parsel

Final Project - Team 13

Jett Andersen (jca2136, Project Manager)
Andy Hadjigeorgiou (ahh2131, Tester)
Xingzhou He (xh2187, Language Guru)
Kunal Jasty (ksj2114, System Integrator)
Robert Ying (ry2242, System Architect)
Introduction - What is Parsel?
The language that generates your envelopes
Introduction - What is Parsel?

- More and more people are making edm
- More and more people are coding
Introduction - What is Parsel?

- So many people love both, but have no way to combine their hobbies!

- Parsel is here to change that.
main(input[char]) -> (signal) =
    intervalMap(10ms, filterInterval, signalFromWav(input))

filterInterval(interval input) -> interval =
    let freq thing = 1000Hz cutoff(time t) -> freq = envelope(0, 10s, t) * thing
    in applyFilterF with leftRightFilter(cutoff(input.start)), input

leftRightFilter(freq cutoff) -> fsignal = fsignal with
    \( \text{freq f} \) -> sample = if \( f < \text{cutoff} \) then (1, 0) else (0, 1)

evelope(time attack, time decay, time t) -> float =
    if \( t < \text{attack} \) then
        \( t / \text{attack} \)
    else if \( t < \text{decay} \) then
        \( 1 - (t - \text{attack}) / \text{decay} \)
    else 0

# maybe library functions
intervalMap(time width, \text{f(interval, time)} -> interval, signal input) -> signal =
    merge(width, map(f, chop(width, input)))

applyFilterF(fsignal filter, interval input) -> interval =
    (interval with input.start, input.stop, ft with ft(input) * filter)
Parsel

- Declarative, functional, and lazy

- Designed to mask the tedious challenges of audio programming behind a clean and efficient language

- v2 will integrate into any producer’s DAW
Evolution of Parsel

- Basic idea of Parsel preserved since the proposal
- Changes mostly made to address needs / difficulties during implementation
Evolution of Parsel (cont’d)

- Haskell for front-end
  - Alex (Lex) / Happy (Yacc)
  - Language design was changed to make it parsable by yacc
    - “\” tag for inline functions
    - No partial function
Evolution of Parsel (cont’d)

● Compiling Parsel code
  ○ After making the grammar, code generation was completed first
    ■ Accelerating backend progress
    ■ Make first integration tests
  ○ Then semantic analysis was developed
    ■ Check for errors
    ■ Make sure generated code compiles in g++
Evolution of Parsel (cont’d)

● Backend
  ○ C++ is used
  ○ Lots of C++14 features
    ■ Generic lambdas
      ● Makes dealing with generated types much easier =)
Evolution of Parsel (cont’d)

● Libraries used
  ○ To speed up development, we used general libraries for audio processing
    ■ libsndfile (Reading / writing .wav file)
    ■ fft4g (Fast Fourier transform implementation)
Runtime/Software Development Environment

- Runs in Ubuntu 14.10 x64

- Build using Cabal, Haskell’s Common Architecture for Building Applications and Libraries

- Supports g++ >= 4.9. Tested on g++ 5.1.1

- Git + Github for version control
Type Checking

- Compile-time type checking for undefined variables, type mismatch errors etc
Data Types That Make Us Special

- signal
  - Designed to be treated as a continuous signal
  - Wraps sampling and buffers of typical audio programming

- frequency-domain signal (fsignal)
  - Allows for efficient filtering and other effects
  - We use a highly optimized C++ Fourier transform
More Data Types

- It’s the little things that count
  - sample
    - multi-channel complex value
  - time
  - freq
Laziness

- Prevents the evaluation of unused data
  - this works :D

- Allows us to work with infinite lists
  - not quite working yet :(
Laziness

- An infinite list of intervals
  - Allows the user to apply effects to audio over time
  - Until the end of time!

- All this without having to write a single loop
Example Program

```haskell
1 main(input[Char]) -> (signal) =
2   intervalMap(10ms, filterInterval, signalFromWav(input))
3
4 filterInterval(interval input) -> interval =
5   let freq thing = 1000Hz cutoff(time t) -> freq = envelope(0, 10s, t) * thing
6   in applyFilterF with leftRightFilter(cutoff(input.start)), input
7
8 leftRightFilter(freq cutoff) -> fsignal = fsignal with
9   \(freq f) -> sample = if f < cutoff then (1, 0) else (0, 1)
10
11 envelope(time attack, time decay, time t) -> float =
12   if t < attack then
13     t / attack
14   else if t < decay then
15     1 - (t - attack) / decay
16   else 0
17
18 # maybe library functions
19 intervalMap(time width, f(interval, time) -> interval, signal input) -> signal =
20   merge(width, map(f, chop(width, input)))
21
22 applyFilterF(fsignal filter, interval input) -> interval =
23   (interval with input.start, input.stop, ft with ft(input) * filter)
24```
generated code:

```c
main(f1[char]) -> (signal) =
let float c = if length(f1) > 5 then 3300 else 10000
in signal with \(\text{float t} \rightarrow \text{complex} = \sin(t \times c)\)
```

```c
psl::Chunk<std::vector<psl::Chunk<char>>> args[argc];
for (int i = 0; i < argc; i++) {
    std::vector<psl::Chunk<char>> chk(strlen(argv[i])+1);
    std::transform(argv[i], argv[i]+strlen(argv[i])+1, chk.begin(), chr2Chunk);
    psl::set(args[i], psl::toChunk([]{ return chk; }));
}

bool B = false, success;
auto fc = out<>(args[1]);
psl::Chunk<psl::Signal> writer1(psl::makeWriter(args[argc+(-2)], fc[0], args[argc-1]));
do {
    B = !B;
    success = writer1().fillBuffer(B);
} while (success);
```
Project Management

- Agile development
- Weekly meetings to assign tasks + discuss

Git Commits
## Modules

<table>
<thead>
<tr>
<th>Module</th>
<th>Input</th>
<th>Output</th>
<th>Authors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lexer</td>
<td>PSL source</td>
<td>Tokens with attributes</td>
<td>Robert Ying, Jett Andersen</td>
</tr>
<tr>
<td>Parser</td>
<td>Tokens with attributes</td>
<td>abstract syntax tree with attributes</td>
<td>Robert Ying, Jett Andersen, Derek He, Andy Hadjigeorgiou</td>
</tr>
<tr>
<td>Semantic analyzer</td>
<td>abstract syntax tree with attributes</td>
<td>abstract syntax tree with symbol table</td>
<td>Jett Andersen, Kunal Jasty, Andy Hadjigeorgiou</td>
</tr>
<tr>
<td>Code generator</td>
<td>abstract syntax tree with symbol table</td>
<td>C++ code</td>
<td>Jett Andersen, Derek He</td>
</tr>
<tr>
<td>C++ compiler</td>
<td>C++ code</td>
<td>executable machine code</td>
<td>GCC Contributors¹</td>
</tr>
<tr>
<td>Standard library functions</td>
<td>C++ code</td>
<td>executable machine code</td>
<td>Robert Ying, Derek He, Kunal Jasty, Jett Andersen</td>
</tr>
</tbody>
</table>
Testing

- Python program runs in shell, watching for file-system changes
- Creates compiler, compiled c++, binary file, and runs parsel program
- Used ‘diff’ to compare compiled c++ with sample c++
- Only updates relevant parts of project
- Improved development productivity
super cool demo
Conclusions

- laziness is hard
  - both real life group laziness
  - and lazy function evaluation
- type-safety is really nice, but difficult to get working
- think more carefully about scope before starting