Productivity of Stream Definitions

Jörg Endrullis Clemens Grabmayer Dimitri Hendriks
Ariya Isihara Jan Willem Klop

Universiteit Utrecht, Vrije Universiteit, Radboud Universiteit

NWO Projects Infinity and ProvCyc

TeReSe Meeting, TU/e
May 24, 2007

Outline

Productivity

- When do we accept an infinite mathematical object to be constructively defined in terms of itself?
- When does a finite set of term equations uniquely represent and constructively define a countably infinite mathematical object?
- One way of answering is:
  - if the equations are productive:
  - if they evaluate to a unique constructor normal form,
  - if the equations allow to generate leading constructors to an arbitrary depth.
- Typical examples of productive objects (objects specified by productive equations) are trees built of constructor symbols.
- A productive process continuously turns input into output, i.e. maps productive objects to productive objects.
- In general, productivity is undecidable.
- Examples: coinductive natural numbers, streams, recursively defined infinite processes, trees, proofs, ...

(Co)recursive stream definitions

- Whereas recursion eliminates (finite) data, corecursion produces potentially infinite data, codata.
- Instead of descending the argument of a call, a corecursive call increases the result.
- Consecutive corecursive calls in a productive stream definition must eventually always produce a constructor symbol.

Example

$$\begin{aligned} \mathsf{zeros} &= 0 : \mathsf{zeros} \\ \mathsf{alt} &= 0 : 1 : \mathsf{alt} \\ \mathsf{nats} &= 0 : \mathsf{map}(+1, \mathsf{nats}) \\ \mathsf{map}(\mathsf{f}, \mathsf{a} : \sigma) &= \mathsf{f}(\mathsf{a}) : \mathsf{map}(\mathsf{f}, \sigma) \end{aligned}$$

Productivity of Stream Definitions

A (co)recursive stream definition $\mathsf{M} = \ldots \mathsf{M} \ldots$ is productive if and only if the process of continuously evaluating M results in an infinite constructor normal form $t_0 : t_1 : t_2 : \ldots$.

Example

$$
\begin{aligned}
\mathsf{alt}' &= 0 : \mathsf{inv}(\mathsf{alt}') \\
\mathsf{alt}'' &= \mathsf{zip}(\mathsf{zeros}, \mathsf{ones}) \\
\mathsf{fib} &= 0 : 1 : \mathsf{add}(\mathsf{fib}, \mathsf{tail}(\mathsf{fib})) \\
\mathsf{morse} &= 0 : 1 : \mathsf{zip}(\mathsf{tail}(\mathsf{morse}), \mathsf{inv}(\mathsf{tail}(\mathsf{morse})))
\end{aligned}
$$

where

$$
\begin{aligned}
\mathsf{tail}(x : \sigma) &= \sigma \\
\mathsf{inv}(x : \sigma) &= (1 - x) : \mathsf{inv}(\sigma) \\
\mathsf{add}(x : \sigma, y : \tau) &= (x + y) : \mathsf{add}(\sigma, \tau) \\
\mathsf{zip}(x : \sigma, \tau) &= x : \mathsf{zip}(\tau, \sigma)
\end{aligned}
$$

Example

$$\mathsf{read}(x:\sigma) = x:\mathsf{read}(\sigma)$$
$$\mathsf{fastread}(x:y:\sigma) = x:y:\mathsf{fastread}(\sigma)$$
$$\mathsf{fives} = 5:\mathsf{read}(\mathsf{fives}) \qquad \text{productive}$$
$$\mathsf{fives}' = 5:\mathsf{fastread}(\mathsf{fives}') \qquad \text{not productive}$$
$$\mathsf{zip}_1(x:\sigma,\tau) = x:\mathsf{zip}_1(\tau,\sigma)$$
$$\mathsf{zip}_2(x:\sigma,y:\tau) = x:y:\mathsf{zip}_2(\sigma,\tau)$$
$$\mathsf{X}_1 = \mathsf{a}:\mathsf{zip}_1(\mathsf{X}_1,\mathsf{tail}(\mathsf{X}_1)) \qquad \text{productive}$$
$$\mathsf{X}_2 = \mathsf{b}:\mathsf{zip}_2(\mathsf{X}_2,\mathsf{tail}(\mathsf{X}_2)) \qquad \text{not productive}$$

Stream Function Specifications

Example

Consider the orthogonal TRS for stream functions

$$\mathsf{even}(x : \sigma) \to x : \mathsf{odd}(\sigma) \qquad \mathsf{tail}(x : \sigma) \to \sigma$$

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

$$\mathsf{add}(x : \sigma, y : \tau) \to \mathsf{a}(x, y) : \mathsf{add}(\sigma, \tau)$$

and operations on data terms:

$$\mathsf{a}(x, 0) \to x \qquad \mathsf{a}(x, \mathsf{s}(y)) \to \mathsf{s}(\mathsf{a}(x, y))\ .$$

We call such a TRS a stream function specification (SFS).

Stream Constant Specifications

Example (Continued)

Based on the SFS for even, odd, zip, add, and tail, consider the extension by:

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

In this stream constant specification (SCS) we have

$$\mathsf{J} \twoheadrightarrow 0 : 1 : 0 : 0 : \mathsf{even}(\mathsf{even}(\ldots))$$
$$\mathsf{D} \twoheadrightarrow 0 : 1 : 1 : 2 : 1 : 3 : 2 : 3 : 3 : 4 : 3 : 5 : 4 : 5 : 5 : 6 : 5 : 7 : 6 : 7 : 7 : \ldots$$

Hence: D is productive, but J is not productive, in this SCS.

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

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

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

Weakly Guarded SFSs and Pure SCSs

Example (Continued)

In the SFS $\mathcal{T}$ we have 'production cycles' of the form:

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

We say that even, odd, zip, and inv are weakly guarded. And we have a collapsing rewrite sequence:

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

We say that tail is collapsing in $\mathcal{T}$.

Such SFSs are called weakly guarded.
SCSs based on weakly guarded SFS are called pure.

Weakly Guarded SFSs

Definition

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

1. $\mathcal{T}$ is orthogonal.
2. The data part $\langle \Sigma_d, R_d \rangle$ is a strongly normalising.
3. Each rule in R_{sf} is of one of the two forms:

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

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

Pure SCSs

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 called a pure recursive stream specification (SCS) iff:

1. $\langle \Sigma_d \uplus \Sigma_{sf} \uplus \{:\}, R_d \uplus R_{sf} \rangle$ is a weakly guarded SFS.
2. $\Sigma_{sc} = \{M_1, \ldots, M_n\}$ set of stream constant symbols; $R_{sc} = \{\rho_{M_i} \mid i \in \{1, \ldots, n\}\}$ where ρ_{M_i} the defining rule for M_i:
$$M_i \to C_i[M_1, \ldots, M_n]$$
where C_i an n-ary stream context in the underlying SFS.

Note: SCSs are orthogonal TRSs.

Production of a Term

Definition

Let $\mathcal{T} = \langle \Sigma, R \rangle$ a pure SCS.
The production $\pi_{\mathcal{T}}(t)$ of a term $t \in Ter(\Sigma)$ is the supremum of the number of data elements t can 'produce':

$$\pi_{\mathcal{T}}(t) := \sup\{n \in \mathbb{N} \mid t \twoheadrightarrow s_1 : \ldots : s_n : t'\} .$$

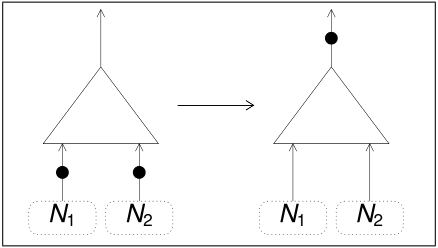
Modelling SCSs with Pebbleflow Nets

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

Pebbleflow Nets:

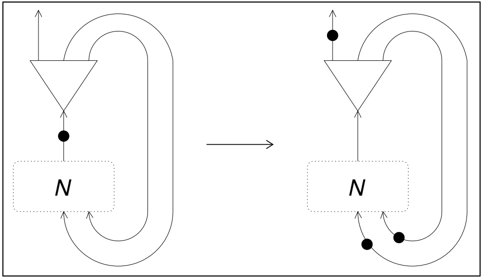
- Stream elements are abstracted from in favour of 'pebbles'.
- A stream definition is modelled by a pebbleflow net: The process of evaluation of a stream definition is modelled by the dataflow of pebbles in a pebbleflow net.
- A stream definition is productive if and only if the net associated to it generates an infinite chain of pebbles.
- Elements are: meets, fans, boxes and gates, sources, wires.

Meet



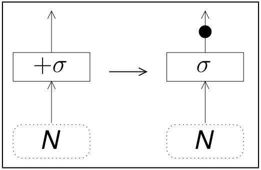
$$\triangle(\bullet(N_1), \bullet(N_2)) \to \bullet(\triangle(N_1, N_2))$$

Recursion



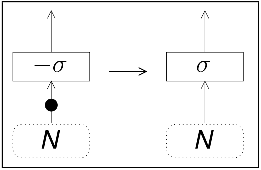
$$\mu x.\bullet(N(x)) \rightarrow \bullet(\mu x.N(\bullet(x)))$$

Box



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

Box(2)



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

I/O sequences

Definition

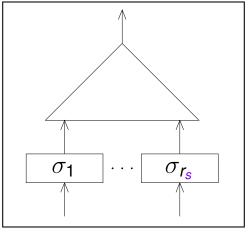
The set $\pm^\omega$ of I/O sequences is the set of infinite sequences over the alphabet $\{+,-\}$ that contain an infinite number of $+$'s:

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

An I/O sequence $\sigma \in \pm^\omega$ is called rational if there exist lists $\alpha, \gamma \in \{+,-\}^*$ such that $\sigma = \alpha\overline{\gamma}$, where γ is not empty.
The pair $\langle \alpha, \gamma \rangle$ is called a rational representation of σ.
And we define:

$$\pm^\omega_{rat} := \{\sigma \in \pm^\omega \mid \sigma \text{ is rational}\} .$$

Gates



A gate for modelling r_s-ary stream functions.

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

Term Representations of Nets

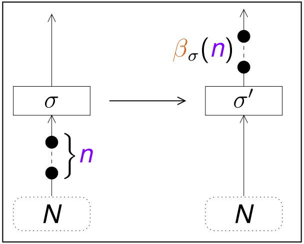
Definition

Let $\mathcal{V}$ be a set of variables, and $\overline{\mathbb{N}} := \mathbb{N} \cup \{\infty\}$.
The set $\mathcal{N}$ of terms for pebbleflow nets is generated by:

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

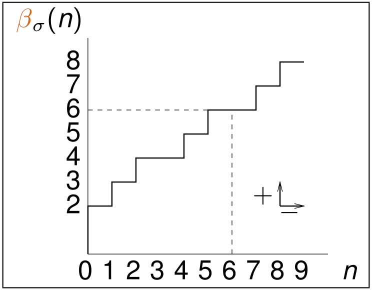
where $k \in \overline{\mathbb{N}}$, $x \in \mathcal{V}$, and $\sigma \in \pm^{\omega}$.

Production Function



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

Production Function



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

Production Function

Definition

The production function $\beta_\sigma : \overline{\mathbb{N}} \to \overline{\mathbb{N}}$ of (a box containing) a sequence $\sigma \in \pm^\omega$ is corecursively defined, for all $n \in \overline{\mathbb{N}}$, by $\beta_\sigma(n) := \beta(\sigma, n)$:

$$\begin{aligned} \beta(+\sigma, n) &= \mathsf{S}(\beta(\sigma, n)) \\ \beta(-\sigma, 0) &= 0 \\ \beta(-\sigma, \mathsf{S}(n)) &= \beta(\sigma, n) \end{aligned}$$

Pebbleflow

Definition

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

$$\begin{aligned} \triangle(\bullet(N_1), \bullet(N_2)) &\to \bullet(\triangle(N_1, N_2)) && \text{(P1)} \\ \mu x.\bullet(N(x)) &\to \bullet(\mu x.N(\bullet(x))) && \text{(P2)} \\ \mathsf{box}((+\sigma), N) &\to \bullet(\mathsf{box}(\sigma, N)) && \text{(P3)} \\ \mathsf{box}((-\sigma), \bullet(N)) &\to \mathsf{box}(\sigma, N) && \text{(P4)} \\ \mathsf{src}(\mathsf{S}(k)) &\to \bullet(\mathsf{src}(k)) && \text{(P5)} \end{aligned}$$

$\to_{\mathsf{p}}$ is an orthogonal CRS, and hence:

Theorem

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

Production of a Net

Definition

The production $\pi(N)$ of a net $N \in \mathcal{N}$ is the supremum of the number of pebbles the net can 'produce':

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

Ariya's Tool

A **net visualization applet** (Java-based).

Is intended to give a feeling for pebbleflow in pebbleflow nets.

Translation of Stream Functions into Gates

Example

Following the collapsing rewrite sequence:

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

the translation of the stream function tail into a rational gate is:

$$[\mathsf{tail}](N) = \triangle_1(\mathsf{box}([\mathsf{tail}]_1, N)) = - - + - + \ldots = -\overline{-+}$$

Translation of Stream Functions into Gates

Example

For the stream function specification

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

which enables the 'production cycle'

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

the translation of the stream function zip into a rational gate is:

$$\begin{aligned} [\mathsf{zip}](N_1, N_2) &= \triangle(\mathsf{box}([\mathsf{zip}]_1, N_1), \mathsf{box}([\mathsf{zip}]_2, N_2)) \\ &= \triangle(\mathsf{box}(-+[\mathsf{zip}]_2, N_1), \mathsf{box}(+[\mathsf{zip}]_1, N_2)) \\ &= \triangle(\mathsf{box}(-++[\mathsf{zip}]_1, N_1), \mathsf{box}(+-+[\mathsf{zip}]_2, N_2)) \\ &= \triangle(\mathsf{box}(\overline{-++}, N_1), \mathsf{box}(\overline{+-+}, N_2)) \end{aligned}$$

Translation of Stream Constants into Gates

Definition

Let $\mathcal{T} = \langle \Sigma_d \uplus \Sigma_{sf} \uplus \{:\}, R_d \uplus R_{sf} \rangle$ an SFS. For every $\mathsf{f} \in \Sigma_{sf}$ with arity $\langle r_s, r_d \rangle$, the translation of f is a rational gate $[\mathsf{f}] : \mathcal{N}^{r_s} \to \mathcal{N}$ def. by:

$$[\mathsf{f}](N_1, \ldots, N_{r_s}) = \triangle_{r_s}(\mathrm{box}([\mathsf{f}]_1, N_1), \ldots, \mathrm{box}([\mathsf{f}]_{r_s}, N_{r_s})) \; ,$$

where $[\mathsf{f}]_i \in \pm^{\omega}_{rat}$ is defined as follows. We distinguish the two formats a rule $\rho_{\mathsf{f}} \in R_{sf}$ can have. Let $\vec{x_i} : \sigma_i$ stand for $x_{i,1} : \ldots : x_{i,n_i} : \sigma_i$. If ρ_{f} has the form: $\mathsf{f}(\vec{x_1} : \sigma_1, \ldots, \vec{x_{r_s}} : \sigma_{r_s}, y_1, \ldots, y_{r_d}) \to t_1 : \ldots : t_{m_{\mathsf{f}}} : u$, where:

$$u \equiv \mathsf{g}(\sigma_{\pi_{\mathsf{f}}(1)}, \ldots, \sigma_{\pi_{\mathsf{f}}(r'_s)}, t'_1, \ldots, t'_{r'_d}) \; ,$$

then

$$[\mathsf{f}]_i = \begin{cases} -^{n_i} +^{m_{\mathsf{f}}} [\mathsf{g}]_j & \text{if } \pi_{\mathsf{f}}(j) = i \\ -^{n_i} \overline{+} & \text{if } \neg\exists j.\, \pi_{\mathsf{f}}(j) = i \end{cases}$$

$$u \equiv \sigma_j \; ,$$

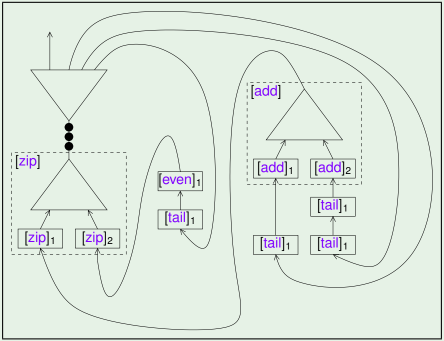
then

$$[\mathsf{f}]_i = \begin{cases} -^{n_i} +^{m_{\mathsf{f}}} \overline{-+} & \text{if } i = j \\ -^{n_i} \overline{+} & \text{if } i \neq j \end{cases}$$

Translation of Stream Constants into Nets

Example

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



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

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, $\pi([\mathsf{M}]) = \pi_{\mathcal{T}}(\mathsf{M})$ *for all* $\mathsf{M} \in \Sigma_{sc}$.

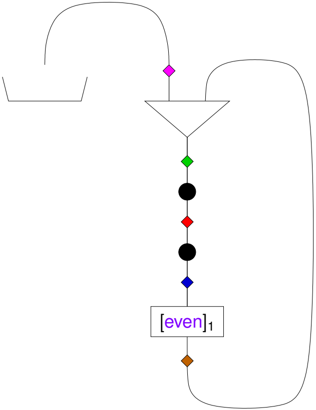
Proof.

$\pi([\mathsf{M}]) \leq \pi_{\mathcal{T}}(\mathsf{M})$: Given a rewrite sequence $[\mathsf{M}] \twoheadrightarrow_{\mathsf{p}} \bullet^{n}(N)$, define inductively a rewrite sequence

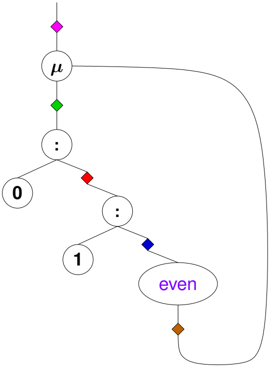
$$\boldsymbol{\mu}(\mathsf{M}) \twoheadrightarrow_{\mu\mathcal{T}} t'_1 : \ldots : t'_n : u'$$

on μ-term representations of infinite terms such that the production of equally coloured contexts within these terms are preserved.

Preservation of Production

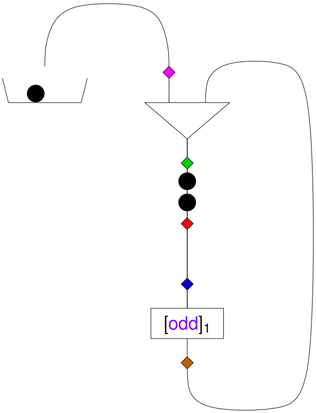


$\mu J. \bullet(\bullet(\text{box}([\text{even}]_1, J)))$

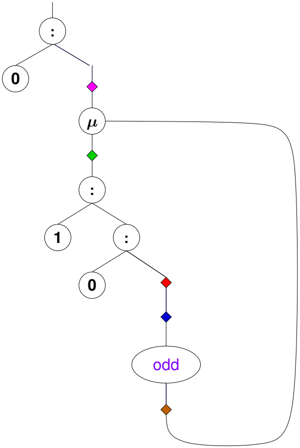


$\mu J. 0 : 1 : \text{even}(J)$

Preservation of Production

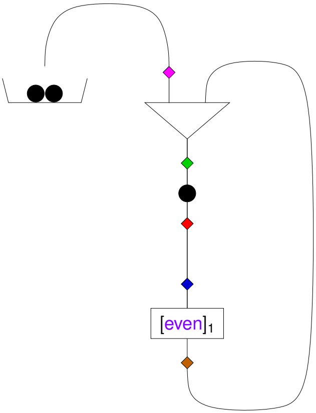


$\bullet(\mu J.\,\bullet(\bullet(\mathrm{box}([\mathrm{odd}]_1, J))))$

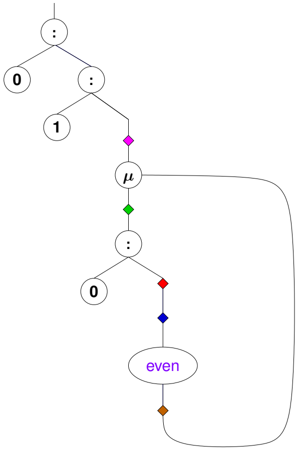


$0 : \mu J.\, 1 : 0 : \mathrm{odd}(J)$

Preservation of Production

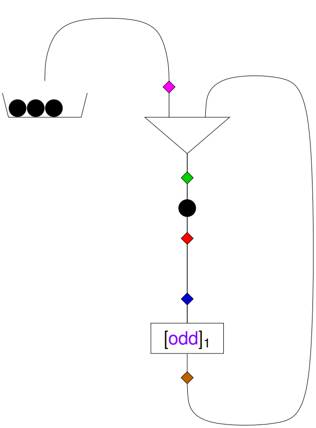


$\bullet(\bullet(\mu J.\, \bullet(\text{box}([\text{even}]_1, J))))$

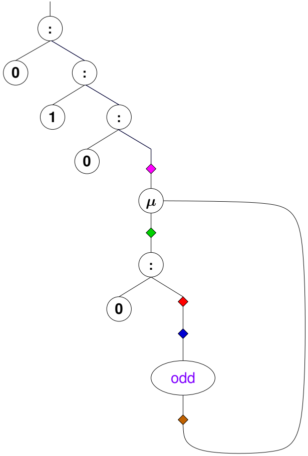


$0 : 1 : \mu J.\, 0 : \text{even}(J)$

Preservation of Production

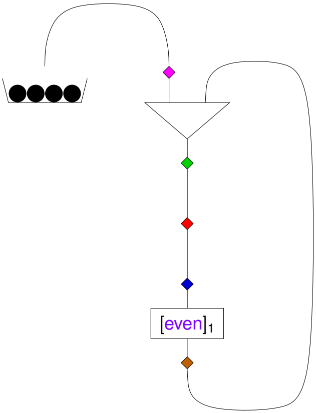


$\bullet(\bullet(\bullet(\mu J.\,\bullet(\mathsf{box}([\mathsf{odd}]_1, J)))))$

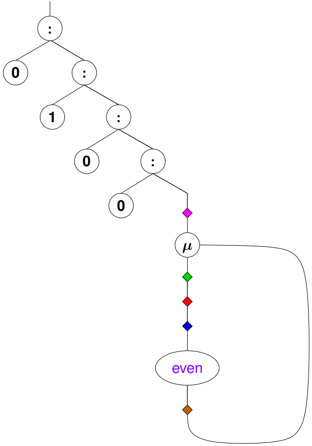


$0 : 1 : 0 : \mu J.\,\mathsf{odd}(J)$

Preservation of Production



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



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

Proof Continued.

$\pi([\mathsf{M}]) \leq \pi_{\mathcal{T}}(\mathsf{M})$: [...] define inductively a rewrite sequence $\boldsymbol{\mu}(\mathsf{M}) \twoheadrightarrow_{\mu\mathcal{T}} t_1' : \ldots : t_n' : u'$ on μ-term representations of infinite terms such that the production of equally coloured contexts within these terms are preserved. Finally, lift this sequence of μ-terms to an infinite rewrite sequence $\mathsf{M} \twoheadrightarrow\!\!\!\!\!\rightarrow_{\mu\mathcal{T}} t_1 : \ldots : t_n : u$ of length $k\omega$, for some $k \in \overline{\mathbb{N}}$. Finally, use compression.

$\pi([\mathsf{M}]) \leq \pi_{\mathcal{T}}(\mathsf{M})$ Given a rewrite sequence $\mathsf{M} \twoheadrightarrow_{\mu\mathcal{T}} t_1 : \ldots : t_n : u$, it is possible to construct, using the fact that in OTRS taking sequences of complete developments is a cofinal rewrite strategy, and starting from a sufficiently large finite unfolding of M in $\mathcal{T}$, a rewrite sequence $\boldsymbol{\mu}(\mathsf{M}) \twoheadrightarrow_{\mu\mathcal{T}} t_1' : \ldots : t_n' : u'$ on μ-term representations of infinite terms. This rewrite sequence can be used to define inductively, similar as in the first case by preserving the production of equally coloured contexts in every step, a rewrite sequence $[\mathsf{M}] \twoheadrightarrow_{\mathsf{p}} \bullet^n(N)$. $\square$

Box Composition

Definition

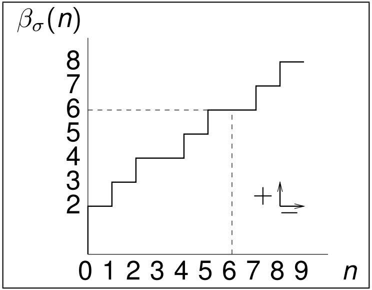
Composition $\cdot : \pm^\omega \times \pm^\omega \to \pm^\omega$, $\langle \sigma, \tau \rangle \mapsto \sigma \cdot \tau$ of I/O sequences is corecursively defined by:

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

Lemma

- $\beta_{\sigma \cdot \tau} = \beta_\sigma \circ \beta_\tau$.
- *Composition is associative.*
- *Composition preserves rationality:* $\sigma \cdot \tau \in \pm^\omega_{rat}$ *if* $\sigma, \tau \in \pm^\omega_{rat}$.
- *On rational representations of rational I/O sequences, composition can be computed effectively.*

Least Fixed Point of Box Composition



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

Fixed Point Computation

Definition

The operations fixed point fix : $\pm^\omega \to \overline{\mathbb{N}}$ and first requirement removal δ : $\pm^\omega \to \pm^\omega$ are corecursively defined by:

$$\mathsf{fix}(+\sigma) = \mathsf{S}(\mathsf{fix}(\delta(\sigma))) \qquad \delta(+\sigma) = +\delta(\sigma)$$
$$\mathsf{fix}(-\sigma) = 0 \qquad \delta(-\sigma) = \sigma$$

Lemma

- $\mathsf{fix}(\sigma)$ *is the least fixed point of* β_σ.
- *Given a rational representation* $\langle \alpha, \gamma \rangle$ *of* $\sigma \in \pm^\omega_{rat}$, *its 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 is defined, for all $\sigma, \tau \in \pm^{\omega}$ and $k, k_1, k_2 \in \overline{\mathbb{N}}$, by:

$$
\begin{aligned}
\bullet(N) &\to \mathsf{box}((+\overline{-+}), N) && && \text{(R1)}\\
\mathsf{box}(\sigma, \mathsf{box}(\tau, N)) &\to \mathsf{box}(\sigma \cdot \tau, N) && && \text{(R2)}\\
\mathsf{box}(\sigma, \triangle(N_1, N_2)) &\to \triangle(\mathsf{box}(\sigma, N_1), \mathsf{box}(\sigma, N_2)) && && \text{(R3)}\\
\mu x.\triangle(N_1, N_2) &\to \triangle(\mu x.N_1, \mu x.N_2) && && \text{(R4)}\\
\mu x.N &\to N && \text{if } x \notin \mathsf{FV}(N) && \text{(R5)}\\
\mu x.\mathsf{box}(\sigma, x) &\to \mathsf{src}(\mathsf{fix}(\sigma)) && && \text{(R6)}\\
\triangle(\mathsf{src}(k_1), \mathsf{src}(k_2)) &\to \mathsf{src}(\mathsf{min}(k_1, k_2)) && && \text{(R7)}\\
\mathsf{box}(\sigma, \mathsf{src}(k)) &\to \mathsf{src}(\beta_{\sigma}(k)) && && \text{(R8)}\\
\mu x.x &\to \mathsf{src}(0) && && \text{(R9)}
\end{aligned}
$$

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.*

Main Result

Theorem

Productivity for pure SCSs is decidable.

Proof.

The following steps describe an decision algorithm for a stream constant M in an SCS $\mathcal{T}$:

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

$\square$

Jörg's Tool

A **translation and collapsing tool** (Haskell-based).

Input: A pure SCS $\mathcal{T}$, a stream constant M in $\mathcal{T}$.

Output: A natural number n or the symbol ∞ dependent on whether the maximal number of leading stream constructor symbols ":" in a reduct of M in $\mathcal{T}$ is n, or respectively, is unbounded.

Conclusion and Ongoing Reserach

- A decision algorithm for a rich class of stream definitions intended as a tool for functional programming practice. Our format of SCSs only restricts the SFS part (i.p. no nesting of recursive calls), but not how SCSs make use of stream functions.
- Previous approaches established criteria for productivity (not applicable for disproving) and are either applicable to general stream def's, but not automatable (Sijtsma '89, Buchholz '05), or give a 'productive'/'don't know' answer only for a very limited subclass (Wadge '81, Hughes–Pareto–Sabry '96, Telford–Turner '97, Buchholz '05).
- Current research: Computable criteria for productivity and its complement by considering lower and upper rational bounds on the production of stream definitions. (Allows to deal with stream functions whose production depends quantitiatively on the values of stream elements and data parameters).

Thanks for your attention!