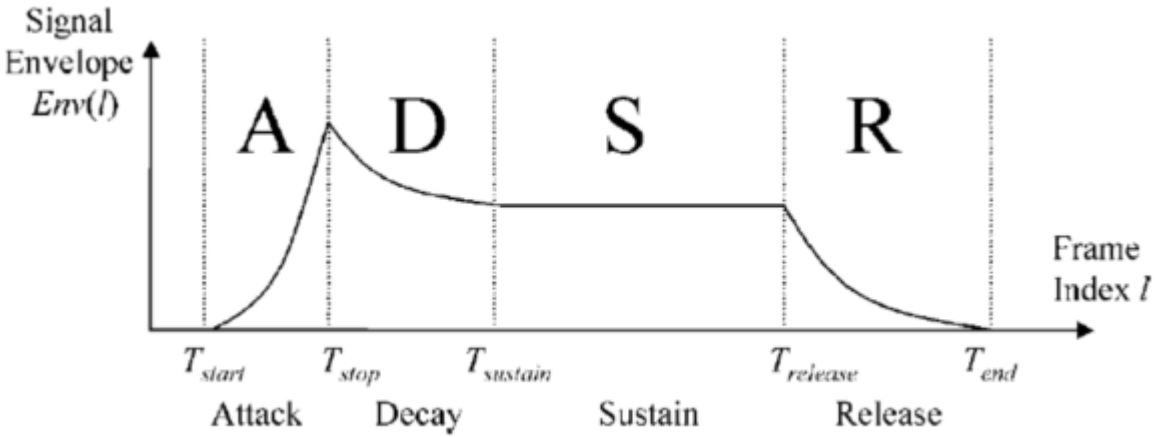
parsel

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?

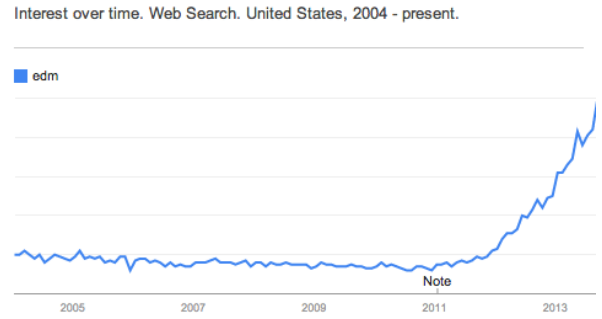
The language that generates your envelopes

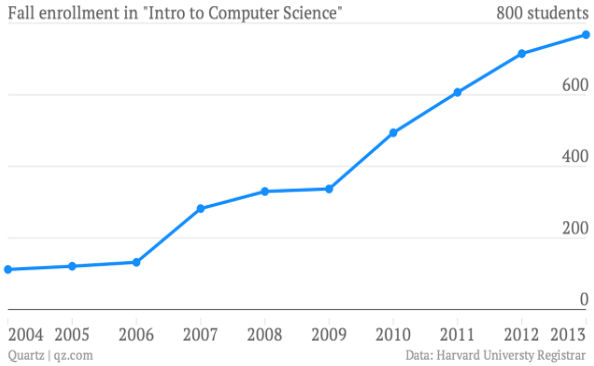


Introduction - What is Parsel?

- 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.

Parsel

```
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
    \(freq f) -> sample = if f < cutoff then (1, 0) else (0, 1)

envelope(time attack, time decay, time t) -> float =
    if t < attack then
        t / attack
    else if t < decay then
        1 - (t - attack) / decay
    else 0

# maybe library functions
intervalMap(time width, 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 **C**ommon **A**rchitecture for **B**uilding **A**pplications and **L**ibraries
- Supports g++ >= 4.9 . Tested on g++ 5.1.1
- Git + Github for version control

```
Name:           parsel
Version:        0.1
Cabal-Version:  >= 1.2
License:        Apache
Author:         ...
Synopsis:       ...
Build-Type:     Simple

Executable parsel
  Main-Is:        Main.hs
  Other-modules:  AlexToken,HappyParser
  Hs-Source-Dirs: src
  Build-Depends:  base >= 4,array,mtl,containers,process
  GHC-Options:    -Wall

-- Executable parsel
--   Main-Is:        Main.hs

--   Hs-Source-Dirs: src
--   Build-Depends:  base >= 4, array, containers, mtl, regex-compat
--   Build-Tools:    alex, happy
--   GHC-Options:    -Wall

```

Type Checking

- Compile-time type checking for undefined variables, type mismatch errors etc

```
Error: Undef (Symbol "interval")
MisusedType "Symbol not found: interval"
MisusedType "Symbol not found: interval"
MisusedType "Symbol not found: interval"
Undef (Symbol "merge")
MisusedType "Symbol not found: merge"
MisusedType "Symbol not found: merge"
Undef (Symbol "psl::lessThan")
MisusedType "Symbol not found: psl::lessThan"
MisusedType "Symbol not found: psl::lessThan"
Undef (Symbol "psl::divide")
MisusedType "Symbol not found: psl::divide"
MisusedType "Symbol not found: psl::divide"
Undef (Symbol "psl::lessThan")
MisusedType "Symbol not found: psl::lessThan"
MisusedType "Symbol not found: psl::lessThan"
Undef (Symbol "psl::minus")
MisusedType "Symbol not found: psl::minus"
MisusedType "Symbol not found: psl::minus"
Undef (Symbol "fsignal")
MisusedType "Symbol not found: fsignal"
MisusedType "Symbol not found: fsignal"
WrongType [Type (Symbol "time")] [Type (Symbol "float")]
WrongType [Type (Symbol "float")] [Type (Symbol "freq")]
WrongType [Type (Symbol "time")] [Type (Symbol "interval")]
```

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

```
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
    \(freq f) -> sample = if f < cutoff then (1, 0) else (0, 1)

envelope(time attack, time decay, time t) -> float =
    if t < attack then
        t / attack
    else if t < decay then
        1 - (t - attack) / decay
    else 0

# maybe library functions
intervalMap(time width, 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)
```

Generated Code

```
1 main(f1[char]) -> (signal) =
2     let float c = if length(f1) > 5 then 3300 else 10000
3     in signal with \(float t) -> complex = sin(t * c)
4
```

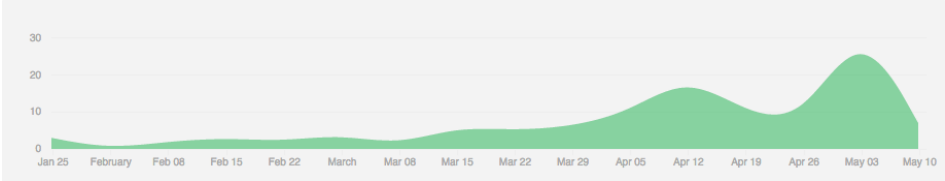
```
    psl::Chunk<std::vector<psl::Chunk<char>>> args[argc];
    for (int i = 0; i < argc; i++) {
        std::vector<psl::Chunk<char>> chk(strlen(argv[i])+1);
        std::transform(argv[i], argv[i]+strlen(argv[i])+1, chk.begin(), chr2Chunk);
        psl::set(args[i], psl::toChunk([=]{ return chk;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

    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

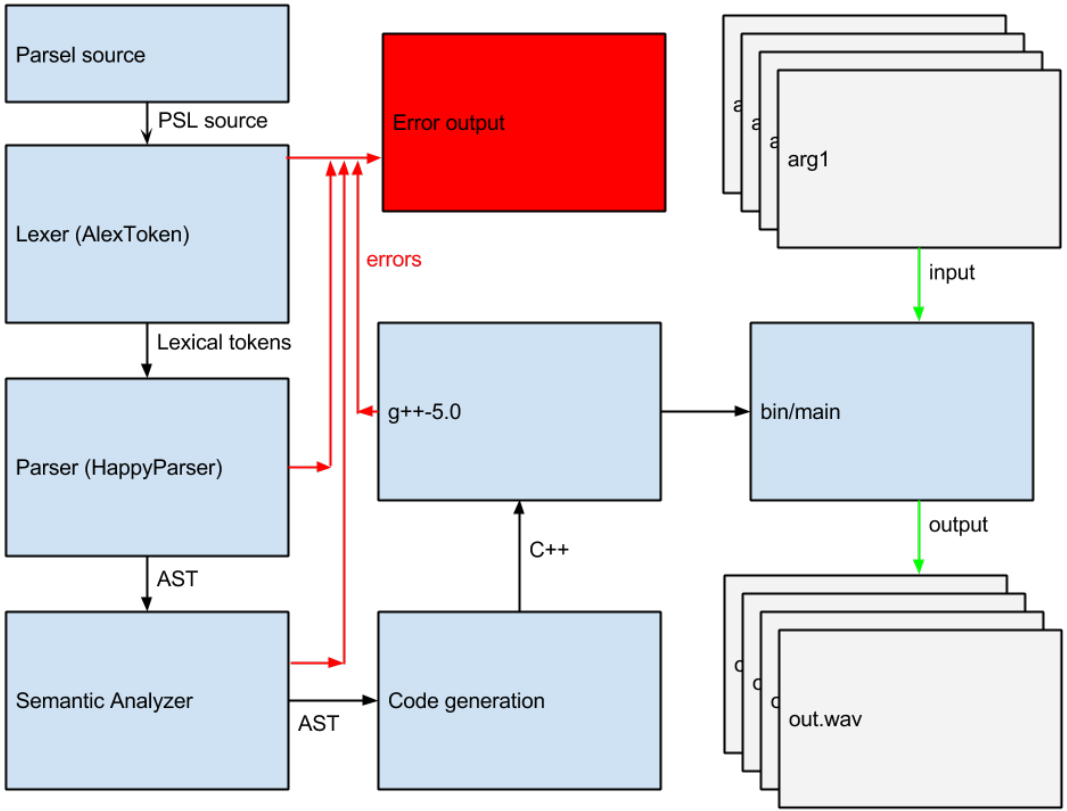
- Agile development
- Weekly meetings to assign tasks + discuss




Git Commits



Translator Architecture



Modules

Module	Input	Output	Authors
Lexer	PSL source	Tokens with attributes	Robert Ying, Jett Andersen
Parser	Tokens with attributes	abstract syntax tree with attributes	Robert Ying, Jett Andersen, Derek He, Andy Hadjigeorgiou
Semantic analyzer	abstract syntax tree with attributes	abstract syntax tree with symbol table	Jett Andersen, Kunal Jasty, Andy Hadjigeorgiou
Code generator	abstract syntax tree with symbol table	C++ code	Jett Andersen, Derek He
C++ compiler	C++ code	executable machine code	GCC Contributors[1]
Standard library functions	C++ code	executable machine code	Robert Ying, Derek He, Kunal Jasty, Jett Andersen

Testing

```
Running binary ...

----------------------

Running cabal build ...

Building parsel-0.1...
Preprocessing executable 'parsel' for parsel-0.1...
line-map.c: file "<command-line>" left but not entered
line-map.c: file "<command-line>" left but not entered
line-map.c: file "<command-line>" left but not entered
[ 8 of 11] Compiling Generators[boot] ( src/Generators.hs-boot, dist/build/parsel/parsel-tmp/Generators.o-boot )
[10 of 11] Compiling Generators       ( src/Generators.hs, dist/build/parsel/parsel-tmp/Generators.o )
Linking dist/build/parsel/parsel ...

----------------------


Deleting old binary ...


----------------------


Compiling parsel file to C++ ...

mkdir -p bin; mkdir -p obj; g++-4.9 -g -std=c++14 -Icpp/include -Icpp -c main.cpp -o obj/main.o
mkdir -p bin; mkdir -p obj; g++-4.9 -g -std=c++14 obj/* -o bin/main -Lcpp/lib -lsndfile

----------------------


----------------------


----------------------


Running binary ...

----------------------

Comparing compiled files ...

----------------------
```

change to code-gen = rebuild compiler

waiting for next file change

```
 1 module Generators where
 2
 3 import Data.List
 4 import Data.Ord
 5
 6 import AST
 7 import Generators2
 8
 9
10 -- sdecs, sdefs, topdecs, code, mainloop
11 genTopDefs :: [TopDef] -> ([Char], [Char], [Char], [Char], [Char])
12 genTopDefs [] = ("", "", "", "", "")
13 genTopDefs [td] = genTopDef td
14 genTopDefs (td:tds) = (sdec ++ sdecs, sdef ++ sdefs, d ++ ds, c ++ cs, ml)
15     where   (sdec, sdef, d, c, mlNew) = genTopDef td
16             (sdecs, sdefs, ds, cs, mlOld) = genTopDefs tds
17             ml = maximumBy (comparing length) [mlNew, mlOld]
18
19
20 -- sdecs, sdefs, topdecs, code, mainloop
21 genTopDef :: TopDef -> ([Char], [Char], [Char], [Char], [Char])
22 genTopDef (Def d) = ("", "", td, c, ml)
23     where (td, c, ml) = genDef d
24 genTopDef (Struct (Symbol sym) tsyms) = (sdec, sdef, "", "", "")
25     where   sdec = "struct " ++ sym ++ ";\n"
26             sdef = "struct " ++ sym ++ " {\n" ++ members ++ "};\n"
27             members = concat $ map ((++";\n") . genTsym) tsyms
28
29
30 -- topdecs, code
31 genDefs :: [Def] -> ([Char], [Char])
32 genDefs [] = ("", "")
33 genDefs [def] = (topdef, code)
34     where (topdef, code, _) = genDef def
35 genDefs (def:defs) = (topdef ++ topdefs, code ++ codes)
36     where   (topdef, code, _) = genDef def
37             (topdefs, codes) = genDefs defs
38
39
40 -- topdecs, code, mainloop
41 genDef :: Def -> ([Char], [Char], [Char])
42 genDef (FuncDef (Symbol sym) tsyms rt expr)
43     | sym == "main" =
44         -- Rewrite with Chunk<vector<Chunk<char>>>
45         let mainloop = "psl::Chunk<std::vector<psl::Chunk<char>>> args[argc];\n"
46                 ++ "for (int i = 0; i < argc; i++) {\n"
47                 ++ "std::vector<psl::Chunk<char>> chk(strlen(argv[i])+1);\n"
48                 ++ "std::transform(argv[i], "
49                 ++ "argv[i]+strlen(argv[i])+1, chk.begin(), chr2Chunk);\n"
50                 ++ "psl::set(args[i], psl::toChunk([=]{ return chk; }));\n"
51                 ++ "}\n\n"
52                 ++ "bool B = false, success;\n"
53                 ++ "auto fc = out()(" ++ args ++ ");\n"
54                 ++ concat (map dec [tsize,tsize-1..1])
55                 ++ "do {\n"
"Generators.hs" 112L, 4369C written                          32,1          Top
```

- 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