Arrp
A Functional Language with Multi-dimensional Signals and Recurrence Equations

Jakob Leben, PhD student
University of Victoria, Canada

Workshop on Functional Art, Music, Modeling and Design (FARM), September 24, 2016, Nara, Japan.
**Motivation**

- **Domains:**
  - Audio synthesis and analysis
  - (Video, sensor arrays, communication, multi-media compression)

- Multi-dimensional and multi-rate signals

- Good code reusability

- High performance, real-time execution
• Signal = Infinite array (multi-dimensional)

• Array semantics inspired by MATLAB, Octave, Numpy

• Polymorphic, higher-order functions, type inference

• Polyhedral modeling and optimization [1,2,3]

1: Sine wave:

\[
sine\_wave(f) = [t \rightarrow \sin(f \cdot t \cdot 2\pi)]
\]

2: Differentiation:

\[
d(x) = [n \rightarrow x[n+1] - x[n]]
\]

3: Multi-rate processing:

\[
downsampling(k,x) = [t \rightarrow x[k \cdot t]]
\]
1: Recursive use of name “y”

\[
lp(a,x) = y = [ \\
0 \rightarrow 0; \\
t \rightarrow a \times x[t] + (1-a) \times y[t-1]
]\]

2: Recursion using keyword “this”

\[
lp(a,x) = [ \\
0 \rightarrow 0; \\
t \rightarrow a \times x[t] + (1-a) \times this[t-1]
]\]
1: 5 harmonics:

```plaintext
a = [5,~: i,t -> sin((i+1)/sr*t*2*pi)];
b = [~,5: t,i -> sin((i+1)/sr*t*2*pi)];
```

2: Differentiation across time or across channels:

```plaintext
dt_a = [5,~: i,t -> a[i,t+1] - a[i,t]];
dc_a = [4,~: i,t -> a[i+1,t] - a[i,t]];
```

How to reuse functions “sine_wave” and “d”?
1: 5 harmonics, curried:

```javascript
a = [5: i -> [t -> sin((i+1)/sr*t*2*pi)]];  
b = [t -> [5: i -> sin((i+1)/sr*t*2*pi)]];  
```

2: Partial application:

```javascript
a[3] ## size [~]  
```
Array currying and partial application

1: “sine_wave” uncurried into “a”

\[
a = [5: i \mapsto \text{sine}_\text{wave}((i+1)/sr)];
\]

2: “d” uncurried into “dt_a”

\[
dt_a = [\#a: i \mapsto \text{d}(a[i])];
\]
1: Pointwise

```plaintext
a = [5: i -> sine_wave((i+1)/sr)];

\[
d(x) = [n -> x[n+1] - x[n]];
\]

dc_a = d(a); ## size [4,~]
```

2: Pointwise and broadcasting

```plaintext
sine_wave(f) = [t -> sin(f*t*2*pi)];

b = sine_wave([5: i -> i+1]/sr);
## size [~5]
```

Broadcasting

\[
c[i,j] = a[i] + b[i,j]
\]
1: \( n < 5 \) and \( n + 1 < 5 \) \( \rightarrow \) \( n < 4 \)

\[
d(x) = [n \rightarrow x[n+1] - x[n]];
\]

\[
dc_a = d(a); \quad \#\# \text{size [4,\~]}
\]

2: \( i < 5 \)

\[
dt_a = [i \rightarrow d(a[i])]; \quad \#\# \text{size [5,\~]}
\]
map(f,a) = [i -> f(a[i])];

scan(f,a) = [
    0 -> a[0];
    i -> f(this[i-1], a[i]);
];

fold(f,a) = s[#a-1]
    where s = scan(f,a);
Abstraction in signal processing

delay(v,a) = [0 -> v; t -> a[t-1]];

win(size,hop,x) =
[t -> [size: k -> x[t*hop + k]]];

1: Recursive LP filter:

lp(a,x) =
y = a*x + (1-a)*delay(0,y);

2: FIR filter:

sum = fold(\a,b -> a + b);

fir(k) =
map(\w -> sum(w*k)) . win(#k,1);
Arrp

Polyhedral scheduling

Model

\[
\begin{align*}
x &= [t \rightarrow f(t)]; \\
y &= [t \rightarrow x[2t + 1] - x[2t]];
\end{align*}
\]

\textbf{Schedule} (Bondhugula et al. 2008)

\[
\begin{align*}
\text{for } t &= 0.. \\
&\quad x[t] = f(t)
\end{align*}
\]

\[
\begin{align*}
\text{for } t &= 0 .. \\
&\quad y[t] = x[2t+1] - x[2t]
\end{align*}
\]

\[
\begin{align*}
\text{for } u &= 0.. \\
&\quad x[u] = f(u) \\
&\quad \text{if } u \mod 2 == 1 \\
&\quad \quad y[u/2] = x[u] - x[u-1]
\end{align*}
\]

\textbf{function period()}

\[
\begin{align*}
\text{for } u &= 0..1 \\
&\quad x[u] = f(u) \\
&\quad y = x[1] - x[0] \\
&\quad \text{output}(y)
\end{align*}
\]
FIR filter

input x: [~] real64,
    coefs: [10] real64;

map = ...
win = ...
sum = ...
fir(k) =
    map(w -> sum(w*k))
    . win(#k,1);
main = fir(coefs, x);
Polyhedral scheduling

FIR filter schedule

s_0

s_1

prelude
period

parallel
template <typename IO>
class alignas(16) program
{
   public:
      IO * io;
      void prelude();
      void period();
   private:
      double b[4];
      double coefs[3];
      int b_ph = 0;
};

template <typename IO>
inline void program<IO>::
    prelude()
{
    ...
    io->input_coefs(coefs); ...
}

template <typename IO>
inline void program<IO>::
    period()
{
    double x;
    double main;
    double a;

    io->input_x(x);
    a = x * coefs[0];
    b[2+b_ph&3] = a;
    for (int c1=1; c1<=2; c1+=1)
    {
        a = x * coefs[c1];
        b[-c1+2+b_ph&3] = b[-c1+2+b_ph&3] + a;
    }
    main = b[0+b_ph];
    io->output(main);
    b_ph = b_ph+1&3;
}
Experimental evaluation

Legend: □ Arrp □ Hand-written C++

Speed (higher = better)

Buffers (lower = better)

Lines of code (lower = better)
Future work

Language:

- Algebraic data types
- Recursive functions
- Foreign function calls

Performance:

- Multi-threading, GPU code (in LLVM?)
- Performance comparison: Faust, Kronos, StreamIt, etc.
- Extensive evaluation of polyhedral scheduling

- Applying the principles to other languages?
Author:
Jakob Leben - jakob.leben@gmail.com

Evaluated Arrp and C++ Code:
http://webhome.csc.uvic.ca/~jleben/farm2016

Arrp Website:
http://mess.cs.uvic.ca/arrp

Arrp Compiler:
https://github.com/jleben/arrp