Using Music to Teach Computation and Mathematics in Secondary Education

John Peterson[1] and Paul Hudak[1]

Yale University, Department of Computer Science, New Haven, CT 06520-8285

Abstract. How should we teach students the basic principles of computation? What sort of problem domain will provide the motivation to learn about computing? Can we combine the creativity of the arts with the more rigorous demands of mathematics education? These questions are the motivation for this research. While it would be difficult to provide conclusive answers to any of them, we demonstrate a system in which music is used to connect two sorts of learning experience: music composition, a *creative* endeavor, and computation, a precise *mathematical* process for defining musical performances. Educationally, we use a language of algorithmic music to teach about *abstraction*: the definition and use of patterns in a formal and rigorous manner.
We are creating a new version of Haskore, a Haskell-based computer music system, suitable for middle and high school students (age 12 and up) that requires no background in music and uses only a simple functional style of computation. These functions are used to capture melodic and harmonic patterns of notes. Although this language is still in the design stage, we have tested the ideas behind it by using the original version of Haskore directly with high-school level students. In this paper, we discuss our experiences using a music language in the classroom and present the design principles behind the educational version of Haskore and outline simple curricula based on Haskore for teaching computation, algebra, and music.

1 Introduction

This research seeks to bring the technologies developed by the programming language community to an area in which they can have a widespread impact on society: secondary education. This problem is particularly interesting because high school students (ages 12 - 18) are the ultimate naive users: they have no background in computing and are not mathematically sophisticated. Thus we need languages which are truly declarative, stripped of all unnecessary detail and clutter.

Music encompasses a number of different learning experiences, allowing us to examine the connection between the mathematical realms of algebra and computation and the artistic world of composition. We use abstraction to describe patterns, both musical and computational, with which students build musical compositions. This supports an artistic *creative design* process that motivates the study of general mathematical and computational principals. Just as a student of creative writing needs to master the formal rules of grammar, a well-constructed computer language for music composition can motivate students to learn about the computational process and formal structure of the music.

Why should young students learn about computation? We believe that there are a number of reasons for incorporating computation early in the curriculum:

- Computation is a general and necessary skill. Calculators and spreadsheets are used by everyone, not just computer programmers. Although the mechanisms of computation vary considerably with the application, the basic principles of computation remain the same.
- Taught correctly, computation re-enforces basic mathematical principals and can shed new light on many mathematical activities.
- Computation can enhance creativity: incorporating computational capabilities into any domain allows students to express complex and interesting artifacts.

Learning about computation is not the same as learning how to program. Programming is fraught with details; even high level languages such as Java require considerable training in software engineering. Our goal it is to create a declarative language that is uncluttered by engineering detail and is highly expressive, allowing students to create intricate and aesthetic compositions from simple mathematical descriptions.

What follows is more a manifesto than anything else. We have not yet constructed any new software or obtained any significant classroom experience. We will present our current ideas for a student appropriate music language but these will be subject to change as work progresses. This work is based on experience in a widely used DSL, Haskore (see `www.haskell.org/haskore`), for music description and some preliminary classroom feedback that indicates a significant level of student interest in our approach. Our website, `haskell.org/edsl`, contains slides we have presented to students and some of their compositions.

2 Computers and Music

Using computers for music composition and performance is by no means a new idea. Our approach is distinguished from previous work by a strong computational and linguistic orientation. Computationally, our work is based on *functional programming*. The original Haskore language is embedded in Haskell, a general purpose purely functional language. Unfortunately, the complexities of Haskell make this system unsuitable for direct use by unsophisticated users. Our initial objective is to capture the essential core of Haskore in a language suitable for student use. Since this "educational Haskore" uses the same semantic foundations of the full Haskore we will not distinguish between these languages in this paper.

For students, we describe this system in a more "linguistic" style. Haskore defines a *basic vocabulary* of notes and other useful values such as numbers or lists.

The user can augment this vocabulary by creating his own *personal dictionary*: new words that are added to the original vocabulary. As in a real dictionary, these new words may freely refer to the original vocabulary or other new words in their definitions. Finally, the user communicates with the system by presenting it with a *sentence* that describes a piece of music. Of course this sentence can use only words in the known vocabulary, either the basic vocabulary or the personal dictionary the user has created. This sentence must also "make sense" in terms of it's structure. Expressions that cannot be typed in the underlying type system correspond to meaningless sentences. Language is a powerful metaphor for this system and by presenting it to students in these terms we connect our learning environment to familiar reference points in a student's educational background.

2.1 A Vocabulary of Music

The most basic musical concept is that of a *pitch*. We use note names to designate a *pitch class* (i.e. one of 12 semitones). Coupling a note name with an octave, represented an integer, yields a pitch. We can run the pitch class and octave together into a single name such as `C3` ("C in the third octave") or `Df5` ("D-flat in the fifth octave"). A `rest` (notated `R`) is a special kind pitch.

A `note` is its pitch paired with its duration. Durations can be named (as in `q` for quarter note) or expressed directly as a number of seconds. We write a note using the `@` operator: `C4@Q` would be the fourth octave C played as a quarter note.

From these we can build more complex musical structures as follows:

- `m1 + m2` is the "sequential composition" of `m1` and `m2`; i.e. `m1` and `m2` are played in sequence.
- `m1 | m2` is the "parallel composition" of `m1` and `m2`; i.e. `m1` and `m2` are played simultaneously.
- `Tempo a m` or `m # a` scales the rate at which `m` is played (i.e. its tempo) by a factor of `a`.
- `Trans i m` or `m ^ i` transposes `m` by interval `i` (in semitones).
- `Instrument i m` or `i : m` plays music `m` using instrument `i`.

From this foundation, we can define simple music objects such as an arpeggio or a chord:

```
cMajArp = C4@Q + E4@Q + G4@Q
cMajChord = C4@Q | E4@Q | G4@Q
```

In addition to this basic music vocabulary, Haskore offers *functional abstraction*. That is, the user can define and use functions which capture various musical patterns. Abstraction is what separates Haskore from traditional music notations: it is able to capture the mathematical structure behind the music.

Using these simple music transformations, we can now create interesting music. Although this example is only five lines long, it defines a very complex and pleasing bit of music:

```
delay dur m = R@dur + m

rep f g 0 m = R@0
rep f g n m = m | g (rep f g (n-1) (f m))

run = rep (^ 5) (delay Q) 8 (C4@Q)

cascade = rep (^ 5) (delay en) 8 run
```

Using a small basic vocabulary and functional abstraction we can describe complex patterns of sound in a succinct and formally well understood manner. The syntax here is derived from Haskell; without further student experience it is impossible to tell whether this syntax is appropriate for our intended users.

3 Using Music to Teach Computation

So far, we have addressed the classroom resource: the language and software. We are also interested in the associated pedagogy: how can it be used to teach? What sort of experience can we provide for the students? Here we show a logical sequence of steps that lead to a general understanding of computation. At each step, the student can create associated musical examples to experience the computational principals in a very concrete way. While all of these examples could easily be converted to more familiar domains such as numbers or lists, we believe that music provides interest and motivation that is not present in more traditional problem domains.

Using music to teach the basics of computation requires no prior knowledge of music or music notation. Although it is useful to use a keyboard to illustrate concepts such as octaves, note names, scales, sharps, and flats, the amount of basic music background needed is minimal. Here we also avoid aesthetic aspects of the music: what makes a good melody or harmony. Since we are dealing primarily with patterns of notes here, we find that an interesting pattern sounds "good" no matter what notes are used to form the pattern. This is similar to the effect of viewing an image through a kaleidoscope: the pattern turns an ordinary image into an interesting geometric artifact. We are not arguing that understanding good melodic or harmonic structure is unimportant – it simply is not necessary in this educational context.

Another issue to be resolved is how much computational power we need to incorporate into the language. While the original Haskore can exploit all of the features of Haskell, we require a much less complex language. Some aspects of Haskell, such as most of the numeric capabilities, are clearly useful. Simple data structures such as lists and tuples also of interest. Our plan is to grow our language slowly, only introducing new features when there is a clear educational need in the user community. Since our intention is to study the only the basic foundations of computation we do not need to replicate the functionality of a full programming language.

A simple programming environment will accompany our language. A program is represented by an unordered set of definitions. The student will interact using an editable definition window to hold the current program and a dialog window to enter expressions for evaluation. Expressions that denote music are played on the MIDI synthesizer while others would be printed out. Additional controls will allow definitions and musical objects to be stored and retrieved from files.

Expressions. The study of music begins with the basic vocabulary presented previously: notes, pitch, sequential and parallel combination, and transposition. At this point, students can type in single note expressions such as `C4@Q` and hear them played on the computer.

Next, we combine notes using the `+` and `|` operators. Students can type in simple melodies or chords at this point and hear the effect of these operations. More complex musical phrases can now be described using both of these operators.

Definitions. Once a student is familiar with expressions, the next step is to add simple (parameterless) definitions. For example, by entering the following definition

```
p = C4@Q + D4@Q + E4@Q + F4@Q  + G4@Q
```

the student can play a phrase such as `p+p+p` to illustrate the idea of naming. From here we proceed to multiple definitions that reference each other. As a pencil and paper assignment, students should be able to expand away definitions in an expression using substitution. We can illustrate some simple musical forms such as ABA at this point.

Parameters. The next step is to introduce parameterized definitions. It is important to go slowly and explain the parameterization process as simply as possible. Written assignments to expand out parameterized definitions can be used. Here are a number of simple patterns that can be discussed at this point:

```
twice m = m + m
aba apart bpart = apart + bpart + apart
majorChord m = m | m ^ 4 | m ^ 7
```

The mathematical sub-language can be introduced at this point, allowing expressions such as `x*2` or `y+1`.

Iteration and Recursion. Once students are familiar with parameters, it is then possible to write recursive functions. This requires two new features: patterns and multiple definitions or a conditional operator. At this point, definitions such as the following can be used.

```
repeat 0 m = noMusic
repeat n m = m + repeat (n-1) m
```

There are many simple iterative musical structures, both rhythmic and melodic, available to illustrate iteration. Most of these can be expressed using simple recursive functions and integer arithmetic.

Functional Programming Once the basic ideas of function definition are out of the bag, students can be introduced to many of the idioms of functional programming. Functional composition allows the student to *pipeline* a set of music transformers. Currying is particularly useful in these pipelines: functions such as `^` and `repeat` become music transformers when applied to all but the last argument. For example, this repeats a piece of music three times, transposes it up 5 semitones, and scales the tempo by 2.

```
f = repeat 3 . (^ 5) . (# 2)
```

As another example, a simple modification to the `repeat` function changes the music with each repetition:

```
repeatm 0 m f = R@0
repeatm n m f = m + repeat (n-1) (f m)
```

This can be used to construct abstractions in which music is played faster or higher with each repetition.

Advanced Topics. This progression leads to many more advanced topics, including:

- Lists and sequences.
- Structural manipulation of music (reversing or changing scales, for example).
- Randomness and music.
- Periodic functions.
- Fractal music.

Each of these topics has two sides: one musical and the other computational. We believe that by careful construction of supporting material such as course notes, examples, activities, and projects we can develop new ways of learning about computation while also discussing basic musical forms and idioms.

4 Conclusions

While this research is in its infancy, we believe we have demonstrated that a properly designed programming language of music composition can be used to teach the basic elements of computing in a new and creative manner. One of the primary challenges of teaching computational principles is to generate interest in the problem domain. The use of music allows students to create simple but compelling artifacts with relatively little computational background. The idea of creating artifacts which can be shared and experienced by others is particularly compelling: this rewards creativity in a way that problem solving approaches to these subjects does not. We also conjecture that the use of music will attract students that are not normally interested in mathematical endeavors.

As we have not yet implemented a student-friendly version of Haskore, we cannot comment on implementation issues but we realize that building a language implementation for use by high school students is much more difficult than it would appear. We have not yet addressed problems with error reporting, type checking, or debugging but expect that all of these will present significant technical challenges.

Once we have working software, we intend to address issues related to pedagogy, assessment, and curriculum. For example, this language can used to study topics such as *algebraic reasoning*. The language embodies a formal domain in which equational reasoning can be used as a simple proof technique. Students can formally determine properties such as transitivity or commutivity for user-defined operators.

Another area that we will investigate is using our language to teach basic musical principals. In this area, there is less emphasis on computational principles and more on specific musical structures such chord types, rhythms, melodic construction, proper musical notation, and harmonies.

The basic contribution of this work will not be the language itself but rather our experiences in using the language in an educational setting. We need to demonstrate results to the educational community, not computer scientists. These results should be in the form of objective studies showing concrete educational impact and will ultimately determine the success or failure of this project.