ALGORITHMIC COMPOSITION

Introducing the score-gen construct

Programs and Data

- We've seen:
  - Ordinary programs:
    - pwl(…) * osc(…)
  - Score-like programs:
    - sim(note(…) ~ 2 @ 0, note(…) ~ 3 @ 1, …)
  - Scores:
    - ```
      {{0 2 {note …}}
       {1 3 {note …}}
       …}
      ```


      ```

## Lots of Choices

- Data and programs have different properties
- Data and programs can work together:
  - Programs create data (scores)
  - `timed-seq` interprets scores to invoke functions
  - Programs can even create (Lisp) programs
- No right/wrong answers
- Today, we look at programs creating scores

  

## The `score-gen` Macro: Introduction

- The problem:
  - Create a score of notes
  - Specify attribute values with SAL expressions (evaluated for each note)
  - Flexible expression of start time, inter-onset time, or duration
- The solution: `score-gen`
- Alternative: build scores with list primitives
  - (been there, done that in Project 2 – was it fun?)

  

## score-gen

```
score-gen(attribute: expression,
          attribute: expression,
          attribute: expression,
          …)
score-gen(score-len: 2, pitch: 60,
          vel: 100, ioi: 0.7,
          name: quote(note))
```

score-gen Loop Variables

- sg:start – starting time for current note
- sg:ioi – current inter-onset interval
- sg:dur – current duration
- sg:count – how many notes computed so far
- Example:
  -
```
score-gen(score-len: 10, ioi: 0.2,
          pitch: c4 + sg:count)
```