Composing Music from Types

Youyou Cong
Tokyo Institute of Technology
TYPES 2022
Bio
Illiac Suite (Hiller & Isaacson ‘57)
Bach Doodle (Huang et al. ‘19)
Euterpea (Hudak & Quick ‘18)

\[
\begin{align*}
x_1 &= \mathsf{c}\ 4\ \mathsf{en} \mathbin{:+} \mathsf{g}\ 4\ \mathsf{en} \mathbin{:+} \mathsf{c}\ 5\ \mathsf{en} \mathbin{:+} \mathsf{g}\ 5\ \mathsf{en} \\
x_2 &= x_1 \mathbin{:+} \text{transpose}\ 3\ x_1 \\
x_3 &= x_2 \mathbin{:+} x_2 \mathbin{:+} \text{invert}\ x_2 \mathbin{:+} \text{retro}\ x_2 \\
x_4 &= \text{forever}\ x_3 \mathbin{:=} \text{forever}\ (\text{tempo}\ (2/3)\ x_3)
\end{align*}
\]
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

\[
v_1 = \text{d qn} :|: \text{g qn} :|: \text{fs qn} :|: \text{g en} :|:
    \text{bf qn} :|: \text{a qn} :|: \text{g hn}
\]

\[
v_2 = \text{d qn} :|: \text{ef qn} :|: \text{d qn} :|: \text{bf\textunderscore en} :|:
    \text{b\_ qn} :|: \text{a\_ qn} :|: \text{g\_ hn}
\]

\[
\text{comp} = \text{defScore (v1 :-: v2)}
\]

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

Cantus firmus
Example of counterpoint

Counterpoint

Cantus firmus
Example of wrong counterpoint
Example of wrong counterpoint
Rule #1

All intervals must be consonant.
Example of wrong counterpoint
Example of wrong counterpoint

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.
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
  | false  = just (dissonant 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 (\( \text{d} = 144 \))

\[ \text{cresc.} \]

\[ \text{dim.} \]
Type-checking Pathetique Sonata

cp-pathetique : CP

cp-pathetique = cp bars-pathetique refl refl
Type-checking Pathetique Sonata

\[
\begin{align*}
\text{cp-pathetique} & : \text{CP} \\
\text{cp-pathetique} &= \text{cp bars-pathetique refl refl}
\end{align*}
\]

Error

\[(\text{dissonant min7}) :: ... \neq [] \text{ of type (List IntervalError)}\]
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

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

```haskell
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))))))

Nil)
Well-typed music may sound wrong.
Towards Musical Soundness
Problems with Synquid’s composition

Non-scale notes
Problems with Synquid’s composition

Repeated notes
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

```plaintext
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

\[ r1 = (\text{isConsonant}, \infty) \land (\text{isScaleNote}, 80) \]
\[ r2 = (\text{notDirect58}, \infty) \land (\text{notRepeated}, 60) \]
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

\[ r1 = (\text{isConsonant}, \infty) \land (\text{isScaleNote}, 80) \]

\[ r2 = (\text{notDirect58}, \infty) \land (\text{notRepeated}, 60) \]
Implications of weighted types

- Type checking and synthesis as Max-SAT problem

\[ C \rightarrow D \rightarrow E \]

- 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 <r₁, r₂>; w
\text{cost}(r₁ e) = w₁
\text{cost}(r₂ e (\text{head } l)) = w₂

(e :: l) : List PitchInterval <r₁, r₂>; 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;
}

56
Precision types (Sampson+ ’11, Boston+ ‘14)

```java
@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/tree/main/types22

Thanks to JST for funding this project.