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

1. Introduction
   - Productivity

2. Recursive Stream Specifications
   - Motivating Example
   - Weakly Guarded Stream Function Specifications
   - Pure Stream Constant Specifications

3. Modelling with Nets
   - Pebbleflow Nets
   - A Rewrite System for Pebbleflow. Ariya’s Tool.
   - Translating Pure Stream Specifications
   - Preservation of Production

4. Deciding Productivity
   - Composition and Fixed Point
   - Net Reduction

5. Conclusion and Ongoing Work
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

```plaintext
zeros = 0 : zeros
alt = 0 : 1 : alt
nats = 0 : map(+1, nats)
map(f, a : σ) = f(a) : map(f, σ)
```
Productivity of Stream Definitions

A (co)recursive stream definition $M = \ldots M \ldots$ 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**

\[
\begin{align*}
alt' &= 0 : \text{inv}(alt') \\
alt'' &= \text{zip}(\text{zeros}, \text{ones}) \\
fib &= 0 : 1 : \text{add}(\text{fib}, \text{tail}(\text{fib})) \\
morse &= 0 : 1 : \text{zip}(\text{tail}(\text{morse}), \text{inv}(\text{tail}(\text{morse})))
\end{align*}
\]

where

\[
\begin{align*}
tail(x : \sigma) &= \sigma \\
\text{inv}(x : \sigma) &= (1 - x) : \text{inv}(\sigma) \\
\text{add}(x : \sigma, y : \tau) &= (x + y) : \text{add}(\sigma, \tau) \\
\text{zip}(x : \sigma, \tau) &= x : \text{zip}(\tau, \sigma)
\end{align*}
\]
Example

\[
\begin{align*}
\text{read}(x : \sigma) &= x : \text{read}(\sigma) \\
\text{fastread}(x : y : \sigma) &= x : y : \text{fastread}(\sigma) \\
\text{fives} &= 5 : \text{read}(\text{fives}) \quad \text{productive} \\
\text{fives}^\prime &= 5 : \text{fastread}(\text{fives}^\prime) \quad \text{not productive} \\
\text{zip}_1(x : \sigma, \tau) &= x : \text{zip}_1(\tau, \sigma) \\
\text{zip}_2(x : \sigma, y : \tau) &= x : y : \text{zip}_2(\sigma, \tau) \\
X_1 &= a : \text{zip}_1(X_1, \text{tail}(X_1)) \quad \text{productive} \\
X_2 &= b : \text{zip}_2(X_2, \text{tail}(X_2)) \quad \text{not productive}
\end{align*}
\]
Stream Function Specifications

Example

Consider the orthogonal TRS for stream functions

\[
\begin{align*}
\text{even}(x : \sigma) & \rightarrow x : \text{odd}(\sigma) & \text{tail}(x : \sigma) & \rightarrow \sigma \\
\text{odd}(x : \sigma) & \rightarrow \text{even}(\sigma) & \text{zip}(x : \sigma, \tau) & \rightarrow x : \text{zip}(\tau, \sigma) \\
\text{add}(x : \sigma, y : \tau) & \rightarrow a(x, y) : \text{add}(\sigma, \tau)
\end{align*}
\]

and operations on data terms:

\[
\begin{align*}
a(x, 0) & \rightarrow x & a(x, s(y)) & \rightarrow s(a(x, y))
\end{align*}
\]

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

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

\[
\begin{align*}
J & \rightarrow 0 : 1 : \text{even}(J) \\
D & \rightarrow 0 : 1 : 0 : \text{zip}(\text{add}(\text{tail}(D), \text{tail}(\text{tail}(D))), \text{even}(\text{tail}(D)))
\end{align*}
\]

In this stream constant specification (SCS) we have

\[
\begin{align*}
J & \rightarrow 0 : 1 : 0 : 0 : \text{even}(\text{even}(\ldots)) \\
\end{align*}
\]

Hence: `D` is productive, but `J` is not productive, in this SCS.
J \rightarrow 0 : 1 : 0 : 0 : \text{even}^\omega

\begin{align*}
J & \rightarrow 0 : 1 : \text{even}(J) \\
\text{even}(J) & \rightarrow \text{even}(0 : 1 : \text{even}(J)) \\
& \rightarrow 0 : \text{odd}(1 : \text{even}(J)) \\
& \rightarrow 0 : \text{even}(\text{even}(J)) \\
\text{even}^2(J) & \equiv \text{even}(\text{even}(J)) \rightarrow \text{even}(0 : \text{even}(\text{even}(J))) \\
& \rightarrow 0 : \text{odd}(\text{even}^2(J)) \\
\text{odd}(\text{even}^2(J)) & \rightarrow \text{odd}(0 : \text{odd}(\text{even}^2(J))) \\
& \rightarrow \text{even}(\text{odd}(\text{even}^2(J))) \\
\text{odd}(\text{even}^2(J)) & \rightarrow \text{even}(\text{odd}(\text{even}^2(J))) \\
& \rightarrow \text{even}^2(\text{odd}(\text{even}^2(J))) \\
& \rightarrow \ldots \rightarrow \text{even}^n(\text{odd}(\text{even}^2(J))) \rightarrow \ldots \\
& \rightarrow \text{even}^\omega
\end{align*}

Hence: \( J \rightarrow 0 : 1 : 0 : 0 : \text{even}^\omega \).
Example (Continued)

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

\[
\text{even}(x : y : \sigma) \rightarrow x : \text{odd}(y : \sigma) \rightarrow x : \text{even}(\sigma) \\
\text{odd}(x : y : \sigma) \rightarrow \text{even}(y : \sigma) \rightarrow y : \text{odd}(\sigma) \\
\text{zip}(x : \sigma, y : \tau) \rightarrow x : \text{zip}(y : \tau, \sigma) \rightarrow x : y : \text{zip}(\sigma, \tau)
\]

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

\[
\text{tail}(x : \sigma) \rightarrow \sigma.
\]

We say that \text{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 \( \mathcal{T} = \langle \Sigma_d \cup \Sigma_{sf} \cup \{.;\}, R_d \cup R_{sf} \rangle \) is called a \textit{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:

\[
\begin{align*}
    & f((x_{1,1} : \ldots : x_{1,n_1} : \sigma_1), \ldots , (x_{r,1} : \ldots : x_{r_s,n_{rs}} : \sigma_{rs}), \vec{y}) \\
        & \rightarrow t_1(\vec{x}, \vec{y}) : \ldots : t_m(\vec{x}, \vec{y}) : \sigma_f , \\
        & \rightarrow t_1(\vec{x}, \vec{y}) : \ldots : t_m(\vec{x}, \vec{y}) : g(\sigma_{\pi_f(1)}, \ldots , \sigma_{\pi_f(r'_s)}, t'_1(\vec{x}, \vec{y}), \ldots , t'_{r'_d}(\vec{x}, \vec{y})) ,
\end{align*}
\]

where \( \pi_f : \{1, \ldots , r'_s\} \rightarrow \{1, \ldots , r_s\} \) is injective in case \( f \sim g \).

4. Weakly guarded: On every dependency cycle \( f \sim g \sim \ldots \sim f \) there is at least one guard.
Pure SCSs

**Definition**

A TRS $\mathcal{T} = \langle \Sigma_d \cup \Sigma_{sf} \cup \Sigma_{sc} \cup \{\_\}, R_d \cup R_{sf} \cup R_{sc} \rangle$ is called a pure recursive stream specification (SCS) iff:

1. $\langle \Sigma_d \cup \Sigma_{sf} \cup \{\_\}, R_d \cup R_{sf} \rangle$ 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 $\rho_{M_i}$ the defining rule for $M_i$:

   $$M_i \rightarrow C_i[M_1, \ldots, M_n]$$

   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 $t \in \text{Ter}(\Sigma)$ is the supremum of the number of data elements $t$ can ‘produce’:

$$\pi_{\mathcal{T}}(t) := \sup \{ n \in \mathbb{N} \mid t \rightarrow s_1 : \ldots : s_n : t' \}.$$
Modelling SCSs with Pebbleflow Nets


**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

\[ \triangle(\bullet(N_1), \bullet(N_2)) \rightarrow \bullet(\triangle(N_1, N_2)) \]
Recursion

\[ \mu x. \bullet(N(x)) \rightarrow \bullet(\mu x. N(\bullet(x))) \]
Box

\[
\text{box}(+\sigma, N) \rightarrow \bullet(\text{box}(\sigma, N))
\]
Box(2)

\[
\text{box}(\neg \sigma, \bullet(N)) \rightarrow \text{box}(\sigma, N)
\]
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:

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

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

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

A gate for modelling $r_s$-ary stream functions.

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

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

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

$$\text{box}(\sigma, \bullet^n(N)) \rightarrow \bullet^{\beta_\sigma(n)}(\text{box}(\sigma', N))$$
Graph of the production function $\beta_\sigma$ for $\sigma = +++-++++-$. 

$\beta_\sigma(n)$
Production Function

Definition

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

$$
\beta(+\sigma, n) = S(\beta(\sigma, n)) \\
\beta(-\sigma, 0) = 0 \\
\beta(-\sigma, S(n)) = \beta(\sigma, n)
$$
Pebbleflow

Definition

The pebbleflow rewrite relation \( \rightarrow_p \) is defined as:

\[
\begin{align*}
\Delta(\bullet(N_1), \bullet(N_2)) & \rightarrow \bullet(\Delta(N_1, N_2)) & (P1) \\
\mu x. \bullet(N(x)) & \rightarrow \bullet(\mu x. N(\bullet(x))) & (P2) \\
\text{box}((+\sigma), N) & \rightarrow \bullet(\text{box}(\sigma, N)) & (P3) \\
\text{box}((-\sigma), \bullet(N)) & \rightarrow \text{box}(\sigma, N) & (P4) \\
\text{src}(S(k)) & \rightarrow \bullet(\text{src}(k)) & (P5)
\end{align*}
\]

\( \rightarrow_p \) is an orthogonal CRS, and hence:

Theorem

*The rewrite relation \( \rightarrow_p \) 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’:

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

\[ \text{tail}(x : \sigma) \rightarrow \sigma . \]

the translation of the stream function \text{tail} into a rational gate is:

\[ [\text{tail}](\mathcal{N}) = \triangle_1(\text{box}([\text{tail}]_1, \mathcal{N})) = \ldots = --+ \]
Translation of Stream Functions into Gates

Example

For the stream function specification

\[ \text{zip}(x : \sigma, \tau) \rightarrow x : \text{zip}(\tau, \sigma) , \]

which enables the ‘production cycle’

\[ \text{zip}(x : \sigma, y : \tau) \rightarrow x : \text{zip}(y : \tau, \sigma) \rightarrow x : y : \text{zip}(\sigma, \tau) , \]

the translation of the stream function \text{zip} into a rational gate is:

\[
\begin{align*}
\text{[zip]}(N_1, N_2) &= \Delta(\text{box}(\text{[zip]}_1, N_1), \text{box}(\text{[zip]}_2, N_2)) \\
&= \Delta(\text{box}(-+\text{[zip]}_2, N_1), \text{box}(+\text{[zip]}_1, N_2)) \\
&= \Delta(\text{box}(-+++\text{[zip]}_1, N_1), \text{box}(+-+\text{[zip]}_2, N_2)) \\
&= \Delta(\text{box}(-+++, N_1), \text{box}(+-+, N_2))
\end{align*}
\]
Translation of Stream Constants into Gates

**Definition**

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

\[
[f](N_1, \ldots, N_{r_s}) = \bigtriangleup_{r_s}(\text{box}([f]_1, N_1), \ldots, \text{box}([f]_{r_s}, N_{r_s}))
\]

where \([f]_i \in \pm \frac{\omega_{rat}}{r} \) is defined as follows. We distinguish the two formats a rule \( \rho_f \in R_{sf} \) can have. Let \( \vec{x}_i : \sigma_i \) stand for \( x_{i,1} : \ldots : x_{i,n_i} : \sigma_i \). If \( \rho_f \) has the form:

\[
f(\vec{x}_1 : \sigma_1, \ldots, \vec{x}_{r_s} : \sigma_{r_s}, y_1, \ldots, y_{r_d}) \rightarrow t_1 : \ldots : t_{m_f} : u,
\]

where:

\[
u \equiv g(\sigma_{\pi_f(1)}, \ldots, \sigma_{\pi_f(r_s')}, t'_1, \ldots, t'_{r_d'})
\]

then

\[
[f]_i = \begin{cases} 
-n_i + m_f[g]_j & \text{if } \pi_f(j) = i \\
-n_i & \text{if } \neg \exists j. \pi_f(j) = i
\end{cases}
\]

then

\[
[f]_i = \begin{cases} 
-n_i + m_f & \text{if } i = j \\
-n_i & \text{if } i \neq j
\end{cases}
\]
Translation of Stream Constants into Nets

Example

\[ D \rightarrow 0 : 1 : 0 : \text{zip}(\text{add}(\text{tail}(D), \text{tail}(\text{tail}(D))), \text{even}(\text{tail}(D))) \]

\[ [D] = \mu D. \bullet (\bullet ([\text{zip}]([\text{add}][\text{tail}](D), [\text{tail}](D))), [\text{even}][\text{tail}](D))) \]
Translation of Stream Constants into Nets

Definition

Let $T = \langle \Sigma_d \uplus \Sigma_{sf} \uplus \Sigma_{sc} \uplus \{:\}, R_d \uplus R_{sf} \uplus R_{sc} \rangle$ be a pure SCS.
For each $M \in \Sigma_{sc}$ with rule $\rho_M \equiv M \rightarrow rhs_M$ the translation $[M] := [M]_\emptyset$ of $M$ into a rational pebbleflow net is recursively def. by:

$$[M]_\alpha = \begin{cases} 
\mu M.[rhs_M]_\alpha \cup \{M\} & \text{if } M \not\in \alpha \\
M & \text{if } M \in \alpha 
\end{cases}$$

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

$$[f(u_1, \ldots, u_{rs}, t_1, \ldots, t_{rd})]_\alpha = [f]([u_1]_\alpha, \ldots, [u_{rs}]_\alpha)$$

where $\alpha$ denotes a set of stream constant symbols.
Translation is Production Preserving

**Theorem**

Let $T$ be a pure SCS. Then, $\pi([M]) = \pi_T(M)$ for all $M \in \Sigma_{sc}$.

**Proof.**

$\pi([M]) \leq \pi_T(M)$: Given a rewrite sequence $[M] \rightarrow_p \bullet^n(N)$, define inductively a rewrite sequence

$$\mu(M) \rightarrow_{\mu_T} t'_1 : \ldots : t'_n : u'$$

on $\mu$-term representations of infinite terms such that the production of equally coloured contexts within these terms are preserved.
Preservation of Production

\[ \begin{align*}
\mu J. \bullet (\bullet (\text{box}([\text{even}]_1, J))) \\
\mu J. 0 : 1 : \text{even}(J)
\end{align*} \]
Preservation of Production

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

\[ 0 : \mu J. 1 : 0 : \text{odd}(J) \]
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

$\text{odd} \mu J. \text{odd}(J)$

$\bullet (\bullet (\bullet (\mu J. \bullet (\text{box}([\text{odd}]_1, J))))))$
Preservation of Production

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

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

\[ \pi([M]) \leq \pi_T(M) : \ldots \] define inductively a rewrite sequence
\[ \mu(M) \rightarrow_{\mu T} t'_1 : \ldots : t'_n : u' \] on \( \mu \)-term representations of infinite terms such that the production of equally coloured contexts within these terms are preserved. Finally, lift this sequence of \( \mu \)-terms to an infinite rewrite sequence \( M \rightarrow_{\mu T} t_1 : \ldots : t_n : u \) of length \( k \omega \), for some \( k \in \mathbb{N} \). Finally, use compression.

\[ \pi([M]) \leq \pi_T(M) \] Given a rewrite sequence \( M \rightarrow_{\mu T} t_1 : \ldots : t_n : u \), 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 \( T \), a rewrite sequence \( \mu(M) \rightarrow_{\mu T} t'_1 : \ldots : t'_n : u' \) on \( \mu \)-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 \( [M] \rightarrow_{p \bullet}^n(N) \). \( \square \)
**Box Composition**

### Definition

**Composition** \( \cdot \) : \( \pm \omega \times \pm \omega \rightarrow \pm \omega \), \( \langle \sigma, \tau \rangle \mapsto \sigma \cdot \tau \) of I/O sequences is corecursively defined by:

\[
(\pm \sigma) \cdot \tau = \pm (\sigma \cdot \tau) \\
(-\sigma) \cdot (+\tau) = \sigma \cdot \tau \\
(-\sigma) \cdot (-\tau) = -((\sigma) \cdot \tau)
\]

### 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 $\beta_\sigma(n)$ for $\sigma = +++---++-$ with least fixed point $\text{fix}(\sigma) = 6$ as indicated.
Fixed Point Computation

Definition

The operations fixed point \( \text{fix} : \pm \omega \to \overline{\mathbb{N}} \) and first requirement removal \( \delta : \pm \omega \to \pm \omega \) are corecursively defined by:

\[
\text{fix}(+\sigma) = S(\text{fix}(\delta(\sigma))) \\
\text{fix}(-\sigma) = 0 \\
\delta(+\sigma) = +\delta(\sigma) \\
\delta(-\sigma) = \sigma
\]

Lemma

- \( \text{fix}(\sigma) \) is the least fixed point of \( \beta_{\sigma} \).
- Given a rational representation \( \langle \alpha, \gamma \rangle \) of \( \sigma \in \pm \omega_{\text{rat}} \), its fixed point \( \text{fix}(\sigma) \) can be computed in finite time.
Definition

Net reduction relation $\rightarrow_R$ on closed pebbleflow nets is defined, for all $\sigma, \tau \in \pm \omega$ and $k, k_1, k_2 \in \mathbb{N}$, by:

1. $(N) \rightarrow \text{box}(\langle +, + \rangle, N)$ \hspace{1cm} (R1)
2. $\text{box}(\sigma, \text{box}(\tau, N)) \rightarrow \text{box}(\sigma \cdot \tau, N)$ \hspace{1cm} (R2)
3. $\text{box}(\sigma, \triangle(N_1, N_2)) \rightarrow \triangle(\text{box}(\sigma, N_1), \text{box}(\sigma, N_2))$ \hspace{1cm} (R3)
4. $\mu x. \triangle(N_1, N_2) \rightarrow \triangle(\mu x. N_1, \mu x. N_2)$ \hspace{1cm} (R4)
5. $\mu x. N \rightarrow N$ if $x \notin \text{FV}(N)$ \hspace{1cm} (R5)
6. $\mu x. \text{box}(\sigma, x) \rightarrow \text{src}(\text{fix}(\sigma))$ \hspace{1cm} (R6)
7. $\triangle(\text{src}(k_1), \text{src}(k_2)) \rightarrow \text{src}(\text{min}(k_1, k_2))$ \hspace{1cm} (R7)
8. $\text{box}(\sigma, \text{src}(k)) \rightarrow \text{src}(\beta_{\sigma}(k))$ \hspace{1cm} (R8)
9. $\mu x. x \rightarrow \text{src}(0)$ \hspace{1cm} (R9)
Properties of Net Reduction

**Theorem**

- $\rightarrow_R$ is production preserving:
  
  $N \rightarrow_R N' \implies \pi(N) = \pi(N')$.

- $\rightarrow_R$ is confluent and terminating.

- Every closed net normalises to a source, its unique $\rightarrow_R$-normal form.

- For every rational net $N$, the $\rightarrow_R$-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 $\mathcal{I}$:

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

- 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 quantitatively on the values of stream elements and data parameters).
Thanks for your attention!