ПИШЕМ EDSL НА HASKELL

СКАЗ О СОЗДАНИИ МУЗЫКАЛЬНОГО СИНТЕЗАТОРА

Антон Холомьёв

anton.kholomiov@gmail.com
github: anton-k

Csound-expression

- Библиотека для создания электронной музыки и синтезаторов
- Генератор кода Csound

Почему Csound?

- Эффективный (написан на C)
- Мощный (около 1000 готовых алгоритмов работы со звуком)
- Поддерживается и развивается

Почему Haskell?

Csound имеет устаревший синтаксис

- Нет пользовательских типов данных
- Нет функций высшего порядка
- Императивный подход (goto, глобальные переменные и тд)
- Нет модулей, библиотек, пакетов (переиспользование кода через: Ctrl-C, Ctrl-V или #include)

Сравним языки

Haskell

- Отличный синтаксис
- Не умеет работать со звуком

Csound

- Плохой синтаксис
- Огромный потенциал для работы со звуком

Путь EDSL

render :: Expr → String

```
e1 = readWav «file.wav»
e2 = lowPassFilter 1500 0.2 e1
e3 = smallHall e2

main = compile (render e3)
```

Суть EDSL(deep)

```
> 1 + 2
3
```

```
data T = Prim Int
       | Add T T

instance Num T where ...

> 1 + 2
Add (Prim 1) (Prim 2)
```

Структура EDSL

- Синтаксис Csound (Динамический AST)
- Наша модель в Haskell (Типизирован через newtype)
- Функции, библиотеки поверх нашей модели

С чего начать?

- Стоит начать с описания в Haskell синтаксиса языка, код которого вы будете генерировать

render :: Ast → String

```
type Ast = CsoundAst
type CsoundAst = (Options, [Instr], [Note])
type Instr = (InstrName, Expr)
type InstrName = Int
type Note = (InstrName, [Param])
...
```

Наша модель

```
newtype Model a = Model (State Context a)
runModel : Model () → Context → Ast


newtype Sig    = Sig    (Model Ast)     – сигналы
newtype D      = D      (Model Ast)     – числа
newtype Str    = Str    (Model Ast)     – строки
...
```

Пример: Учёт массивов

```
allocTab :: Tab → Model Int
allocTab table = do
    ctx ← getContext
    let (n, ctx') = newTableIndex table ctx
    setContext ctx'
    return n
```

Пример: Учёт массивов

```
oscil :: Sig → Sig → Tab → Sig
oscil (Sig amp) (Sig freq) table = Sig $ do
    ampExpr ← amp
    freqExpr ← freq
    n ← allocTab table
    return $ csoundOscil ampExpr freqExpr n
```

Под капотом

- Генерация различных идентификаторов
- Создание иллюзии модульности компонент (выход за пределы модели инструмент+ноты)
- Создание таблиц для «аналоговых» осцилляторов
- ...

Всё может измениться

Генерация функций для аудио алгоритмов

происходит из документации!

Бибилотеки

- Ноты
- UI
- Сэмплер
- FRP

Ноты

```
data Sco a


mel :: [Sco a] → Sco a
har :: [Sco a] → Sco a
del  :: Duration → Sco a → Sco a
str   :: Factor → Sco a → Sco a


sco :: (Arg a, Sigs b) => (a → IO' b) → Sco a → b
```

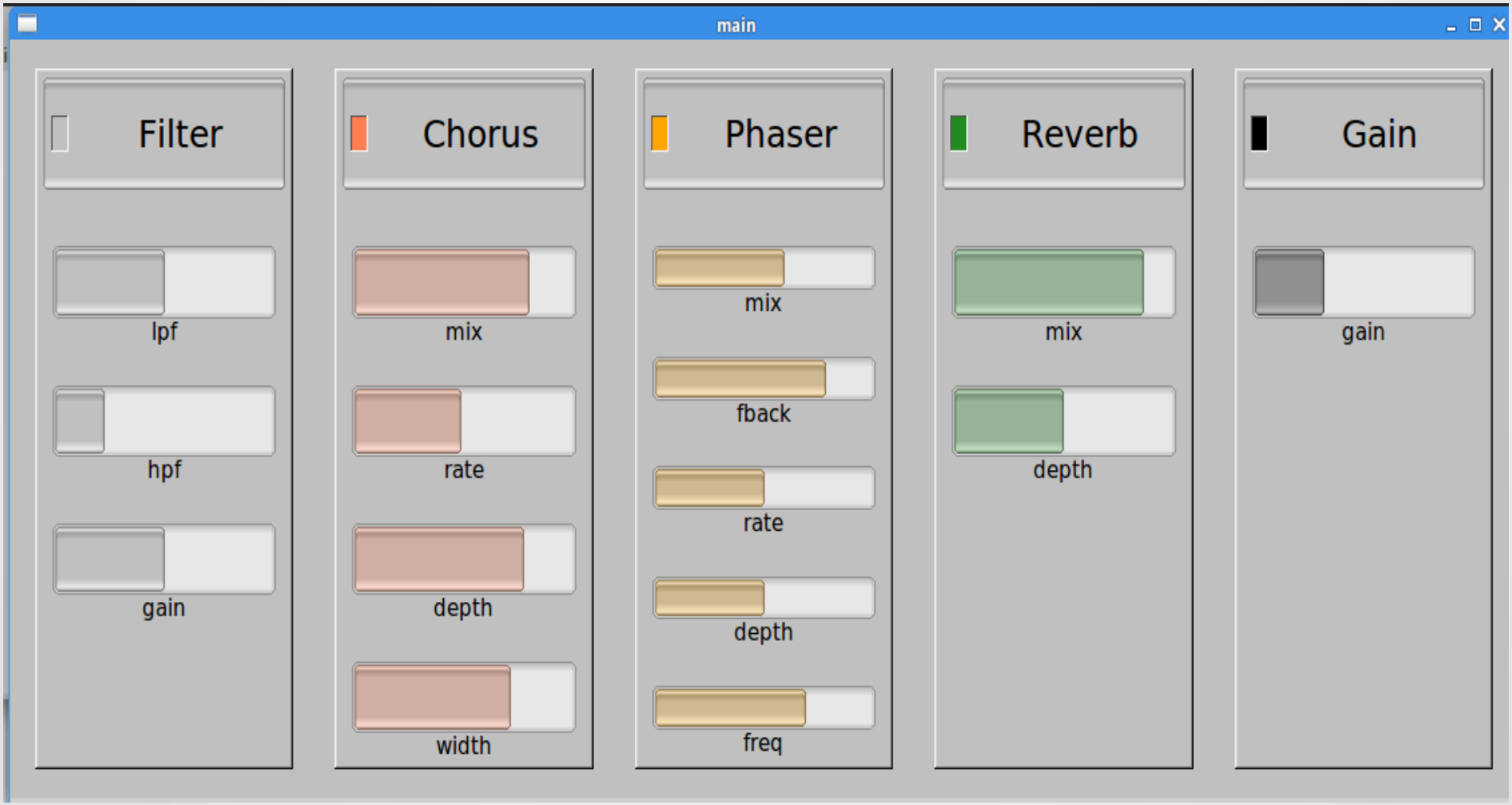
UI = данные + картинка

```
data Src a = (Gui, IO' a)

hor, ver :: [Gui] → Gui

hlift,   vlift   :: (a → b)         → Src a → Src b
hlift2, vlift2 :: (a → b → c) → Src a → Src b → Src c
```

UI


main
Filter
lpf
hpf
gain
Chorus
mix
rate
depth
width
Phaser
mix
fback
rate
depth
freq
Reverb
mix
depth
Gain
gain

UI код

```
module Fx where

import Csound.Base

main = dac $ lift1 (\fx -> fx $ fromMono $ saw 110) $ fxHor
        [ uiFilter False 0.5 0.5 0.5
        , uiChorus False 0.5 0.5 0.5 0.5
        , uiPhaser False 0.5 0.5 0.5 0.5 0.5
        , uiReverb True  0.5 0.5
        , uiGain   True  0.5 ]
```

Сэмплер

```
newtype Sam = Sam (Reader Bpm (Sig, Dur))

wav :: String → Sam
lim  :: Dur → Sam → Sam
mel, har :: [Sam] → Sam
del :: Dur → Sam → Sam
```

FRP

```
type Procedure a = a → IO' ()

data Evt a = Evt (Procedure a → IO' ())

instance Functor Evt
instance Monoid (Evt a)
filter :: (a → Bool') → Evt a → Evt a
cycle :: [a] → Evt b → Evt a
...
```

Удаление общих подвыражений

a = foo 34

b = bar a a → b = bar (foo 34) (foo 34)

- Алгоритм Ершова (см Oleg Kiselyov: Implementing Explicit and Finding Implicit Sharing in Embedded DSLs)

Библиотека data-fix-cse

```
data Expr = Prim Int
          | Add Expr Expr
          | Mul Expr Expr
```

```
data E a = Prim Int
         | Add a a
         | Mul a a

data Fix f  = Fix (f (Fix f))

type Expr = Fix E
```

Библиотека data-fix-cse

```
type VarName = Int
type Dag f = IntMap (f VarName)


cse :: (Eq (f Int), Ord (f Int), Traversable f)
       => Fix f -> Dag f
```

Библиотека data-fix-cse

```
{-# Language
            DeriveFunctor,
            DeriveFoldable,
            DeriveTraversable #-}

data E a = Prim Int
             | Add a a
             | Mul a a
            deriving (Functor, Foldable, Traversable)
```

Найдите помощника!

- Основная идея для реализации модели была подсказана в email рассылке Csound
- Энтузиазм к началу новой версии был найден на московской встрече хаскелистов
- Алгоритм CSE был случайно найден в статье Олега Киселёва

Планы

- Написать библиотеку инструментов
- Обновить документацию
- Сделать приятный/понятный пользователю сайт
- Обновить библиотеку для GHC-7.10

Итоги

- Начните с описания целевого языка в Haskell (или поищите на Hackage). Так мы делаем AST
- EDSL = AST + Наша модель + Библиотеки-надстройки
- Наша модель это State с контекстом. Новые типы – обёртки вокруг AST целевого языка
- Генерируйте код стандартных функций из доков
- Алгоритм Ершова для CSE

Итоги

- Ищите помощников (идеи, встречи, email-рассылки)
- Старайтесь как можно раньше сделать прототип. Вначале делается урезанный функционал, каркас.

Спасибо за внимание