MLang : Music Language

Team members

Language Guru
Aditya Mukerjee

System Architect/Integrator
Nikhil Sarda

System Tester
Nishtha Agarwal

Project Manager
Sushmita Swaminathan

© Original Artist
Reproduction rights obtainable from
www.CartoonStock.com

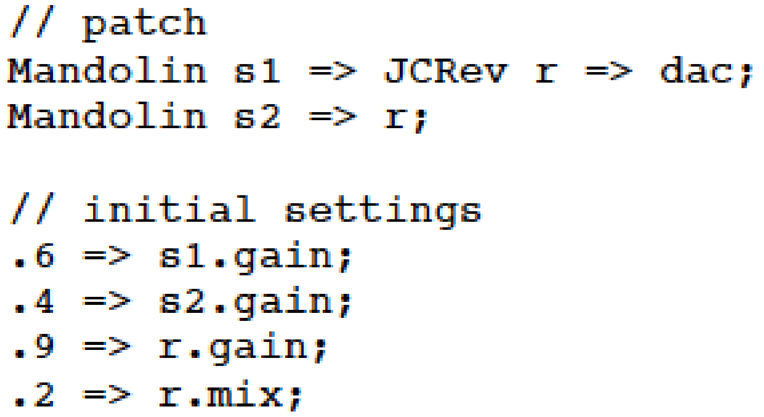
What was the motivation?

- We wanted to create language that could synthesize music not just modify it
- It also had to be simple for non-programmers to use and programmers to extend

Existing Music Synthesis Languages

ChucK – Music synthesis language- loosely C-like object oriented language

Sample program



```
// patch
Mandolin s1 => JCRev r => dac;
Mandolin s2 => r;

// initial settings
.6 => s1.gain;
.4 => s2.gain;
.9 => r.gain;
.2 => r.mix;
```

```
// Play a fragment
fun void playVoice(Mandolin m, int voice[][], int transport) {
    for( 0 => int i; i < voice.cap(); i++) {
        if ( voice[i][0] > 0 ) {
            std.mtof( voice[i][0] + transport ) => m.freq;
            1.0 => m.pluck;
        }
        duration[voice[i][1]] => now;
    }
    finish.broadcast();
}

// Main: play the whole song
spork ~ playVoice(s1, voice1, 0);
spork ~ playVoice(s2, voice2, -12);
finish => now;
```

This is what we wanted to avoid

© Ron Leishman * www.ClipartOf.com/439927

© Dennis Cox * www.ClipartOf.com/6261

Can we do it in a more sane manner?

- Homoiconicity

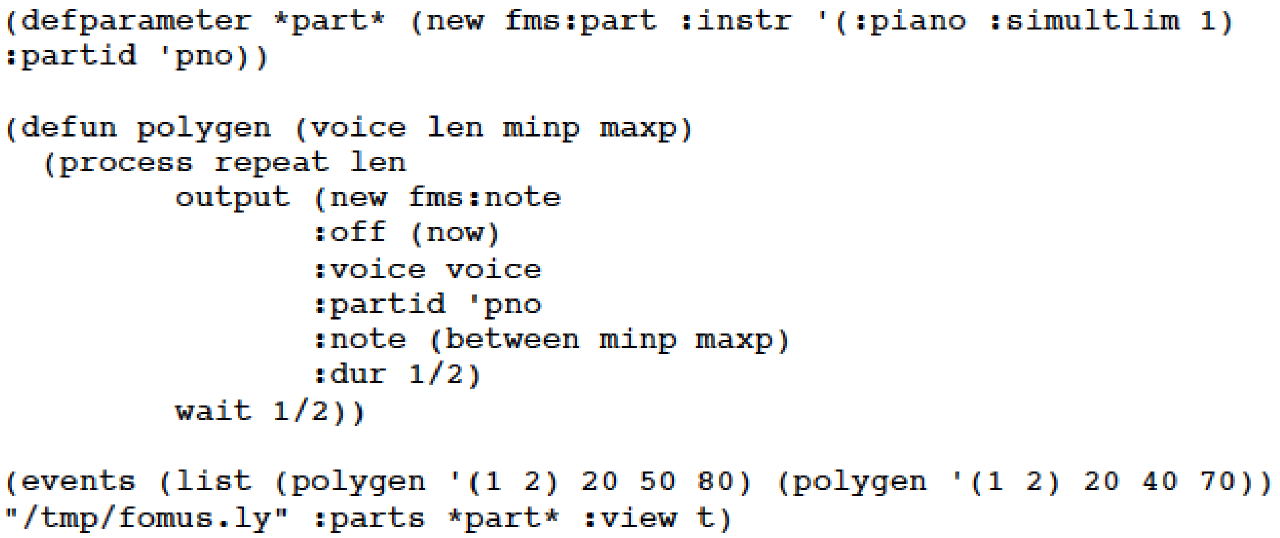
The program is a representation of the data itself

- Modularity

Programmers can each write their own modules that can be used and transformed by others

Other Music languages based on Functional Programming

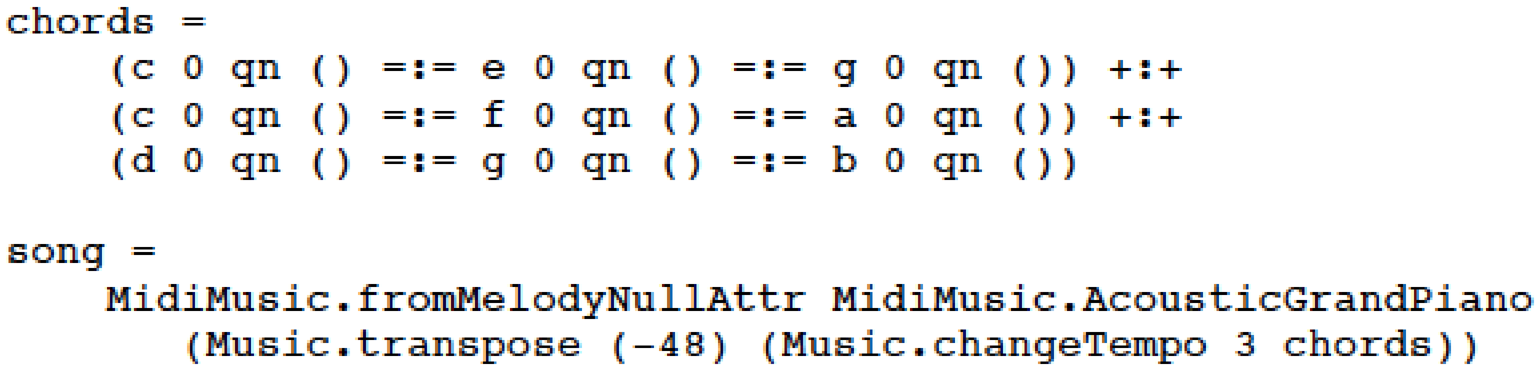
- CLM – Common Lisp Music



```
(defparameter *part* (new fms:part :instr '(:piano :simultlim 1)
:partid 'pno))

(defun polygen (voice len minp maxp)
  (process repeat len
         output (new fms:note
                :off (now)
                :voice voice
                :partid 'pno
                :note (between minp maxp)
                :dur 1/2)
         wait 1/2))

(events (list (polygen '(1 2) 20 50 80) (polygen '(1 2) 20 40 70))
"/tmp/fomus.ly" :parts *part* :view t)
```

Other Music languages based on Functional Programming

- Haskore – Based on Haskell



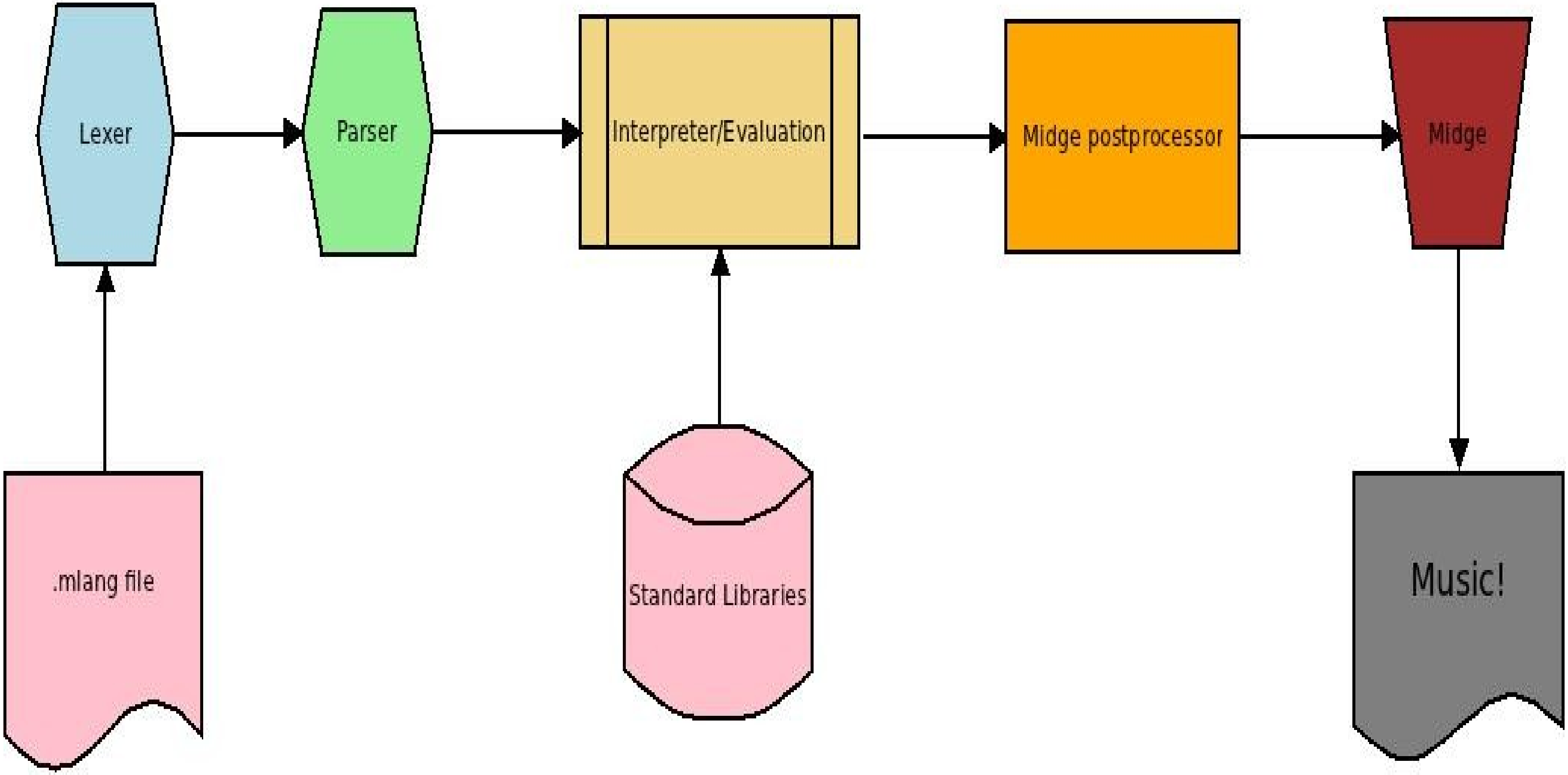
```
chords =
    (c 0 qn () =:= e 0 qn () =:= g 0 qn ()) +:+
    (c 0 qn () =:= f 0 qn () =:= a 0 qn ()) +:+
    (d 0 qn () =:= g 0 qn () =:= b 0 qn ())

song =
    MidiMusic.fromMelodyNullAttr MidiMusic.AcousticGrandPiano
       (Music.transpose (-48) (Music.changeTempo 3 chords))
```

So how exactly does MLang look?

Sample Mlang Program

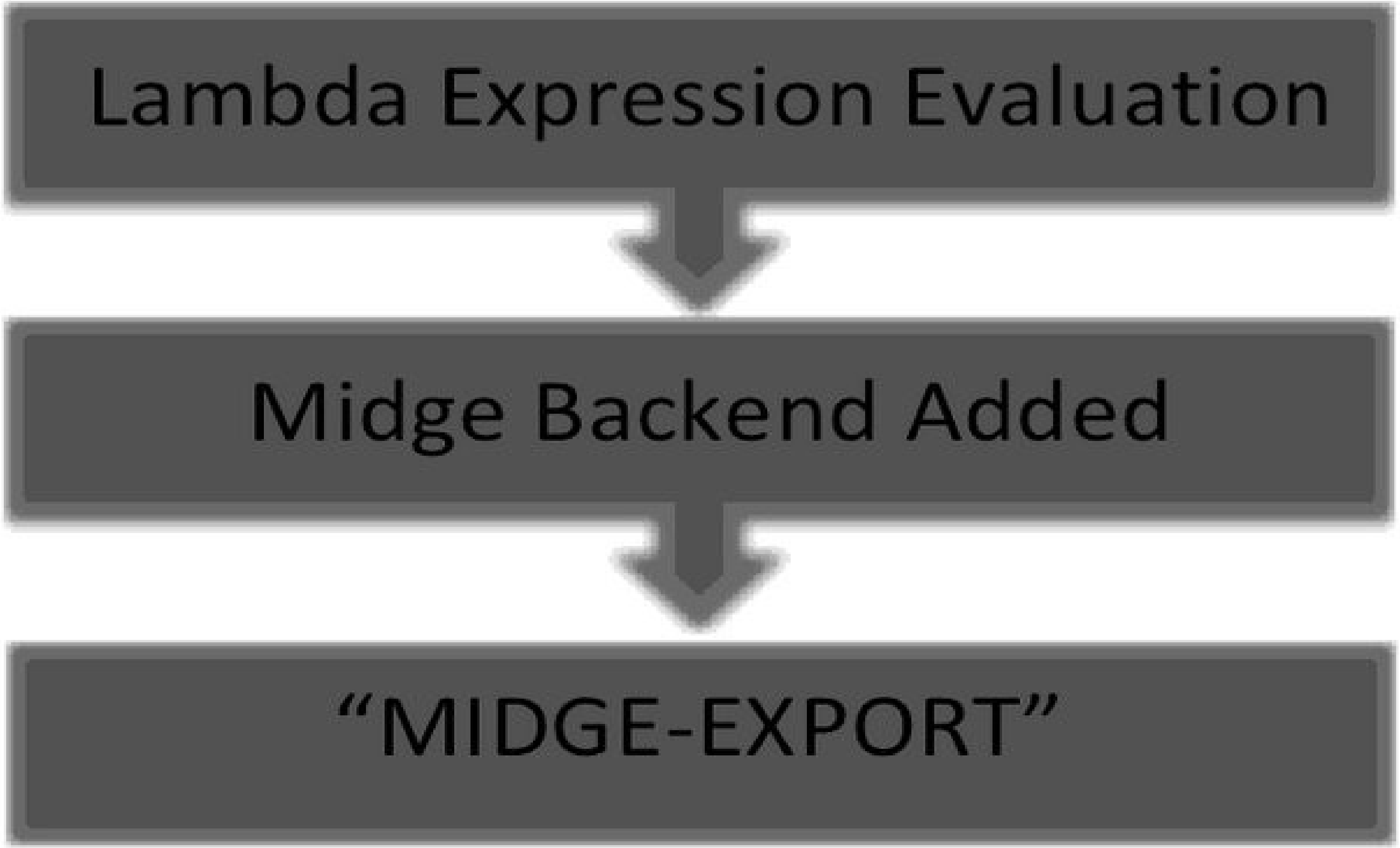
```
(
;Read standard library
(READ-FILE stdlib.mlang)

;Song tempo of 160 and time signature of 4/4
(label head (160 4 4))

;REPEAT4 → repeat a note 4 times, CONCAT → concatenate notes
(label phrase1 (CONCAT (REPEAT4 (3 e 8)) (REPEAT4 (3 e 8)) (REPEAT4 (3 e 8)) ((2
a 8) (2 b 8) (3 d 8) (3 e 8)) (REPEAT4 (3 d 8)) (REPEAT4 (3 d 8))
        ((3 g 8) (3 d 4) REST8 (3 e 8) REST8 (3 d 4))))

;Channel with acoustic bass, volume 90, repeat 4 times
(label channel1 (bass_ac 90 4 phrase1))

(MIDGE-EXPORT paranoid.mg (head (channel1)))
)
```

Implementation



Testing

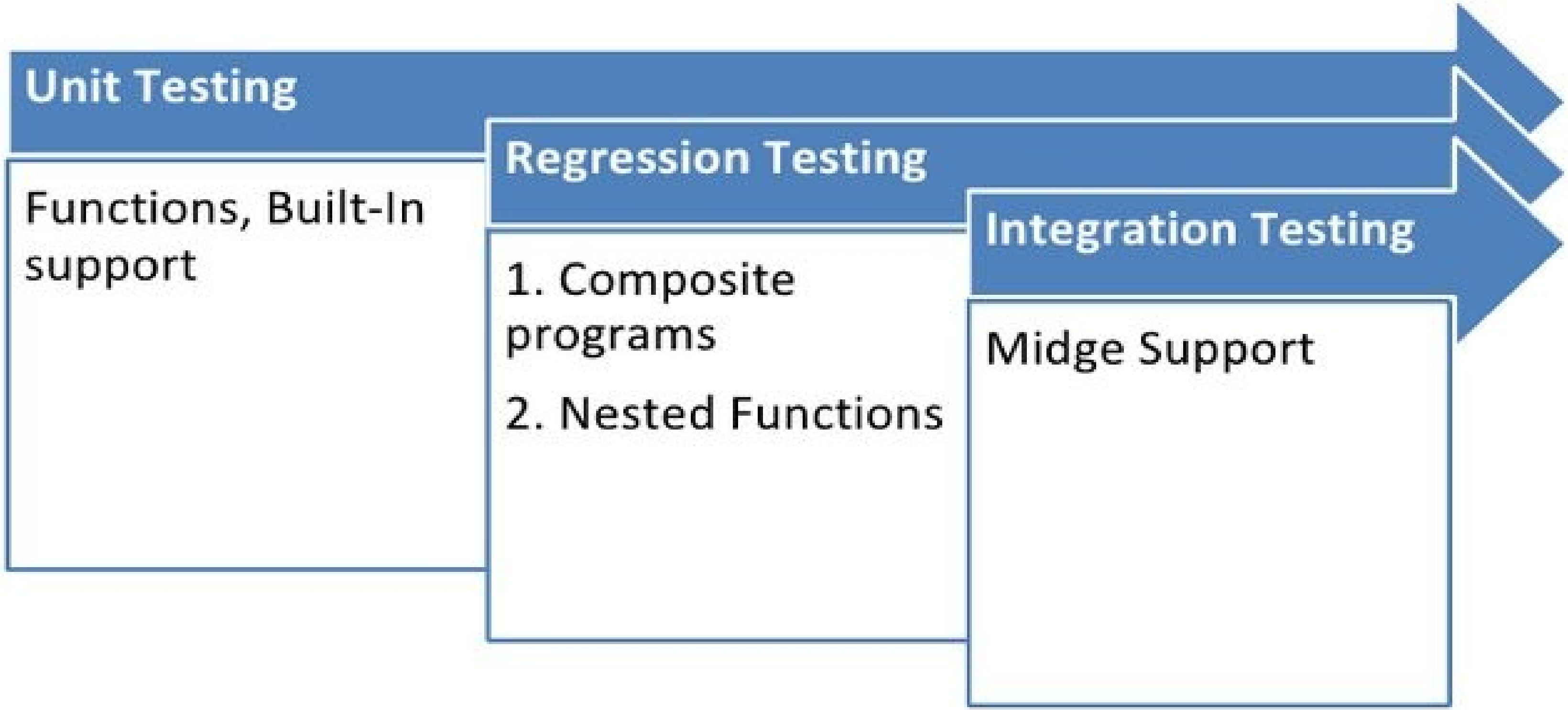
- Iterative Approach



Lambda Expression Evaluation

Midge Backend Added

“MIDGE-EXPORT”



Unit Testing

Functions, Built-In support

Regression Testing

1. Composite programs

2. Nested Functions

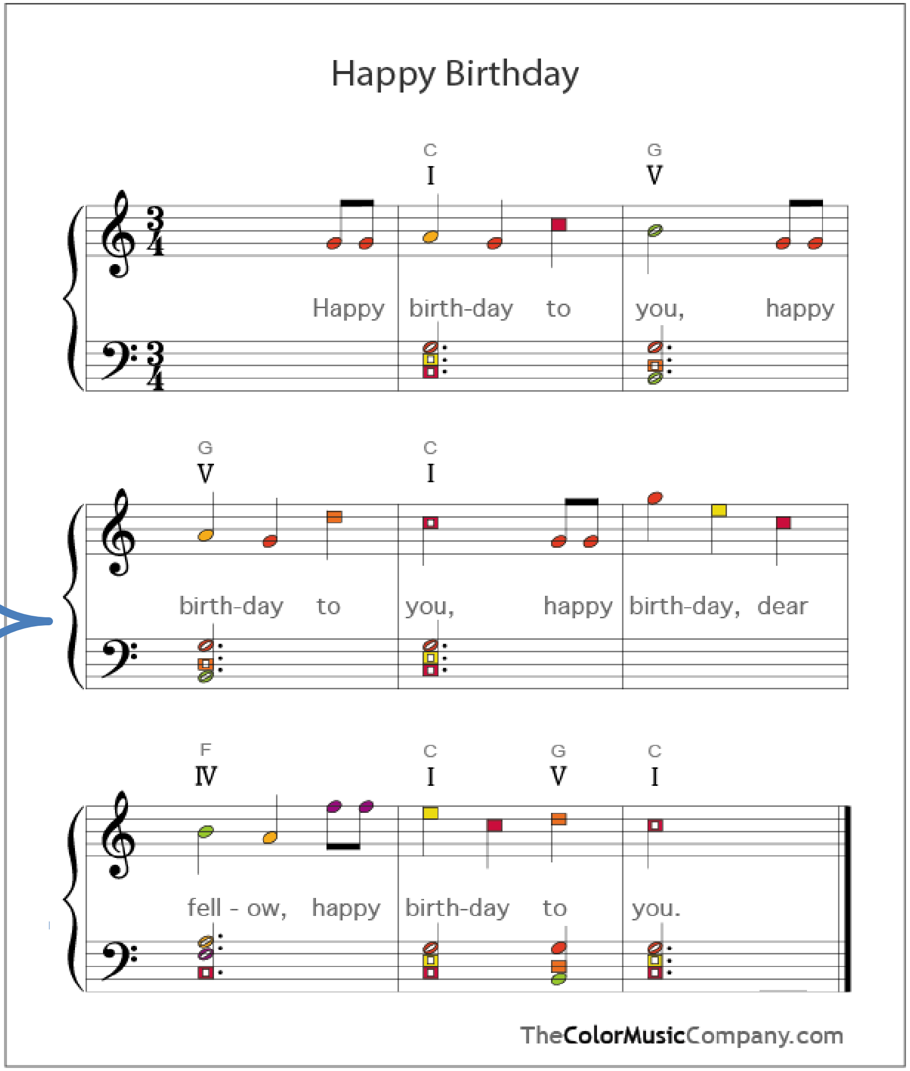
Integration Testing

Midge Support

- Playing Music!

Sample Test Code

```
1. (CONCAT (A S D) (A D))
2. (NTH 2 (A D F))
3. (REVERSE (CDR (A S D F)))
4. (READ-FILE abc.mlang)
5. (WRITE-FILE xyz.mlang (quote (a b c)))
6.    (
   (READ-FILE stdlib.mlang)
   (label head (42 4 4))
   (label phrase1 ((3 d 16) (3 d 16) (3 e 8) (3 d 8)
   (3 g 8) (3 f+ 4)
   (3 d 16) (3 d 16) (3 e 8) (3 d 8) (3 a 8) (3 g 4)
   (3 d 16) (3 d 16) (3 d 8) (3 b 8) (3 g 8) (3 f+ 8)
   (3 e 4)
   (3 c 16) (3 c 16) (3 b 8) (3 g 8) (3 a 8) (3 g 4))
   )
   (label channel1 (1 96 1 phrase1))
   (MIDGE-EXPORT happy.mg (head (channel1)))
   )
```



Results

Lessons Learned

- Version Control a must!
- Keep everyone in the loop.
- When designing, keep things as simple as possible.
- Commit code in incremental steps instead of all at once.
- Have fun!

Conclusions

- 1237 lines of ML code
- 533 lines of Mlang code
- 106 lines of glue (shell scripts, build)
- 100 lines of tests
- Lots of fun!

THANK YOU!