Productivity

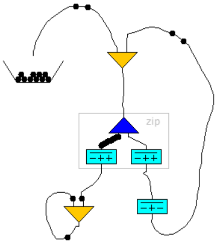
introduction, history

pure stream format, pebbleflow nets, decidability

Jörg Endrullis   Clemens Grabmayer   Dimitri Hendriks

Vrije Universiteit Amsterdam – Universiteit Utrecht

ISR 2010, Utrecht, July 7



course overview

today:

1. introduction, history (D)
2. the pure stream format, pebbleflow nets, decidability (D)
3. extended formats (C)

tomorrow:

4. data-oblivious productivity (C)
5. productivity of infinite data structures via termination (J)
6. complexity and variants of productivity (C)
7. practicum: defining streams (you)

outline parts 1 & 2

1. introduction
2. history
3. the pure stream format
4. modelling with nets
5. deciding productivity
6. conclusion

what is productivity?

a finite expression is productive if it

- represents a unique infinite object, and
- allows for the construction of this infinite object

in a slogan:

productivity = constructive well-definedness

in this talk:

- infinite objects: streams (infinite contructor normal forms)
- finite expressions: functional programs / term rewriting systems
- construction: evaluation / term rewriting

why study productivity?

productivity is a crucial property for correctness of programs dealing with infinite data structures:

- productivity ensures unlimited progress
- a productive program is produces values indefinitely
- a productive program is immune to starvation

streams

- a stream over A is an infinite sequence of elements from A
- we write streams as

$$a_0 : a_1 : a_2 : \dots$$

where ‘$:$’ is the stream constructor symbol

example

$$\begin{aligned} \mathsf{zeros} &= 0 : 0 : 0 : \dots \\ \mathsf{fib} &= 0 : 1 : 1 : 2 : 3 : 5 : 8 : \dots \\ \mathsf{morse} &= 0 : 1 : 1 : 0 : 1 : 0 : 0 : 1 : \dots \\ \mathsf{facts} &= 1 : 1 : 2 : 6 : 24 : 120 : \dots \end{aligned}$$

productivity of stream specifications

definition

a stream specification is an orthogonal, $\{S, D\}$-sorted constructor TRS

definition

a stream specification is productive for a term t if outermost-fair evaluation results in an infinite constructor normal form:

$$t \twoheadrightarrow a_0 : a_1 : a_2 : \ldots$$

- productivity of stream specifications is undecidable in general

example stream specifications

example (productive? yes!)

$$\mathsf{zeros} \to 0 : \mathsf{zeros}$$

The rule for zeros produces 0's indefinitely:

$$\mathsf{zeros} \to 0 : \mathsf{zeros} \to 0 : 0 : \mathsf{zeros} \twoheadrightarrow 0 : 0 : 0 : \ldots$$

example stream specifications

example (productive? no!)

$$\mathsf{N} \to 0 : \mathsf{tail}(\mathsf{N})$$
$$\mathsf{tail}(x : xs) \to xs$$

we cannot make any progress:

$$\begin{aligned} \mathsf{N} &\to 0 : \mathsf{tail}(\mathsf{N}) \\ &\to 0 : \mathsf{tail}(0 : \mathsf{tail}(\mathsf{N})) \\ &\to 0 : \mathsf{tail}(\mathsf{N}) \\ &\twoheadrightarrow \ldots \end{aligned}$$

example stream specifications

example (productive? yes!)

$$\begin{aligned} \mathsf{A} &\to 0 : \mathsf{read1}(\mathsf{A}) \\ \mathsf{read1}(x : xs) &\to x : \mathsf{read1}(xs) \end{aligned}$$

$$\begin{aligned} \mathsf{A} &\to 0 : \mathsf{read1}(\mathsf{A}) \\ &\to 0 : \mathsf{read1}(0 : \mathsf{read1}(\mathsf{A})) \\ &\to 0 : 0 : \mathsf{read1}(\mathsf{read1}(\mathsf{A})) \\ &\twoheadrightarrow 0 : 0 : 0 : \ldots \end{aligned}$$

example stream specifications

example (productive? no!)

$$\mathsf{B} \to 0 : \mathsf{read2}(\mathsf{B})$$

$$\mathsf{read2}(x : y : xs) \to x : y : \mathsf{read2}(xs)$$

The rule for read2 can never be applied:

$$\begin{aligned} \mathsf{B} &\to 0 : \mathsf{read2}(\mathsf{B}) \\ &\to 0 : \mathsf{read2}(0 : \mathsf{read2}(\mathsf{B})) \\ &\twoheadrightarrow 0 : \mathsf{read2}(0 : \mathsf{read2}(0 : \mathsf{read2}(\ldots))) \end{aligned}$$

operational versus extensional

$$\mathsf{read1}(x : xs) \to x : \mathsf{read1}(xs)$$
$$\mathsf{read2}(x : y : xs) \to x : y : \mathsf{read2}(xs)$$

- read1 and read2 are extensionally equal, but intensionally/operationally they are not!

productivity versus unique solvability

example (productive? no!)

$$\mathsf{Z} = \mathsf{h}(\mathsf{Z})$$
$$\mathsf{h}(x : xs) = 0 : \mathsf{h}(xs)$$

- Z has a unique solution $0 : 0 : 0 : \ldots$
- but this solution cannot be found by evaluating the specification:

$$\mathsf{Z} = \mathsf{h}(\mathsf{Z}) = \mathsf{h}(\mathsf{h}(\mathsf{Z})) = \ldots$$

example stream specifications

example (productive? yes!)

$$
\begin{aligned}
\mathsf{morse} &\to 0 : 1 : \mathsf{h}(\mathsf{tail}(\mathsf{morse})) \\
\mathsf{h}(x : xs) &\to x : \mathsf{not}(x) : \mathsf{h}(xs) \\
\mathsf{tail}(x : xs) &\to xs \\
\mathsf{not}(0) &\to 1 \\
\mathsf{not}(1) &\to 0
\end{aligned}
$$

productivity of stream specifications

example (productive? no!)

$$\mathsf{J} \to 0 : 1 : \mathsf{even}(\mathsf{J})$$
$$\mathsf{even}(x : xs) \to x : \mathsf{odd}(xs)$$
$$\mathsf{odd}(x : xs) \to \mathsf{even}(xs)$$

$$\mathsf{even}(\mathsf{J}) \twoheadrightarrow \mathsf{even}(0 : 1 : \mathsf{even}(\mathsf{J})) \twoheadrightarrow 0 : \mathsf{even}^2(\mathsf{J})$$
$$\mathsf{even}^2(\mathsf{J}) \twoheadrightarrow \mathsf{even}(0 : \mathsf{even}^2(\mathsf{J})) \twoheadrightarrow 0 : \mathsf{odd}(\mathsf{even}^2(\mathsf{J}))$$
$$\mathsf{odd}(\mathsf{even}^2(\mathsf{J})) \twoheadrightarrow \mathsf{odd}(0 : \mathsf{odd}(\mathsf{even}^2(\mathsf{J}))) \twoheadrightarrow \mathsf{even}(\mathsf{odd}(\mathsf{even}^2(\mathsf{J})))$$
$$\twoheadrightarrow\!\!\!\twoheadrightarrow \mathsf{even}^\omega$$

- and hence:

  $$\mathsf{J} \twoheadrightarrow\!\!\!\twoheadrightarrow 0 : 1 : 0 : 0 : \mathsf{even}^\omega$$

  J is strongly normalizing, but not productive
- question: how many bits do we have to add to make J productive, i.e. for which n is $\mathsf{J} \to a_0 : a_1 : \ldots : a_n : \mathsf{even}(\mathsf{J})$ productive?

Productivity recognition: previous approaches

- Wadge (1981): ‘cyclic sum test’ (limited, computable criterion)
- Sijtsma (1989): mathematical theory of productivity based on ‘production moduli’ (not directly computable criteria)
- Coquand (1994): ‘guardedness’ as a syntactic criterion for productivity (automatable, but restrictive criterion)
- Hughes, Pareto, and Sabry (1996): introduce a type system for proving productivity (automatable criterion)
- Telford and Turner (1997): extend the notion of guardedness by a method in the flavour of Wadge
- Buchholz (2004): type system for proving productivity (automatable for a restricted subsystem)

all use a data-oblivious analysis, a ‘quantitative’ analysis where the knowledge about concrete values of data elements is ignored

Production moduli

definition

let $\mathsf{f} : (A^\omega)^r \to A^\omega$ be a stream function.
a production modulus for f is a function $\nu_{\mathsf{f}} : (\overline{\mathbb{N}})^r \to \overline{\mathbb{N}}$ such that the first $\nu_{\mathsf{f}}(t_1, \ldots, t_r)$ elements of $\mathsf{f}(t_1, \ldots, t_r)$ can be computed whenever the first n_i elements of t_i are defined.

example

$$\mathsf{tail}(x : xs) \to xs \qquad \nu_{\mathsf{tail}}(n) = n \dot{-} 1$$

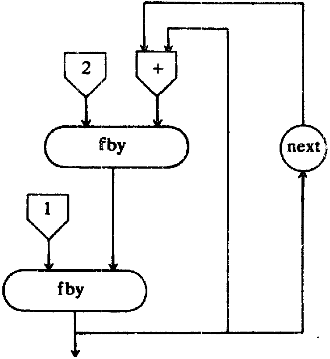
$$\mathsf{dup}(x : xs) \to x : x : \mathsf{dup}(xs) \qquad \nu_{\mathsf{dup}}(n) = 2n$$

$$\mathsf{odd}(x : y : xs) \to y : \mathsf{odd}(xs) \qquad \nu_{\mathsf{odd}}(n) = \lfloor \frac{n}{2} \rfloor$$

$$\mathsf{add}(x : xs, y : ys) \to (x + y) : \mathsf{add}(xs, ys) \qquad \nu_{\mathsf{add}}(n, m) = \min(n, m)$$

the cyclic sum test of Wadge

- a network is a device for computing the least fixed point of a system of equations (Kahn 1974)
- Wadge 1981 studies deadlock in dataflow networks. (free of deadlock $\approx$ productive).



- a loop means that some node is consuming its own output
- a node might starve, when it is waiting for itself to produce data

the cyclic sum test of Wadge

- associate with each of the arguments of the operations an integer which says how far the output leads (or lags) that argument;
- network passes the test if, for every cycle, the sum of associated numbers is positive
- a network passing the test is guaranteed to be immune to deadlock

the cyclic sum test of Wadge

- by using numbers it can only be expressed that consumption and production differs by a constant value,
- with constant functions as production moduli one cannot express that production depends on the number of elements consumed
- for example, to the argument of dup

  $$\mathsf{dup}(x : xs) \to x : x : \mathsf{dup}(xs)$$

  one has to associate 0 to its argument, for it first has to consume a stream element before it can produce one (two)
- but then e.g.:

  $$\mathsf{D} \to 1 : \mathsf{tail}(\mathsf{dup}(\mathsf{D}))$$

  is not recognized to be productive

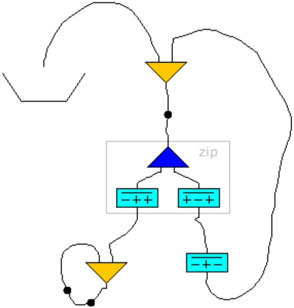
deciding productivity via pebbleflow

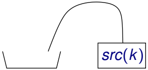
$$\mathsf{M} \to C[\mathsf{M}]$$

$$\dots$$

$$\mathsf{zip}(x : xs, ys) \to \dots$$

$$\dots$$





pure stream specification $\xmapsto{\text{translate}}$ pebbleflow net $\xmapsto{\text{compute production}}$ pebble source

desiderata specification format

we want a syntactic format of stream specifications such that:

- the exact production moduli of stream functions can be computed
- the set of moduli is closed under composition and infimum
- least fixed points of production moduli can be computed

the pure stream format (PSF)

example

$\mathsf{morse} \to 0 : 1 : \mathsf{h}(\mathsf{tail}(\mathsf{morse}))$	*stream constants*
$\mathsf{tail}(x : xs) \to xs$ $\mathsf{h}(x : xs) \to x : \mathsf{not}(x) : \mathsf{h}(xs)$	*stream functions*
$\mathsf{not}(0) \to 1 \qquad \mathsf{not}(1) \to 0$	*data functions*

a pure stream specification is a 3-layered, $\{S, D\}$-sorted orthogonal constructor TRS:

- data layer: stream independent, terminating TRS
- stream function layer: no pattern matching on data, no nesting of stream functions
- stream constant layer: no restrictions

no nesting in stream function rules in PSF

- in PSF no nesting is allowed in the stream function layer
- (nesting in the stream constant layer *is* allowed!)

example (productive, but not pure)

$$\begin{aligned} \mathsf{zeros} &\to 0 : \mathsf{log}(\mathsf{exp}(\mathsf{zeros})) \\ \mathsf{exp}(x : xs) &\to x : \mathsf{dup}(\mathsf{exp}(xs)) \\ \mathsf{log}(x : xs) &\to x : \mathsf{log}(\mathsf{odd}(xs)) \end{aligned}$$

no stream dependent data functions in PSF

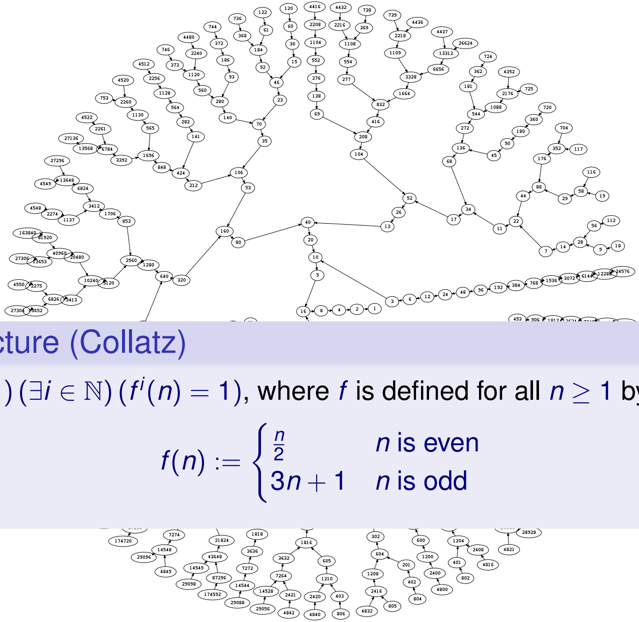
- in PSF data terms cannot be built using stream terms
- stream dependent data functions possibly create 'look-ahead'
- productivity is undecidable for specification including stream dependent data functions

example (productive? iff n is even)

$$
\begin{aligned}
\mathsf{S} &\to 0 : \mathsf{S}(n) : \mathsf{S} \\
\mathsf{head}(x : xs) &\to x \\
\mathsf{tail}(x : xs) &\to xs
\end{aligned}
$$

where $\mathsf{S}(n) := \mathsf{head}(\mathsf{tail}^n(\mathsf{S}))$

encoding Collatz conjecture



conjecture (Collatz)

$(\forall n \geq 1)\,(\exists i \in \mathbb{N})\,(f^i(n) = 1)$, where f is defined for all $n \geq 1$ by:

$$f(n) := \begin{cases} \frac{n}{2} & n \text{ is even} \\ 3n+1 & n \text{ is odd} \end{cases}$$

encoding Collatz conjecture

writing '$\bullet$' for successful termination, and dividing $3n+1$ immediately by 2 in case n is odd, the Collatz conjecture can be reformulated as:

$$(\forall n \geq 1)\,(\exists i \in \mathbb{N})\,(F^i(n) = \bullet)$$

for $F : \mathbb{N} \to \mathbb{N} \cup \{\bullet\}$ defined by:

$$\begin{aligned} F(1) &= \bullet & & \\ F(2n) &= n & & (n \geq 1) \\ F(2n+1) &= 3n+2 & & (n \geq 1) \end{aligned}$$

encoding Collatz conjecture

let

$$\begin{aligned} \mathsf{C} &\to \bullet : \mathsf{zip}(\mathsf{C}, \mathsf{third}(\mathsf{tail}^4(\mathsf{C}))) \\ \mathsf{zip}(xs, ys) &\to \mathsf{head}(xs) : \mathsf{zip}(ys, \mathsf{tail}(xs)) \\ \mathsf{third}(xs) &\to \mathsf{head}(xs) : \mathsf{third}(\mathsf{tail}^3(xs)) \end{aligned}$$

then

$$\mathsf{C} \twoheadrightarrow \bullet : \mathsf{C}(1) : \mathsf{C}(5) : \mathsf{C}(2) : \mathsf{C}(8) : \mathsf{C}(3) : \mathsf{C}(11) : \ldots$$

resembling Collatz function F written as a stream:

$$F = \bullet : 1 : 5 : 2 : 8 : 3 : 11 : \ldots$$

picturing the 'runs' through C, we get

$$\mathsf{C} \twoheadrightarrow \bullet : \mathsf{C}(1) : \mathsf{C}(5) : \mathsf{C}(2) : \mathsf{C}(8) : \mathsf{C}(3) : \mathsf{C}(11) : \ldots$$

encoding Collatz conjecture

$$\mathsf{C} \to \bullet : \mathsf{zip}(\mathsf{C}, \mathsf{third}(\mathsf{tail}^4(\mathsf{C})))$$

proposition

Collatz conjecture is true

$\iff$ all runs are finite (ending in $\bullet$)

$\iff$ the specification for C is productive: $\mathsf{C} \twoheadrightarrow \bullet : \bullet : \bullet : \bullet : \ldots$

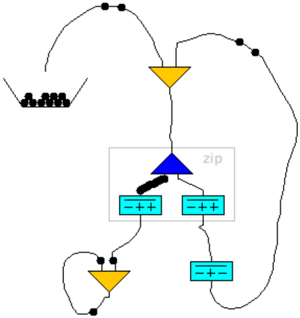
the stream of positive rational numbers

yet another pure stream specification:

$$
\begin{aligned}
\mathsf{rats} &\to \mathsf{mkpairs}(\mathsf{aux}) \\
\mathsf{aux} &\to 0 : \mathsf{aux}' \\
\mathsf{aux}' &\to 1 : \mathsf{zip}(\mathsf{aux}', \mathsf{add}(\mathsf{aux}', \mathsf{tail}(\mathsf{aux}'))) \\
\mathsf{tail}(x : xs) &\to xs \\
\mathsf{mkpairs}(x : y : xs) &\to \langle x, y\rangle : \mathsf{mkpairs}(y : xs) \\
\mathsf{zip}(x : xs, ys) &\to x : \mathsf{zip}(ys, xs) \\
\mathsf{add}(x : xs, y : ys) &\to (x + y) : \mathsf{add}(xs, ys) \\
0 + y &\to y \\
\mathsf{s}(x) + y &\to \mathsf{s}(x + y)
\end{aligned}
$$

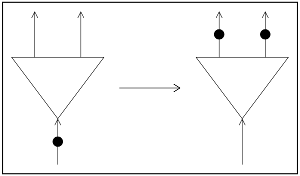
$\mathsf{rats} \twoheadrightarrow \langle 0, 1\rangle : \langle 1, 1\rangle : \langle 1, 2\rangle : \langle 2, 1\rangle : \langle 1, 3\rangle : \langle 3, 2\rangle : \langle 2, 3\rangle : \langle 3, 1\rangle : \langle 1, 4\rangle \ldots$

modelling stream specifications by pebbleflow nets



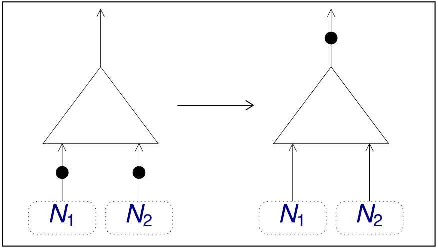
- tool for visualizing the production of a stream specification
- stream elements are abstracted from in favour of 'pebbles' •
- evaluation is modelled by the flow of pebbles through the net
- pebbleflow nets can be implemented by interaction nets
- a pure stream specification is productive if and only if its net generates an infinite chain of pebbles
- a pebbleflow net is a network built of pebble processing units (fans, boxes, meets, sources) connected by wires

fan



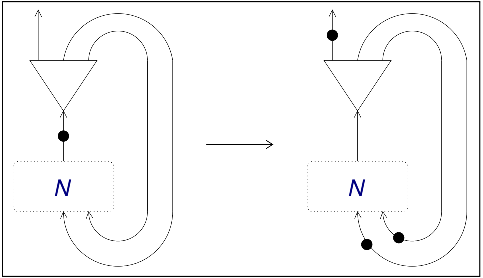
- duplicates an incoming pebble along its output ports
- explicit sharing device
- enables construction of cyclic nets
- used to implement recursion, in particular feedback

meet



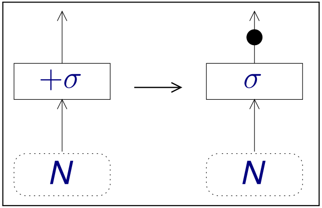
$$\triangle(\bullet(N_1), \bullet(N_2)) \to \bullet(\triangle(N_1, N_2))$$

recursion/feedback

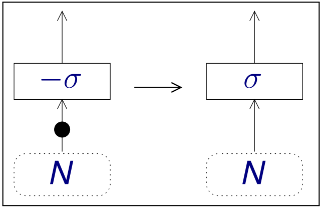


$\mu x.\bullet(N(x)) \to \bullet(\mu x.N(\bullet(x)))$

box



$\text{box}(+\sigma, N) \to \bullet(\text{box}(\sigma, N))$



$\text{box}(-\sigma, \bullet(N)) \to \text{box}(\sigma, N)$

- boxes contain I/O sequences: infinite sequences over $\{-, +\}$
- I/O sequences model production moduli of stream functions
- $+$: a ready state for an output pebble
- $-$: a requirement for an input pebble

translating unary stream functions to I/O sequences

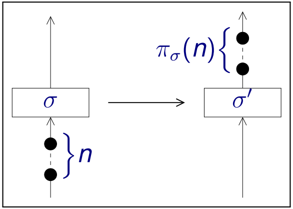
Example

$$\mathsf{dup}(x : xs) \to x : x : \mathsf{dup}(xs)$$
$$\mathsf{even}(x : xs) \to x : \mathsf{odd}(xs)$$
$$\mathsf{odd}(x : xs) \to \mathsf{even}(xs)$$

$$[\![\mathsf{dup}]\!] = -{+}{+}[\![\mathsf{dup}]\!] \qquad [\![\mathsf{dup}]\!] = \overline{-{+}{+}}$$
$$[\![\mathsf{even}]\!] = -{+}[\![\mathsf{odd}]\!] \qquad [\![\mathsf{even}]\!] = \overline{-{+}-}$$
$$[\![\mathsf{odd}]\!] = -[\![\mathsf{even}]\!] \qquad [\![\mathsf{odd}]\!] = \overline{-{-}+}$$

- every pure stream function f is mapped to an eventually periodic I/O sequence $[\![\mathsf{f}]\!]$, corresponding to an eventually periodically increasing function $\pi_{[\![\mathsf{f}]\!]}$, which forms the exact production modulus of f

from I/O sequences to production moduli



- the production function $\pi_\sigma : \overline{\mathbb{N}} \to \overline{\mathbb{N}}$ of $\sigma \in \pm^\omega$ is defined by $\pi_\sigma(n) := \pi(\sigma, n)$:

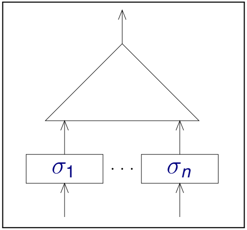
$$
\begin{aligned}
\pi(+\sigma, n) &= \pi(\sigma, n) + 1 \\
\pi(-\sigma, 0) &= 0 \\
\pi(-\sigma, n+1) &= \pi(\sigma, n)
\end{aligned}
$$

- example:

$$
\begin{aligned}
\pi_{[\![\mathsf{dup}]\!]}(n) &= \pi_{\overline{-++}}(n) = 2n \\
\pi_{[\![\mathsf{odd}]\!]}(n) &= \pi_{\overline{--+}}(n) = \lfloor \frac{n}{2} \rfloor
\end{aligned}
$$

translation of stream functions into gates

- gates are used to model stream functions



a gate with n input ports

example

$$\mathrm{zip}(x : xs, ys) = x : \mathrm{zip}(ys, xs)$$

$$[\![\mathrm{zip}]\!]_1 = -+[\![\mathrm{zip}]\!]_2 \qquad [\![\mathrm{zip}]\!]_1 = \overline{-++}$$

$$[\![\mathrm{zip}]\!]_2 = +[\![\mathrm{zip}]\!]_1 \qquad [\![\mathrm{zip}]\!]_2 = \overline{+-+}$$

pebbleflow rewrite system

- terms for pebbleflow nets ($k \in \mathbb{N} \cup \{\infty\}$, $x \in \mathrm{VAR}$, $\sigma \in \pm^{\omega}$):

$$N ::= \mathsf{src}(k) \mid x \mid \bullet(N) \mid \mathsf{box}(\sigma, N) \mid \mu x.N \mid \triangle(N, N)$$

- pebbleflow rewrite rules:

$$\begin{aligned}
\triangle(\bullet(N_1), \bullet(N_2)) &\to \bullet(\triangle(N_1, N_2)) \\
\mu x.\bullet(N(x)) &\to \bullet(\mu x.N(\bullet(x))) \\
\mathsf{box}(+\sigma, N) &\to \bullet(\mathsf{box}(\sigma, N)) \\
\mathsf{box}(-\sigma, \bullet(N)) &\to \mathsf{box}(\sigma, N) \\
\mathsf{src}(1 + k) &\to \bullet(\mathsf{src}(k))
\end{aligned}$$

pebbleflow tool

- net visualization tool written by Ariya Isihara,
- available via: http://infinity.few.vu.nl/productivity

$$\mathsf{J} \to 0 : 1 : \mathsf{even}(\mathsf{J})$$
$$[\![\mathsf{J}]\!] = \mu x.\bullet(\bullet(\mathsf{box}(\overline{-+-}, x)))$$

$$\mathsf{rats} \to \mathsf{mkpairs}(\mathsf{aux})$$
$$\mathsf{aux} \to 0 : \mathsf{aux}'$$
$$\mathsf{aux}' \to 1 : \mathsf{zip}(\mathsf{aux}', \mathsf{add}(\mathsf{aux}', \mathsf{tail}(\mathsf{aux}')))$$

$$[\![\mathsf{rats}]\!] = -\overline{-+}(\bullet(\mu x.\bullet(\triangle(\overline{-++}(x), \overline{+-+}(\triangle(\overline{-+}(x), \overline{-+}(-\overline{-+}(x))))))))$$

(where $\mathsf{box}(\sigma, N)$ is abbreviated to $\sigma(N)$)

translation preserves production

- for a specification $\mathcal{R} = \langle \Sigma, R \rangle$ the production $\Pi_{\mathcal{R}}(\mathsf{t})$ of a stream term t is defined by:

$$\Pi_{\mathcal{R}}(\mathsf{t}) := \sup\{\, n \in \mathbb{N} \mid \mathsf{t} \twoheadrightarrow \mathsf{d}_1 : \ldots : \mathsf{d}_n : \mathsf{t}' \,\}$$

- the production $\Pi_\bullet(N)$ of a net N is defined by:

$$\Pi_\bullet(N) := \sup\{\, n \in \mathbb{N} \mid N \twoheadrightarrow \bullet^n(N') \,\}$$

- translation of pure stream specifications to pebbleflow nets preserves production:

$$\Pi_\bullet([\![\mathsf{M}]\!]) = \Pi_{\mathcal{R}}(\mathsf{M})$$

- pure stream specifications are translated into rational nets, i.e. nets with eventually periodic I/O sequences only

net reduction

$$
\begin{aligned}
\bullet(N) &\to \mathsf{box}(+\overline{-+}, N) \\
\mathsf{box}(\sigma_1, \mathsf{box}(\sigma_2, N)) &\to \mathsf{box}(\sigma_1 \circ \sigma_2, N) \\
\mathsf{box}(\sigma, \triangle(N_1, N_2)) &\to \triangle(\mathsf{box}(\sigma, N_1), \mathsf{box}(\sigma, N_2)) \\
\mu x.\triangle(N_1, N_2) &\to \triangle(\mu x.N_1, \mu x.N_2) \\
\mu x.N &\to N && \text{if } x \notin \mathsf{FV}(N) \\
\mu x.\mathsf{box}(\sigma, x) &\to \mathsf{src}(\mathsf{fix}(\sigma)) \\
\triangle(\mathsf{src}(k_1), \mathsf{src}(k_2)) &\to \mathsf{src}(\mathsf{min}(k_1, k_2)) \\
\mathsf{box}(\sigma, \mathsf{src}(k)) &\to \mathsf{src}(\pi_\sigma(k)) \\
\mu x.x &\to \mathsf{src}(0)
\end{aligned}
$$

box composition & fixed point computation

composition $\circ : \pm^\omega \times \pm^\omega \to \pm^\omega$ is defined by:

$$
\begin{aligned}
+\sigma \circ \tau &= +(\sigma \circ \tau) \\
-\sigma \circ +\tau &= \sigma \circ \tau \\
-\sigma \circ -\tau &= -(-\sigma \circ \tau)
\end{aligned}
$$

- associative, preserves periodicity, and implements composition of the production functions: $\pi_{\sigma \circ \tau} = \pi_\sigma \circ \pi_\tau$

the operation fixed point $\mathsf{fix} : \pm^\omega \to \overline{\mathbb{N}}$ is defined by:

$$
\begin{aligned}
\mathsf{fix}(+\sigma) &= 1 + \mathsf{fix}(\delta(\sigma)) & \qquad \delta(+\sigma) &= +\delta(\sigma) \\
\mathsf{fix}(-\sigma) &= 0 & \qquad \delta(-\sigma) &= \sigma
\end{aligned}
$$

- $\mathsf{fix}(\sigma)$ is the least fixed point of π_σ

properties of net reduction

- net reduction is production preserving:

$$N \twoheadrightarrow N' \quad \text{implies} \quad \Pi_\bullet(N) = \Pi_\bullet(N')$$

- net reduction is terminating and confluent, hence:
  every closed net normalises to a unique normal form
- normal forms are of the form $\mathrm{src}(k)$, a source of k pebbles, with $k \in \mathbb{N}$ or $k = \infty$
- normal forms of rational nets can be computed

ProPro

- ProPro: a tool for proving productivity
- available via: http://infinity.few.vu.nl/productivity
-

$$\mathsf{J} \to 0 : 1 : \mathsf{even}(\mathsf{J})$$

$$\begin{aligned}
\mathsf{aux} &\to 0 : \mathsf{aux}' \\
\mathsf{aux}' &\to 1 : \mathsf{zip}(\mathsf{aux}', \mathsf{add}(\mathsf{aux}', \mathsf{tail}(\mathsf{aux}'))) \\
\mathsf{rats} &\to \mathsf{mkpairs}(\mathsf{aux})
\end{aligned}$$

productivity of PSF is decidable

theorem

productivity of pure stream specifications is decidable

decision algorithm for stream constant M in pure stream specification $\mathcal{R}$:

- translate M to the rational net $[\![\mathsf{M}]\!]$
- reduce $[\![\mathsf{M}]\!]$ to a source $\mathsf{src}(k)$
- (recall $\Pi_{\mathcal{R}}(\mathsf{M}) = \Pi_{\bullet}([\![\mathsf{M}]\!]) = k$)
- if $k = \infty$, output: $\mathcal{R}$ is productive for M
- if $k \in \mathbb{N}$, output: $\mathcal{R}$ is not productive for M

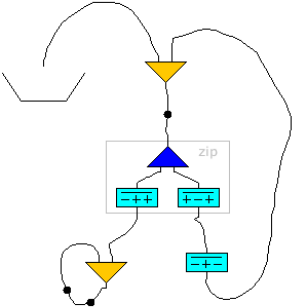
deciding productivity via pebbleflow

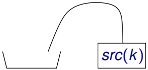
$$\mathsf{M} \to C[\mathsf{M}]$$

$$\ldots$$

$$\mathsf{zip}(x : xs, ys) \to \ldots$$

$$\ldots$$





pure stream specification $\xmapsto[\text{translate}]{}$ pebbleflow net $\xmapsto[\text{compute production}]{}$ pebble source

conclusion

- previous approaches: sufficient conditions for productivity, not automatable or only for a limited subclass
- pebbleflow approach: decision algorithm for productivity of a rich class of stream specifications, only stream function layer is restricted

coming up next:

- extended formats of stream specifications: stream functions defined using pattern matching on data