Watching Streams Grow: The Pebbleflow Method

Jörg Endrullis   Clemens Grabmayer   Dimitri Hendriks
Ariya Isihara   Jan Willem Klop

Universiteit Utrecht, Vrije Universiteit

NWO Projects Infinity and ProvCyc

TF-lunch, UU
6 November 2007

Outline

The Problem

- When do we accept an infinite object defined in terms of itself?
- When does a finite set of equations constructively define a unique infinite object?
- When is a recursive program productive?

- If it evaluates to a unique infinite constructor normal form

Example

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

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

- Other examples: infinite trees, processes, coinductive proofs, ...
- In general, productivity is (highly) undecidable (Π^0_3-complete).

Streams

- The set A^ω of streams (over set A) is defined by:

$$A^\omega := \{\sigma \mid \sigma : \mathbb{N} \to A\}$$

- A^ω is the greatest fixed point of:

$$\lambda X.\, A \times X$$

- ‘:’ is the stream constructor symbol: $\mathsf{a} : \sigma$ denotes the result of prepending $\mathsf{a} \in A$ to $\sigma \in A^\omega$
- A recursive stream specification

$$\mathsf{M} = \ldots \mathsf{M} \ldots$$

  is productive if the process of continually evaluating M results in an infinite constructor normal form:

$$\mathsf{M} \twoheadrightarrow \mathsf{a}_0 : \mathsf{a}_1 : \mathsf{a}_2 : \ldots$$

Examples

Example

$$\mathsf{read}(x : \sigma) = x : \mathsf{read}(\sigma)$$

$$\mathsf{fastread}(x : y : \sigma) = x : y : \mathsf{fastread}(\sigma)$$

$$\mathsf{fives} = 5 : \mathsf{read}(\mathsf{fives})$$ productive

$$\mathsf{fives}' = 5 : \mathsf{fastread}(\mathsf{fives}')$$ not productive

$$\mathsf{zip}_2(x : \sigma, y : \tau) = x : y : \mathsf{zip}_2(\sigma, \tau)$$

$$\mathsf{zip}_1(x : \sigma, \tau) = x : \mathsf{zip}_1(\tau, \sigma)$$

$$\mathsf{sevens} = 7 : \mathsf{zip}_2(\mathsf{sevens}, \mathsf{tail}(\mathsf{sevens}))$$ not productive

$$\mathsf{sevens}' = 7 : \mathsf{zip}_1(\mathsf{sevens}', \mathsf{tail}(\mathsf{sevens}'))$$ productive

Weakly Guarded Stream Function Specifications

Example

$\mathsf{tail}(x : \sigma) \to \sigma$ $\mathsf{even}(x : \sigma) \to x : \mathsf{odd}(\sigma)$ $\mathsf{odd}(x : \sigma) \to \mathsf{even}(\sigma)$ $\mathsf{zip}(x : \sigma, \tau) \to x : \mathsf{zip}(\tau, \sigma)$ $\mathsf{add}(x : \sigma, y : \tau) \to \mathsf{a}(x, y) : \mathsf{add}(\sigma, \tau)$	SFS-layer
$\mathsf{a}(x, 0) \to x$ $\quad$ $\mathsf{a}(x, \mathsf{s}(y)) \to \mathsf{s}(\mathsf{a}(x, y))$	data-layer

Weakly Guarded Stream Function Specifications

Example (Continued)

In an SFS we have 'production cycles' like:

$$\mathsf{even}(x : y : \sigma) \to x : \mathsf{odd}(y : \sigma) \to x : \mathsf{even}(\sigma)$$
$$\mathsf{odd}(x : y : \sigma) \to \mathsf{even}(y : \sigma) \to y : \mathsf{odd}(\sigma)$$
$$\mathsf{zip}(x : \sigma, y : \tau) \to x : \mathsf{zip}(y : \tau, \sigma) \to x : y : \mathsf{zip}(\sigma, \tau)$$

We say that even, odd, zip, and inv are weakly guarded.

tail is collapsing in $\mathcal{T}$:

$$\mathsf{tail}(x : \sigma) \to \sigma \ .$$

Weakly Guarded Stream Function Specifications

Definition

A TRS $\mathcal{T} = \langle \Sigma_D \uplus \Sigma_{sf} \uplus \{:\},\ R_d \uplus R_{sf} \rangle$
is a weakly guarded stream function specification (SFS) if

1. $\langle \Sigma_D,\ R_d \rangle$ is a sub-TRS, the data layer of $\mathcal{T}$.
2. Each rule in R_{sf}, the SFS-layer of $\mathcal{T}$, is of one of two forms:

$$\mathsf{f}((x_{1,1} : \ldots : x_{1,n_1} : \sigma_1), \ldots, (x_{r,1} : \ldots : x_{r_s,n_{r_s}} : \sigma_{r_s}), \vec{y})$$
$$\to t_1(\vec{x},\vec{y}) : \ldots : t_{m_\mathsf{f}}(\vec{x},\vec{y}) : \sigma_l \ ,$$
$$\to t_1(\vec{x},\vec{y}) : \ldots : t_{m_\mathsf{f}}(\vec{x},\vec{y}) : \mathsf{g}(\sigma_{\pi_\mathsf{f}(1)}, \ldots, \sigma_{\pi_\mathsf{f}(r'_s)}, t'_1(\vec{x},\vec{y}), \ldots, t'_{r'_d}(\vec{x},\vec{y})) \ ,$$

   where $\pi_\mathsf{f} : \{1, \ldots, r'_s\} \to \{1, \ldots, r_s\}$ is injective in case $\mathsf{f} \leadsto \mathsf{g}$.
3. Weakly guarded: On every dependency cycle $\mathsf{f} \leadsto \mathsf{g} \leadsto \cdots \leadsto \mathsf{f}$ there is at least one guard.

Pure Stream Constant Specifications

Example

$\mathsf{T} \to 0 : \mathsf{zip}(\mathsf{inv}(\mathsf{T}), \mathsf{tail}(\mathsf{T}))$	SCS-layer
$\mathsf{tail}(x : \sigma) \to \sigma$ $\mathsf{zip}(x : \sigma, \tau) \to x : \mathsf{zip}(\tau, \sigma)$ $\mathsf{inv}(x : \sigma) \to \mathsf{i}(x) : \mathsf{inv}(\sigma)$	SFS-layer
$\mathsf{i}(0) \to 1 \qquad \mathsf{i}(1) \to 0$	data-layer

This is a productive pSCS, obtaining the Thue-Morse sequence:

$$\mathsf{T} \twoheadrightarrow 0 : 1 : 1 : 0 : 1 : 0 : 0 : 1 : 1 : 0 : 0 : 1 : 0 : 1 : 1 : 0 : \ldots$$

Pure Stream Constant Specifications

Example

$\mathsf{J} = 0 : 1 : \mathsf{even}(\mathsf{J})$	SCS-layer
$\mathsf{even}(x : \sigma) \to x : \mathsf{odd}(\sigma)$ $\mathsf{odd}(x : \sigma) \to \mathsf{even}(\sigma)$	SFS-layer
	data-layer

This pSCS is not productive: $\mathsf{J} \twoheadrightarrow 0 : 1 : 0 : 0 : \mathsf{even}^{\omega}$

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

$$
\begin{aligned}
\mathsf{J} &\to 0 : 1 : \mathsf{even}(\mathsf{J}) \\
\mathsf{even}(\mathsf{J}) &\to \mathsf{even}(0 : 1 : \mathsf{even}(\mathsf{J})) \\
&\to 0 : \mathsf{odd}(1 : \mathsf{even}(\mathsf{J})) \\
&\to 0 : \mathsf{even}(\mathsf{even}(\mathsf{J})) \\
\mathsf{even}^2(\mathsf{J}) \equiv \mathsf{even}(\mathsf{even}(\mathsf{J})) &\twoheadrightarrow \mathsf{even}(0 : \mathsf{even}(\mathsf{even}(\mathsf{J}))) \\
&\to 0 : \mathsf{odd}(\mathsf{even}^2(\mathsf{J})) \\
\mathsf{odd}(\mathsf{even}^2(\mathsf{J})) &\twoheadrightarrow \mathsf{odd}(0 : \mathsf{odd}(\mathsf{even}^2(\mathsf{J}))) \\
&\to \mathsf{even}(\mathsf{odd}(\mathsf{even}^2(\mathsf{J}))) \\
\mathsf{odd}(\mathsf{even}^2(\mathsf{J})) &\twoheadrightarrow \mathsf{even}(\mathsf{odd}(\mathsf{even}^2(\mathsf{J}))) \\
&\twoheadrightarrow \mathsf{even}^2(\mathsf{odd}(\mathsf{even}^2(\mathsf{J}))) \\
&\twoheadrightarrow \ldots \twoheadrightarrow \mathsf{even}^n(\mathsf{odd}(\mathsf{even}^2(\mathsf{J}))) \twoheadrightarrow \ldots \\
&\twoheadrightarrow\!\!\!\!\twoheadrightarrow \mathsf{even}^\omega
\end{aligned}
$$

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

Pure Stream Constant Specifications

Example

$\mathsf{D}' \to 0 : 1 : 1 : \mathsf{zip}(\mathsf{add}(\mathsf{tail}(\mathsf{D}'), \mathsf{tail}(\mathsf{tail}(\mathsf{D}'))), \mathsf{E})$ $\mathsf{E} \to \mathsf{even}(\mathsf{tail}(\mathsf{D}'))$	SCS-layer
$\mathsf{tail}(x : \sigma) \to \sigma$ $\mathsf{even}(x : \sigma) \to x : \mathsf{odd}(\sigma)$ $\mathsf{odd}(x : \sigma) \to \mathsf{even}(\sigma)$ $\mathsf{add}(x : \sigma, y : \tau) \to \mathsf{a}(x, y) : \mathsf{add}(\sigma, \tau)$ $\mathsf{zip}(x : \sigma, \tau) \to x : \mathsf{zip}(\tau, \sigma)$	SFS-layer
$\mathsf{a}(x, 0) \to x \qquad \mathsf{a}(x, \mathsf{s}(y)) \to \mathsf{s}(\mathsf{a}(x, y))$	data-layer

This pSCS is productive, we obtain:

$$\mathsf{D}' \twoheadrightarrow 0 : 1 : 1 : 2 : 1 : 3 : 2 : 3 : 3 : 4 : 3 : 5 : 4 : 5 : 5 : 6 : 5 : 7 : 6 : 7 : 7 : \ldots$$

Pure Stream Constant Specifications

Definition

A TRS $\mathcal{T} = \langle \Sigma_D \uplus \Sigma_{sf} \uplus \Sigma_{sc} \uplus \{:\}, R_D \uplus R_{sf} \uplus R_{sc} \rangle$ is a pure recursive stream specification (SCS) if:

1. $\mathcal{T}_0 := \langle \Sigma_D \uplus \Sigma_{sf} \uplus \{:\}, R_D \uplus R_{sf} \rangle$ is a weakly guarded SFS.
2. $\Sigma_{sc} = \{\mathsf{M}_0, \ldots, \mathsf{M}_n\}$ set of stream constant symbols, where M_0 is called the root of $\mathcal{T}$; $R_{sc} = \{\rho_{\mathsf{M}_i} \mid i \in \{0, 1, \ldots, n\}\}$, where ρ_{M_i} the defining rule for M_i:

$$\mathsf{M}_i \to C_i[\mathsf{M}_0, \ldots, \mathsf{M}_n] \qquad (C_i \text{ an } n\text{-ary stream context in } \mathcal{T}_0).$$

We call R_D, R_{sf}, and R_{sc} the data-, SFS-, and SCS-layer of $\mathcal{T}$, resp..

Production of a Term. Productivity of an SCS.

Let $\overline{\mathbb{N}} := \mathbb{N} \cup \{\infty\}$ the coinductive natural numbers.

Definition

Let $\mathcal{T} = \langle \Sigma, R \rangle$ a pure SCS.

- The production $\Pi_{\mathcal{T}}(t)$ of a stream term $t \in Ter(\Sigma)$:

$$\Pi_{\mathcal{T}}(t) := \sup\{n \in \mathbb{N} \mid t \twoheadrightarrow u_1 : \ldots : u_n : t'\} \in \overline{\mathbb{N}}\ .$$

- $\mathcal{T}$ is called productive if $\Pi_{\mathcal{T}}(\mathsf{M}_0) = \infty$.

Proposition

A pure SCS $\mathcal{T}$ is productive if and only if $\mathsf{M}_0 \twoheadrightarrow u_1 : u_2 : u_3 : u_4 : \ldots$.

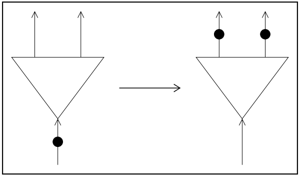
Modelling SCSs with Pebbleflow Nets

- Kahn (1974): Networks are devices for computing least fixed points of systems of equations.

Pebbleflow Nets:

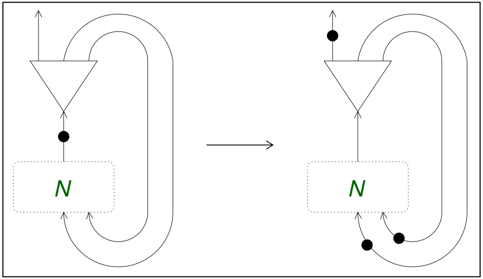
- Stream elements are abstracted from in favour of ‘pebbles’: $[u : s] = \bullet([s])$.
- An SCS is modelled by a pebbleflow net: Evaluation of an SCS is modelled by the flow of pebbles in a net.
- A stream definition is productive if and only if the net associated to it generates an infinite chain of pebbles.
- Elements are: fans, meets, boxes and gates, sources, wires.

Fan



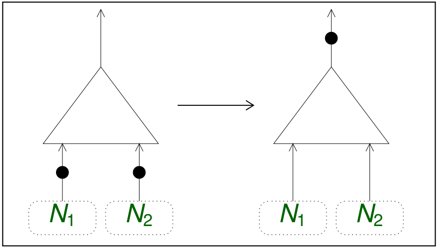
- duplicates an incoming pebble along its output ports
- explicit sharing device
- enables construction of cyclic nets
- used to implement recursion, in particular feedback

Recursion/Feedback



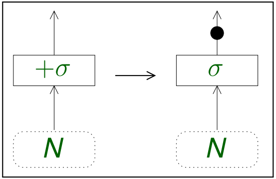
$$\mu x.\bullet(N(x)) \to \bullet(\mu x.N(\bullet(x)))$$

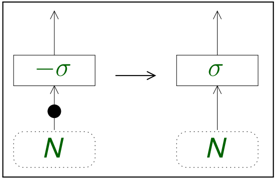
Meet



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

Box





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

- $+$: a ready state for an output pebble
- $-$: a requirement for an input pebble
- σ : an infinite sequence over $\{+, -\}$

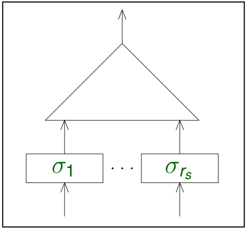
I/O sequences

- I/O sequences contain infinitely many $+$'s:

$$\pm^{\omega} := \{\sigma \in \{+,-\}^{\omega} \mid \forall n.\, \exists m \geq n.\ \sigma(m) = +\}$$

- $\sigma \in \pm^{\omega}$ is rational if there exist $\alpha, \gamma \in \pm^{*}$ such that $\sigma = \alpha\overline{\gamma} = \alpha\gamma\gamma\gamma\ldots$ (γ non-empty)
- I/O sequences model quantitative behaviour of stream functions

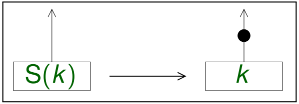
Gates



A gate for modelling r_s-ary stream functions.

$$\triangle(\mathrm{box}(\sigma_1, [\,]_1), \ldots, \mathrm{box}(\sigma_{r_s}, [\,]_{r_s}))$$

Source



$$\mathsf{src}(\mathsf{S}(k)) \to \bullet(\mathsf{src}(k))$$

Term Representations of Nets

Definition

Let $\mathcal{V}$ be a set of variables.

The set $\mathcal{N}$ of terms for pebbleflow nets is generated by:

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

where $x \in \mathcal{V}$, $\sigma \in \pm^{\omega}$, and $k \in \overline{\mathbb{N}} := \mathbb{N} \cup \{\infty\}$.

Pebbleflow

Definition

The pebbleflow rewrite relation $\to_{\mathsf{p}}$ is defined as:

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

Theorem

The rewrite relation $\to_{\mathsf{p}}$ is confluent.

Pebbleflow Tool: Watching Streams Grow

- Net visualization tool (Java applet) by Ariya Isihara

  click & play: http://infinity.few.vu.nl/productivity

$$\mathsf{T} = 0 : \mathsf{zip}(\mathsf{inv}(\mathsf{T}), \mathsf{tail}(\mathsf{T}))$$
$$[\mathsf{T}] = \mu x.\bullet(\triangle(\mathsf{box}(\overline{-++}, \mathsf{box}(\overline{-+}, x)), \mathsf{box}(\overline{+-+}, \mathsf{box}(-\overline{-+}, x))))$$

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

$$\mathsf{D} = 0 : 1 : 0 : \mathsf{zip}(\mathsf{add}(\mathsf{tail}(\mathsf{D}), \mathsf{tail}(\mathsf{tail}(\mathsf{D}))), \mathsf{even}(\mathsf{tail}(\mathsf{D})))$$
$$[\mathsf{D}] = \mu D.\bullet(\bullet(\bullet([\mathsf{zip}]([\mathsf{add}]([\mathsf{tail}](D), [\mathsf{tail}]([\mathsf{tail}](D))), [\mathsf{even}]([\mathsf{tail}](D))))))$$

Production of a Net

Definition

The production $\Pi(N)$ of a net $N \in \mathcal{N}$:

$$\Pi(N) := \sup\{n \in \mathbb{N} \mid N \twoheadrightarrow_{\mathrm{p}} \bullet^{n}(N')\} \,.$$

Translation of Unary Stream Functions into I/O-Seq's

Example

$$\begin{aligned}
\mathsf{tail}(x : \sigma) &= \sigma \\
\mathsf{even}(x : \sigma) &= x : \mathsf{odd}(\sigma) \\
\mathsf{odd}(x : \sigma) &= \mathsf{even}(\sigma) \\
\mathsf{dup}(x : \sigma) &= x : x : \mathsf{dup}(\sigma)
\end{aligned}$$

$$\begin{aligned}
[\mathsf{tail}]_1 &= -\sigma_{\mathsf{id}} \\
[\mathsf{even}]_1 &= -+[\mathsf{odd}]_1 \\
[\mathsf{odd}]_1 &= -[\mathsf{even}]_1 \\
[\mathsf{dup}]_1 &= -++[\mathsf{dup}]_1
\end{aligned}
\qquad\qquad
\begin{aligned}
[\mathsf{tail}]_1 &= -\overline{-+} \\
[\mathsf{even}]_1 &= \overline{-+-} \\
[\mathsf{odd}]_1 &= \overline{--+} \\
[\mathsf{dup}]_1 &= \overline{-++}
\end{aligned}$$

Translation of Stream Functions into Gates

Example

$$\mathsf{zip}(x : \sigma, \tau) = x : \mathsf{zip}(\tau, \sigma)$$

$$[\mathsf{zip}] = \triangle_2(\mathsf{box}([\mathsf{zip}]_1, [\,]_1), \mathsf{box}([\mathsf{zip}]_2, [\,]_2))\,,$$

where:

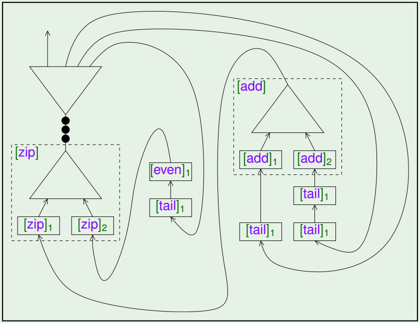
$$[\mathsf{zip}]_1 = -+[\mathsf{zip}]_2 \qquad\qquad [\mathsf{zip}]_1 = \overline{-++}$$

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

Translation of Stream Constants into Nets

Example

$$\mathsf{D} \to 0 : 1 : 0 : \mathsf{zip}(\mathsf{add}(\mathsf{tail}(\mathsf{D}), \mathsf{tail}(\mathsf{tail}(\mathsf{D}))), \mathsf{even}(\mathsf{tail}(\mathsf{D})))$$



$$[\mathsf{D}] = \mu D.\bullet(\bullet(\bullet([\mathsf{zip}]([\mathsf{add}]([\mathsf{tail}](D), [\mathsf{tail}]([\mathsf{tail}](D))), [\mathsf{even}]([\mathsf{tail}](D))))))$$

Translation of Stream Constants into Nets

Definition

Let $\mathcal{T} = \langle \Sigma_D \uplus \Sigma_{sf} \uplus \Sigma_{sc} \uplus \{:\}, R_D \uplus R_{sf} \uplus R_{sc} \rangle$ be a pure SCS.
For each $\mathsf{M} \in \Sigma_{sc}$ with rule $\rho_{\mathsf{M}} \equiv \mathsf{M} \to rhs_{\mathsf{M}}$ the translation $[\mathsf{M}] := [\mathsf{M}]_{\varnothing}$ of M into a rational pebbleflow net is recursively def. by:

$$[\mathsf{M}]_\alpha = \begin{cases} \mu M.[rhs_{\mathsf{M}}]_{\alpha \cup \{\mathsf{M}\}} & \text{if } \mathsf{M} \notin \alpha \\ M & \text{if } \mathsf{M} \in \alpha \end{cases}$$

$$[t : u]_\alpha = \bullet([u]_\alpha)$$

$$[\mathsf{f}(u_1, \ldots, u_{r_s}, t_1, \ldots, t_{r_d})]_\alpha = [\mathsf{f}]([u_1]_\alpha, \ldots, [u_{r_s}]_\alpha)$$

where α denotes a set of stream constant symbols.

Translation is Production Preserving

Theorem

Let $\mathcal{T}$ be a pure SCS. Then it holds: $\Pi([\mathsf{M}_0]) = \Pi_{\mathcal{T}}(\mathsf{M}_0)$.

General Idea.

- Going back and forth between rewrite sequences

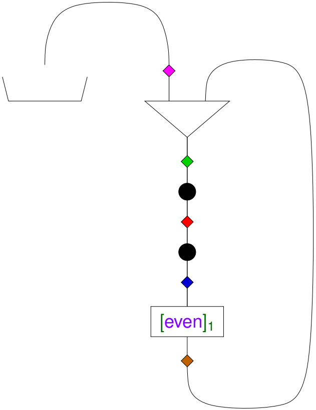
$$[\mathsf{M}_0] \twoheadrightarrow_{\mathsf{p}} \bullet^n(N)$$

  on the pebbleflow net translation of M_0 that produce n pebbles, and rewrite sequences

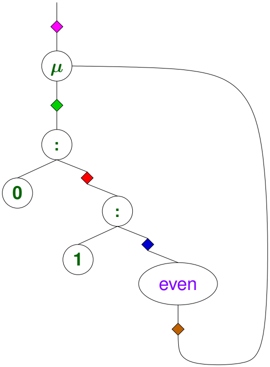
$$\mathsf{M}_0 \twoheadrightarrow_{\mathcal{T}} t_1 : \ldots : t_n : u$$

  in $\mathcal{T}$ that produce a prefix of n stream constructor.
- For this purpose establish a bisimulation-like correspondence between a μ-term representations of the SCS and the corresponding pebbleflow net using tracker symbols.

Preservation of Production (Idea)

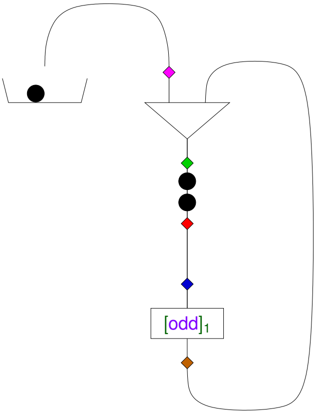


$\mu J. \bullet(\bullet(\mathrm{box}([\mathrm{even}]_1, J)))$

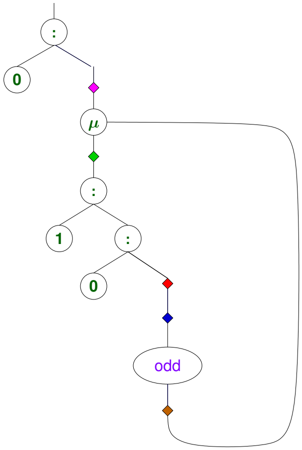


$\mu J. 0 : 1 : \mathrm{even}(J)$

Preservation of Production (Idea)

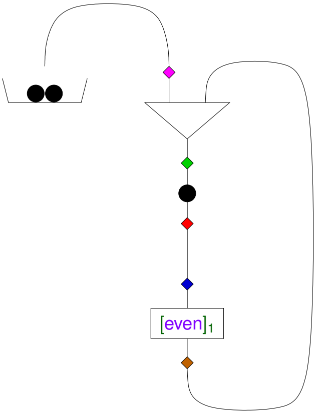


$\bullet(\mu J.\,\bullet(\bullet(\mathrm{box}([\mathrm{odd}]_1, J))))$

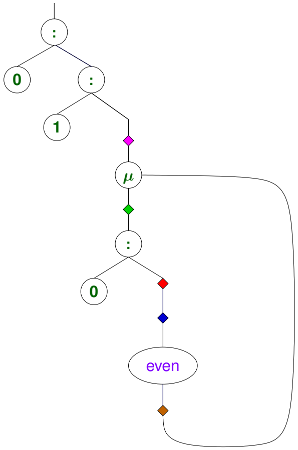


$0 : \mu J.\, 1 : 0 : \mathrm{odd}(J)$

Preservation of Production (Idea)

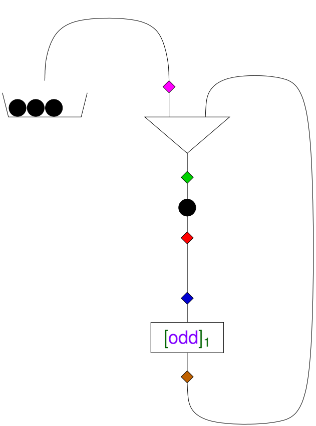


$\bullet(\bullet(\mu J.\,\bullet(\text{box}([\text{even}]_1, J))))$

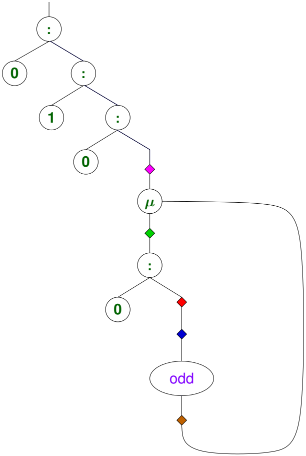


$0 : 1 : \mu J.\, 0 : \text{even}(J)$

Preservation of Production (Idea)

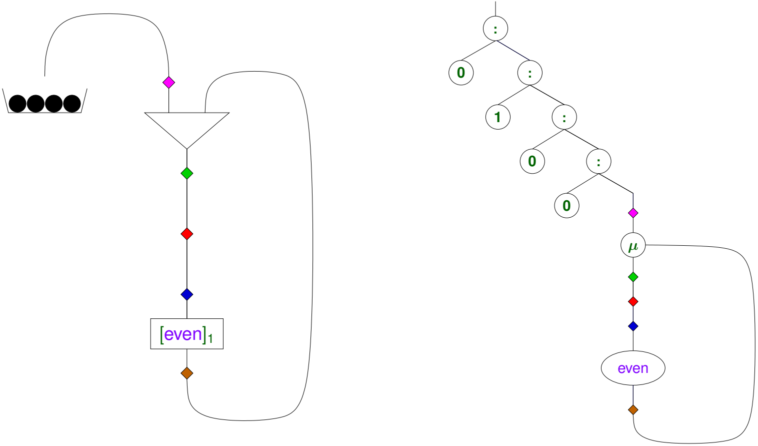


$\bullet(\bullet(\bullet(\mu J.\, \bullet(\mathsf{box}([\mathsf{odd}]_1, J)))))$



$0 : 1 : 0 : \mu J.\, \mathsf{odd}(J)$

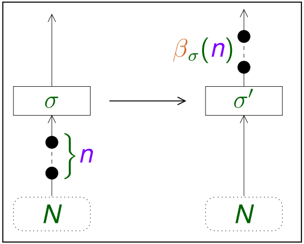
Preservation of Production (Idea)

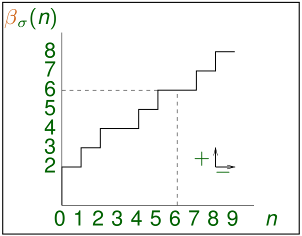


$\bullet(\bullet(\bullet(\bullet(\mu J.\, \mathsf{box}([\mathsf{even}]_1, J)))))$

$0 : 1 : 0 : 0 : \mu J.\, \mathsf{even}(J)$

Production Function

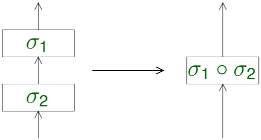




$$\mathrm{box}(\sigma, \bullet^{n}(N)) \to \bullet^{\beta_\sigma(n)}(\mathrm{box}(\sigma', N))$$

Graph of the
production function β_σ
for $\sigma = ++\overline{-+-+-}$.

Box Composition

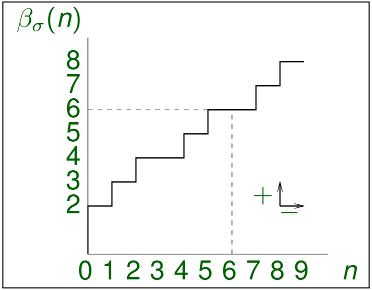


$$(+\sigma_1) \circ \sigma_2 = +(\sigma_1 \circ \sigma_2)$$
$$(-\sigma_1) \circ (+\sigma_2) = \sigma_1 \circ \sigma_2$$
$$(-\sigma_1) \circ (-\sigma_2) = -((-\sigma_1) \circ \sigma_2)$$

Proposition

- $\beta_{\sigma_1 \circ \sigma_2} = \beta_{\sigma_1} \circ \beta_{\sigma_2}$
- *associative*
- *preserves rationality*
- *rat. rep. of* $\sigma_1 \circ \sigma_2$ *can be computed from rat. rep.'s of* σ_1 *and* σ_2

Least Fixed Point of Box Composition



Graph of the production function β_σ for $\sigma = {+}{+}\overline{{-}{+}{-}{+}{-}}$
with least fixed point $\mathsf{fix}(\sigma) = 6$ as indicated.

Lemma

- *Given a rational representation $\langle \alpha, \gamma \rangle$ of $\sigma \in \pm_r^\omega$, its least fixed point $\mathsf{fix}(\sigma)$ can be computed in finite time.*

From Nets to Sources

Definition

Net reduction relation $\to_{\mathsf{R}}$ on closed pebbleflow nets:

$$
\begin{aligned}
\bullet(N) &\to \mathrm{box}((+\overline{-+}), N) \\
\mathrm{box}(\sigma, \mathrm{box}(\tau, N)) &\to \mathrm{box}(\sigma \cdot \tau, N) \\
\mathrm{box}(\sigma, \triangle(N_1, N_2)) &\to \triangle(\mathrm{box}(\sigma, N_1), \mathrm{box}(\sigma, N_2)) \\
\mu x.\triangle(N_1, N_2) &\to \triangle(\mu x.N_1, \mu x.N_2) \\
\mu x.N &\to N \qquad\qquad\qquad\qquad \text{if } x \notin \mathrm{FV}(N) \\
\mu x.\mathrm{box}(\sigma, x) &\to \mathrm{src}(\mathrm{fix}(\sigma)) \\
\triangle(\mathrm{src}(k_1), \mathrm{src}(k_2)) &\to \mathrm{src}(\min(k_1, k_2)) \\
\mathrm{box}(\sigma, \mathrm{src}(k)) &\to \mathrm{src}(\beta_\sigma(k)) \\
\mu x.x &\to \mathrm{src}(0)
\end{aligned}
$$

for all $\sigma, \tau \in \pm^\omega$ and $k, k_1, k_2 \in \overline{\mathbb{N}}$.

Properties of Net Reduction

Theorem

- $\rightarrow_{\mathsf{R}}$ *is production preserving:*

$$N \rightarrow_{\mathsf{R}} N' \quad\Longrightarrow\quad \Pi(N) = \Pi(N')\ .$$

- $\rightarrow_{\mathsf{R}}$ *is confluent and terminating.*
- *Every closed net normalises to a source, its unique* $\rightarrow_{\mathsf{R}}$*-normal form.*
- *For every rational net* N*, the* $\rightarrow_{\mathsf{R}}$*-normal form of* N *can be computed effectively.*

Deciding Productivity for pure SCSs

Theorem

Productivity for pure SCSs is decidable.

Proof.

A decision algorithm for productivity of an SCS $\mathcal{T}$:

1. Translate M_0 to the rational net $[\mathsf{M}_0]$.
2. Reduce $[\mathsf{M}_0]$ to a source $\mathsf{src}(n)$.
   (Note that $\pi_{\mathcal{T}}(\mathsf{M}_0) = \pi([\mathsf{M}_0]) = n$.)
3. If $n = \infty$, then output: "$\mathcal{T}$ is productive";
   else, $n \in \mathbb{N}$, output: "$\mathcal{T}$ is not productive, it only produces n data-elements for M_0".

□

Net Reduction Tool: Computing Net Production

- Translation and reduction tool (Haskell-based) by Jörg Endrullis.
  Use it at: `http://infinity.few.vu.nl/productivity`

$$\mathsf{T} = 0 : \mathsf{zip}(\mathsf{inv}(\mathsf{T}), \mathsf{tail}(\mathsf{T}))$$
$$\Pi_{\mathcal{T}}(\mathsf{T}) = \Pi([\mathsf{T}]) = \infty$$

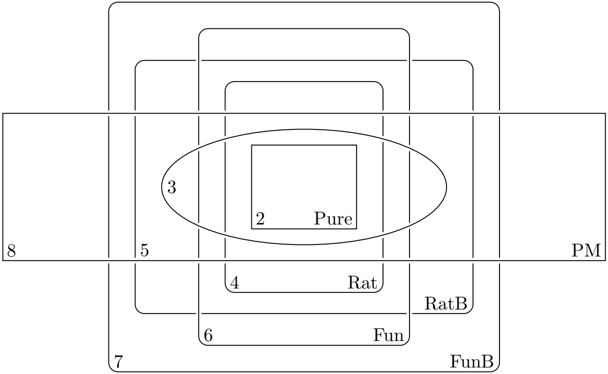
$$\mathsf{J} = 0 : 1 : \mathsf{even}(\mathsf{J})$$
$$\Pi_{\mathcal{T}}(\mathsf{J}) = \Pi([\mathsf{J}]) = 4$$

$$\mathsf{D} = 0 : 1 : 0 : \mathsf{zip}(\mathsf{add}(\mathsf{tail}(\mathsf{D}), \mathsf{tail}(\mathsf{tail}(\mathsf{D}))), \mathsf{even}(\mathsf{tail}(\mathsf{D})))$$
$$\Pi_{\mathcal{T}}(\mathsf{D}) = \Pi([\mathsf{D}]) = \infty$$

Summary and Extensions

- Previous Approaches: sufficient conditions for productivity, not automatable or only for a limited subclass
- Our Contribution: decision algorithm for a rich class of SCSs, only SFS part is restricted
- Recent Results:
  - We increase the applicability of the pebbleflow method by translating to nets that bound the production of SCSs from below and above.
  - We obtain, for some classes of SCSs, computable sufficient conditions for productivity, and for non-productivity.
  - For a class of SFSs with pattern matching on data we give a computable, data-obliviously optimal, sufficient condition for productivity.

Application to Larger Classes



PM SCS

Example

$\mathsf{M}_0 \to 0 : 1 : \mathsf{f}(\mathsf{M}_0)$	SCS-layer
$\mathsf{f}(0 : x : \sigma) \to 1 : 0 : x : \mathsf{f}(\sigma)$ $\mathsf{f}(1 : \sigma) \to 0 : \mathsf{f}(\sigma)$	SFS-layer
	data-layer

This **PM** SCS is productive.

Our Papers and Tools.

Please visit `http://infinity.few.vu.nl/productivity` to find:

- Productivity of Stream Definitions, Proceedings of FCT 2007, LNCS 4637, pages 274–287, 2007;
- Productivity of Stream Definitions, Technical Report;
- Sufficiently productive? Or not productive enough?, submitted;
- and to access and use our tools.

Thanks for your attention!