Faust Term Rewriting Extension

Albert Gräf

Dept. of Music Informatics

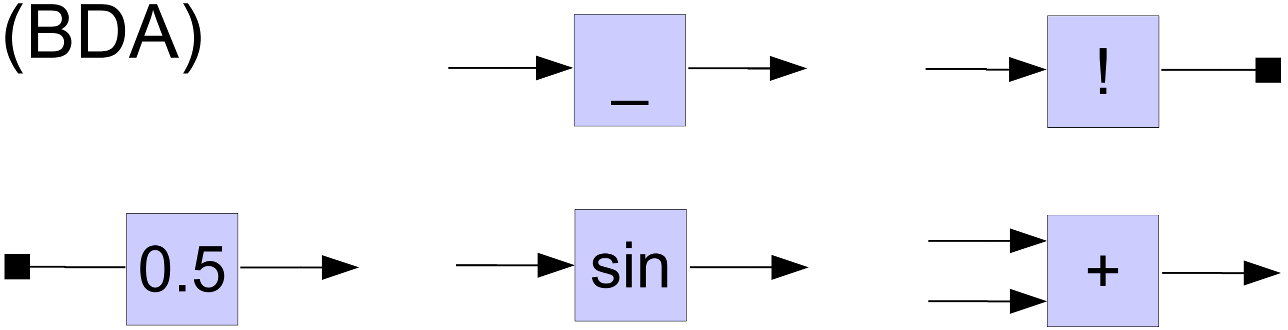
Signal Processing with Faust

```
// basic amplifier
vol = hslider("vol", 0.3, 0, 3.5, 0.01);
process(x,y) = vol*x, vol*y;
```

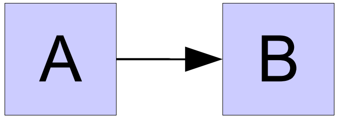
- *Functional signal processing* language, processing of *synchronous streams* of samples.
- *Formal semantics* turns Faust programs into formal specifications of signal processors.
- Specifications are *executable*, sophisticated optimizations, generates competitive C++ code.
- Works with *different platforms and environments*, just recompile.

Faust Block Diagram Algebra (BDA)

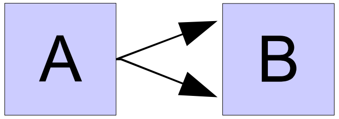
Basic blocks



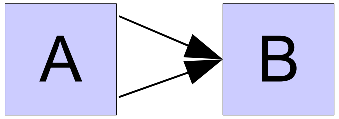
Combining blocks



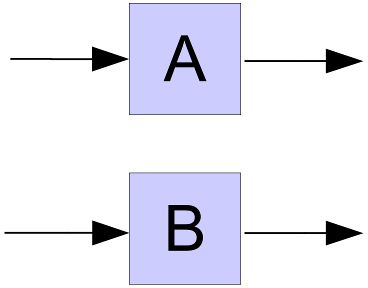
`A:B`



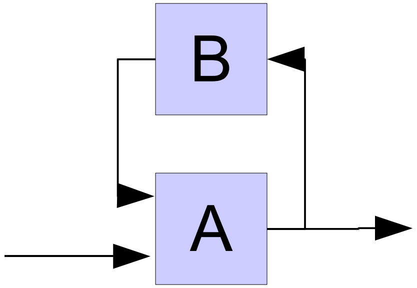
`A<:B`



`A:>B`



`A,B`



`A~B`

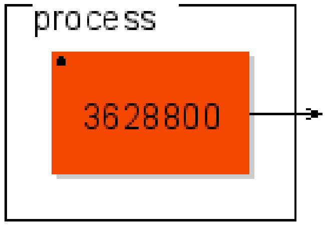
Faust Term Rewriting Extension

Term Rewriting Extension

macro definition

```
fact(0) = 1;
fact(n) = n*fact(n-1);
process = fact(10);
```

macro call



- Faust signal processors are **terms** in the block diagram algebra (BDA)
- **Term rewriting** provides us with a means to manipulate BDA terms in an **algebraic fashion** at **compile time**

Term Rewriting in a Nutshell

$$\begin{aligned}\mathrm{top}(\mathrm{push}(s,x)) &\to x\\ \mathrm{pop}(\mathrm{push}(s,x)) &\to s\end{aligned}$$

term rewriting system

reduction relation

terms as “data”

$$\mathrm{top}(\mathrm{pop}(\mathrm{push}(\mathrm{empty},1))) \to \mathrm{top}(\mathrm{empty})$$

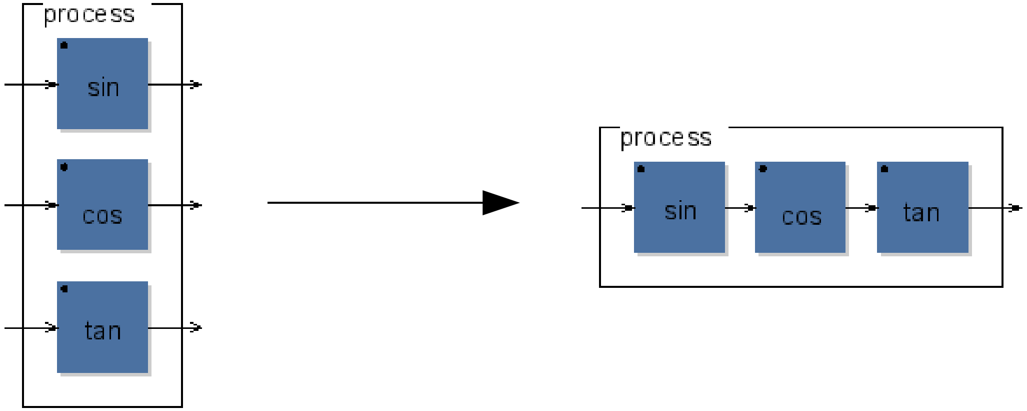
normal form

- Whitehead et al: *universal algebra* (1898)
- Term rewriting and equational logic (1970s)
- Term rewriting as programming language (O'Donnell, 1985)
- Used in computer algebra, compiler backends, FPLs, ...
- **Here:** TR as a ***macro language***

Rewriting BDA Terms

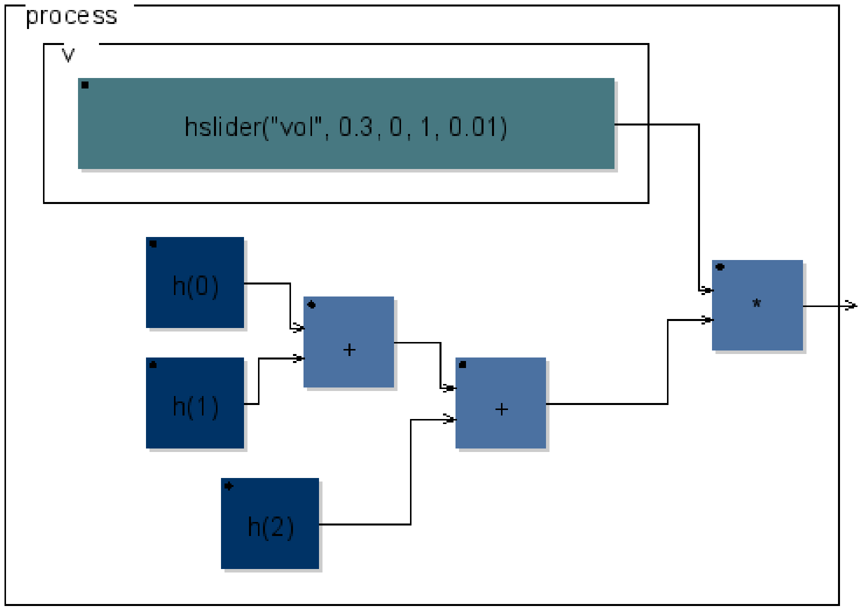
```
serial((x,y))      = serial(x) : serial(y);
serial(x)          = x;
process            = serial((sin,cos,tan));
```

```
serial(((sin,cos),tan))
 → serial((sin,cos)) : serial(tan)
 → (serial(sin) : serial(cos)) : serial(tan)
 → sin : serial(cos) : serial(tan)
 → sin : cos : serial(tan)
 → sin : cos : tan
```



Custom BDA Ops

```
fold(1,f,x) = x(0);
fold(n,f,x) = f(fold(n-1,f,x),x(n-1));
fsum(n)     = fold(n,+);

f0 = 440; a(0) = 1; a(1) = 0.5; a(2) = 0.3;

h(i)        = a(i)*osc((i+1)*f0);
v           = hslider("vol", 0.3, 0, 1, 0.01);
process     = v*fsum(3,h);
```

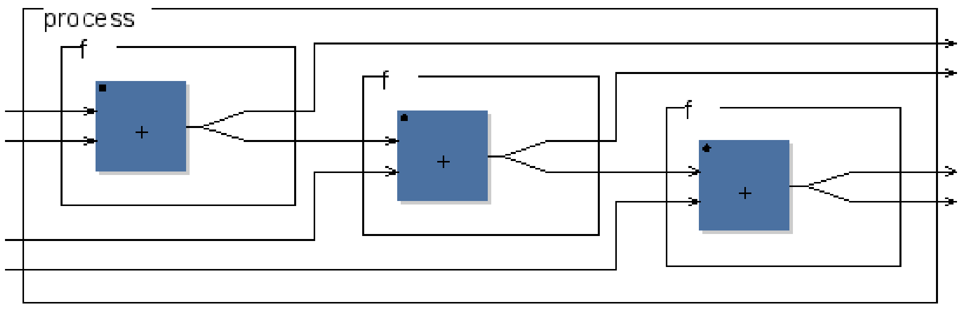


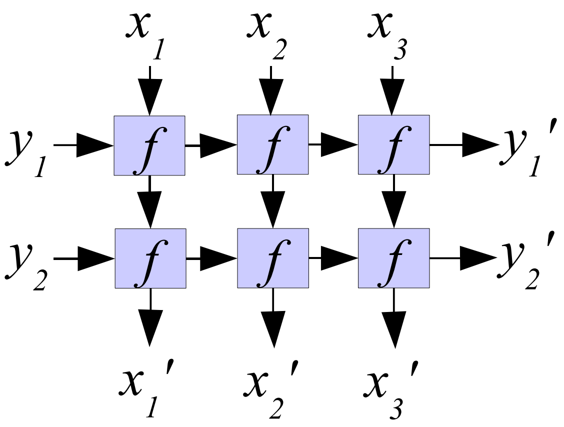
Faust Term Rewriting Extension

```
g(1,f)   = f;
g(m,f)   = (f, r(m-1)) : (_, g(m-1,f));

h(1,m,f) = g(m,f);
h(n,m,f) = (r(n+m) <: (!,r(n-1),s(m),
             (_,s(n-1),r(m) : g(m,f)))) :
           (h(n-1,m,f), _);

r(1) = _; r(n) = _,r(n-1); // route through
s(1) = !; s(n) = !,s(n-1); // skip

f        = + <: _,_; // cell function
process  = h(2,3,f);
```

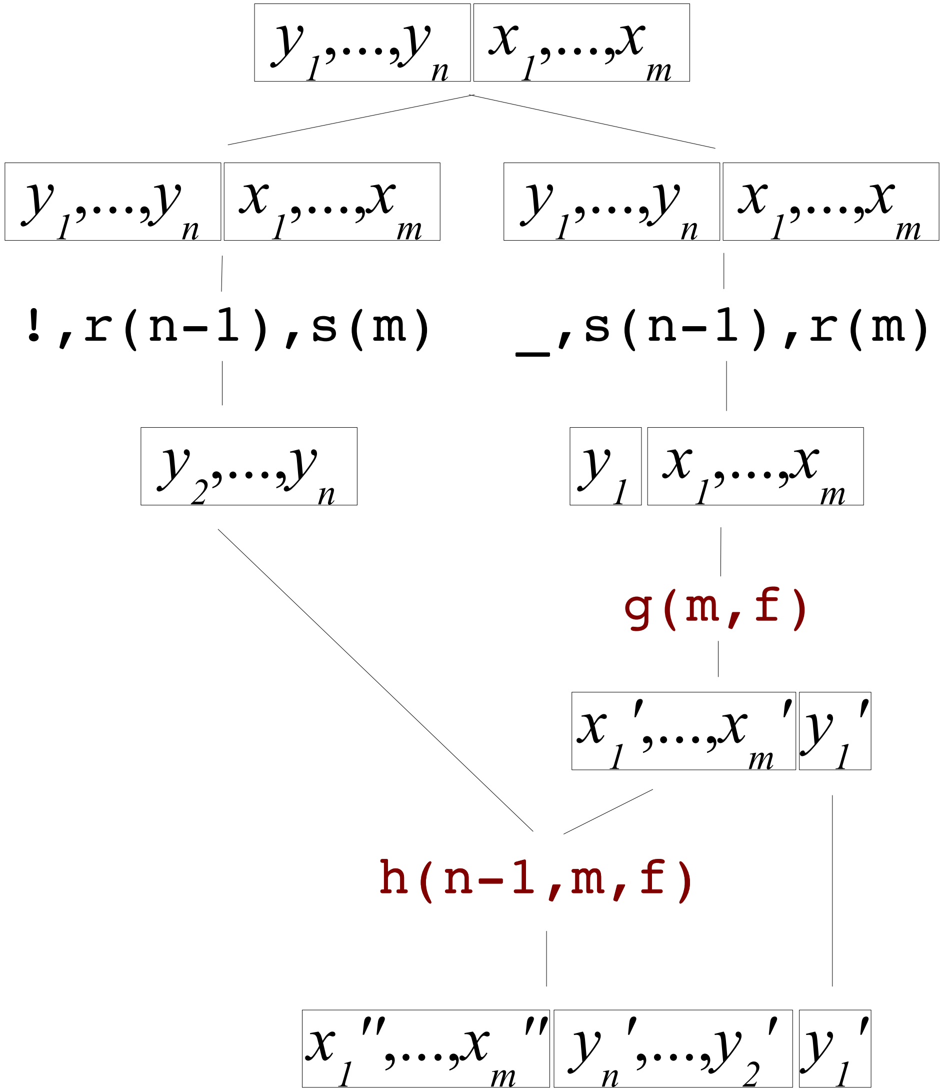
Systolic Array:
parallel
processing
in a 2D grid

`g(3,+):`





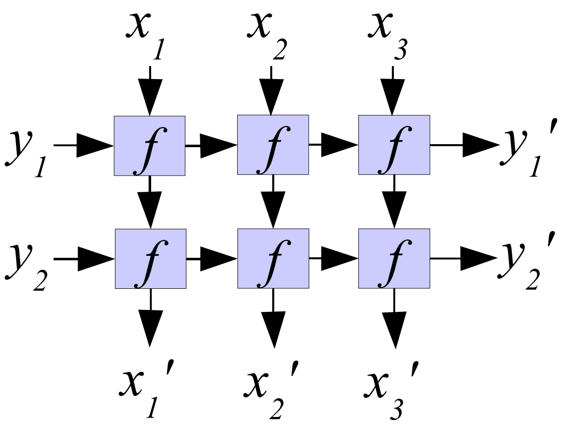
Faust Term Rewriting Extension



$y_1,\ldots,y_n$ $x_1,\ldots,x_m$
$y_1,\ldots,y_n$ $x_1,\ldots,x_m$
$y_1,\ldots,y_n$ $x_1,\ldots,x_m$
!,r(n-1),s(m)
_,s(n-1),r(m)
$y_2,\ldots,y_n$
y_1 $x_1,\ldots,x_m$
g(m,f)
$x_1',\ldots,x_m'$ y_1'
h(n-1,m,f)
$x_1'',\ldots,x_m''$ $y_n',\ldots,y_2'$ y_1'

Systolic Array: arranging the rows

```
h(n,m,f)
  = (r(n+m) <: (!,r(n-1),s(m),
     (_,s(n-1),r(m) : g(m,f)))) :
    (h(n-1,m,f), _);
```



Macro Hygiene

- C example:

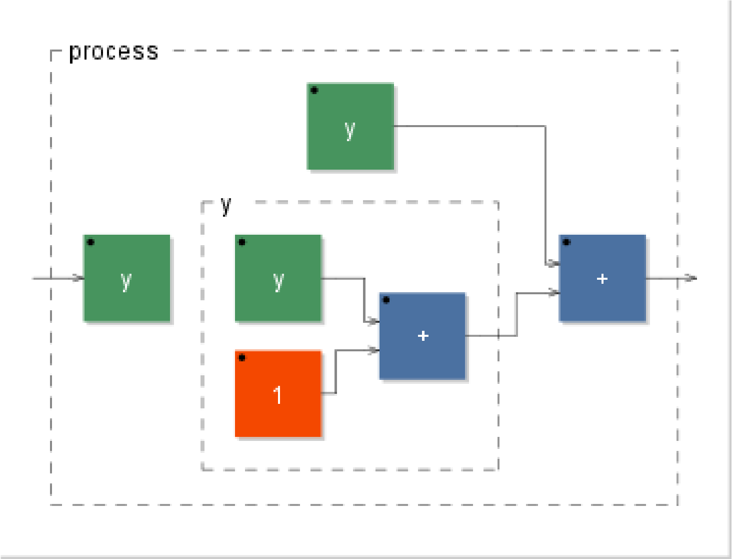
```
#define F(x) { int y = x+1; return x+y; }
```

  What does `F(y)` expand to?

```
F(y) ⇒ { int y = y+1; return y+y; }
```

- Faust: symbols in macro definitions are bound *lexically* (using Faust's block structure), so this *name capture* is avoided.

```
F = case
 { (x) => x+y with { y = x+1; }; };
process(y) = F(y);
```



Conclusion

- Term rewriting as a hygienic *macro language*.
- Rewriting rules are applied at *compile time* only.
- *Turing-complete*, so in principle anything computable can be done (including throwing the Faust compiler into an infinite loop, so beware!).
- Most useful for *optimization* and *transformation rules*, and to *construct complicated BDA expressions* automatically.
- **Future work:** Conditional rules, interface to Faust's internal BDA optimization passes.