Rock Your Composition

**PRESENTERS**

- **Project Manager**: Xiaoji Li (xl2336)
- **Language Guru**: Junyang Lu (jl3937)
- **System Architect**: Ruizhi Liao (rl2643)
- **System Integrator**: Dong Li (dl2768)
- **Tester**: Zhuojun Huang (zh2193)
INTRODUCTION

RYC: rock your composition

tempo = 300;
key = #3;
song = m2 a b c 1;
a = [ _1 #3 _5 ^3 | ];
b = [&6 _1 ^4 _1 | ];
c = [(mordent _5 4) | ];
e = [(1, 2) (chord 2) | ];
m2 a b c d = [
a a a b
b c a a
a b a b
b c d 1 (1, / 1 2) (1, / 1 2) 1 1 |
a e a a
b a a e 11
a e a a
b a e (_1, 2) (1, 2) |
];
chord p d = {
(p, d)
(+ p 2, d)
(+ p 4, d)
(+ p 7, d) 
};
mordent p d = [ (p, / d 8)
(+ p 1, / d 8) (p, * 3 (/ d 4)) ];
INTRODUCTION

RYC: rock your composition

There are two ways of musical notation: numerical and staff

The staff notation is rather complicated, especially for one without professional musical background.

We present RYC, the language for computer scientists and music lovers to play with music.

With RYC, one can write a piece of melody, do synchronization (parallelism). Since it's purely functional, you can even do lambda calculus.

key = #2;

song = [1 2 3 4 | 5 6 7 \^1];
INTRODUCTION

RYC: rock your composition

MUSICAL NOTATION CONVENTION:
0 ; REST  ^ : OTTAVA ALTA
1-7; NOTE _ : OTTAVA SOTTO
(6,8) : a note with a duration in the unit of basic length of the time signature
# : SHARP & : FLAT

key = #2;
song = [ 1 2 3 4 | 5 6 7 ^1 ];
The corresponding staff notation is:
tempo = 300;
key = 5;
song = repeat 100 m;

repeat times melody =
if (<= times 0)
  []
  (seq melody (repeat (- times 1) melody));

m = { [ #1 0 3 1] ^[ (_5, 4)]} ;
Main Features

\textbf{RYC: rock your composition}

tempo = 300;
key = 5;
song = repeat 100 m;
repeat times melody =
if (<= times 0)
  []
  (seq melody (repeat (- times 1) melody));
m = { [ #1 0 3 1] ^[ (_5, 4)]} ;

1 – 7: do - ti
0: rest
#: half tone higher
&: half tone lower
^: an octave higher
_: an octave lower
`tempo = 300;`  
`key = 5;`  

`song = repeat 100 m;`  

`repeat times melody =`  
`if (<= times 0)`  
`    []`  
`    (seq melody (repeat (- times 1) melody));`  

`m = { [ #1 0 3 1] ^[(_5, 4)] } ;`
Main Features

**RYC: rock your composition**

tempo = 300;
key = 5;
song = repeat 100 m;
repeat times melody =
if (<= times 0)
  []
  (seq melody (repeat (- times 1) melody));
m = { [ #1 0 3 1] [ (_5, 4)]};

[: Sequential melody]
Main Features

**RYC: rock your composition**

```
tempo = 300;
key = 5;
song = repeat 100 m;
repeat times melody =
if (<= times 0)
    []
    (seq melody (repeat (- times 1) melody));
m = { [ #1 0 3 1] ^[ (_5, 4)]} ;
```

{(): Parallel melody}
Main Features

**RYC: rock your composition**

tempo = 300;
key = 5;
song = repeat 100 m;

```
repeat times melody =
if (<= times 0)
    []
    (seq melody (repeat (- times 1) melody));
m = { [ #1 0 3 1] ^[ (_5, 4)]} ;
```
tempo = 300;
key = 5;
song = repeat 100 m;
repeat times melody =
if (<= times 0)
[
(seq melody (repeat (- times 1) melody));
m = { [ #1 0 3 1] ^[ (_5, 4)]};
Input: Source Program

**RYC**: rock your composition

\[
\text{song} = \{ [1 (f \ 4)] \ [(\&3, \ 2)] \} ; \\
f \ x = + \ x \ 1 ;
\]
Token Stream

**RYC:** rock your composition

```plaintext
song = { [ 1 ( f 4 ) ] [ ( & 3 , 2 ) ] } ; f x = + x 1 ;
```

---

**Lexer**  
**Parser**  
**Lambda Calculus Evaluator**  
**Time and Channel Allocator**  
**MIDI Generator**
**AST**

**RYC: rock your composition**

```
song = {[1 (f 4)] [(&3, 2)]} ;
f x = + x 1 ;
```
Melody Tree

**RYC: rock your composition**

![Melody Tree Diagram]

- **Lexer**
- **Parser**
- **Lambda Calculus Evaluator**
- **Time and Channel Allocator**
- **MIDI Generator**
## MIDI Event List

**RYC: rock your composition**

<table>
<thead>
<tr>
<th>Pitch</th>
<th>Start time</th>
<th>End time</th>
<th>Channel</th>
</tr>
</thead>
<tbody>
<tr>
<td>60</td>
<td>0</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>63</td>
<td>0</td>
<td>200</td>
<td>1</td>
</tr>
<tr>
<td>67</td>
<td>100</td>
<td>200</td>
<td>0</td>
</tr>
</tbody>
</table>
Output: MIDI file

RYC: rock your composition

a.mid

Lexer -> Parser -> Lambda Calculus Evaluator -> Time and Channel Allocator -> MIDI Generator
MIDI LIBRARY-JDKSMIDI

RYC: rock your composition

MIDI Event: msg

```c
for (size_t i = 0; i < intervals.size(); ++i) {
    assert(intervals[i].channel != -1);
    msg.SetTime((unsigned long)(intervals[i].start + 0.5));
    msg.SetNoteOn(intervals[i].channel, intervals[i].note, 100);
    tracks.GetTrack(1)->PutEvent(msg);
    msg.SetTime((unsigned long)(intervals[i].end + 0.5));
    msg.SetNoteOff(intervals[i].channel, intervals[i].note, 100);
    tracks.GetTrack(1)->PutEvent(msg);
}
```

From gen_midi.c
DEVELOPMENT ENVIRONMENT

**RYC: rock your composition**

- Lexing and Parsing: lex, yacc
- Online Editor: Google Drive, Google Group
- Text Editors: Sublime, Xcode
- Version Control: Dropbox
- Test Suite: Make file for test suite of RYC files
UNIT TESTING

AST

\[ \text{song} ::= \{ [ (1.0, 1.0) \ (f \ 4.0) ] \\ [ (3.0, 2.0) ] \} ; \]
\[ f ::= (\lambda x \to ((+ x) \ 1.0)) ; \]

Melody Tree

\[ f\{ [ (1.0, 1.0) (5.0, 1.0) ] \\ [ (3.0 - 1, 2.0) ] \} ; \]

INTEGRATION TESTING

- sequential melody
- parallel melody
- user-defined funcion
- built-in funcion
TEAM MANAGEMENT

RYC: rock your composition

Timeline
Lessons learnt

**RYC: rock your composition**

**Junyang:** the idea of lambda calculus and easy evaluation fascinated me

**Dong:** A lot of stuff needs to be taken into consideration when building a functional language

**Ruizhi:** The semantic definition of each production is very tricky and educational

**Zhuojun:** I have better understanding about different phases of compiler

**Xiaoji:** It's been a lot of fun working with these people
Thank you!

PRESENTERS

Project Manager  
Xiaoji Li(xl2336)

Language Guru  
Junyang Lu(jl3937)

System Architect  
Ruizhi Liao(rl2643)

System Integrator  
Dong Li(dl2768)

Tester  
Zhuojun Huang(zh2193)