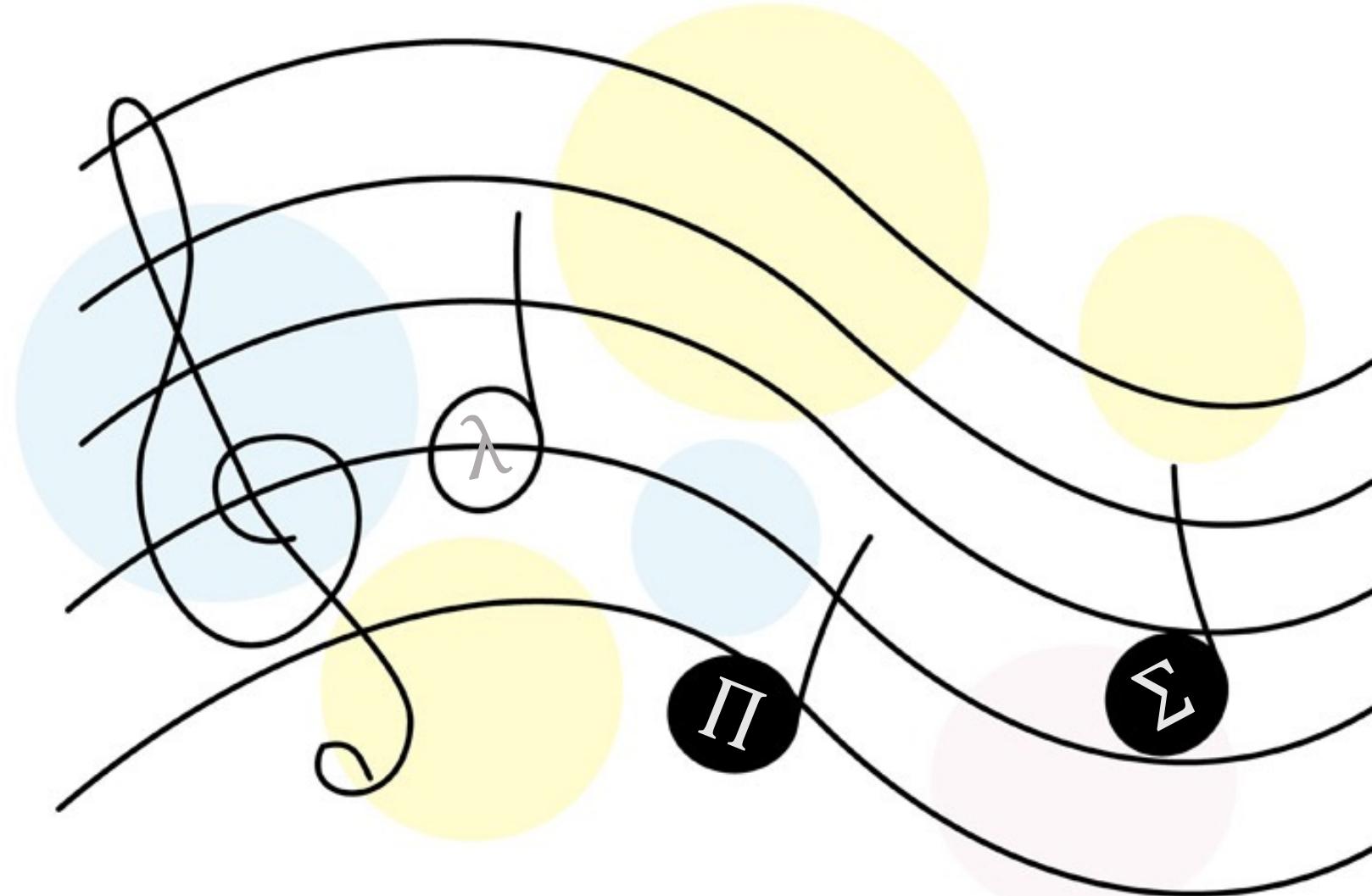


Composing Music from Types

Youyou Cong

Tokyo Institute of Technology

TYPES 2022



Bio



1995



1996



2011



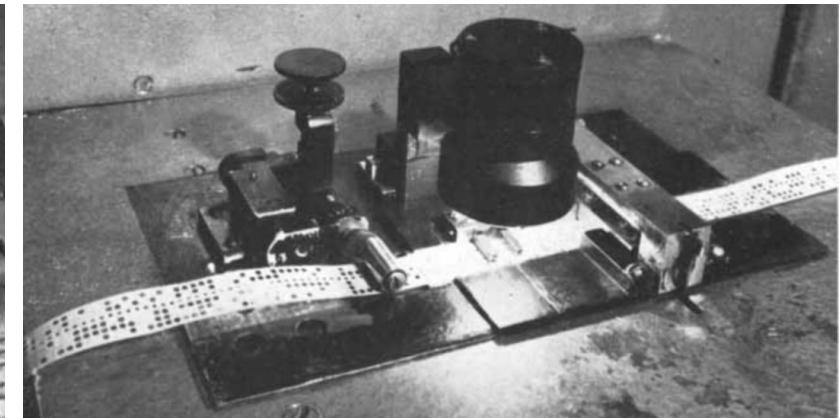
2014

AIM XXIX

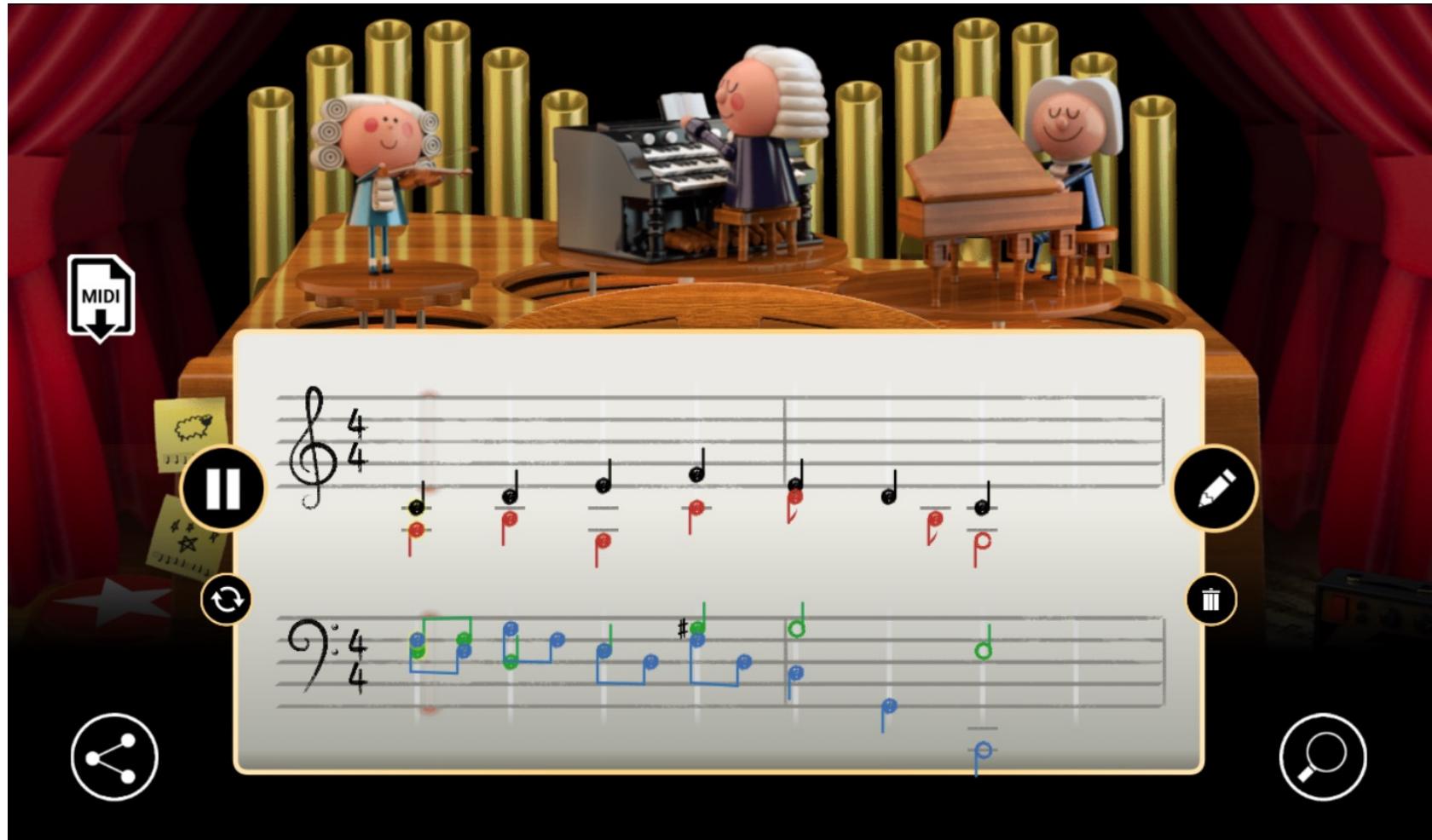


2019

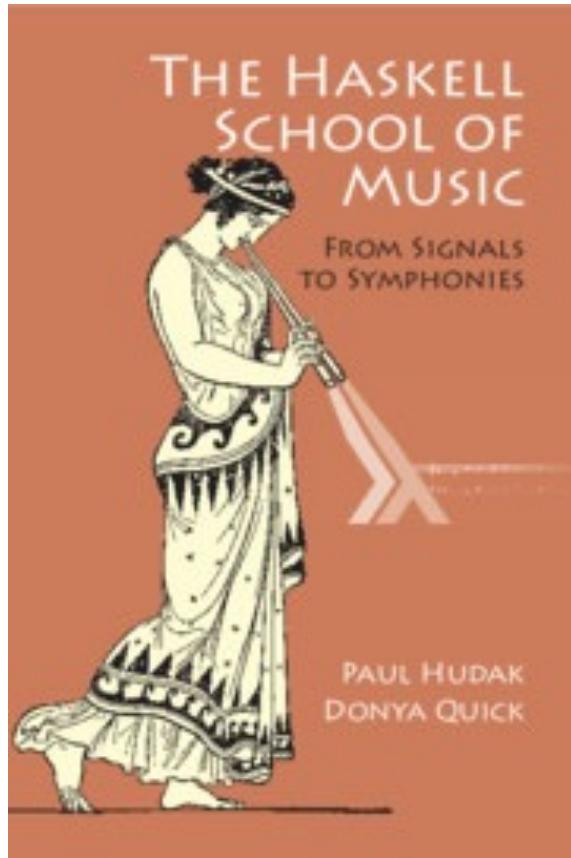
Illiad Suite (Hiller & Isaacson '57)



Bach Doodle (Huang et al. '19)



Euterpea (Hudak & Quick '18)



```
x1 = c 4 en :+: g 4 en :+: c 5 en :+: g 5 en  
x2 = x1 :+: transpose 3 x1  
x3 = x2 :+: x2 :+: invert x2 :+: retro x2  
x4 = forever x3 :=: forever (tempo (2/3) x3)
```



Mezzo (Szamozvancev & Gale '17)

Well-Typed Music Does Not Sound Wrong (Experience Report)

Dmitrij Szamozvancev
University of Cambridge, UK
ds709@cam.ac.uk

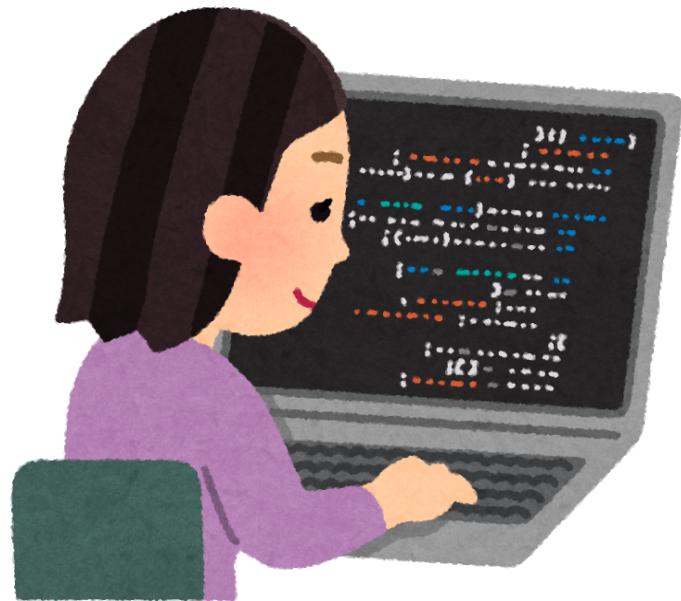
Michael B. Gale*
University of Cambridge, UK
michael.gale@cl.cam.ac.uk

```
v1 = d qn :|: g qn :|: fs qn :|: g en :|:  
      bf qn :|: a qn :|: g hn  
v2 = d qn :|: ef qn :|: d qn :|: bf_ en :|:  
      b_ qn :|: a_ qn :|: g_ hn  
comp = defScore (v1 :-: v2)
```

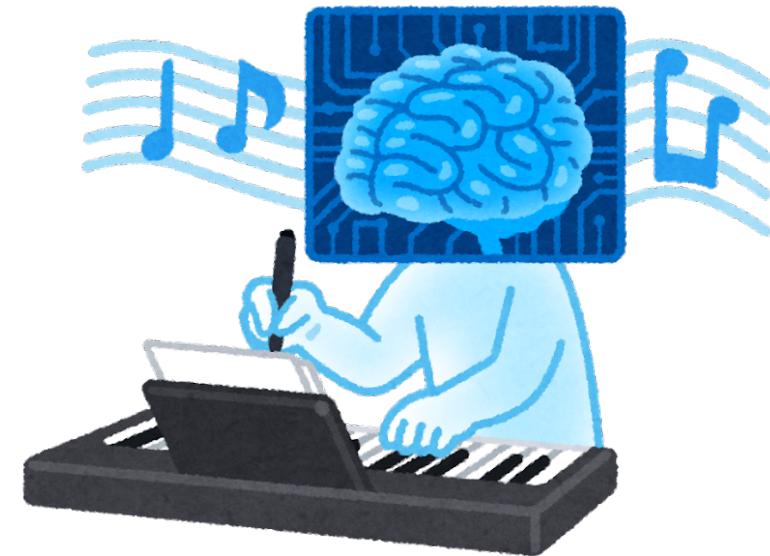
Can't have major sevenths
in chords: Bb - B_.

Parallel octaves are
forbidden: A - A_, then G - G_.

Our work: Representing & generating correct music



Agda



Synquid (Polikarpova+ '16)

Musical correctness?



Introduction to Counterpoint

Counterpoint 101

- Technique for combining multiple melodies
- Has explicit rules
(by J. J. Fux 1725)



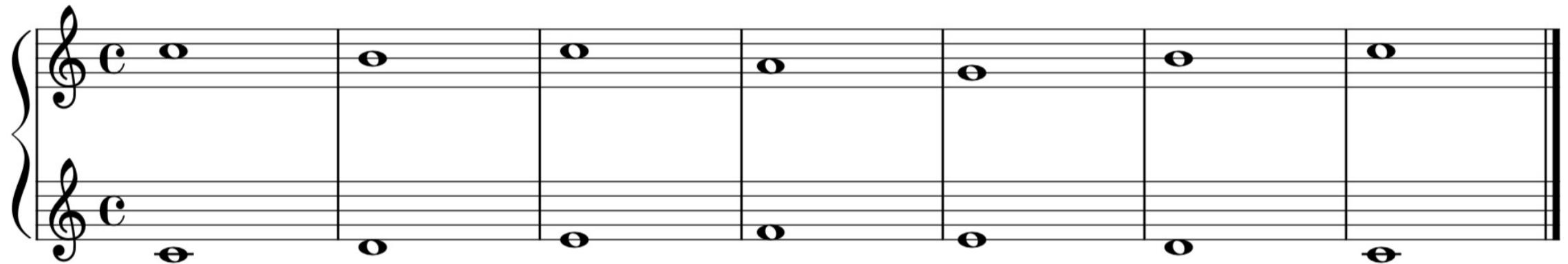
Example of counterpoint

A musical staff consisting of two staves. The top staff has a treble clef and a 'C' key signature, with a brace indicating it is part of a two-staff system. The bottom staff also has a treble clef and a 'C' key signature. The music consists of seven measures. The top staff contains only vertical bar lines. The bottom staff contains note heads: an open circle at the start, followed by six solid black circles, and an open circle at the end. The notes are positioned on the first, third, and fifth lines of the staff.

Cantus firmus

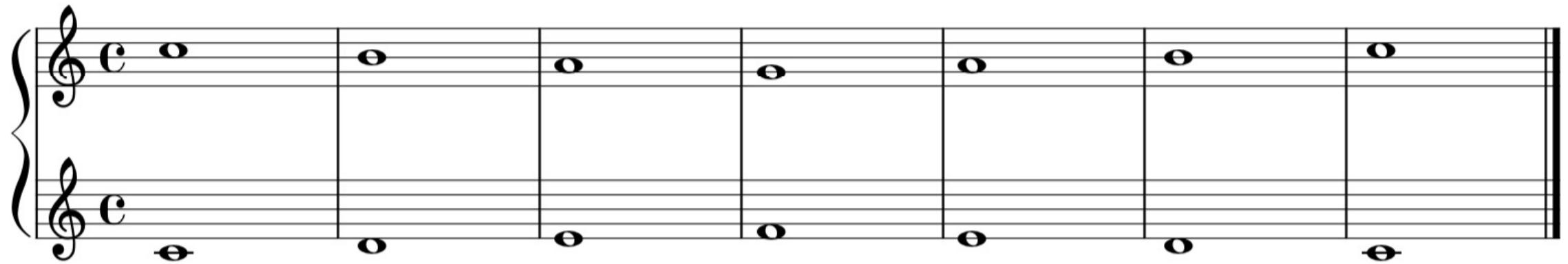
Example of counterpoint

Counterpoint

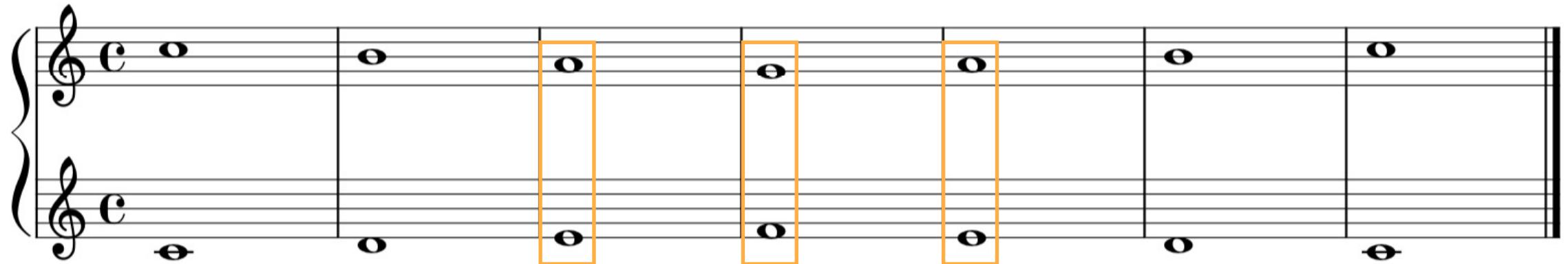


Cantus firmus

Example of wrong counterpoint



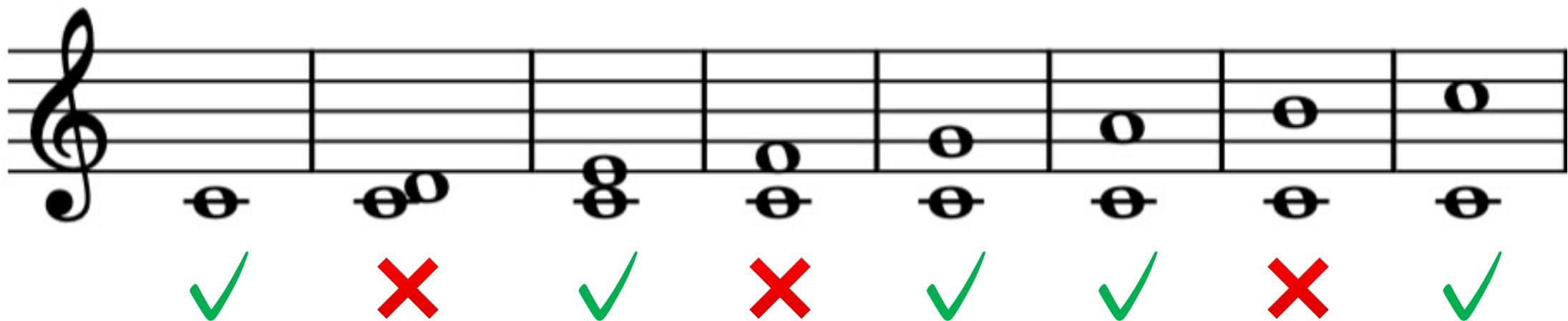
Example of wrong counterpoint



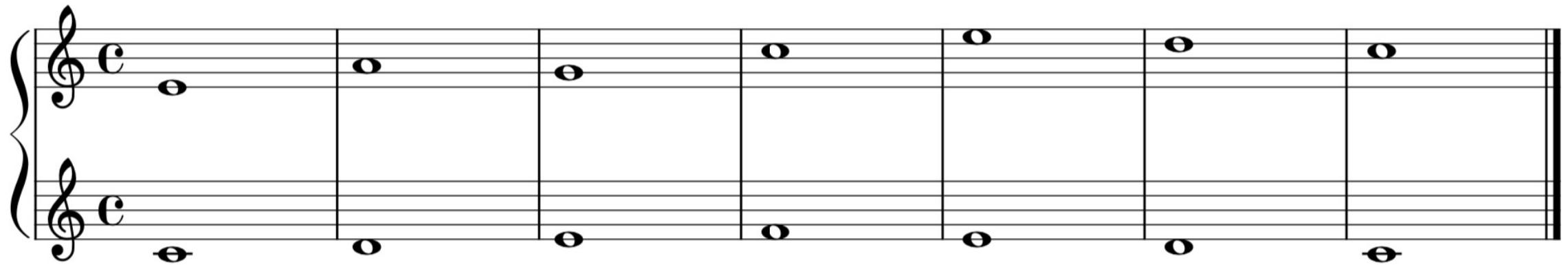
Dissonant

Rule #1

All intervals must be consonant.



Example of wrong counterpoint



Example of wrong counterpoint

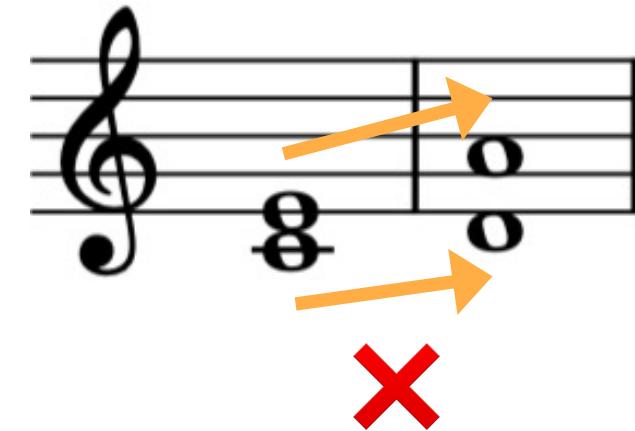
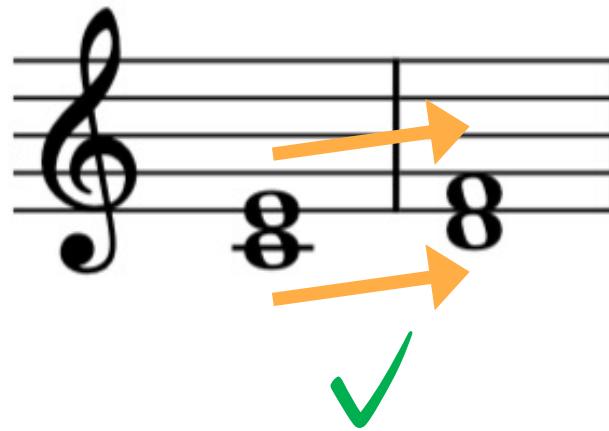
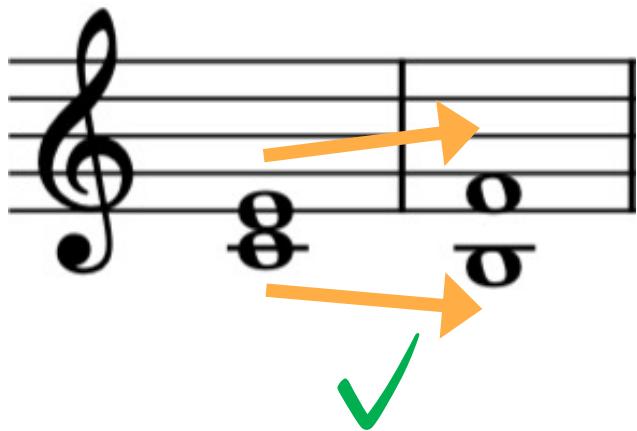
The image shows a musical staff with two voices. The top voice (treble clef) has notes on the 4th and 5th lines. The bottom voice (treble clef) has notes on the 3rd and 4th lines. Orange arrows point from each note in the top voice to the corresponding note in the bottom voice, illustrating direct fifth and octave intervals.

Direct fifth

Direct octave

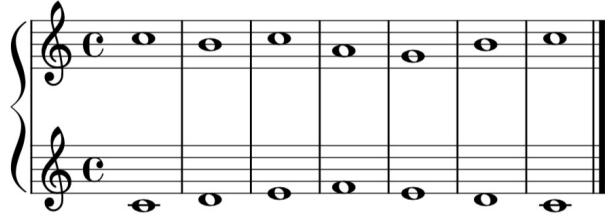
Rule #2

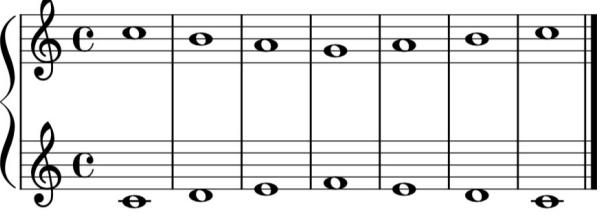
No direct fifth or octave is allowed.



Rules of counterpoint

1. All intervals must be consonant.
2. No direct fifth or octave is allowed.

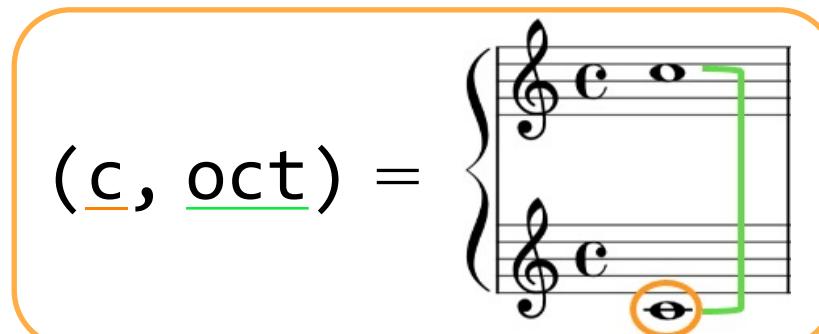
⊤  : CP

⊤  : CP

Representing Correct Counterpoint

Correct counterpoint as records

```
record CP : Set where
  constructor cp
  field
    bars      : List PitchInterval
    intervalOK : collectIntervalError bars ≡ []
    motionOK   : collectMotionError bars ≡ []
```



Interval rule as a function

```
data IntervalError : Set where
  dissonant : Interval → IntervalError

checkInterval : PitchInterval → Maybe IntervalError
checkInterval (p , i) with isConsonant i
checkInterval (p , i) | false = just (dissonant i)
checkInterval (p , i) | true  = nothing

collectIntervalError : List PitchInterval → List IntervalError
collectIntervalError = mapMaybe checkInterval
```

Motion rule as a function

```
data MotionError : Set where
```

```
  direct58 : PitchInterval → PitchInterval → MotionError
```

```
checkMotion : PitchInterval → PitchInterval → Maybe MotionError
```

```
checkMotion pi1 pi2 with isDirect pi1 pi2 | isPerfect (proj2 pi2)
```

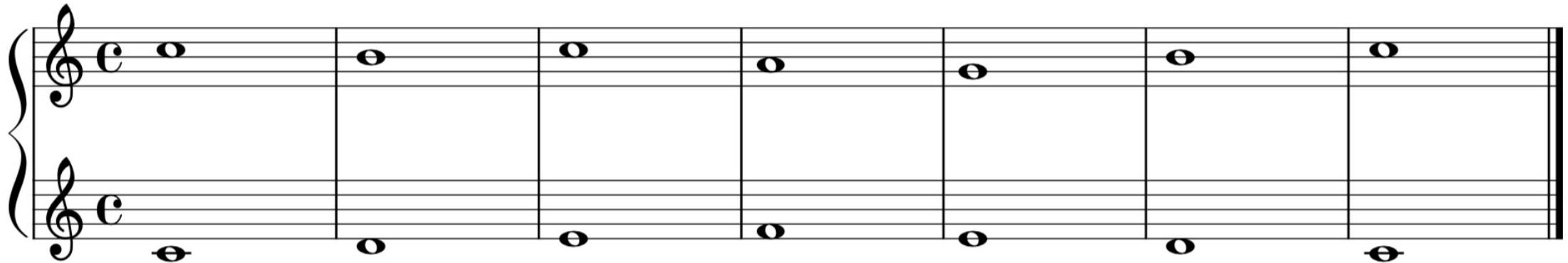
```
checkMotion pi1 pi2 | true | true = just (direct58 pi1 pi2)
```

```
checkMotion pi1 pi2 | _ | _ = nothing
```

```
collectMotionError : List PitchInterval → List MotionError
```

```
collectMotionError = mapMaybe (uncurry checkMotion) ∘ pairs
```

Type-checking correct counterpoint

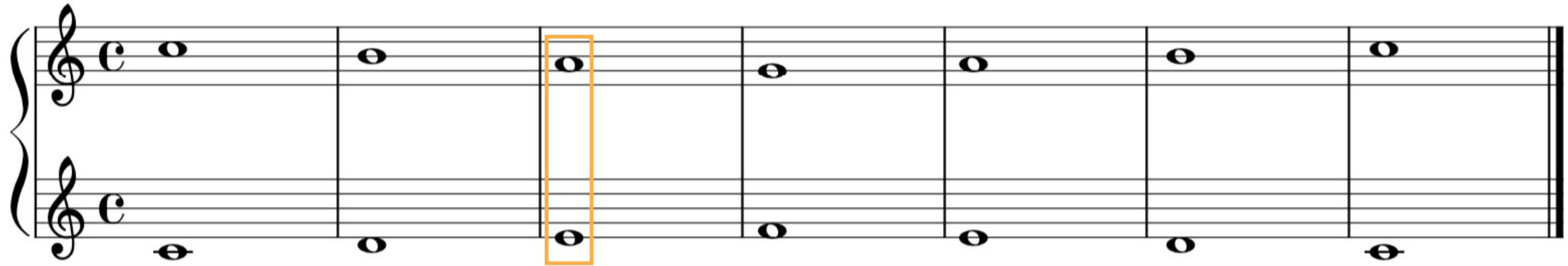


`cp-correct` : CP

`cp-correct` = `cp` `bars-correct refl refl`

All Done

Type-checking wrong counterpoint



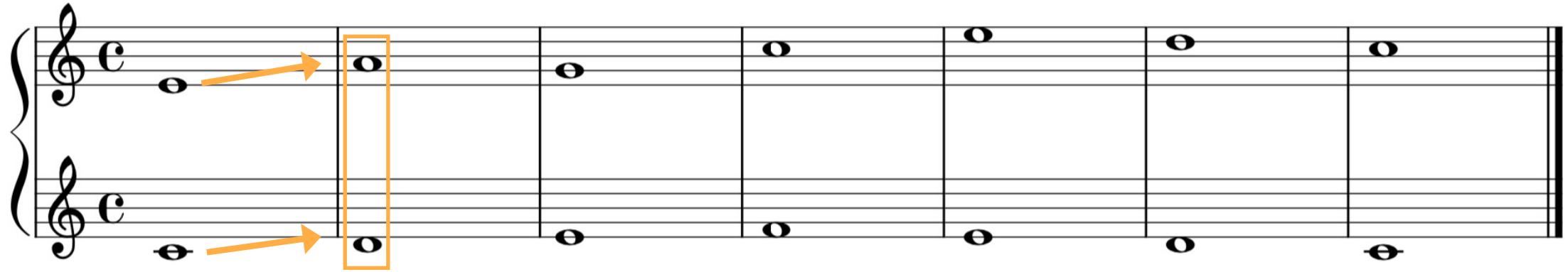
`cp-dissonant : CP`

`cp-dissonant = cp bars-dissonant refl refl`

Error

`(dissonant per4) :: ... != [] of type (List IntervalError)`

Type-checking wrong counterpoint



cp-direct58 : CP

cp-direct58 = cp bars-direct58 refl refl

Error

(direct58 (c,maj3) (d,per5)) :: ... != [] of type (List MotionError)

Beethoven's Pathetique Sonata

Allegro di molto e con brio ($\text{d} = 144$)

The musical score consists of two staves for piano. The top staff is for the treble clef (G-clef) and the bottom staff is for the bass clef (F-clef). The key signature is one flat (B-flat). Measure 1 starts with a dynamic p and a forte dynamic f . Measure 2 features a series of grace notes with fingerings (4, 5, 1, 1, 2, 5) over a sustained note. Measures 3-4 show a continuation of the melodic line with fingerings (4, 3, 5, 2, 1, 1, 2, 5). Measures 5-6 include a crescendo dynamic and a forte dynamic f . Measures 7-8 show a continuation of the melodic line with fingerings (3, 4, 5, 2, 1, 1, 2, 5). Measures 9-10 feature a dynamic p and a forte dynamic f . Measures 11-12 show a continuation of the melodic line with fingerings (4, 2, 1, 1, 2, 5). Measures 13-14 feature a dynamic p and a forte dynamic f . Measures 15-16 show a continuation of the melodic line with fingerings (5, 3, 1, 1, 2, 5). Measures 17-18 feature a dynamic p and a forte dynamic f . Measures 19-20 show a continuation of the melodic line with fingerings (5, 2, 1, 1, 2, 5).

Type-checking Pathetique Sonata



`cp-pathetique` : CP

`cp-pathetique` = `cp` `bars-pathetique refl refl`

Type-checking Pathetique Sonata



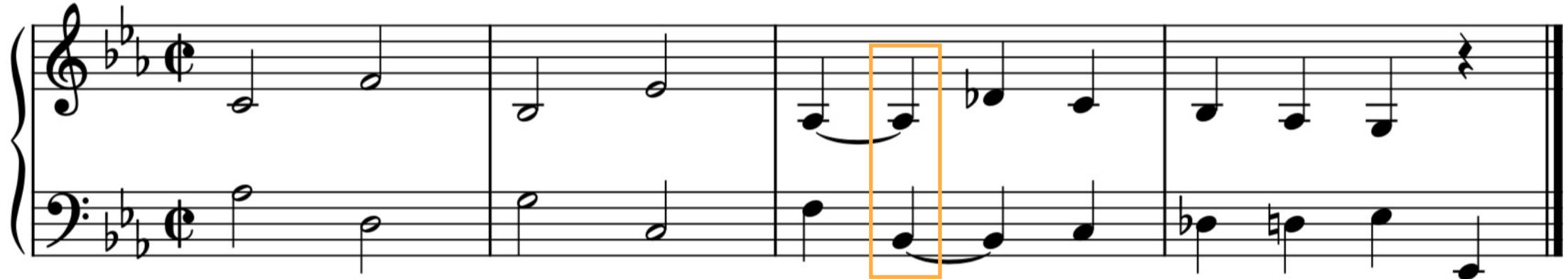
`cp-pathetique` : CP

`cp-pathetique` = cp bars-pathetique refl refl

Error

`(dissonant min7) :: ... != []` of type `(List IntervalError)`

Type-checking Pathetique Sonata



`cp-pathetique` : CP

`cp-pathetique` = cp bars-pathetique refl refl

Error

`(dissonant min7) :: ... != []` of type `(List IntervalError)`

Beethoven's exercises with Haydn



Generating Correct Counterpoint

Correct counterpoint as refined lists

```
type CP =  
{ List PitchInterval |  
  intervalOK _v && motionOK _v }
```

Interval rule as a measure

```
measure intervalOK :: List PitchInterval -> Bool where
    Nil -> True
    Cons pi pis -> isConsonant (proj2 pi) &&
                           intervalOK pis
```

Motion rule as a measure

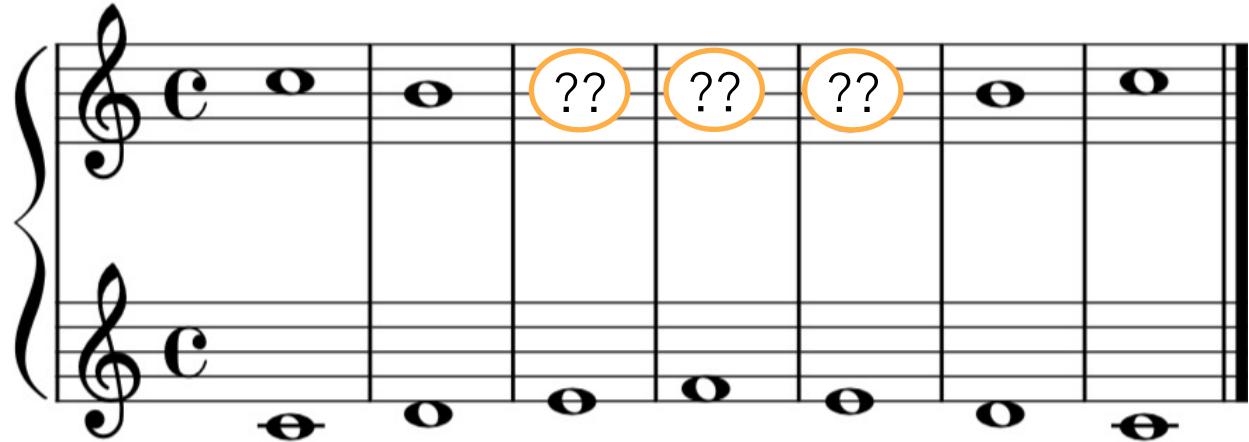
```
measure motionOK :: List PitchInterval -> Bool where
    Nil -> True
    Cons pi pis ->
        (pis == Nil) ||
        if {- pi and (head pis) form direct motion -}
        then not (is58 (interval (head pis))) &&
            motionOK pis
        else motionOK pis
```

Generating correct counterpoint

cp :: CP

cp =

```
Cons (C, Oct)
  (Cons (D, Maj6)
    (Cons (E, ??)
      (Cons (F, ??)
        (Cons (E, ??)
          (Cons (D, Maj6)
            (Cons (C, Oct)
              Nil)))))))
```

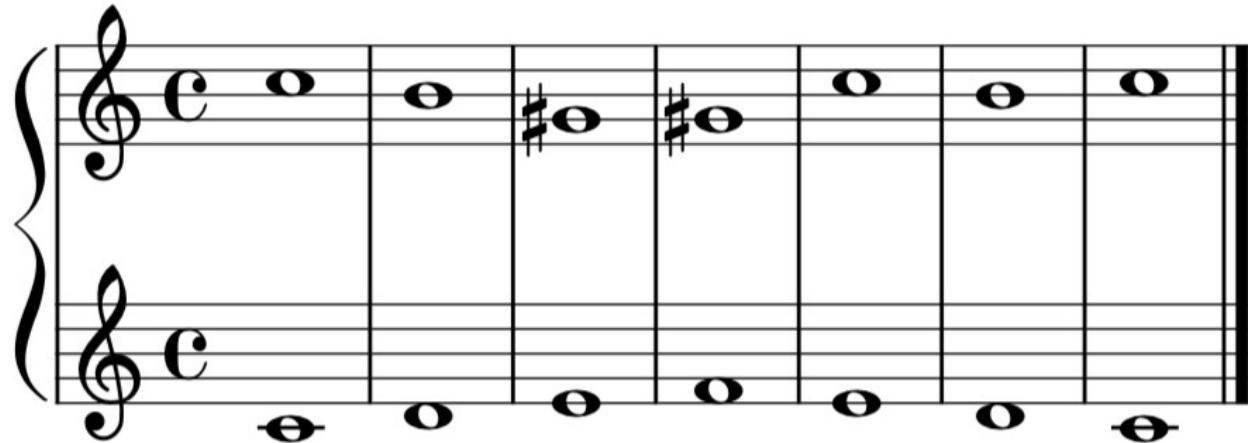


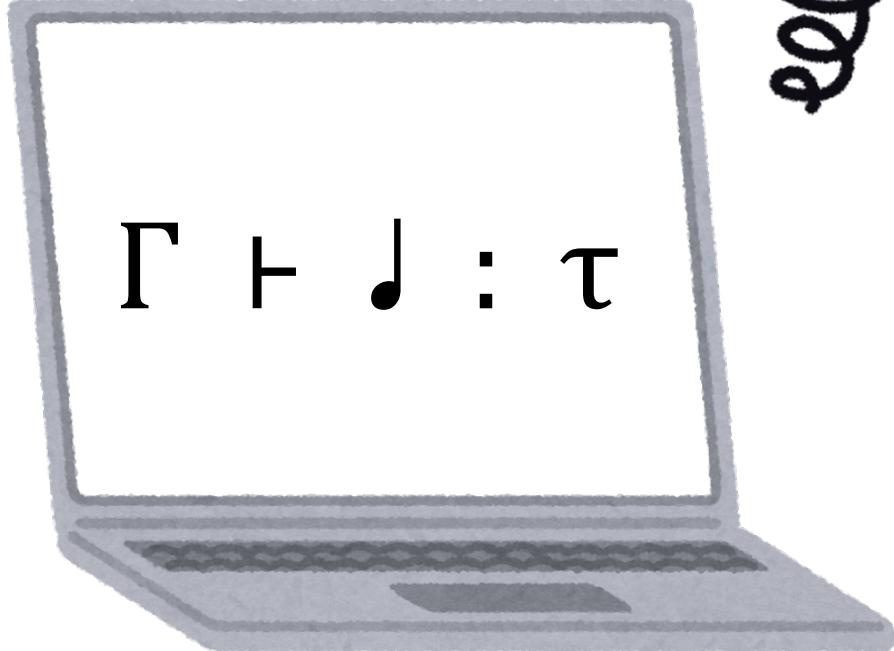
Generating correct counterpoint

cp :: CP

cp =

```
Cons (C, Oct)
  (Cons (D, Maj6)
    (Cons (E, Maj3)
      (Cons (F, Min3)
        (Cons (E, Min6)
          (Cons (D, Maj6)
            (Cons (C, Oct)
              Nil))))))
```





Well-typed music
may sound wrong.

Towards Musical Soundness

Problems with Synquid's composition

Non-scale notes

The musical score consists of two staves. The top staff begins with a treble clef and a key signature of one sharp (F#). It has seven measures. The bottom staff begins with a treble clef and a key signature of one flat (B-flat). It also has seven measures. Measures 4 and 5 of the top staff are circled in orange. An orange callout box with the text "Non-scale notes" points to these circled notes.

Problems with Synquid's composition

Repeated notes

The musical score consists of two staves. The top staff begins with a treble clef and a key signature of one sharp. It has six measures. The bottom staff begins with a treble clef and a key signature of one sharp. It also has six measures. In both staves, the fourth and fifth measures contain identical note patterns: a sharp followed by a regular note. These specific notes are highlighted with orange circles. An orange callout box with the text "Repeated notes" points to these circled measures.

Hard and soft rules of counterpoint

1. All intervals must be consonant. (hard)
2. No direct fifth or octave is allowed. (hard)
3. Non-scale notes are not preferred. (soft)
4. Repeated notes are not preferred. (soft)

Limitation of standard refinement types

```
type CP = { List PitchInterval |  
            allConsonant _v && noDirect58 _v &&  
            allScaleNotes _v && noRepeat _v }
```

Representing hard and soft rules

```
type CP = List PitchInterval < r1, r2 >
```

Representing hard and soft rules

```
type CP = List PitchInterval < r1, r2 >
```

Rules on individual intervals

Representing hard and soft rules

```
type CP = List PitchInterval < r1, r2 >
```



Rules on adjacent intervals

Representing hard and soft rules

```
type CP = List PitchInterval < r1, r2 >
```

where

```
r1 = (isConsonant, ∞) ∧ (isScaleNote, 80)
```

```
r2 = (notDirect58, ∞) ∧ (notRepeated, 60)
```

Representing hard and soft rules

```
type CP = List PitchInterval < r1, r2 >
```

where

```
r1 = (isConsonant, ∞) ∧ (isScaleNote, 80)
```

```
r2 = (notDirect58, ∞) ∧ (notRepeated, 60)
```

Hard

Representing hard and soft rules

```
type CP = List PitchInterval < r1, r2 >
```

where

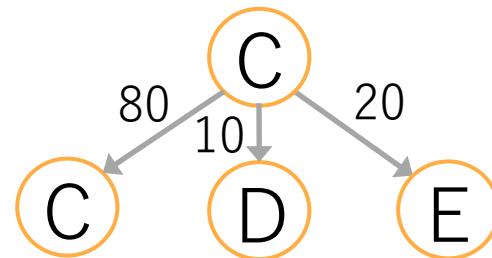
```
r1 = (isConsonant, ∞) ∧ (isScaleNote, 80)
```

```
r2 = (notDirect58, ∞) ∧ (notRepeated, 60)
```

Soft

Implications of weighted types

- Type checking and synthesis as Max-SAT problem



- Soundness as gradable property

Theorem. *Lowly weighted music does not sound too wrong.*

Ideas for implementation

- Extend Synquid
 - ☺ Has reusable components
 - ☹ May not be suited for music
- Use Turnstile and Rosette (Chang et al. '20, Torlak et al. '13)
 - ☺ Easier to customize
 - ☹ May not combine well

Ideas for soundness

e : PitchInterval

l : List PitchInterval < r_1, r_2 >; w

$\text{cost}(r_1 \ e) = w_1$

$\text{cost}(r_2 \ e \ (\text{head } l)) = w_2$

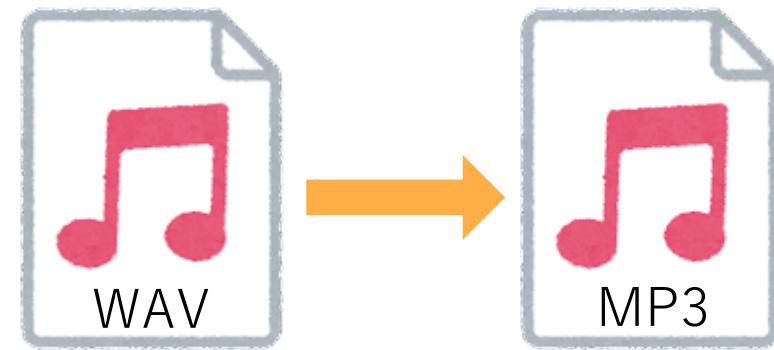
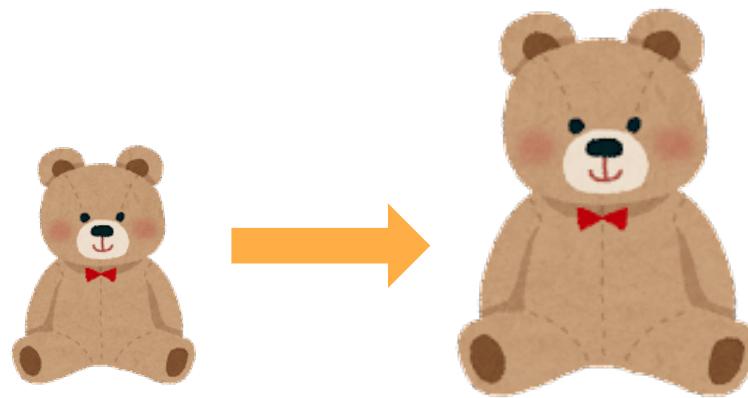
(e :: l) : List PitchInterval < r_1, r_2 >; w + w₁ + w₂

Discussion

How would weighted refinement types
be useful for programming?



Possible application: Approximate computing



Precision types (Sampson+ '11, Boston+ '14)

```
@Approx(0.8) int square(@Approx(0.9) int x) {  
    @Approx(0.8) int xSquared = x * x;  
    return xSquared;  
}
```

Precision types (Sampson+ '11, Boston+ '14)

```
@Approx(0.8) int square(@Approx(0.9) int x) {  
    @Approx(0.8) int xSquared = x * x;  
  
    return xSquared;  
}
```

Precision types (Sampson+ '11, Boston+ '14)

```
@Approx(0.8) int square(@Approx(0.9) int x) {  
    @Approx(0.8) int xSquared = x * x;  
  
    return xSquared;  
}
```

Precision types (Sampson+ '11, Boston+ '14)

```
@Approx(0.8) int square(@Approx(0.9) int x) {  
    @Approx(0.8) int xSquared = x * x;  
    return xSquared;  
}
```

Takeaway

Type theorists can get inspiration from music!

Code & Google Form:

<https://github.com/YouyouCong/counterpoint/types22>



Thanks to JST for funding this project.