_boolean}(\mathsf{True}), C_1, C_2), \sigma \rangle \to \langle C_1, \sigma \rangle}$$

[ifFalse]

$$\frac{}{\langle \mathsf{if}(\textit{\_\_boolean}(\mathsf{False}), C_1, C_2), \sigma \rangle \to \langle C_2, \sigma \rangle}$$

[ifResolve]

$$\frac{\langle E, \sigma \rangle \to \langle E', \sigma' \rangle}{\langle \mathsf{if}(E, C_1, C_2), \sigma \rangle \to \langle \mathsf{if}(E', C_1, C_2), \sigma' \rangle}$$

[appendArrayArray]

$$\frac{\textit{\_\_termRoot}(a_1) \triangleright \textit{\_\_flexArray} \quad \textit{\_\_termRoot}(a_2) \triangleright \textit{\_\_flexArray}}{\langle \mathsf{add}(a_1, a_2), \sigma \rangle \to \langle \textit{\_\_append}(a_1, a_2), \sigma \rangle}$$

[appendArrayElement]

$$\frac{\textit{\_\_termRoot}(a) \triangleright \textit{\_\_flexArray}}{\langle \mathsf{add}(a, x), \sigma \rangle \to \langle \mathsf{add}(a, \textit{\_\_flexArray}(x)), \sigma \rangle}$$

[addInt]

$$\frac{n_1 \triangleright \textit{\_\_int32}(\_) \quad n_2 \triangleright \textit{\_\_int32}(\_)}{\langle \mathsf{add}(n_1, n_2), \sigma \rangle \to \langle \textit{\_\_add}(n_1, n_2), \sigma \rangle}$$

[addRightInt]

$$\frac{n \triangleright \textit{\_\_int32}(\_) \quad \langle E_2, \sigma \rangle \to \langle I_2, \sigma' \rangle}{\langle \mathsf{add}(n, E_2), \sigma \rangle \to \langle \mathsf{add}(n, I_2), \sigma' \rangle}$$

[addReal]

$$\frac{n_1 \triangleright \textit{\_\_real64}(\_) \quad n_2 \triangleright \textit{\_\_real64}(\_)}{\langle \mathsf{add}(n_1, n_2), \sigma \rangle \to \langle \textit{\_\_add}(n_1, n_2), \sigma \rangle}$$

[addRightReal]

$$\frac{n \triangleright \textit{\_\_real64}(\_) \quad \langle E_2, \sigma \rangle \to \langle I_2, \sigma' \rangle}{\langle \mathsf{add}(n, E_2), \sigma \rangle \to \langle \mathsf{add}(n, I_2), \sigma' \rangle}$$

[addLeft]

$$\frac{\langle E_1, \sigma \rangle \to \langle I_1, \sigma' \rangle}{\langle \mathsf{add}(E_1, E_2), \sigma \rangle \to \langle \mathsf{add}(I_1, E_2), \sigma' \rangle}$$

[divInt]

$$\frac{n_1 \triangleright \textit{\_\_int32}(\_) \quad n_2 \triangleright \textit{\_\_int32}(\_)}{\langle \mathsf{div}(n_1, n_2), \sigma \rangle \to \langle \textit{\_\_div}(n_1, n_2), \sigma \rangle}$$

[divRightInt]

$$\frac{n \triangleright \textit{\_\_int32}(\_) \quad \langle E_2, \sigma \rangle \to \langle I_2, \sigma' \rangle}{\langle \mathsf{div}(n, E_2), \sigma \rangle \to \langle \mathsf{div}(n, I_2), \sigma' \rangle}$$

[divReal]

$$\frac{n_1 \triangleright \textit{\_\_real64}(\_) \quad n_2 \triangleright \textit{\_\_real64}(\_)}{\langle \mathsf{div}(n_1, n_2), \sigma \rangle \to \langle \textit{\_\_div}(n_1, n_2), \sigma \rangle}$$

[divRightInt]

$$\frac{n \triangleright \textit{\_\_real64}(\_) \quad \langle E_2, \sigma \rangle \to \langle I_2, \sigma' \rangle}{\langle \mathsf{div}(n, E_2), \sigma \rangle \to \langle \mathsf{div}(n, I_2), \sigma' \rangle}$$

[divLeft]

$$\frac{\langle E_1, \sigma \rangle \to \langle I_1, \sigma' \rangle}{\langle \mathsf{div}(E_1, E_2), \sigma \rangle \to \langle \mathsf{div}(I_1, E_2), \sigma' \rangle}$$

[while]
$$\frac{}{\langle \mathsf{while}(E, C), \sigma \rangle \rightarrow \langle \mathsf{if}(E, \mathsf{seq}(C, \mathsf{while}(E, C)), \_\_\textit{done}), \sigma \rangle}$$

[getArrayElement]
$$\frac{\_\_\textit{termRoot}(a) \triangleright \_\_\textit{flexArray} \quad i \triangleright \_\_\textit{int32}(\_)}{\langle \mathsf{get}(a, i), \sigma \rangle \rightarrow \langle \_\_\textit{get}(a, i), \sigma \rangle}$$

[getArrayElementRight]
$$\frac{\_\_\textit{termRoot}(a) \triangleright \_\_\textit{flexArray} \quad \langle E, \sigma \rangle \rightarrow \langle I, \sigma' \rangle}{\langle \mathsf{get}(a, E), \sigma \rangle \rightarrow \langle \mathsf{get}(a, I), \sigma' \rangle}$$

[getArrayElementLeft]
$$\frac{\langle E, \sigma \rangle \rightarrow \langle I, \sigma' \rangle}{\langle \mathsf{get}(E, x), \sigma \rangle \rightarrow \langle \mathsf{get}(I, x), \sigma' \rangle}$$

[for]
$$\frac{v \triangleright \mathsf{assign}(\_, \_\_\textit{int32}(\_))}{\langle \mathsf{for}(v, E, i, C), \sigma \rangle \rightarrow \langle \mathsf{seq}(v, \mathsf{while}(E, \mathsf{seq}(C, i))), \sigma \rangle}$$

[expInt]
$$\frac{n \triangleright \_\_\textit{int32}(\_) \quad exponent \triangleright \_\_\textit{int32}(\_)}{\langle \mathsf{exp}(n, exponent), \sigma \rangle \rightarrow \langle \_\_\textit{exp}(n, exponent), \sigma \rangle}$$

[expIntRight]
$$\frac{n \triangleright \_\_\textit{int32}(\_) \quad \langle E, \sigma \rangle \rightarrow \langle I, \sigma' \rangle}{\langle \mathsf{exp}(n, E), \sigma \rangle \rightarrow \langle \mathsf{exp}(n, I), \sigma' \rangle}$$

[expIntLeft]
$$\frac{\langle E, \sigma \rangle \rightarrow \langle I, \sigma' \rangle}{\langle \mathsf{exp}(E, exponent), \sigma \rangle \rightarrow \langle \mathsf{exp}(I, exponent), \sigma' \rangle}$$

[getArraySize]
$$\frac{\_\_\textit{termRoot}(a) \triangleright \_\_\textit{flexArray}}{\langle \mathsf{len}(a), \sigma \rangle \rightarrow \langle \_\_\textit{size}(a), \sigma \rangle}$$

[getArraySizeResolve]
$$\frac{\langle E, \sigma \rangle \rightarrow \langle I, \sigma' \rangle}{\langle \mathsf{len}(E), \sigma \rangle \rightarrow \langle \mathsf{len}(I), \sigma' \rangle}$$

[subroutine]
$$\frac{}{\langle \mathsf{func}(name, S), \sigma \rangle \rightarrow \langle \_\_\textit{done}, \_\_\textit{put}(\sigma, name, S) \rangle}$$

[incrementInt]
$$\frac{n_1 \triangleright \_\_\textit{int32}(\_)}{\langle \mathsf{inc}(n_1), \sigma \rangle \rightarrow \langle \_\_\textit{add}(n_1, \_\_\textit{int32}(1)), \sigma \rangle}$$

[incrementReal]
$$\frac{n_1 \triangleright \_\_\textit{real64}(\_)}{\langle \mathsf{inc}(n_1), \sigma \rangle \rightarrow \langle \_\_\textit{add}(n_1, \_\_\textit{real64}(1.0)), \sigma \rangle}$$

[incrementResolve]
$$\frac{\langle E, \sigma \rangle \rightarrow \langle I, \sigma' \rangle}{\langle \mathsf{inc}(E), \sigma \rangle \rightarrow \langle \mathsf{inc}(I), \sigma' \rangle}$$

[runSubroutine]
$$\frac{}{\langle \mathsf{execute}(func), \sigma \rangle \rightarrow \langle \mathsf{deref}(func), \sigma \rangle}$$

[setBpm]
$$\frac{n \triangleright \_\_\textit{int32}(\_)}{\langle \mathsf{setBpm}(n), \sigma \rangle \rightarrow \langle \_\_\textit{user}(\mathsf{bpm}, n), \sigma \rangle}$$

[setBpmResolve]
$$\frac{\langle E, \sigma \rangle \rightarrow \langle I, \sigma' \rangle}{\langle \mathsf{setBpm}(E), \sigma \rangle \rightarrow \langle \mathsf{setBpm}(I), \sigma' \rangle}$$

[gtInt] $$\frac{n_1 \triangleright \textit{\_\_int32}(\_) \quad n_2 \triangleright \textit{\_\_int32}(\_)}{\langle \mathsf{gt}(n_1, n_2), \sigma \rangle \rightarrow \langle \textit{\_\_gt}(n_1, n_2), \sigma \rangle}$$

[gtRightInt] $$\frac{n \triangleright \textit{\_\_int32}(\_) \quad \langle E_2, \sigma \rangle \rightarrow \langle I_2, \sigma' \rangle}{\langle \mathsf{gt}(n, E_2), \sigma \rangle \rightarrow \langle \mathsf{gt}(n, I_2), \sigma' \rangle}$$

[gtReal] $$\frac{n_1 \triangleright \textit{\_\_int32}(\_) \quad n_2 \triangleright \textit{\_\_real64}(\_)}{\langle \mathsf{gt}(n_1, n_2), \sigma \rangle \rightarrow \langle \textit{\_\_gt}(n_1, n_2), \sigma \rangle}$$

[gtRightReal] $$\frac{n \triangleright \textit{\_\_real64}(\_) \quad \langle E_2, \sigma \rangle \rightarrow \langle I_2, \sigma' \rangle}{\langle \mathsf{gt}(n, E_2), \sigma \rangle \rightarrow \langle \mathsf{gt}(n, I_2), \sigma' \rangle}$$

[gtLeft] $$\frac{\langle E_1, \sigma \rangle \rightarrow \langle I_1, \sigma' \rangle}{\langle \mathsf{gt}(E_1, E_2), \sigma \rangle \rightarrow \langle \mathsf{gt}(I_1, E_2), \sigma' \rangle}$$

[drumVolume] $$\frac{n \triangleright \textit{\_\_int32}(\_)}{\langle \mathsf{drumVolume}(drum, n), \sigma \rangle \rightarrow \langle \textit{\_\_user}(\mathsf{drumVolume}, drum, n), \sigma \rangle}$$

[drumVolumeResolve] $$\frac{\langle E, \sigma \rangle \rightarrow \langle I, \sigma' \rangle}{\langle \mathsf{drumVolume}(drum, E), \sigma \rangle \rightarrow \langle \mathsf{drumVolume}(drum, I), \sigma' \rangle}$$

[decrementInt] $$\frac{n_1 \triangleright \textit{\_\_int32}(\_)}{\langle \mathsf{dec}(n_1), \sigma \rangle \rightarrow \langle \textit{\_\_sub}(n_1, \textit{\_\_int32}(1)), \sigma \rangle}$$

[decrementReal] $$\frac{n_1 \triangleright \textit{\_\_real64}(\_)}{\langle \mathsf{dec}(n_1), \sigma \rangle \rightarrow \langle \textit{\_\_sub}(n_1, \textit{\_\_real64}(1.0)), \sigma \rangle}$$

[decrementResolve] $$\frac{\langle E, \sigma \rangle \rightarrow \langle I, \sigma' \rangle}{\langle \mathsf{dec}(E), \sigma \rangle \rightarrow \langle \mathsf{dec}(I), \sigma' \rangle}$$

[drumDuration] $$\frac{n \triangleright \textit{\_\_int32}(\_)}{\langle \mathsf{drumDuration}(drum, n), \sigma \rangle \rightarrow \langle \textit{\_\_user}(\mathsf{drumDuration}, n), \sigma \rangle}$$

[drumDurationResolve] $$\frac{\langle E, \sigma \rangle \rightarrow \langle I, \sigma' \rangle}{\langle \mathsf{drumDuration}(drum, E), \sigma \rangle \rightarrow \langle \mathsf{drumDuration}(drum, I), \sigma' \rangle}$$

[negateInt] $$\frac{n \triangleright \textit{\_\_int32}(\_)}{\langle \mathsf{neg}(n), \sigma \rangle \rightarrow \langle \textit{\_\_sub}(\textit{\_\_int32}(0), n), \sigma \rangle}$$

[negateReal] $$\frac{n \triangleright \textit{\_\_real64}(\_)}{\langle \mathsf{neg}(n), \sigma \rangle \rightarrow \langle \textit{\_\_sub}(\textit{\_\_real64}(0.0), n), \sigma \rangle}$$

[negateIntResolve] $$\frac{\langle E, \sigma \rangle \rightarrow \langle I, \sigma' \rangle}{\langle \mathsf{neg}(E), \sigma \rangle \rightarrow \langle \mathsf{neg}(I), \sigma' \rangle}$$

[MakeTrack] $$\frac{title \triangleright \textit{\_\_string}(\_)}{\langle \mathsf{makeTrack}(name, title), \sigma \rangle \rightarrow \langle \textit{\_\_user}(\mathsf{track}, name, title), \textit{\_\_put}(\sigma, name, title) \rangle}$$

[MakeTrackResolveTitle] $$\frac{\langle E, \sigma \rangle \rightarrow \langle s, \sigma' \rangle}{\langle \mathsf{makeTrack}(name, E), \sigma \rangle \rightarrow \langle \mathsf{makeTrack}(name, s), \sigma' \rangle}$$

[geInt] $$\frac{n_1 \triangleright \textit{\_\_int32}(\_) \quad n_2 \triangleright \textit{\_\_int32}(\_)}{\langle \mathsf{ge}(n_1, n_2), \sigma\rangle \rightarrow \langle \textit{\_\_ge}(n_1, n_2), \sigma\rangle}$$

[geRightInt] $$\frac{n \triangleright \textit{\_\_int32}(\_) \quad \langle E_2, \sigma\rangle \rightarrow \langle I_2, \sigma'\rangle}{\langle \mathsf{ge}(n, E_2), \sigma\rangle \rightarrow \langle \mathsf{ge}(n, I_2), \sigma'\rangle}$$

[geReal] $$\frac{n_1 \triangleright \textit{\_\_real64}(\_) \quad n_2 \triangleright \textit{\_\_real64}(\_)}{\langle \mathsf{ge}(n_1, n_2), \sigma\rangle \rightarrow \langle \textit{\_\_ge}(n_1, n_2), \sigma\rangle}$$

[geRightReal] $$\frac{n \triangleright \textit{\_\_real64}(\_) \quad \langle E_2, \sigma\rangle \rightarrow \langle I_2, \sigma'\rangle}{\langle \mathsf{ge}(n, E_2), \sigma\rangle \rightarrow \langle \mathsf{ge}(n, I_2), \sigma'\rangle}$$

[eqInt] $$\frac{n_1 \triangleright \textit{\_\_int32}(\_) \quad n_2 \triangleright \textit{\_\_int32}(\_)}{\langle \mathsf{eq}(n_1, n_2), \sigma\rangle \rightarrow \langle \textit{\_\_eq}(n_1, n_2), \sigma\rangle}$$

[eqRightInt] $$\frac{n \triangleright \textit{\_\_int32}(\_) \quad \langle E_2, \sigma\rangle \rightarrow \langle I_2, \sigma'\rangle}{\langle \mathsf{eq}(n, E_2), \sigma\rangle \rightarrow \langle \mathsf{eq}(n, I_2), \sigma'\rangle}$$

[eqReal] $$\frac{n_1 \triangleright \textit{\_\_real64}(\_) \quad n_2 \triangleright \textit{\_\_real64}(\_)}{\langle \mathsf{eq}(n_1, n_2), \sigma\rangle \rightarrow \langle \textit{\_\_eq}(n_1, n_2), \sigma\rangle}$$

[eqRightReal] $$\frac{n \triangleright \textit{\_\_real64}(\_) \quad \langle E_2, \sigma\rangle \rightarrow \langle I_2, \sigma'\rangle}{\langle \mathsf{eq}(n, E_2), \sigma\rangle \rightarrow \langle \mathsf{eq}(n, I_2), \sigma'\rangle}$$

[eqLeft] $$\frac{\langle E_1, \sigma\rangle \rightarrow \langle I_1, \sigma'\rangle}{\langle \mathsf{eq}(E_1, E_2), \sigma\rangle \rightarrow \langle \mathsf{eq}(I_1, E_2), \sigma'\rangle}$$

[and] $$\frac{Bool_1 \triangleright \textit{\_\_boolean}(\_) \quad Bool_2 \triangleright \textit{\_\_boolean}(\_)}{\langle \mathsf{AND}(Bool_1, Bool_2), \sigma\rangle \rightarrow \langle \textit{\_\_and}(Bool_1, Bool_2), \sigma\rangle}$$

[andRight] $$\frac{Bool \triangleright \textit{\_\_boolean}(\_) \quad \langle ERight, \sigma\rangle \rightarrow \langle BoolRight, \sigma'\rangle}{\langle \mathsf{AND}(Bool, ERight), \sigma\rangle \rightarrow \langle \mathsf{AND}(Bool, BoolRight), \sigma'\rangle}$$

[andLeft] $$\frac{\langle ERightLeft, \sigma\rangle \rightarrow \langle ILeft, \sigma'\rangle}{\langle \mathsf{AND}(ELeft, ERight), \sigma\rangle \rightarrow \langle \mathsf{AND}(ILeft, ERight), \sigma'\rangle}$$

[setSoundSnare] $$\frac{track \triangleright \textit{\_\_string}(\_) \quad pos \triangleright \textit{\_\_int32}(\_) \quad sound \triangleright \mathsf{snare}}{\langle \mathsf{setSound}(track, pos, sound), \sigma\rangle \rightarrow \langle \textit{\_\_user}(\mathsf{setSound}, track, pos, sound), \sigma\rangle}$$

[setSoundBass] $$\frac{track \triangleright \textit{\_\_string}(\_) \quad pos \triangleright \textit{\_\_int32}(\_) \quad sound \triangleright \mathsf{bass}}{\langle \mathsf{setSound}(track, pos, sound), \sigma\rangle \rightarrow \langle \textit{\_\_user}(\mathsf{setSound}, track, pos, sound), \sigma\rangle}$$

[setSoundTom1] $$\frac{track \triangleright \textit{\_\_string}(\_) \quad pos \triangleright \textit{\_\_int32}(\_) \quad sound \triangleright \mathsf{tom1}}{\langle \mathsf{setSound}(track, pos, sound), \sigma\rangle \rightarrow \langle \textit{\_\_user}(\mathsf{setSound}, track, pos, sound), \sigma\rangle}$$

[setSoundTom2] $$\frac{track \triangleright \textit{\_\_string}(\_) \quad pos \triangleright \textit{\_\_int32}(\_) \quad sound \triangleright \mathsf{tom2}}{\langle \mathsf{setSound}(track, pos, sound), \sigma\rangle \rightarrow \langle \textit{\_\_user}(\mathsf{setSound}, track, pos, sound), \sigma\rangle}$$

[setSoundFloorTom] $$\frac{track \triangleright \textit{\_\_string}(\_) \quad pos \triangleright \textit{\_\_int32}(\_) \quad sound \triangleright \mathsf{floorTom}}{\langle \mathsf{setSound}(track, pos, sound), \sigma\rangle \rightarrow \langle \textit{\_\_user}(\mathsf{setSound}, track, pos, sound), \sigma\rangle}$$

[setSoundCrash] $$\frac{track \triangleright \mathit{\_\_string}(\_) \quad pos \triangleright \mathit{\_\_int32}(\_) \quad sound \triangleright \mathsf{crash}}{\langle \mathsf{setSound}(track, pos, sound), \sigma \rangle \rightarrow \langle \mathit{\_\_user}(\mathsf{setSound}, track, pos, sound), \sigma \rangle}$$

[setSoundRide] $$\frac{track \triangleright \mathit{\_\_string}(\_) \quad pos \triangleright \mathit{\_\_int32}(\_) \quad sound \triangleright \mathsf{ride}}{\langle \mathsf{setSound}(track, pos, sound), \sigma \rangle \rightarrow \langle \mathit{\_\_user}(\mathsf{setSound}, track, pos, sound), \sigma \rangle}$$

[setSoundOpenHiHatOpen] $$\frac{track \triangleright \mathit{\_\_string}(\_) \quad pos \triangleright \mathit{\_\_int32}(\_) \quad sound \triangleright \mathsf{hiHatOpen}}{\langle \mathsf{setSound}(track, pos, sound), \sigma \rangle \rightarrow \langle \mathit{\_\_user}(\mathsf{setSound}, track, pos, sound), \sigma \rangle}$$

[setSoundhiHatClosed] $$\frac{track \triangleright \mathit{\_\_string}(\_) \quad pos \triangleright \mathit{\_\_int32}(\_) \quad sound \triangleright \mathsf{hiHatClosed}}{\langle \mathsf{setSound}(track, pos, sound), \sigma \rangle \rightarrow \langle \mathit{\_\_user}(\mathsf{setSound}, track, pos, sound), \sigma \rangle}$$

[setSoundhiHatPedal] $$\frac{track \triangleright \mathit{\_\_string}(\_) \quad pos \triangleright \mathit{\_\_int32}(\_) \quad sound \triangleright \mathsf{hiHatPedal}}{\langle \mathsf{setSound}(track, pos, sound), \sigma \rangle \rightarrow \langle \mathit{\_\_user}(\mathsf{setSound}, track, pos, sound), \sigma \rangle}$$

[setSoundResolveSound] $$\frac{track \triangleright \mathit{\_\_string}(\_) \quad pos \triangleright \mathit{\_\_int32}(\_) \quad \langle E, \sigma \rangle \rightarrow \langle I, \sigma' \rangle}{\langle \mathsf{setSound}(track, pos, E), \sigma \rangle \rightarrow \langle \mathsf{setSound}(track, pos, I), \sigma' \rangle}$$

[setSoundResolvePos] $$\frac{track \triangleright \mathit{\_\_string}(\_) \quad \langle E, \sigma \rangle \rightarrow \langle I, \sigma' \rangle}{\langle \mathsf{setSound}(track, E, x), \sigma \rangle \rightarrow \langle \mathsf{setSound}(track, I, x), \sigma' \rangle}$$

[setSoundResolveTrack] $$\frac{\langle E, \sigma \rangle \rightarrow \langle I, \sigma' \rangle}{\langle \mathsf{setSound}(E, x, y), \sigma \rangle \rightarrow \langle \mathsf{setSound}(I, x, y), \sigma' \rangle}$$

[setSoundDerefTrack] $$\frac{}{\langle \mathsf{setSound}(track, x, y), \sigma \rangle \rightarrow \langle \mathsf{setSound}(\mathsf{deref}(track), x, y), \sigma \rangle}$$

[ne] $$\frac{n_1 \triangleright \mathit{\_\_int32}(\_) \quad n_2 \triangleright \mathit{\_\_int32}(\_)}{\langle \mathsf{ne}(n_1, n_2), \sigma \rangle \rightarrow \langle \mathit{\_\_ne}(n_1, n_2), \sigma \rangle}$$

[neRight] $$\frac{n \triangleright \mathit{\_\_int32}(\_) \quad \langle E_2, \sigma \rangle \rightarrow \langle I_2, \sigma' \rangle}{\langle \mathsf{ne}(n, E_2), \sigma \rangle \rightarrow \langle \mathsf{ne}(n, I_2), \sigma' \rangle}$$

[neLeft] $$\frac{\langle E_1, \sigma \rangle \rightarrow \langle I_1, \sigma' \rangle}{\langle \mathsf{ne}(E_1, E_2), \sigma \rangle \rightarrow \langle \mathsf{ne}(I_1, E_2), \sigma' \rangle}$$

[ltInt] $$\frac{n_1 \triangleright \mathit{\_\_int32}(\_) \quad n_2 \triangleright \mathit{\_\_int32}(\_)}{\langle \mathsf{lt}(n_1, n_2), \sigma \rangle \rightarrow \langle \mathit{\_\_lt}(n_1, n_2), \sigma \rangle}$$

[ltRightInt] $$\frac{n \triangleright \mathit{\_\_int32}(\_) \quad \langle E_2, \sigma \rangle \rightarrow \langle I_2, \sigma' \rangle}{\langle \mathsf{lt}(n, E_2), \sigma \rangle \rightarrow \langle \mathsf{lt}(n, I_2), \sigma' \rangle}$$

[ltReal] $$\frac{n_1 \triangleright \mathit{\_\_real64}(\_) \quad n_2 \triangleright \mathit{\_\_real64}(\_)}{\langle \mathsf{lt}(n_1, n_2), \sigma \rangle \rightarrow \langle \mathit{\_\_lt}(n_1, n_2), \sigma \rangle}$$

[ltRightReal] $$\frac{n \triangleright \mathit{\_\_real64}(\_) \quad \langle E_2, \sigma \rangle \rightarrow \langle I_2, \sigma' \rangle}{\langle \mathsf{lt}(n, E_2), \sigma \rangle \rightarrow \langle \mathsf{lt}(n, I_2), \sigma' \rangle}$$

[ltLeft] $$\frac{\langle E_1, \sigma \rangle \rightarrow \langle I_1, \sigma' \rangle}{\langle \mathsf{lt}(E_1, E_2), \sigma \rangle \rightarrow \langle \mathsf{lt}(I_1, E_2), \sigma' \rangle}$$

[or] $$\frac{Bool_1 \triangleright \mathit{\_\_boolean}(\_) \quad Bool_2 \triangleright \mathit{\_\_boolean}(\_)}{\langle \mathsf{OR}(Bool_1, Bool_2), \sigma \rangle \rightarrow \langle \mathit{\_\_or}(Bool_1, Bool_2), \sigma \rangle}$$

[orRight] $$\frac{Bool \triangleright \mathit{\_\_boolean}(\_) \quad \langle ERight, \sigma \rangle \rightarrow \langle BoolRight, \sigma' \rangle}{\langle \mathsf{OR}(Bool, ERight), \sigma \rangle \rightarrow \langle \mathsf{OR}(Bool, BoolRight), \sigma' \rangle}$$

[orLeft] $$\frac{\langle ELeft, \sigma \rangle \rightarrow \langle ILeft, \sigma' \rangle}{\langle \mathsf{OR}(ELeft, ERight), \sigma \rangle \rightarrow \langle \mathsf{OR}(ILeft, ERight), \sigma' \rangle}$$

5 Externel to Internal Parser

```
    (* ART parser for the GCD+user language which generates abstract terms *)

statement ::= seq^^ | assign^^ | if^^ | while^^ | for^^ | func^^ | execute^^|
  drumVolume^^ | drumDuration^^| makeTrack^^ | setSound^^ | runTrack^^| setBpm^^
drum ::= snare^^ | bass^^ | tom1^^ | tom2^^ | floorTom^^ | crash^^ | ride^^ |
   hiHatOpen^^ | hiHatClosed^^ | hiHatPedal^^

seq ::= statement statement
assign ::= ID '='^ expr0 ';'^
func ::= 'song'^ ID '{'^ statement '}'^
if ::= 'if'^ '('^ expr0 ')'^ '{'^ statement '}'^ 'else:'^ '{'^ statement '}'^
while ::= 'while'^ '('^ expr0 ')'^ '{'^ statement '}'^
for ::= 'for'^ '('^ statement expr0 ';'^ statement ')'^ '{'^ statement '}'^
execute ::= 'perform'^ ID ';'^
setSound ::= 'set_sound'^ ID '['^ expr0 ']'^ drum ';'^
makeTrack ::= 'make_track'^ ID  '{'^ expr0 '}'^
runTrack ::= 'play_track'^ ID ';'^
setBpm ::= 'set_bpm'^ expr0 ';'^

drumVolume ::= 'drum_volume'^ '('^ drum ','^ expr0 ')'^ ';'^
drumDuration ::= 'drum_duration'^ '('^ drum ','^ expr0 ')'^ ';'^

operator0 ::= AND^^
operator1 ::= OR^^|XOR^^
operator2 ::= NOT^^
operator3 ::= gt^^|ne^^|ge^^|eq^^|lt^^|le^^
operator4 ::= add^^|sub^^
operator5 ::= mult^^|div^^
operator6 ::= neg^^ | len^^
operator7 ::= exp^^ | get^^
operator8 ::= inc^^|dec^^
operator9 ::= castReal^^ | castInt^^

expr0 ::= expr1^^|expr1 operator0^^ expr1
expr1 ::= expr2^^|expr1 operator1^^ expr2
expr2 ::= expr3^^|operator2^^ expr2
expr3 ::= expr4^^|expr3 operator3^^ expr4
expr4 ::= expr5^^|expr4 operator4^^ expr5
expr5 ::= expr6^^|expr5 operator5^^ expr6
expr6 ::= expr7^^|operator6^^ expr6
expr7 ::= expr8^^|expr7 operator7^^ expr8
expr8 ::= expr9^^|operator8^^ expr8
expr9 ::= expr10^^|operator9^^ '('^ expr9 ')'^
expr10 ::= '('^ expr1 ')'^ | operand^^
```

```
add ::= '+'^
sub ::= '-'^
mult ::= '*'^
div ::= '/'^
exp ::= '**'^
neg ::= '-'^
inc ::= '++'^
dec ::= '--'^

gt ::= '>'^
ne ::= '!='^
ge ::= '>='^
eq ::= '=='^
lt ::= '<'^
le ::= '<='^

OR ::= 'OR'^
XOR ::= 'XOR'^
AND ::= 'AND'^
NOT ::= 'NOT'^

len ::= 'len'^
get ::= 'at'^

castReal ::= 'real'^
castInt ::= 'int'^

snare ::= 'snare'^
bass ::= 'bass'^
tom1 ::= 'tom1'^
tom2 ::= 'tom2'^
floorTom ::= 'floor_tom'^
crash ::= 'crash'^
ride ::= 'ride'^
hiHatOpen ::= 'hi_hat_open'^
hiHatClosed ::= 'hi_hat_closed'^
hiHatpedal ::= 'hi_hat_pedal'^

True ::= 'True'^
False ::= 'False'^

list ::= INTEGER ','^ list^ | INTEGER
__flexArray ::= '['^ list^ ']'^
operand ::= deref^^ | INTEGER^^| True^^ | False^^ | __flexArray^^ | STRING^^ | REAL^^
deref ::= ID
```

```
(* lexical items below this line *)
ID <leftExtent:int rightExtent:int lexeme:String v:ARTValueString> ::=
  &ID^^ {ID.lexeme = artLexeme(ID.leftExtent, ID.rightExtent);
  ID.v = new ARTValueString(artLexemeAsID(ID.leftExtent, ID.rightExtent)); }

INTEGER <leftExtent:int rightExtent:int lexeme:String v:ARTValueInteger32> ::=
  &INTEGER^^ { INTEGER.lexeme = artLexeme(INTEGER.leftExtent, INTEGER.rightExtent);
  INTEGER.v = new ARTValueInteger32(
    artLexemeAsInteger(INTEGER.leftExtent, INTEGER.rightExtent));
  }

REAL ::= &REAL^^

STRING ::= &STRING_DQ^^|&STRING_SQ^^|&STRING_PLAIN_SQ^^
```

6 Experiments

In this section I perform experiments using the grammar and eSOS rules defined above. Each experiment will escalate in complexity to show how the features tie together.

The first experiment simply tests making a track and running it.

```
make_track track01{"track01"}

set_sound track01[1] bass;
set_sound track01[5] snare;
set_sound track01[9] bass;
set_sound track01[11] bass;

play_track track01;
```

This successfully showed DML can be used to produce midi drum output.

The next test demonstrates how a for loop can be used to loop a pattern in a track, to easily produce longer tracks.

```
set_bpm 115;

make_track track01{"track01"}

for(i = 1; i < 50; i = i + 16;) {
  set_sound track01[i] bass;
  set_sound track01[i + 4] snare;
  set_sound track01[i + 8] bass;
```

```
  set_sound track01[i + 10] bass;
  set_sound track01[i + 14] snare;
}

play_track track01;
```

The beats per minute has also been set. The default value is set at 90, so in the example the tempo has been increased.

The next step is to try use multiple loops and layer sounds.

```
set_bpm 120;

make_track track01{"track01"}

num_of_ticks = 120;

for(i = 1; i<num_of_ticks; i = i + 4;) {
  set_sound track01[i] hi_hat_closed;
}

for(i = 21; i<num_of_ticks; i = i + 20;) {
  set_sound track01[i] ride;
}
for(i = 1; i < num_of_ticks; i = i + 16;) {
  set_sound track01[i] bass;
  set_sound track01[i + 4] snare;
  set_sound track01[i + 8] bass;
  set_sound track01[i + 10] snare;
  set_sound track01[i + 14] bass;
}

set_sound track01[num_of_ticks] crash;
set_sound track01[num_of_ticks] bass;

play_track track01;
```

In the above example a hi-hat and ride cymbal are layered over the drums.

In the final example multiple tracks are combined to make a song. Further functionality is demonstrated in setting the volume and duration of different sounds.

```
set_bpm 120;

make_track track01{"track01"}

drum_volume(crash, 150);
drum_volume(ride, 200);
```

```
drum_duration(snare, 3);

num_of_ticks = 80;

for(i = 0; i<num_of_ticks; i = i + 2;) {
  set_sound track01[i] hi_hat_closed;
}

for(i = 12; i<num_of_ticks; i = i + 12;) {
  set_sound track01[i] ride;
}
for(i = 0; i < num_of_ticks; i = i + 16;) {
  set_sound track01[i] bass;
  set_sound track01[i + 4] snare;
  set_sound track01[i + 8] bass;
  set_sound track01[i + 10] snare;
  set_sound track01[i + 14] bass;
}

set_sound track01[num_of_ticks] crash;
set_sound track01[num_of_ticks] bass;

make_track track02{"track02"}

for(i = 0; i < num_of_ticks; i = i + 4;) {
  set_sound track02[i] ride;
}

for(i = 0; i < num_of_ticks; i = i + 16;) {
  set_sound track02[i] bass;
  set_sound track02[i + 5] snare;
  set_sound track02[i + 8] bass;
  set_sound track02[i + 12] snare;
  set_sound track02[i + 14] snare;
}

make_track interlude{"track03"}
for(i = 0; i < 32; i = i + 4;) {
  set_sound interlude[i] ride;
}

song mySong {
  play_track track01;
```

These experiments show how DML is able to produce drum beats similar to those of a drum machine. the language allows users to set the tempo ,the volume and note duration of each drum, and to assemble tracks, which can be joined to make songs.

7 Conclusion and Reflection

The project was challenging with a difficult learning curve, but gratifying to produce. The language was successful at using midi to mimic a drum machine was successful with useful features included in the program. Some features included ended up no being useful, for example arrays were implemented, but are not used in the experiments carried out. The project could be improved by adding an attribute grammar, and more drum sounds could be added to give more variety.