Musicode

——基于 C 的音乐编程语言

何逸宸　ZY2106342

梁世俊　ZY2106323

目录

1 语言设计驱动

1.1 编程和音乐创作的类比

对于一种编译语言来说，通常的操作是将源代码通过词法分析、语法分析、中间代码生成、中间代码优化和生成目标代码五个基本步骤，生成可在目标机器上运行的目标代码。而对于音乐创作来说，各式各样的乐器甚至人的嗓音相当于硬件，而引导这些“硬件”演奏出优美旋律的，就是“机器码”——乐谱。通过 Musicode 编程语言，可以将类似 C 语言的源代码，先通过前端生成“音名&间隔”中间代码，然后通过后端生成用于描述音乐的 midi 文件以及五线谱。

1.2 Musicode 的设计动机

比起工程文件和 midi 文件单纯存储音符，力度，速度等单位化的信息，我们希望创建一个从乐理上的角度来表示一段音乐从作曲上的角度是如何实现的语言。只要不是现代主义无调性音乐，大部分的音乐都是极其具有乐理上的规律性的，这些规律抽象成乐理逻辑语句可以大大地精简化。

在人工智能高速发展的时代，我们希望作曲 AI 能在真正理解乐理的情况下进行乐曲创作，而不是使用深度学习，用大量的现成曲目进行训练，这个语言可以作为 AI 理解音乐的接口。

我们希望设计一个类似于 C 语言文法的音乐编程语言 Musicode，使得熟悉 C 语言的人可以快速上手，使用它进行自己的音乐创作。Musicode 是一种静态语言，可以在编译时确定变量类型以及进行错误检查。Musicode 可以用非常简洁的语法来表达一段音乐的音符，和弦，旋律，节奏，力度等信息，可以通过乐理逻辑来生成曲子，并且进行高级的乐理操作。Musicode 除了用来创作音乐之外，还可以从乐理层面上来创作音乐和分析音乐，并且可以在 Musicode 的基础上设计乐理算法来探索音乐的可能性。

1.3 Musicode 的语言特点

1.3.1 特定语义的字符串

设计了一些可识别的字符串来表示常见和弦，方便使用：

```
NAME_OF_INTERVAL = {
    'perfect unison': 0,
    'minor second': 1,
    'major second': 2,
    'minor third': 3,
    'major third': 4,
    'perfect fourth': 5,
    'diminished fifth': 6,
    'perfect fifth': 7,
    'minor sixth': 8,
    'major sixth': 9,
    'minor seventh': 10,
    'major seventh': 11,
    'perfect octave': 12,
    'minor ninth': 13,
    'major ninth': 14,
    'perfect eleventh': 17,
    'minor thirteenth': 20,
    'major thirteenth': 21
}
```

```
standard = {
    'C': 0,
    'C#': 1,
    'D': 2,
    'D#': 3,
    'E': 4,
    'F': 5,
    'F#': 6,
    'G': 7,
    'G#': 8,
    'A': 9,
    'A#': 10,
    'B': 11,
    'Bb': 10,
    'Eb': 3,
    'Ab': 8,
    'Db': 1,
    'Gb': 6
}
```

1.3.2 乐器类型

提供了海量可供选择的乐器类型，方便生成各种音色的乐曲：

```
instruments = {
    'Acoustic Grand Piano': 1,
    'Bright Acoustic Piano': 2,
    'Electric Grand Piano': 3,
    'Honky-tonk Piano': 4,
    'Electric Piano 1': 5,
    'Electric Piano 2': 6,
    'Harpsichord': 7,
    'Clavi': 8,
    'Celesta': 9,
    'Glockenspiel': 10,
    'Music Box': 11,
    'Vibraphone': 12,
    'Marimba': 13,
    'Xylophone': 14,
    'Tubular Bells': 15,
    'Dulcimer': 16,
    'Drawbar Organ': 17,
    'Percussive Organ': 18,
    'Rock Organ': 19,
    'Church Organ': 20,
    'Reed Organ': 21,
    'Accordion': 22,
    'Harmonica': 23,
    'Tango Accordion': 24,
    'Acoustic Guitar (nylon)': 25,
    'Acoustic Guitar (steel)': 26,
    'Electric Guitar (jazz)': 27,
    'Electric Guitar (clean)': 28,
    'Electric Guitar (muted)': 29,
    'Overdriven Guitar': 30,
    'Distortion Guitar': 31,
    'Guitar harmonics': 32,
    'Acoustic Bass': 33,
    'Electric Bass (finger)': 34,
    'Electric Bass (pick)': 35,
```

```
'Fretless Bass': 36,
'Slap Bass 1': 37,
'Slap Bass 2': 38,
'Synth Bass 1': 39,
'Synth Bass 2': 40,
'Violin': 41,
'Viola': 42,
'Cello': 43,
'Contrabass': 44,
'Tremolo Strings': 45,
'Pizzicato Strings': 46,
'Orchestral Harp': 47,
'Timpani': 48,
'String Ensemble 1': 49,
'String Ensemble 2': 50,
'SynthStrings 1': 51,
'SynthStrings 2': 52,
'Choir Aahs': 53,
'Voice Oohs': 54,
'Synth Voice': 55,
'Orchestra Hit': 56,
'Trumpet': 57,
'Trombone': 58,
'Tuba': 59,
'Muted Trumpet': 60,
'French Horn': 61,
'Brass Section': 62,
'SynthBrass 1': 63,
'SynthBrass 2': 64,
'Soprano Sax': 65,
'Alto Sax': 66,
'Tenor Sax': 67,
'Baritone Sax': 68,
'Oboe': 69,
'English Horn': 70,
'Bassoon': 71,
'Clarinet': 72,
'Piccolo': 73,
'Flute': 74,
'Recorder': 75,
'Pan Flute': 76,
'Blown Bottle': 77,
'Shakuhachi': 78,
'Whistle': 79,
'Ocarina': 80,
'Lead 1 (square)': 81,
'Lead 2 (sawtooth)': 82,
'Lead 3 (calliope)': 83,
'Lead 4 (chiff)': 84,
'Lead 5 (charang)': 85,
'Lead 6 (voice)': 86,
'Lead 7 (fifths)': 87,
'Lead 8 (bass + lead)': 88,
'Pad 1 (new age)': 89,
'Pad 2 (warm)': 90,
'Pad 3 (polysynth)': 91,
'Pad 4 (choir)': 92,
'Pad 5 (bowed)': 93,
'Pad 6 (metallic)': 94,
'Pad 7 (halo)': 95,
'Pad 8 (sweep)': 96,
'FX 1 (rain)': 97,
'FX 2 (soundtrack)': 98,
'FX 3 (crystal)': 99,
'FX 4 (atmosphere)': 100,
'FX 5 (brightness)': 101,
'FX 6 (goblins)': 102,
'FX 7 (echoes)': 103,
'FX 8 (sci-fi)': 104,
'Sitar': 105,
'Banjo': 106,
'Shamisen': 107,
'Koto': 108,
'Kalimba': 109,
'Bag pipe': 110,
'Fiddle': 111,
'Shanai': 112,
'Tinkle Bell': 113,
'Agogo': 114,
'Steel Drums': 115,
'Woodblock': 116,
'Taiko Drum': 117,
'Melodic Tom': 118,
'Synth Drum': 119,
'Reverse Cymbal': 120,
'Guitar Fret Noise': 121,
'Breath Noise': 122,
'Seashore': 123,
'Bird Tweet': 124,
'Telephone Ring': 125,
'Helicopter': 126,
'Applause': 127,
'Gunshot': 128
}
```

1.3.3 常用和弦

定义了一些人们常用的和弦的俗称与其配置的对应关系，增强了代码的可读性。

```
chordTypes = match({
    ('major', 'M', 'maj', 'majorthird'): ((4, 7), ),
    ('minor', 'm', 'minorthird', 'min', '-'): ((3, 7), ),
    ('maj7', 'M7', 'major7th', 'majorseventh'): ((4, 7, 11), ),
    ('m7', 'min7', 'minor7th', 'minorseventh', '-7'): ((3, 7, 10), ),
    ('7', 'seven', 'seventh', 'dominant seventh', 'dom7', 'dominant7', 'germansixth'):
```

```
((4, 7, 10), ),
('minormajor7', 'minor major 7', 'mM7'): ((3, 7, 11), ),
('dim', 'o'): ((3, 6), ),
('dim7', 'o7'): ((3, 6, 9), ),
('half-diminished7', 'ø7', 'ø', 'half-diminished', 'half-dim', 'm7b5'):
((3, 6, 10), ),
('aug', 'augmented', '+', 'aug3', '+3'): ((4, 8), ),
('aug7', 'augmented7', '+7'): ((4, 8, 10), ),
('augmaj7', 'augmented-major7', '+maj7', 'augM7'): ((4, 8, 11), ),
('aug6', 'augmented6', '+6', 'italian-sixth'): ((4, 10), ),
('frenchsixth', ): ((4, 6, 10), ),
('aug9', '+9'): ((4, 8, 10, 14), ),
('sus', 'sus4'): ((5, 7), ),
('sus2', ): ((2, 7), ),
('9', 'dominant9', 'dominant-ninth', 'ninth'): ((4, 7, 10, 14), ),
('maj9', 'major-ninth', 'major9th', 'M9'): ((4, 7, 11, 14), ),
('m9', 'minor9', 'minor9th', '-9'): ((3, 7, 10, 14), ),
('augmaj9', '+maj9', '+M9', 'augM9'): ((4, 8, 11, 14), ),
('add6', '6', 'sixth'): ((4, 7, 9), ),
('m6', 'minorsixth'): ((3, 7, 9), ),
('add2', '+2'): ((2, 4, 7), ),
('add9', ): ((4, 7, 14), ),
('madd2', 'm+2'): ((2, 3, 7), ),
('madd9', ): ((3, 7, 14), ),
('7sus4', '7sus'): ((5, 7, 10), ),
('7sus2', ): ((2, 7, 10), ),
('maj7sus4', 'maj7sus', 'M7sus4'): ((5, 7, 11), ),
('maj7sus2', 'M7sus2'): ((2, 7, 11), ),
('9sus4', '9sus'): ((5, 7, 10, 14), ),
('9sus2', ): ((2, 7, 10, 14), ),
('maj9sus4', 'maj9sus', 'M9sus', 'M9sus4'): ((5, 7, 11, 14), ),
('13sus4', '13sus'): ((5, 7, 10, 14, 21), (7, 10, 14, 17, 21)),
('13sus2', ): ((2, 7, 10, 17, 21), ),
('maj13sus4', 'maj13sus', 'M13sus', 'M13sus4'):
((5, 7, 11, 14, 21), (7, 11, 14, 17, 21)),
('maj13sus2', 'M13sus2'): ((2, 7, 11, 17, 21), ),
('add4', '+4'): ((4, 5, 7), ),
('madd4', 'm+4'): ((3, 5, 7), ),
('maj7b5', 'M7b5'): ((4, 6, 11), ),
('maj7#11', 'M7#11'): ((4, 7, 11, 18), ),
('maj9#11', 'M9#11'): ((4, 7, 11, 14, 18), ),
('69', '6/9', 'add69'): ((4, 7, 9, 14), ),
('m69', 'madd69'): ((3, 7, 9, 14), ),
('6sus4', '6sus'): ((5, 7, 9), ),
```

```
    ('6sus2', ): ((2, 7, 9), ),
    ('5', 'power chord'): ((7, ), ),
    ('5(+octave)', 'power chord(with octave)'): ((7, 12), ),
    ('maj11', 'M11', 'eleventh', 'major 11', 'major eleventh'):
    ((4, 7, 11, 14, 17), ),
    ('m11', 'minor eleventh', 'minor 11'): ((3, 7, 10, 14, 17), ),
    ('11', 'dominant11', 'dominant 11'): ((4, 7, 10, 14, 17), ),
    ('13', 'dominant13', 'dominant 13'): ((4, 7, 10, 14, 17, 21), ),
    ('maj13', 'major 13', 'M13'): ((4, 7, 11, 14, 17, 21), ),
    ('m13', 'minor 13'): ((3, 7, 10, 14, 17, 21), ),
    ('maj13#11', 'M13#11'): ((4, 7, 11, 14, 18, 21), ),
    ('13#11', ): ((4, 7, 10, 14, 18, 21), ),
    ('fifth_9th', ): ((7, 14), ),
    ('minormajor9', 'minor major 9', 'mM9'): ((3, 7, 11, 14), ),
    ('dim(Maj7)', ): ((3, 6, 11), )
})
```

1.4 Musicode 的基本数据类型

除了常见的数据类型外，根据中间代码和目标代码的特征，Musicode 还新增了 note、chord 和 setting 三种数据类型。

1.4.1 Note

Note 表示一个音符，其中包含了音名 (C, E, Gb, ... 一个表示音名的字符串)，八度数(和音名一起确定一个音的音高)，时长(音符长度，单位为小节) 和音量(音符的力度，范围为 0-127) 等信息。可以通过 1.3.1 节中的特定语义的字符串对音符进行初始化，也可以用 setting 对音符进行初始化，还可直接使用音高数进行初始化。

```
note a5 = “A5”;
note a5 = {"A", 5, 1};
note a5 = “A5”;
```

1.4.2 Chord

Chord 被定义为一组音符的集合，这个定义或许比乐理里面的和弦定义更为广义化，因为按照这个定义，一首完整的乐曲也可以完全装进和弦类里面。Chord 包含了音符(音符列表，为一个记载着这个和弦所有音符的列表)，时长(和弦的每个音符各自的音符长度)和间隔(每两个连续音符之间的间隔，单位为小节)。

```
chord Cmaj = "Cmaj";
chord bar1_melody1 = {"E4", "G4", "C5", "D5",
                      "D5", "E5", "G4", "G4"};
```

1.4.3 Setting

Setting 表示一个数，一个列表，或者列表的列表，用于%和@操作符对 note 或 chord 做出更改。

```
setting s1 = {1/8, 1/8, 1/8, 1/8,
              1/8+1/16, 3/16, 1/16, 1/16};
s1 = {s1, s1, 80};
bar1_melody1 = bar1_melody1 % s1;
Cmaj = Cmaj
       @ {1, 2, 3, 1.1, 2.1, 3, 1.1, 2.1}
       % {1/8, 1/8};
```

2 语言特性

2.1 文法

<乐段> → {(<表达式> | <播放语句> | <声明语句> | <打印语句>) ';'}

<标识符> → <字母> | <标识符><字母> | <标识符><数字>

<乐曲声明> → 'piece' <标识符> ['=' <配置>]

<配置列表> → <表达式> | '{' <表达式> {',' <表达式>} '}'

<配置> → <配置列表> | '{' <配置列表> {',' <配置列表>} '}'

<配置声明> → 'setting' <标识符> ['=' <配置>]

<音符声明> → 'note' <标识符> ['=' (<数> | <字符串>)]

<音符列表> → '{' (<字符串> | <标识符>) {',' (<字符串> | <标识符>)} '}'

<和弦声明> → 'chord' <标识符> ['=' (<字符串> | <音符列表>)]

<声明语句> → <乐曲声明> | <配置声明> | <音符声明> | <和弦声明>

<表达式> → <拼接表达式> | <一元表达式> <赋值运算符> <表达式>

<赋值运算符> → '=' | '/=' | '*=' | '%=' | '+=' | '-=' | '&=' | '|='

<拼接表达式> → <加法表达式>|<拼接表达式><拼接运算符><加法表达式>

<拼接运算符> → '|' | '&'

<增减表达式> → <修改表达式>|<增减表达式><增减运算符><修改表达式>

<增减运算符> → '+' | '-'

<修改表达式> → <一元表达式>|<修改表达式><修改运算符><一元表达式>

<修改运算符> → '/' | '*' | '%' | '@'

<一元表达式> → <后缀表达式> | {'+' | '-'} ['~'] <一元表达式> {'+' | '-'}

<后缀表达式> → <基本表达式> | <后缀表达式> '[' <表达式> ']'

<基本表达式> → <标识符> | <配置> | <数> | <字符串> | '(' <表达式> ')'

<播放语句> → 'play' '(' <标识符> ')' | 'score' '(' <标识符> ')'

<打印语句> → 'print' '(' <标识符> ')'

<数> → <数字> {<数字>} ['.' <数字> {<数字>}]

<字母> → A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V |

W | X | Y | Z | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z

<数字> → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

2.2 保留字及运算符

2.2.1 保留字

Musicode 语言共有 7 个保留字，分别为：

note	chord	play	piece
setting	print	score	

2.2.2 运算符

Musicode 语言的运算符及优先级如下表(从高到低排列)：

优先级	运算符	名称或含义	结合方向	说明
1	[]	数组下标	左到右	
	()	圆括号		
2	-	负号运算符(音高减少半音)	右到左	单目运算符
	+	正号运算符(音高增加半音)		单目运算符
	~	和弦倒序		单目运算符
3	/	除(分数)；和弦转位	左到右	双目运算符
	*	和弦重复		双目运算符
	%	对每个音进行修改		双目运算符
	@	取和弦的音进行配置		双目运算符
4	+	增加音高；和弦内增加音符	左到右	双目运算符
	-	降低音高；和弦内减少音符		双目运算符
5	&	和弦对齐拼接	左到右	双目运算符
	|	和弦间隔拼接		双目运算符

6	=	赋值运算符	右到左	赋值运算符
	/=	除(分数)；和弦转位后赋值		赋值运算符
	*=	和弦重复后赋值		赋值运算符
	%=	对每个音进行修改后赋值		赋值运算符
	+=	增加音高、音符后赋值		赋值运算符
	-=	减少音高、音符后赋值		赋值运算符
	&=	和弦对齐拼接后赋值		赋值运算符
	|=	和弦间隔拼接后赋值		赋值运算符
7	,	逗号运算符	左到右	

2.3 中间代码

中间代码中每个单元的格式为 “音高[时长，间隔，音量]”，详见 4.3 节。

2.4 目标代码

目标代码的格式为音乐描述文件 midi。midi 文件是一种用于记录音乐的二进制文件，与波形文件不同，midi 文件不对音乐进行抽样，而是对音乐的每个音符记录为一个数字，所以与波形文件相比文件要小得多，可以满足长时间音乐的需要。MIDI 标准规定了各种音调的混合及发音，通过输出装置可以将这些数字重新合成为音乐。

五线谱，是世界上通用的一种记谱法，通过在五根等距离的平行横线上标以不同时值的音符及其他记号来记载音乐。在本语言中可以通过中间代码生成五线谱文件，详见 4.4 节。

3 指称语义

3.1 本语言使用到的域

3.1.1 基本域

Number：域元素为实数

Character：域元素取自某字符集

Pitch-Name(音名) = C，C#(Db)，D，D#(Eb)，E，F，F#(Gb)，G，G#(Ab)，A，A#(Bb)，B

Register(音区) = 1，2，3，4，5，6，7，8

3.1.2 笛卡尔积域

Pitch(音高) = Pitch-Name × Register

Note(音符) = Pitch × Number(时长)

3.1.3 联合域

Tone(音)：Note(Pitch × Number) + Interval(Number)

Setting_List(配置列表) = Tone × … × Tone

Setting(配置) = Setting_List × … × Setting_List

3.1.4 序列域

Chord(和弦)中的元素是选自域 Tone 中的元素的有限序列，即同构序列元素。或为 nil 元素，或为 x • s，其中元素 x ∈ Tone，序列 s ∈ Chord。

String(字符串)中的元素是选自域 Character 中的元素的有限序列。

3.1.5 存储域

Store：所有存储状态的集合，元素为 sto

Location：存储单元，元素为 loci

Storable：可储值域，元素为 stble

3.1.6 环境域

Environ：变量的作用域，本语言中只有全局作用域，元素为 env

Bindable：标识符束定的值，元素为 bdble

Identifier：标识符，元素为 I

3.2 语义域，语义函数及辅助函数：（S ∈ Statement，E ∈ Expression，D ∈ Declaration）

Value = note Note + chord Chord + string String + number Number + setting Setting

Storable = Value

Bindable = value Value + variable Location (值和变量是可束定体)

3.2.1 语句执行的语义函数

execute：Statement → (Environ → Store → Store)

语句的执行是将 Statement 域中的元素(一条命令 S)映射为程序状态的改变，即某一环境 env 下的 sto 映射为另一 sto'。

execute〖S〗env sto = sto’

3.2.2 表达式求值的语义函数是

evaluate：Expression → (Environ → Store → Value)

每个表达式的求值是将 Expression 域中的元素 E 映射为从环境中取出改变了状态值，即在某一环境 env 的 sto 下求值 value。

evaluate〖E〗 env sto= …

3.2.3 声明确立的语义函数

elaborate: Declaration→ (Environ → Store → Environ × store)

每个声明的确立是将 Declaration 域中的元素 D 映射为将环境 env 下 sto 变为有了新束定的环境 env' × sto'。

elaborate〖D〗 env sto = (env', sto')

3.2.4 辅助函数

Store = Location → (stored Storable + undefined(未定义) + unused(未使用))

empty_store：Store (所有元素都映射为 unused 的 Store 元素)

empty_store = λloc.unused

allocate：Store → Store × Location (将 sto 映射为 sto'，其中某些 loc 从未使用变为未定义)

```
allocate sto =
    let loc = any_unused_location (sto) in
        (sto [loc→ undefined]，loc)
```

deallocate：Store × Location → Store (将 sto 映射为 sto'，其中某些 loc 变为未使用)

deallocate (sto，loc) = sto [loc → unused]

update：Store × Location × Storable → Store (将 sto 映射为 sto'，sto 中的 loc 单元更新为 stble 值)

update (sto，loc，stble) = sto [loc→stored stble]

fetch：Store × Location → Storable (从 sto 的 loc 单元上取出 stble 值)

```
fetch (sto，loc) =
    let stored_value (stored stble) = stble
```

stored_value (undefined) = fail

stored_value (unused) = fail

in

stored_value (sto(loc))

Environ = Identifier → (bound Bindable + unbound)

empty_environ：Environ (所有标识符均处于未束定状态的空环境)

enpty_environ = λI. unbound

bind：Identifier × Bindable → Environ (只要有标识符束定于可束定体，环境就改变了)

*bind(I,bdble) = λI'. **if** I'=I **then** bound bdble **else** unbound*

overlay：Environ × Environ → Environ (两束定环境组成新环境 evn + evn'，若某标识符在两环境中均有束定，则新环境中以后束定复盖先束定)

overlay (env'，env) =

*λI. **if** env'(I) /=unbound **then** env'(I) **else** env(I)*

find：Environ × Identifier → Bindable (在环境 env 中找出标识符 I 所对应的可束定体 bdble)

find(env,I) =

***let** bound_value(bound bdble) = bdble*

bound_value(unbound) = ⊥

in

bound_value(env(I))

coerce：Store × Bindable → Value (获取已束定变量的值)

coerce(sto，find(env，I)) = fetch(sto，loc)

3.3 各语法短语的语义

3.3.1 数和标识符的语义等式

evaluate〖N〗env sto = double(valuation N)

evaluete〖I〗env sto = coerce(sto，find(env，I))

3.3.2 语句短语语义等式

<语句> → <赋值语句> | <声明语句>

执行赋值语句。在 env sto 中对 E 求值，并放入临时变元 val 中；再在 env 找出 I 束定的单元放于临时变元 loc 中；最后以 sto，loc，val 作参数调用辅助函数 update，loc 中的值被修改，I 因而得赋值。

execute〖I = E〗env sto =

let val = evaluate E env sto **in**

let variable loc = find(env，I) **in**

update(sto，loc，val)

执行声明语句。在 env sto 中确立声明 D。由于确立 D 既改变了 env 又改变了 sto，则将它们作为序偶取出，覆盖原有环境 env 和存储 sto。

execute〖D〗 env sto =

let (env', sto') = elaborate D env sto **in**

env', sto'

连续执行多条语句。连续执行两命令的语义是：第一命令 S1 在 env sto 下执行，不会改变束定的环境，只改变存储。S2 在 S1 改变了的存储(括号内执行结果)中执行。

execute〖S1; S2〗env sto =

execute S2 env(execute S1 env sto)

3.3.3 表达式语义等式

```
<表达式>  →  <表达式> | <表达式>
          | <表达式> & <表达式>
          | <表达式> @ <表达式>
          | <表达式> % <表达式>
          | <表达式> * <表达式>
          | <表达式> / <表达式>
          | <表达式> + <表达式>
          | <表达式> - <表达式>
          | + <表达式>
          | - <表达式>
```

和弦的拼接“|”。得到音乐片段 B 追加到音乐片段 A 之后的新的音乐片段 concatenate(A, B)。

evaluate〖E1 | E2〗env sto =
    let chord c1 = evaluate E1 env sto **in**
    let chord c2 = evaluate E2 env sto **in**
    chord(concatenate (c1，c2))

和弦的合并“&”。得到音乐片段 B 和音乐片段 A 合并之后的新的音乐片段 merge(A, B)。

evaluate〖E1 & E2〗env sto =
    let chord c1 = evaluate E1 env sto **in**
    let chord c2 = evaluate E2 env sto **in**
    chord(merge (c1，c2))

和弦添加音符“+”。得到音乐片段 A 添加音符 b 后的音乐片段 add(A, b)。

evaluate〖E1 + E2〗env sto =
    let chord c1 = evaluate E1 env sto **in**
    let note c2 = evaluate E2 env sto **in**
    chord(add (c1，c2))

和弦去除音符“-”。得到音乐片段 A 去除音符 b 后的音乐片段 minus(A, b)。

evaluate〖E1 - E2〗env sto =
    let chord c1 = evaluate E1 env sto **in**
    let note c2 = evaluate E2 env sto **in**
    chord(minus (c1，c2))

和弦重复“*”。得到音乐片段 A 重复 n 次后的音乐片段 repeat(A, n)。

```
evaluate〖E1 * E2〗env sto =
    let chord c1 = evaluate E1 env sto in
    let number c2 = evaluate E2 env sto in
    chord(repeat ( c1，c2 ))
```

和弦转位“/”。得到音乐片段 A 中音符转 n 位的音乐片段 reverse(A, n)。

```
evaluate〖E1 / E2〗env sto =
    let chord c1 = evaluate E1 env sto in
    let number c2 = evaluate E2 env sto in
    chord(reverse ( c1，c2 ))
```

和弦修改“%”。得到音乐片段 A 根据配置 s 修改后的音乐片段 modify(A, s)。

```
evaluate〖E1 % E2〗env sto =
    let chord c1 = evaluate E1 env sto in
    let setting s2 = evaluate E2 env sto in
    chord(modify ( c1，s2 ))
```

和弦配置“@”。得到音乐片段 A 根据配置 s 配置后的音乐片段 setting(A, s)。

```
evaluate〖E1 @ E2〗env sto =
    let chord c1 = evaluate E1 env sto in
    let setting s2 = evaluate E2 env sto in
    chord(setting ( c1，s2 ))
```

双目音符升调“+”。得到音符 a 升 n 个半音后的音符 rise(a, n)。

```
evaluate〖E1 + E2〗env sto =
    let note c1 = evaluate E1 env sto in
    let number c2 = evaluate E2 env sto in
    note(rise ( c1，c2 ))
```

单目音符升调“+”。得到音符 a 升 1 个半音后的音符 rise(a)。

```
evaluate〖+ E1〗env sto =
    let note c1 = evaluate E1 env sto i
    note(rise ( c1 ))
```

双目音符降调“-”。得到音符 a 降 n 个半音后的音符 fall(a, n)。

```
evaluate〖E1 - E2〗env sto =
    let note c1 = evaluate E1 env sto in
    let number c2 = evaluate E2 env sto in
    note(fall ( c1，c2 ))
```

单目音符降调“-”。得到音符 a 降 1 个半音后的音符 fall(a)。

```
evaluate〖- E1〗env sto =
    let note c1 = evaluate E1 env sto i
    note(fall ( c1 ))
```

双目和弦升调“+”。得到音乐片段 A 中音符升 n 个半音后的音乐片段 rise(A,n)。

```
evaluate〖E1 + E2〗env sto =
    let chord c1 = evaluate E1 env sto in
    let number c2 = evaluate E2 env sto in
    chord(rise ( c1，c2 ))
```

单目和弦升调“+”。得到音乐片段 A 中音符升 1 个半音后的音乐片段 rise(A)。

```
evaluate〖+ E1〗env sto =
    let chord c1 = evaluate E1 env sto in
    chord(rise ( c1 ))
```

双目和弦降调“-”。得到音乐片段 A 中音符降 n 个半音后的音乐片段 fall(A,n)。

```
evaluate〖E1 - E2〗env sto =
    let chord c1 = evaluate E1 env sto in
    let number c2 = evaluate E2 env sto in
    chord(fall ( c1，c2 ))
```

单目和弦降调“-”。得到音乐片段 A 中音符降 1 个半音后的音乐片段 fall (A)。

```
evaluate〖- E1〗env sto =
    let chord c1 = evaluate E1 env sto in
    chord(fall ( c1 ))
```

将和弦看成数组时，数组变量的语义描述。

component：Number × Chord → Note (和弦索引)

component(number, nil) = ⊥

component(number, var · arrvar) =

if number = 0 then var

else component(prodecessor(number, arrvar))

fetch_chord：Store × Identifier → Chord

fetch_chord (sto，nil) = nil

fetch_chord (sto，var · arrvar) =

fetch(sto，var) · fetch_array (sto，arrvar)

update_chord：Store × Identifier × Chord → Store

update_chord (sto，nil，nil) = sto

update_chord (sto，var·arrvar，val·arrval) =

let sto'= update (sto，var，val) in

update_chord (sto'，arrvar，arrval)

4 作曲过程

4.1 Musicode 源代码

选取北航校歌的一个乐句作为例子，该乐句有两个声部(melody1 和 melody2)，每个声部四个小节(s1, s2, s3 和 s4)，并添加一个伴奏(accompany)，得到如下的源代码：

```
//第一小节
setting s1 = {1/8, 1/8, 1/8, 1/8, 1/8+1/16, 3/16, 1/16, 1/16};
                                        //第一小节音符时长和间隔的配置
s1 = {s1, s1, 80};                      //{时长, 间隔, 音量}
chord bar1_melody1 = {"E4", "G4", "C5", "D5", "D5", "E5", "G4",
"G4"};                                  //第一小节一声部的音名
bar1_melody1 = bar1_melody1 % s1;       //对音名进行配置生成旋律
chord bar1_melody2 = {"E3", "G3", "G3", "G3", "G3", "C4", "E3",
"E3"};                                  //第一小节二声部的音名
bar1_melody2 = bar1_melody2 % s1;       //对音名进行配置生成旋律

//第二小节
setting s2 = {1/8, 1/8, 1/8, 1/16, 1/16, 1/8+1/16, 5/16};
                                        //第二小节音符时长和间隔的配置
s2 = {s2, s2, 80};                      //{时长, 间隔, 音量}
chord bar2_melody1 = {"A4", "A4", "A4", "C5", "C5", "D5", "D5"};
                                        //第二小节一声部的音名
bar2_melody1 = bar2_melody1 % s2;       //对音名进行配置生成旋律
chord bar2_melody2 = {"F3", "F3", "F3", "F3", "A3", "B3", "B3"};
                                        //第二小节二声部的音名
bar2_melody2 = bar2_melody2 % s2;       //对音名进行配置生成旋律
```

```
//第三小节
setting s3 = {1/4, 1/8, 1/16, 1/16, 1/8, 1/8+1/16, 1/16, 1/16,
1/16};
s3 = {s3, s3, 80};
chord bar3_melody1 = {"C5", "C5", "A4", "G4", "E4", "G4", "E4",
"E4", "G4"};
bar3_melody1 = bar3_melody1 % s3;
chord bar3_melody2 = {"A3", "A3", "E3", "E3", "C3", "E3", "C3",
"C3", "E3"};
bar3_melody2 = bar3_melody2 % s3;

//第四小节
setting s4 = {1/4, 1/8+1/16, 1/16, 1/8, 3/8};
s4 = {s4, s4, 80};
chord bar4_melody1 = {"A4", "G4", "E4", "G4", "D4"};
bar4_melody1 = bar4_melody1 % s4;
chord bar4_melody2 = {"F3", "E3", "C3", "D3", "B2"};
bar4_melody2 = bar4_melody2 % s4;

//将四小节旋律合并到一起
chord melody1 = bar1_melody1 | bar2_melody1 | bar3_melody1 |
bar4_melody1;
chord melody2 = bar1_melody2 | bar2_melody2 | bar3_melody2 |
bar4_melody2;

//添加伴奏
chord Cmaj = "Cmaj";                    //创建 C 和弦并进行配置
Cmaj = Cmaj @ {1, 2, 3, 1.1, 2.1, 3, 1.1, 2.1} % {1/8, 1/8};
chord Fmaj = "Fmaj";                    //创建 F 和弦并进行配置
Fmaj = Fmaj @ {1, 2, 3, 1.1} % {1/8, 1/8};
chord Gmaj1 = "Gmaj";                   //创建 G 和弦并进行配置
Gmaj1 = Gmaj1 @ {1, 2, 3, 1.1} % {1/8, 1/8};
chord Amin = "Amin";                    //创建小 A 和弦并进行配置
Amin = Amin @ {1, 2, 3, 1.1, 2.1, 3, 1.1, 2.1} % {1/8, 1/8};
chord Gmaj2 = "Gmaj";                   //创建 G 和弦并进行配置
Gmaj2 = Gmaj2 @ {1, 2, 3, 1.1, 2.1, 3, 1.1, 2.1} % {1/8, 1/8};
chord accompany = Cmaj - 12 | Fmaj - 24 | Gmaj1 - 24 | Amin - 24
| Gmaj2 - 12;                           //对和弦进行降调和拼接

piece c;                                //将两个和弦和伴奏并列生成乐曲
c = {{melody1, melody2, accompany}, {74, 69, 1}, 60};
play(c);                                //演奏乐曲并生成 midi 文件
score(c);                               //生成五线谱
```

4.2 语法树

根据源代码可生成语法树(由于语法树过大，此处只保留第一小节)：

```
  /-setting s1 =
 |
 |     /-1
 |    |
 |  /-|--/
 |  |  |
 |  |  \-8
 |  |
 |  |  /-1
 |  |  |
 |  |--|--/
 |  |  |
 |  |  \-8
 |  |
/-|  |  /-1
|  |  |  |
|  |  |--|--/
|  |  |  |
|  |  |  \-8
|  |  |
|  |  |  /-1
|  |  |  |
|  |  |--|--/
|  |  |  |
|  |  |  \-8
|  |  |
|  |  |     /-1
|  |  |     |
|  |  |  /-|--/
|  \-|  |  |
|    |  |  \-8
|    |  |
|    |--|--+
|    |  |
|    |  |  /-1
|    |  |  |
|    |  \-|--/
|    |    |
|    |     \-16
|    |
|    |  /-3
|    |  |
|    |--|--/
|    |  |
|    |  \-16
|    |
|    |  /-1
|    |  |
|    |--|--/
|    |  |
|    |  \-16
|    |
|    |  /-1
|    |  |
|     \-|--/
|       |
|        \-16
|
|  /-s1
|  |
|  |--=
|--|
|  |  /-s1
|  |  |
|  \-|--s1
|    |
|     \-80
|
|  /-chord bar1_melody1 =
|  |
|  |  /-E4
|  |  |
|--|  |--G4
|  |  |
|  |  |--C5
|  |  |
|  |  |--D5
|  \-|
```

```
|     |--D5
|     |
|     |--E5
|     |
|     |--G4
|     |
|      \-G4
|
|   /-bar1_melody1
|  |
|  |--=
|--|
|  |   /-bar1_melody1
|  |  |
|   \-|--%
|     |
|      \-s1
|
|   /-chord bar1_melody2 =
|  |
|  |   /-E3
|  |  |
|--|  |--G3
|  |  |
|  |  |--G3
|  |  |
|  |  |--G3
|   \-|
|     |--G3
|     |
|     |--C4
|     |
|     |--E3
|     |
|      \-E3
|
|   /-bar1_melody2
|  |
--|  |--=
  |--|
  |  |   /-bar1_melody2
  |  |  |
  |   \-|--%
  |     |
```

```
|      \-s1
|
|   /-chord melody1 =
|--|
|   \-bar1_melody1
|
|   /-chord melody2 =
|--|
|   \-bar1_melody2
|
|   /-chord Cmaj =
|--|
|   \-Cmaj
|
|   /-Cmaj
|  |
|  |--=
|  |
|  |      /-Cmaj
|  |     |
|  |     |--@
|  |     |
|  |     |   /-1
|  |     |  |
|  |     |  |--2
|  |     |  |
|  |     |  |--3
|  |   /-|  |
|  |  |  |  |   /-1
|--|  |  |  |  |
|  |  |  |  |--|--.
|  |  |  |  |  |  |
|  |  |  |  |  |   \-1
|  |  |  |  |  |
|  |  |  |  |  |   /-2
|  |  |  |  |  |  |
|  |  |  |   \-|--|--.
|  |  |  |     |  |
|  |  |  |     |   \-1
|  |  |  |     |
|  |  |  |     |--3
|  |  |  |     |
|  |  |  |     |   /-1
|  |  |  |     |  |
```

```
|   \-|     |--|--.                  |  |  |   /-74
|     |     |  |                     |   \-|  |
|     |     |   \-1                  |     |--|--69
|     |     |                        |     |  |
|     |     |   /-2                  |     |   \-1
|     |     |  |                     |     |
|     |      \-|--.                  |      \-60
|     |        |                     |
|     |         \-1                   \-play - c
|     |
|     |--%
|     |
|     |      /-1
|     |     |
|     |   /-|--/
|     |  |  |
|     |  |   \-8
|      \-|
|        |   /-1
|        |  |
|         \-|--/
|           |
|            \-8
|
|   /-chord accompany =
|  |
|--|   /-Cmaj
|  |  |
|   \-|---
|     |
|      \-12
|
|--piece c
|
|   /-c
|  |
|  |--=
|  |
|  |     /-melody1
|--|     |
|  |   /-|--melody2
|  |  |  |
|  |  |   \-accompany
|  |  |
```

4.3 中间代码

中间代码中每个单元的格式为 “音高[时长，间隔，音量]”，如下：

音轨 1: E4[1/8;1/8;80], G4[1/8;1/8;80], C5[1/8;1/8;80], D5[1/8;1/8;80], D5[3/16;3/16;80], E5[3/16;3/16;80], G4[1/16;1/16;80], G4[1/16;1/16;80], A4[1/8;1/8;80], A4[1/8;1/8;80], A4[1/8;1/8;80], C5[1/16;1/16;80], C5[1/16;1/16;80], D5[3/16;3/16;80], D5[5/16;5/16;80], C5[1/4;1/4;80], C5[1/8;1/8;80], A4[1/16;1/16;80], G4[1/16;1/16;80], E4[1/8;1/8;80], G4[3/16;3/16;80], E4[1/16;1/16;80], E4[1/16;1/16;80], G4[1/16;1/16;80], A4[1/4;1/4;80], G4[3/16;3/16;80], E4[1/16;1/16;80], G4[1/8;1/8;80], D4[3/8;3/8;80]
音轨 2: E3[1/8;1/8;80], G3[1/8;1/8;80], G3[1/8;1/8;80], G3[1/8;1/8;80], G3[3/16;3/16;80], C4[3/16;3/16;80], E3[1/16;1/16;80], E3[1/16;1/16;80], F3[1/8;1/8;80], F3[1/8;1/8;80], F3[1/8;1/8;80], F3[1/16;1/16;80], A3[1/16;1/16;80], B3[3/16;3/16;80], B3[5/16;5/16;80], A3[1/4;1/4;80], A3[1/8;1/8;80], E3[1/16;1/16;80], E3[1/16;1/16;80], C3[1/8;1/8;80], E3[3/16;3/16;80], C3[1/16;1/16;80], C3[1/16;1/16;80], E3[1/16;1/16;80], F3[1/4;1/4;80], E3[3/16;3/16;80], C3[1/16;1/16;80], D3[1/8;1/8;80], B2[3/8;3/8;80]
音轨 3: C3[1/8;1/8;100], E3[1/8;1/8;100], G3[1/8;1/8;100], C4[1/8;1/8;100], E4[1/8;1/8;100], G3[1/8;1/8;100], C4[1/8;1/8;100], E4[1/8;1/8;100], F2[1/8;1/8;100], A2[1/8;1/8;100], C3[1/8;1/8;100], F3[1/8;1/8;100], G2[1/8;1/8;100], B2[1/8;1/8;100], D3[1/8;1/8;100], G3[1/8;1/8;100], A2[1/8;1/8;100], C3[1/8;1/8;100], E3[1/8;1/8;100], A3[1/8;1/8;100], C4[1/8;1/8;100], E3[1/8;1/8;100], A3[1/8;1/8;100], C4[1/8;1/8;100], G3[1/8;1/8;100], B3[1/8;1/8;100], D4[1/8;1/8;100], G4[1/8;1/8;100], B4[1/8;1/8;100], D4[1/8;1/8;100], G4[1/8;1/8;100], B4[1/8;1/8;100]

4.4 目标代码

目标代码为相应的 midi 文件：temp.mid，以及五线谱。