Coinductive Program Synthesis for the Masses

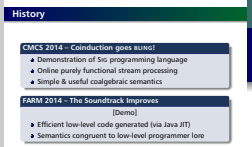
Excerpt of: Higher-Order Causal Stream Functions in SIG from First Principles

▷ Baltasar Trancón y Widemann[1,2] Markus Lepper[2]

[1] Ilmenau University of Technology

[2] semantics GmbH, Berlin

CMCS Short Talk
2016-04-02//03



- Demo: first bars of *Toccata in d-minor*, BWV 565
- Played on SIG software synthesizer, fully computational (no wave tables), real time, CD quality, four octaves fully polyphonic— $49 \times 44\,100$ samples per second, using 20%–25% of a single CPU core.
- Shows out-of-the box scalability; compare functional reactive programming! Plausible reasons both technical and formal.
- This talk is to argue that the last statement is true, deep and interesting.

History

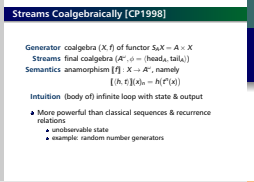
CMCS 2014 – Coinduction goes BLING!

- Demonstration of SIG programming language
- Online purely functional stream processing
- Simple & useful coalgebraic semantics

FARM 2014 – The Soundtrack Improves

[Demo]

- Efficient low-level code generated (via Java JIT)
- Semantics congruent to low-level programmer lore



- Last observation redundant for coalgebra audience, but found useful in teaching.

Streams Coalgebraically [CP1998]

Generator coalgebra (X, f) of functor $S_A X = A \times X$

Streams final coalgebra $(A^\omega, \phi = \langle \text{head}_A, \text{tail}_A \rangle)$

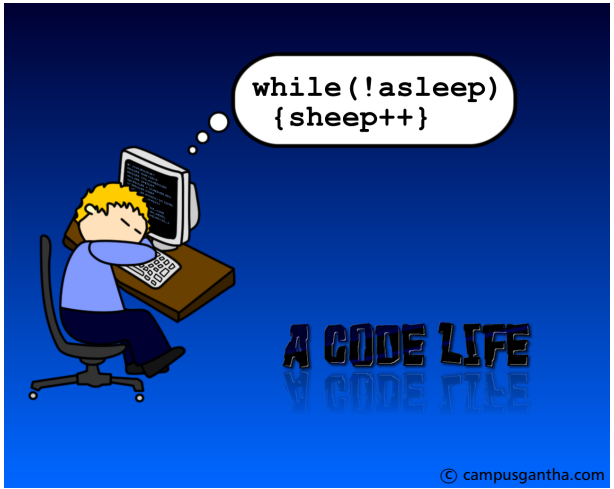
Semantics anamorphism $[\![f]\!] : X \to A^\omega$, namely

$$[\![\langle h, t \rangle]\!](x)_n = h\big(t^n(x)\big)$$

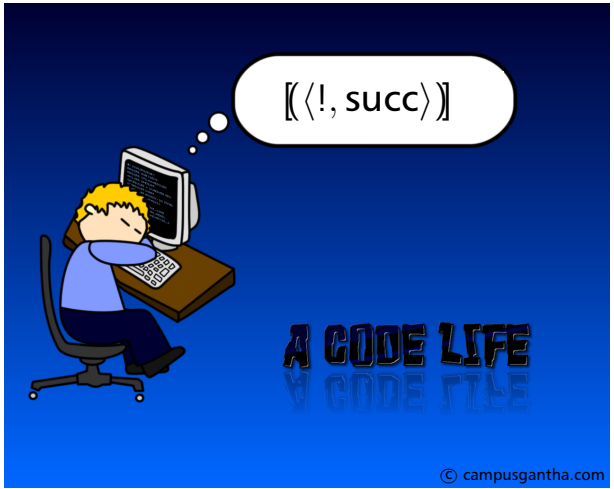
Intuition (body of) infinite loop with state & output

- More powerful than classical sequences & recurrence relations
  - unobservable state
  - example: random number generators

Illustration



Illustration



- Simply add input in the right places.

Stream Processing Coalgebraically

Mealy Transducer coalgebra (X, f) of functor $T_{AB} = (B \times X)^A$
Stream Functions final coalg. $(A \overset{\omega}{\rightsquigarrow} B, \phi = \cdots)$
by construction restricted to causal functions
$A \overset{\omega}{\rightsquigarrow} B = \{f : A^\omega \to B^\omega \mid x_{<n} = y_{<n} \Longrightarrow f(x)_{<n} = f(y)_{<n}\}$
▷ Index makes sense as (real) global time
Semantics anamorphism $[\![f]\!] : X \to A \overset{\omega}{\rightsquigarrow} B$, namely
$[\![\langle h, t\rangle]\!](x)(s)_n = h\Big(t\big(\cdots t(x)(s_0)\cdots\big)(s_{n-1})\Big)(s_n)$
Intuition (body of) infinite loop with state, input & output

Stream Processing Coalgebraically

Mealy Transducer coalgebra (X, f) of functor $T_{AB} = (B \times X)^A$

Stream Functions final coalg. $(A \overset{\omega}{\rightsquigarrow} B, \phi = \cdots)$

- by construction restricted to **causal** functions

  $A \overset{\omega}{\rightsquigarrow} B = \{f : A^\omega \to B^\omega \mid x_{<n} = y_{<n} \Longrightarrow f(x)_{<n} = f(y)_{<n}\}$

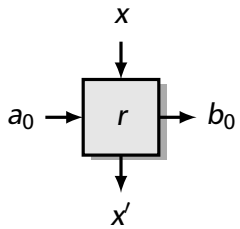
  ▷ Index makes sense as (real) global time

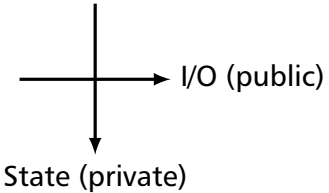
Semantics anamorphism $[\![f]\!] : X \to A \overset{\omega}{\rightsquigarrow} B$, namely

$$[\![\langle h, t\rangle]\!](x)(s)_n = h\Big(t\big(\cdots t(x)(s_0)\cdots\big)(s_{n-1})\Big)(s_n)$$

Intuition (body of) infinite loop with state, ***input*** & output

Coinductive Stream Functions in SIG





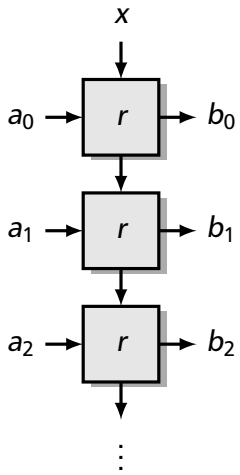
$$\lambda r : X \times A \to B \times X$$

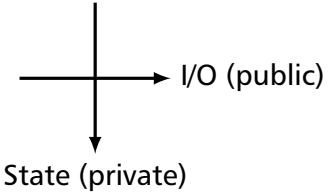
$$[\![\lambda r]\!] : X \to A \overset{\omega}{\rightsquigarrow} B$$

- Side remark: right currying to $A \to X \to B \times X$ would set the stage for the Kleisli category of a state monad instead.

Coinductive Stream Functions in SIG

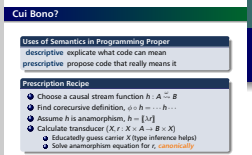
- Side remark: right currying to $A \to X \to B \times X$ would set the stage for the Kleisli category of a state monad instead.
- Virtual ω-replication of data-flow network; clearly shows state privacy & causality.





$\lambda r : X \times A \to B \times X$

$[\![\lambda r]\!] : X \to A \overset{\omega}{\rightsquigarrow} B$



- Treatment in SIG papers so far descriptive; here prescriptive.

Cui Bono?

Uses of Semantics in Programming Proper

descriptive explicate what code can mean

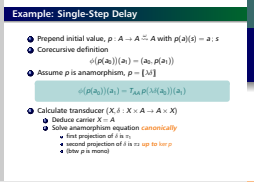
prescriptive propose code that really means it

Prescription Recipe

1. Choose a causal stream function $h : A \overset{\omega}{\rightsquigarrow} B$
2. Find corecursive definition, $\phi \circ h = \cdots h \cdots$
3. Assume h is anamorphism, $h = [\![\lambda r]\!]$
4. Calculate transducer $(X, r : X \times A \to B \times X)$
   1. Educatedly guess carrier X (type inference helps)
   2. Solve anamorphism equation for r, ***canonically***

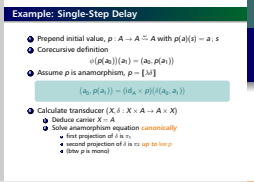
Example: Single-Step Delay

1. Prepend initial value, $p : A \to A \overset{\omega}{\rightsquigarrow} A$ with $p(a)(s) = a\,;\,s$
2. Corecursive definition
$$\phi(p(a_0))(a_1) = (a_0, p(a_1))$$
3. Assume p is anamorphism, $p = [\![\lambda\delta]\!]$
$$\phi \circ p = T_{AA}\, p \circ \lambda\delta$$
4. Calculate transducer $(X, \delta : X \times A \to A \times X)$
   1. Deduce carrier $X = A$
   2. Solve anamorphism equation ***canonically***
      - first projection of δ is π_1
      - second projection of δ is π_2 ***up to*** $\ker p$
      - (btw p is mono)



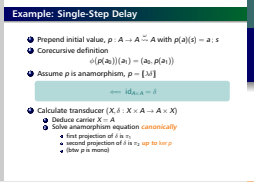
Example: Single-Step Delay

1. Prepend initial value, $p : A \to A \overset{\omega}{\rightsquigarrow} A$ with $p(a)(s) = a\,;\,s$
2. Corecursive definition
$$\phi(p(a_0))(a_1) = (a_0, p(a_1))$$
3. Assume p is anamorphism, $p = [\![\lambda\delta]\!]$
$$\phi(p(a_0))(a_1) = T_{AA}\,p(\lambda\delta(a_0))(a_1)$$
4. Calculate transducer $(X, \delta : X \times A \to A \times X)$
   1. Deduce carrier $X = A$
   2. Solve anamorphism equation ***canonically***
      - first projection of δ is π_1
      - second projection of δ is π_2 ***up to*** $\ker p$
      - (btw p is mono)



Example: Single-Step Delay

1. Prepend initial value, $p : A \to A \overset{\omega}{\rightsquigarrow} A$ with $p(a)(s) = a\,;\,s$
2. Corecursive definition
$$\phi(p(a_0))(a_1) = (a_0, p(a_1))$$
3. Assume p is anamorphism, $p = [\![\lambda\delta]\!]$
$$(a_0, p(a_1)) = (\mathrm{id}_A \times p)(\delta(a_0, a_1))$$
4. Calculate transducer $(X, \delta : X \times A \to A \times X)$
   1. Deduce carrier $X = A$
   2. Solve anamorphism equation ***canonically***
      - first projection of δ is π_1
      - second projection of δ is π_2 ***up to*** $\ker p$
      - (btw p is mono)

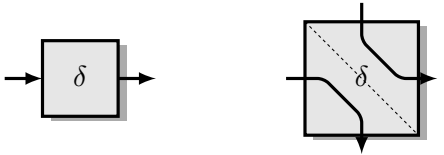


Example: Single-Step Delay

1. Prepend initial value, $p : A \to A \overset{\omega}{\rightsquigarrow} A$ with $p(a)(s) = a\,;\,s$
2. Corecursive definition
$$\phi(p(a_0))(a_1) = (a_0, p(a_1))$$
3. Assume p is anamorphism, $p = [\![\lambda\delta]\!]$
$$\Longleftarrow \mathrm{id}_{A\times A} = \delta$$
4. Calculate transducer $(X, \delta : X \times A \to A \times X)$
   1. Deduce carrier $X = A$
   2. Solve anamorphism equation ***canonically***
      - first projection of δ is π_1
      - second projection of δ is π_2 ***up to*** $\ker p$
      - (btw p is mono)

Delay, Graphically
• Programmer concerned with IO flow only
• State flow is inferred from delay
$\mathrm{id}_{A\times A} = \mathrm{id}_A \times \mathrm{id}_A$
• Independent identities $\Longrightarrow$ delayed cycles vanish

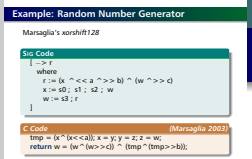
Delay, Graphically



- Programmer concerned with IO flow only
- State flow is inferred from delay

$$\mathrm{id}_{A\times A} = \mathrm{id}_A \times \mathrm{id}_A$$

- Independent identities $\Longrightarrow$ delayed cycles vanish



Example: Random Number Generator

Marsaglia's *xorshift128*

SIG Code

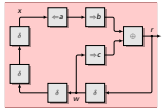
```
[ –> r
  where
    r := (x ^<< a ^>> b) ^ (w ^>> c)
    x := s0 ; s1 ; s2 ; w
    w := s3 ; r
]
```

C Code *(Marsaglia 2003)*

```
tmp = (x^(x<<a)); x = y; y = z; z = w;
return w = (w^(w>>c)) ^ (tmp^(tmp>>b));
```

- SIG code is referentially transparent; system of let-equations.
- C code is destructive assignments.
- Parameters a, b, c are from table of suggestions.
- Elements $s_0, \ldots, s_3$ are random seed.

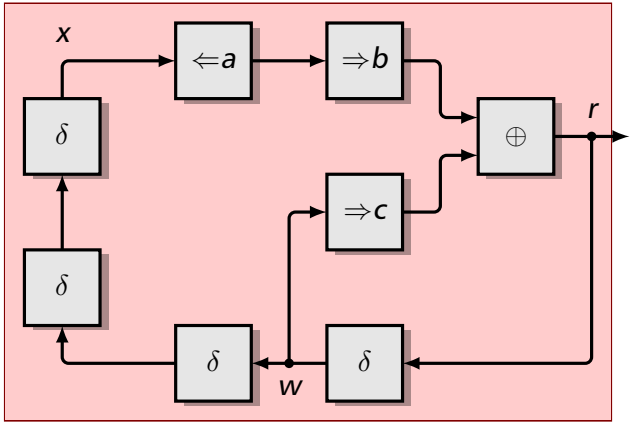
Example: Random Number Generator
Marsaglia's *xorshift128*

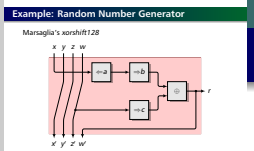


- Now allocate state and eliminate delay.

Example: Random Number Generator

Marsaglia's *xorshift128*

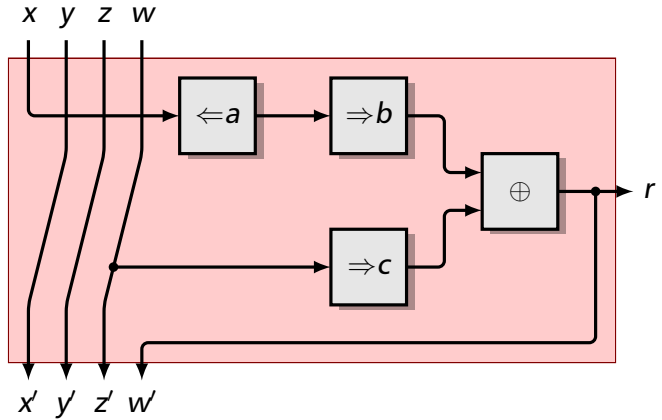


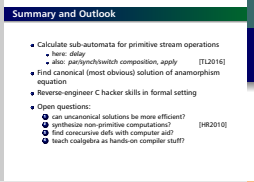


Example: Random Number Generator

- Result: data-flow network; straightforward code generation gets just as good as C version.

Marsaglia's *xorshift128*




Summary and Outlook

- Calculate sub-automata for primitive stream operations
  - here: *delay*
  - also: *par/synch/switch composition*, *apply* [TL2016]
- Find canonical (most obvious) solution of anamorphism equation
- Reverse-engineer C hacker skills in formal setting
- Open questions:
  1. can uncanonical solutions be more efficient?
  2. synthesize non-primitive computations? [HR2010]
  3. find corecursive defs with computer aid?
  4. teach coalgebra as hands-on compiler stuff?

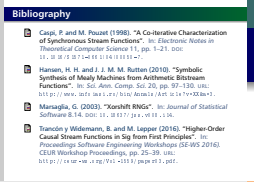
Summary and Outlook

- Calculate sub-automata for primitive stream operations
  - here: *delay*
  - also: *par/synch/switch composition*, *apply* [TL2016]
- Find canonical (most obvious) solution of anamorphism equation
- Reverse-engineer C hacker skills in formal setting
- Open questions:
  1. can uncanonical solutions be more efficient?
  2. synthesize non-primitive computations? [HR2010]
  3. find corecursive defs with computer aid?
  4. teach coalgebra as hands-on compiler stuff?

The End
Just doin' my coinduction exercises – ain't so bad folks!

The End

Just doin' my coinduction exercises – ain't so bad folks!


Bibliography

Caspi, P. and M. Pouzet (1998). "A Co-iterative Characterization of Synchronous Stream Functions". In: *Electronic Notes in Theoretical Computer Science* 11, pp. 1–21. DOI: 10.1016/S1571-0661(04)00050-7.

Hansen, H. H. and J. J. M. M. Rutten (2010). "Symbolic Synthesis of Mealy Machines from Arithmetic Bitstream Functions". In: *Sci. Ann. Comp. Sci.* 20, pp. 97–130. URL: http://www.infoiasi.ro/bin/Annals/Article?v=XX&a=3.

Marsaglia, G. (2003). "Xorshift RNGs". In: *Journal of Statistical Software* 8.14. DOI: 10.18637/jss.v008.i14.

Trancón y Widemann, B. and M. Lepper (2016). "Higher-Order Causal Stream Functions in Sig from First Principles". In: *Proceedings Software Engineering Workshops (SE-WS 2016)*. CEUR Workshop Proceedings, pp. 25–39. URL: http://ceur-ws.org/Vol-1559/paper03.pdf.

Bibliography

Caspi, P. and M. Pouzet (1998). "A Co-iterative Characterization of Synchronous Stream Functions". In: *Electronic Notes in Theoretical Computer Science* 11, pp. 1–21. DOI: 10.1016/S1571-0661(04)00050-7.

Hansen, H. H. and J. J. M. M. Rutten (2010). "Symbolic Synthesis of Mealy Machines from Arithmetic Bitstream Functions". In: *Sci. Ann. Comp. Sci.* 20, pp. 97–130. URL: http://www.infoiasi.ro/bin/Annals/Article?v=XX&a=3.

Marsaglia, G. (2003). "Xorshift RNGs". In: *Journal of Statistical Software* 8.14. DOI: 10.18637/jss.v008.i14.

Trancón y Widemann, B. and M. Lepper (2016). "Higher-Order Causal Stream Functions in Sig from First Principles". In: *Proceedings Software Engineering Workshops (SE-WS 2016)*. CEUR Workshop Proceedings, pp. 25–39. URL: http://ceur-ws.org/Vol-1559/paper03.pdf.