Proving Productivity, part 2

extended formats, variants of productivity, and complexity

Jörg Endrullis   Clemens Grabmayer   Dimitri Hendriks

Vrije Amsterdam Amsterdam — Universiteit Utrecht — Vrije Universiteit Amsterdam

The End of ***Infinity***
VU Amsterdam, December 15, 2009

Overview

- Extended stream formats
  - for a special class of stream functions
    (simulation by open pebbleflow nets)
  - for larger classes of stream specifications
    (using data-oblivious productivity)
- Productivity and variant definitions in TRSs
- Complexity of productivity and its variants

Overview

Overview

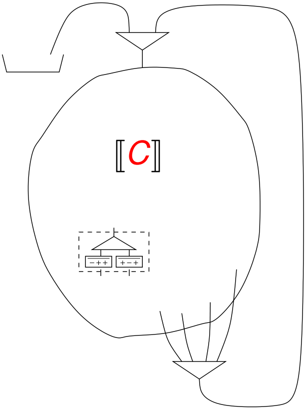
Deciding productivity via pebbleflow

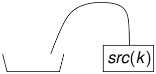
$$\mathsf{M} \to C[\mathsf{M}]$$

$$\dots$$

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

$$\dots$$





(PSF)
pure stream constant spec $\longmapsto$ pebbleflow net $\longmapsto$ pebble source

translation — computation of production

Pure stream constant specification

Example

$$\mathsf{M} \to \mathsf{zip}(0 : \mathsf{M}, \mathsf{M})$$

stream layer

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

data layer

Suppose that $\mathsf{nats} \twoheadrightarrow 0 : 1 : 2 : \ldots$. Then it holds:

$$\mathsf{f}(\mathsf{nats}) \twoheadrightarrow 0 : 0 : 1 : 0 : 2 : 1 : 3 : 0 : 4 : 2 : 5 : 1 : 6 : 3 : 7 : 0 : 8 : \ldots =: \mathsf{a}$$

For all n: $\mathsf{a}(2n) = n$, $\mathsf{a}(2n+1) = \mathsf{a}(n)$ (Sequence A025480).

stream function specification

Example

$$\mathsf{f}(\sigma) \to \mathsf{zip}(\sigma, \mathsf{f}(\sigma))$$

stream layer

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

data layer

Suppose that $\mathsf{nats} \twoheadrightarrow 0 : 1 : 2 : \dots$. Then it holds:

$$\mathsf{f}(\mathsf{nats}) \twoheadrightarrow 0 : 0 : 1 : 0 : 2 : 1 : 3 : 0 : 4 : 2 : 5 : 1 : 6 : 3 : 7 : 0 : 8 : \dots =: \mathsf{a}$$

For all n: $\mathsf{a}(2n) = n$, $\mathsf{a}(2n + 1) = \mathsf{a}(n)$ (Sequence A025480).

stream function specification

Example

$$\mathsf{f}(\sigma) \to \mathsf{zip}(\sigma, \mathsf{f}(\sigma))$$

stream layer

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

data layer

Suppose that $\mathsf{nats} \twoheadrightarrow 0 : 1 : 2 : \ldots$. Then it holds:

$$\mathsf{f}(\mathsf{nats}) \twoheadrightarrow 0 : 0 : 1 : 0 : 2 : 1 : 3 : 0 : 4 : 2 : 5 : 1 : 6 : 3 : 7 : 0 : 8 : \ldots =: \mathsf{a}$$

For all n: $\mathsf{a}(2n) = n$, $\mathsf{a}(2n+1) = \mathsf{a}(n)$ (Sequence A025480).

stream function specification

Example

$$\mathsf{f}(\sigma) \to \mathsf{zip}(\sigma, \mathsf{f}(\sigma))$$

stream layer

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

data layer

Suppose that $\mathsf{nats} \twoheadrightarrow 0 : 1 : 2 : \ldots$. Then it holds:

$$\mathsf{f}(\mathsf{nats}) \twoheadrightarrow 0 : 0 : 1 : 0 : 2 : 1 : 3 : 0 : 4 : 2 : 5 : 1 : 6 : 3 : 7 : 0 : 8 : \ldots =: \mathsf{a}$$

For all n: $\mathsf{a}(2n) = n$, $\mathsf{a}(2n+1) = \mathsf{a}(n)$ (Sequence A025480).

stream function specification

Example

$$\mathsf{f}(\sigma) \to \mathsf{zip}(\sigma, \mathsf{f}(\sigma))$$

stream layer

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

data layer

Suppose that $\mathsf{nats} \twoheadrightarrow 0 : 1 : 2 : \ldots$. Then it holds:

$$\mathsf{f}(\mathsf{nats}) \twoheadrightarrow 0 : 0 : 1 : 0 : 2 : 1 : 3 : 0 : 4 : 2 : 5 : 1 : 6 : 3 : 7 : 0 : 8 : \ldots =: \mathsf{a}$$

For all n: $\mathsf{a}(2n) = n$, $\mathsf{a}(2n+1) = \mathsf{a}(n)$ (Sequence A025480).

stream function specification

Example

$$\mathsf{f}(\sigma) \to \mathsf{zip}(\sigma, \mathsf{f}(\sigma))$$

stream layer

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

data layer

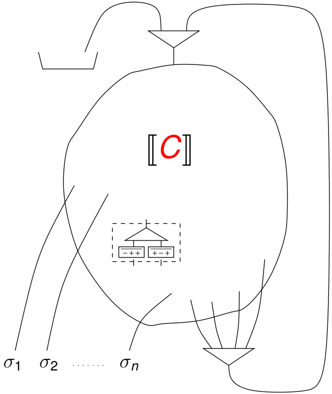
Suppose that $\mathsf{nats} \twoheadrightarrow 0 : 1 : 2 : \ldots$. Then it holds:

$$\mathsf{f}(\mathsf{nats}) \twoheadrightarrow 0 : 0 : 1 : 0 : 2 : 1 : 3 : 0 : 4 : 2 : 5 : 1 : 6 : 3 : 7 : 0 : 8 : \ldots =: \mathsf{a}$$

For all n: $\mathsf{a}(2n) = n$, $\mathsf{a}(2n+1) = \mathsf{a}(n)$ (Sequence A025480).

Recognising productivity via pebbleflow

$$\mathsf{f}(\boldsymbol{\sigma}) \to C[\boldsymbol{\sigma}, \mathsf{f}(\boldsymbol{\sigma})]$$
$$\ldots$$
$$\mathsf{zip}(x : \sigma, \tau) \to \ldots$$
$$\ldots$$

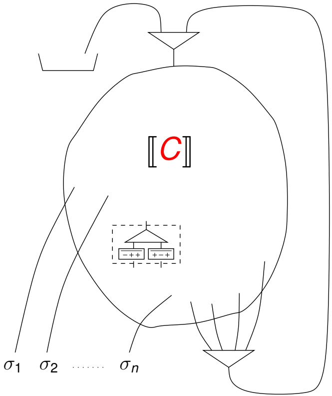


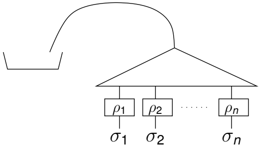
stream function spec $\longmapsto$ open pebbleflow net

translation

Recognising productivity via pebbleflow

$$\mathsf{f}(\boldsymbol{\sigma}) \to C[\boldsymbol{\sigma}, \mathsf{f}(\boldsymbol{\sigma})]$$
$$\dots$$
$$\mathsf{zip}(x : \sigma, \tau) \to \dots$$
$$\dots$$





stream function spec ⟼ (translation) open pebbleflow net ⟼ (computation of throughput) gate

Recognising productivity of some stream functions

Let $\mathcal{S}$ be a specification for a stream function f.

1. Try to transform $\mathcal{S}$ into a stream constant spec in PSF with stream parameters. If unsuccessful, answer: "`sorry, don't know`".
2. Build the corresponding open pebbleflow net.
3. Collapse the pebbleflow net into a gate γ (*ProPro*-extension by Niels Rademaker using the I/O-list infimum operation).
4. If either of the I/O-lists in the gate γ is finite, answer: "$\mathcal{S}$ `is not productive for` f"; else "$\mathcal{S}$ `is productive for` f".

Recognising productivity of some stream functions

Let $\mathcal{S}$ be a specification for a stream function f.

1. Try to transform $\mathcal{S}$ into a stream constant spec in PSF with stream parameters. If unsuccessful, answer: "`sorry, don't know`".
2. Build the corresponding open pebbleflow net.
3. Collapse the pebbleflow net into a gate γ (*ProPro*-extension by Niels Rademaker using the I/O-list infimum operation).
4. If either of the I/O-lists in the gate γ is finite, answer: "$\mathcal{S}$ `is not productive for` f"; else "$\mathcal{S}$ `is productive for` f".

Recognising productivity of some stream functions

$$\mathsf{f}(\sigma_1,\sigma_2) \to \mathsf{zip}(\sigma_1,\mathsf{zip}(\sigma_2,\mathsf{g}(\sigma_1)))$$
$$\mathsf{g}(\sigma_1) \to \mathsf{zip}(\mathsf{even}(\underline{\mathsf{f}(\sigma_1,\sigma_1)}),\mathsf{g}(\sigma_1))$$

By introducing a new stream function $\mathsf{f}_1(\sigma_1) := \mathsf{f}(\sigma_1,\sigma_1)$, we obtain:

$$\mathsf{f}(\sigma_1,\sigma_2) \to \mathsf{zip}(\sigma_1,\mathsf{zip}(\sigma_2,\mathsf{g}(\sigma_1)))$$
$$\mathsf{f}_1(\sigma_1) \to \mathsf{zip}(\sigma_1,\mathsf{zip}(\sigma_1,\mathsf{g}(\sigma_1)))$$
$$\mathsf{g}(\sigma_1) \to \mathsf{zip}(\mathsf{even}(\underline{\mathsf{f}_1(\sigma_1)}),\mathsf{g}(\sigma_1))$$

Now, by letting $\mathsf{M} := \mathsf{f}(\sigma_1,\sigma_2)$, $\mathsf{M}_1 := \mathsf{f}_1(\sigma_1)$, $\mathsf{N} := \mathsf{g}(\sigma_1)$, we obtain:

$$\mathsf{M} \to \mathsf{zip}(\sigma_1,\mathsf{zip}(\sigma_2,\mathsf{N}))$$
$$\mathsf{M}_1 \to \mathsf{zip}(\sigma_1,\mathsf{zip}(\sigma_1,\mathsf{N}))$$
$$\mathsf{N} \to \mathsf{zip}(\mathsf{even}(\mathsf{M}_1),\mathsf{N})$$

Recognising productivity of some stream functions

$$\begin{aligned} f(\sigma_1, \sigma_2) &\to zip(\sigma_1, zip(\sigma_2, g(\sigma_1))) \\ g(\sigma_1) &\to zip(even(\underline{f(\sigma_1, \sigma_1)}), g(\sigma_1)) \end{aligned}$$

By introducing a new stream function $f_1(\sigma_1) := f(\sigma_1, \sigma_1)$, we obtain:

$$\begin{aligned} f(\sigma_1, \sigma_2) &\to zip(\sigma_1, zip(\sigma_2, g(\sigma_1))) \\ f_1(\sigma_1) &\to zip(\sigma_1, zip(\sigma_1, g(\sigma_1))) \\ g(\sigma_1) &\to zip(even(\underline{f_1(\sigma_1)}), g(\sigma_1)) \end{aligned}$$

Now, by letting $M := f(\sigma_1, \sigma_2)$, $M_1 := f_1(\sigma_1)$, $N := g(\sigma_1)$, we obtain:

$$\begin{aligned} M &\to zip(\sigma_1, zip(\sigma_2, N)) \\ M_1 &\to zip(\sigma_1, zip(\sigma_1, N)) \\ N &\to zip(even(M_1), N) \end{aligned}$$

Recognising productivity of some stream functions

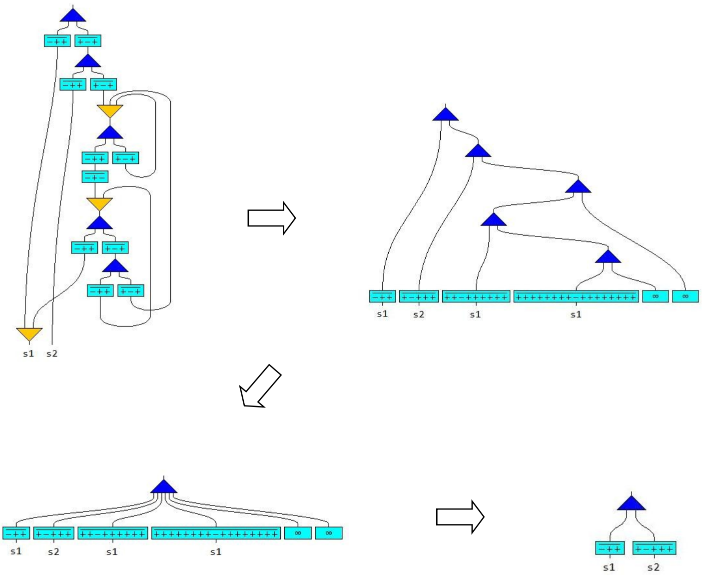
$$\begin{aligned} \mathsf{f}(\sigma_1, \sigma_2) &\to \mathsf{zip}(\sigma_1, \mathsf{zip}(\sigma_2, \mathsf{g}(\sigma_1))) \\ \mathsf{g}(\sigma_1) &\to \mathsf{zip}(\mathsf{even}(\underline{\mathsf{f}(\sigma_1, \sigma_1)}), \mathsf{g}(\sigma_1)) \end{aligned}$$

By introducing a new stream function $\mathsf{f}_1(\sigma_1) := \mathsf{f}(\sigma_1, \sigma_1)$, we obtain:

$$\begin{aligned} \mathsf{f}(\sigma_1, \sigma_2) &\to \mathsf{zip}(\sigma_1, \mathsf{zip}(\sigma_2, \mathsf{g}(\sigma_1))) \\ \mathsf{f}_1(\sigma_1) &\to \mathsf{zip}(\sigma_1, \mathsf{zip}(\sigma_1, \mathsf{g}(\sigma_1))) \\ \mathsf{g}(\sigma_1) &\to \mathsf{zip}(\mathsf{even}(\underline{\mathsf{f}_1(\sigma_1)}), \mathsf{g}(\sigma_1)) \end{aligned}$$

Now, by letting $\mathsf{M} := \mathsf{f}(\sigma_1, \sigma_2)$, $\mathsf{M}_1 := \mathsf{f}_1(\sigma_1)$, $\mathsf{N} := \mathsf{g}(\sigma_1)$, we obtain:

$$\begin{aligned} \mathsf{M} &\to \mathsf{zip}(\sigma_1, \mathsf{zip}(\sigma_2, \mathsf{N})) \\ \mathsf{M}_1 &\to \mathsf{zip}(\sigma_1, \mathsf{zip}(\sigma_1, \mathsf{N})) \\ \mathsf{N} &\to \mathsf{zip}(\mathsf{even}(\mathsf{M}_1), \mathsf{N}) \end{aligned}$$


−++
+−+
−+−
s1
s2
+−+++
++−++++++
++++++++−++++++++
∞

How special is this class of stream functions?

Very restrictive. Their defining rules are of the form (simplified):

$$\mathsf{f}_1(\sigma) \to C_1[\sigma, \mathsf{f}_1(\sigma), \ldots, \mathsf{f}_n(\sigma)]$$
$$\cdots$$
$$\mathsf{f}_n(\sigma) \to C_n[\sigma, \mathsf{f}_1(\sigma), \ldots, \mathsf{f}_n(\sigma)]$$

where $C_1, \ldots, C_n$ are stream contexts consisting of pure stream functions (like zip, even, ...).

In their defining rules:

- no consumption of data-elements from stream parameters
- consequently also no additional supply of consumed

Overview

Extending PSF

Example (poor man's pat-mat)

$$\begin{array}{rl}
\mathsf{T} \to 0 : 1 : \mathsf{f}(\mathsf{tail}(\mathsf{T})) & \\
\underline{\mathsf{f}}(0 : \sigma) \to 0 : 1 : \mathsf{f}(\sigma) & \textit{stream layer} \\
\underline{\mathsf{f}}(1 : \sigma) \to 1 : 0 : \mathsf{f}(\sigma) & \\
\mathsf{tail}(x : \sigma) \to \sigma & \\
\hline
& \textit{data layer} \\
\hline
\end{array}$$

is a productive stream definition of the Thue–Morse stream:

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

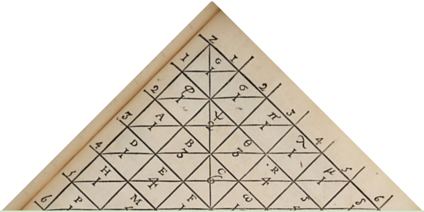
Extending PSF

- In extended-pure specifications, the rules for stream functions allow:
  - a restricted form of exhaustive pattern matching
  - duplication of stream variables
    $$f(\sigma) \to g(\sigma, \sigma).$$
  - additional supply in stream variables is allowed
    $$\mathsf{diff}(x : y : \sigma) \to \mathsf{xor}(x, y) : \mathsf{diff}(y : \sigma)$$
  - use of non-productive stream functions
    $$\mathsf{onlyread2}(x : y : \sigma) \to x : y : \mathsf{idle}(\sigma) \qquad \mathsf{idle}(\sigma) \to \mathsf{idle}(\sigma)$$
- In flat specifications, additional feature:
  - exhaustive pattern matching on constructors

Extending PSF

- In extended-pure specifications, the rules for stream functions allow:
  - a restricted form of exhaustive pattern matching
  - duplication of stream variables
    $\mathsf{f}(\sigma) \to \mathsf{g}(\sigma, \sigma)$.
  - additional supply in stream variables is allowed
    $\mathsf{diff}(x : y : \sigma) \to \mathsf{xor}(x, y) : \mathsf{diff}(y : \sigma)$
  - use of non-productive stream functions
    $\mathsf{onlyread2}(x : y : \sigma) \to x : y : \mathsf{idle}(\sigma) \qquad \mathsf{idle}(\sigma) \to \mathsf{idle}(\sigma)$

- In flat specifications, additional feature:
  - exhaustive pattern matching on constructors



1
1 1
1 2 1
1 3 3 1
1 4 6 4 1

Example (Pascal's triangle)

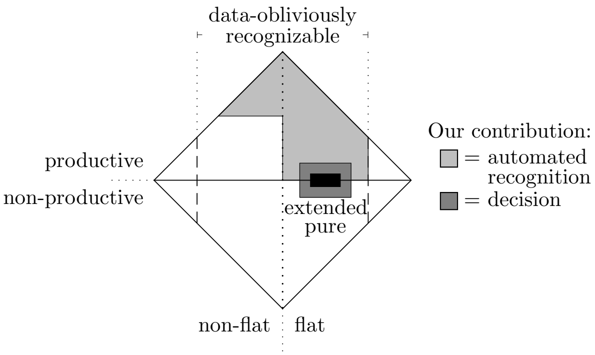
$$\begin{array}{rcl|l}
\mathsf{P} & \to & \mathsf{0} : \mathsf{s}(\mathsf{0}) : \mathsf{g}(\mathsf{P}) & \\
\mathsf{g}(\underline{\mathsf{s}(x)} : \underline{y} : \sigma) & \to & \mathsf{a}(\mathsf{s}(x), y) : \mathsf{g}(y : \sigma) & \textit{stream layer} \\
\mathsf{g}(\underline{\mathsf{0}} : \sigma) & \to & \mathsf{0} : \mathsf{s}(\mathsf{0}) : \mathsf{g}(\sigma) & \\
\hline
\mathsf{a}(x, \mathsf{s}(y)) & \to & \mathsf{s}(\mathsf{a}(x, y)) & \textit{data layer} \\
\mathsf{a}(x, \mathsf{0}) & \to & x &
\end{array}$$

is a productive stream specification of the Pascal's triangle:

$$\mathsf{P} \twoheadrightarrow \mathsf{0} : \mathsf{1} : \mathsf{0} : \mathsf{1} : \mathsf{1} : \mathsf{0} : \mathsf{1} : \mathsf{2} : \mathsf{1} : \mathsf{0} : \mathsf{1} : \mathsf{3} : \mathsf{3} : \mathsf{1} : \mathsf{0} : \ldots$$

New concepts and definitions

- stream specification formats: ext. pure $\subsetneq$ flat $\subsetneq$ friendly-nesting;
- data-oblivious rewriting;
- data-oblivious productivity.



Data-Oblivious Analysis

Example (Pascal's triangle)

$$\mathsf{P} \to 0 : \mathsf{s}(0) : \mathsf{g}(\mathsf{P})$$
$$\mathsf{g}(\mathsf{s}(x) : y : \sigma) \to \mathsf{a}(\mathsf{s}(x), y) : \mathsf{g}(y : \sigma)$$
$$\mathsf{g}(0 : \sigma) \to 0 : \mathsf{s}(0) : \mathsf{g}(\sigma)$$

data abstracted we have:

$$\mathsf{P}' \to \bullet : \bullet : \mathsf{g}(\mathsf{P}')$$
$$\mathsf{g}(\bullet : \bullet : \sigma) \to \bullet : \mathsf{g}(\bullet : \sigma)$$
$$\mathsf{g}(\bullet : \sigma) \to \bullet : \bullet : \mathsf{g}(\sigma)$$

The data oblivious lower/upper bounds on the production of g are:

$$n \mapsto n \dot{-} 1 \; / \; n \mapsto 2n$$

The lower bound implies productivity of P′ follows; we say: P is data-obliviously productive. This implies productivity of P.

Data-Oblivious Productivity

$\Pi_{\mathcal{S}}(t) := \sup\{n \in \overline{\mathbb{N}} \mid t \twoheadrightarrow s_1 : \ldots : s_n : r\}$ data-aware production of t.

Definition

The data-oblivious production range ($\subseteq \overline{\mathbb{N}}$) of a term t:

$\underline{\overline{do}}_{\mathcal{S}}(t)$:= set of all productions of t under outermost-fair data-oblivious rewrite sequences starting at t

The d-o lower/upper bounds:

$$\underline{do}_{\mathcal{S}}(t) := \inf \underline{\overline{do}}_{\mathcal{S}}(t) \qquad \overline{do}_{\mathcal{S}}(t) := \sup \underline{\overline{do}}_{\mathcal{S}}(t)$$

A term t is data-obliviously productive if $\underline{do}_{\mathcal{S}}(t) = \infty$.

Proposition (Data-oblivious productivity implies productivity)

$$\underline{do}_{\mathcal{S}}(t) \leq \Pi_{\mathcal{S}}(t) \leq \overline{do}_{\mathcal{S}}(t)$$

Stream specifications

For stream specifications we consider:

- $\{S, D\}$-sorted, orthogonal, constructor TRSs $R = \langle \Sigma, R \rangle$
- Σ_S stream symbols and Σ_D data symbols

Definition (Stream Specification)

R_S	*stream layer*
R_D	*data layer*

1. $\mathsf{M}_0 \in \Sigma_S$ with arity 0, the root of R.
2. $\langle \Sigma_D, R_D \rangle$ is a terminating, D-sorted TRS, the data layer of R.
3. R is exhaustive

Flat stream spec's

R is called flat: in rules for stream functions, no nested occurrences of stream function rules on their right hand sides.

Theorem

For flat stream spec's we can decide data-oblivious productivity.

Extended-pure stream spec's

R is called extended-pure: the defining rules for a stream function all have the same data abstraction.

Example

$$\begin{aligned}\mathsf{inv}(0:\sigma) &\to 1:\mathsf{inv}(\sigma)\\ \mathsf{inv}(1:\sigma) &\to 0:\mathsf{inv}(\sigma)\\ \mathsf{inv}(\bullet:\sigma) &\to \bullet:\mathsf{inv}(\sigma)\end{aligned}$$

Non-example:

$$\begin{aligned}\mathsf{g}(0:x:\sigma) &\to x:x:\mathsf{g}(\sigma)\\ \mathsf{g}(1:x:\sigma) &\to x:\mathsf{g}(\sigma)\\ \mathsf{g}(\bullet:\bullet:\sigma) &\to \bullet:\bullet:\mathsf{g}(\sigma)\\ \mathsf{g}(\bullet:\bullet:\sigma) &\to \bullet:\mathsf{g}(\sigma)\ .\end{aligned}$$

Proposition

For pure stream spec's: productivity = data-oblivious productivity.

Theorem

We can decide productivity of extended-pure stream specifications.

Stream specification (friendly-nesting)

The convolution product $\times$ is the stream operation $\times : \mathbb{R}^\omega \times \mathbb{R}^\omega \to \mathbb{R}^\omega$:

$$(\sigma \times \tau)(i) = \sum_{j=0}^{i} \sigma(j) \cdot \tau(i-j) \qquad \text{(for all } i \in \mathbb{N}\text{)}$$

Hence: $(x : \sigma') \times (y : \tau') = (x.y) : (x \cdot \tau' + \sigma' \times (y : \tau'))$.

Example

$$\begin{aligned}
\mathsf{nats} &\to 0 : \times(\mathsf{ones}, \mathsf{ones}) \\
\mathsf{ones} &\to \mathsf{s}(0) : \mathsf{ones} \\
\times(x : \sigma', y : \tau') &\to \mathsf{m}(x, y) : \mathsf{add}(\mathsf{times}(\tau', x), \times(\sigma', y : \tau')) \\
\mathsf{times}(x : \sigma', y) &\to \mathsf{m}(x, y) : \mathsf{times}(\sigma', y) \\
\mathsf{add}(x : \sigma', y : \tau') &\to \mathsf{a}(x, y) : \mathsf{add}(\sigma', \tau')
\end{aligned}$$

stream layer

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

data layer

Stream specification (friendly-nesting)

The convolution product $\times$ is the stream operation $\times : \mathbb{R}^\omega \times \mathbb{R}^\omega \to \mathbb{R}^\omega$:

$$(\sigma \times \tau)(0) := \sigma(0).\tau(0)$$
$$(\sigma \times \tau)' := \sigma(0) \cdot \tau' + \sigma' \times \tau$$

Hence: $(x : \sigma') \times (y : \tau') = (x.y) : (x \cdot \tau' + \sigma' \times (y : \tau'))$.

Example

$$\mathsf{nats} \to 0 : \times(\mathsf{ones}, \mathsf{ones})$$
$$\mathsf{ones} \to \mathsf{s}(0) : \mathsf{ones}$$
$$\times(x : \sigma', y : \tau') \to \mathsf{m}(x, y) : \mathsf{add}(\mathsf{times}(\tau', x), \times(\sigma', y : \tau'))$$
$$\mathsf{times}(x : \sigma', y) \to \mathsf{m}(x, y) : \mathsf{times}(\sigma', y)$$
$$\mathsf{add}(x : \sigma', y : \tau') \to \mathsf{a}(x, y) : \mathsf{add}(\sigma', \tau')$$

stream layer

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

data layer

Stream specification (friendly-nesting)

The convolution product $\times$ is the stream operation $\times : \mathbb{R}^\omega \times \mathbb{R}^\omega \to \mathbb{R}^\omega$:

$$\begin{aligned} (\sigma \times \tau)(0) &:= \sigma(0).\tau(0) \\ (\sigma \times \tau)' &:= \sigma(0) \cdot \tau' + \sigma' \times \tau \end{aligned}$$

Hence: $(x : \sigma') \times (y : \tau') = (x.y) : (x \cdot \tau' + \sigma' \times (y : \tau'))$.

Example

$$\begin{aligned} \mathsf{nats} &\to 0 : \times(\mathsf{ones}, \mathsf{ones}) \\ \mathsf{ones} &\to \mathsf{s}(0) : \mathsf{ones} \\ \times(x : \sigma', y : \tau') &\to \mathsf{m}(x, y) : \mathsf{add}(\mathsf{times}(\tau', x), \times(\sigma', y : \tau')) \\ \mathsf{times}(x : \sigma', y) &\to \mathsf{m}(x, y) : \mathsf{times}(\sigma', y) \\ \mathsf{add}(x : \sigma', y : \tau') &\to \mathsf{a}(x, y) : \mathsf{add}(\sigma', \tau') \end{aligned}$$

stream layer

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

data layer

Friendly-nesting stream spec's

Friendly-nesting stream specifications are extensions of flat ones with friendly (nesting) rules γ:

- γ consumes in each argument at most one stream element,
- it produces at least one stream element, and
- the defining rules of stream function symbols on the right hand side are friendly again.

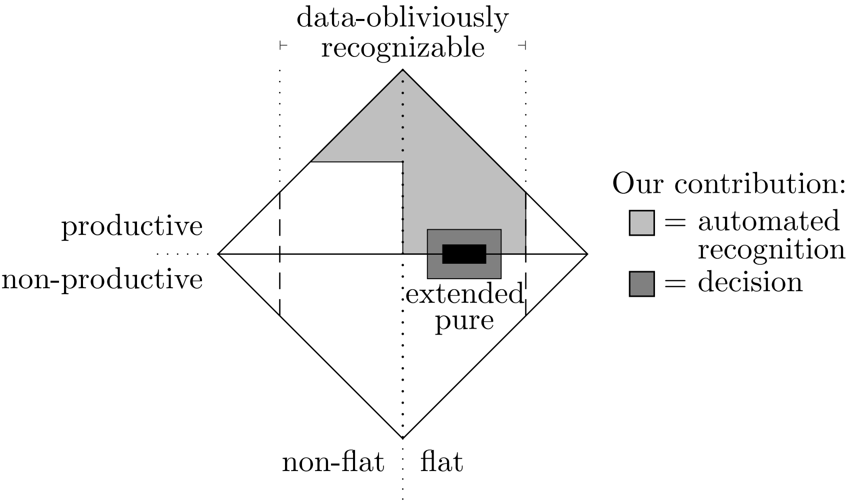
Example

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

Theorem (For friendly nesting stream specifications . . .)

. . . we have a sufficient condition for (data-oblivious) productivity.

Map of stream specifications



Overview

1. Extended stream formats
2. Variants of Productivity
3. Computational Complexity

Productivity and variants

1 $$\mathsf{zeros} \to 0 : \mathsf{zeros}$$

- productive: there is only one maximal rewrite sequence:
  $\mathsf{zeros} \to 0 : \mathsf{zeros} \to 0 : 0 : \mathsf{zeros} \to \ldots \twoheadrightarrow 0 : 0 : 0 : \ldots$

2 $$\mathsf{zeros} \to 0 : \mathsf{id}(\mathsf{zeros}) \qquad \mathsf{id}(\sigma) \to \sigma$$

- $\mathsf{zeros} \twoheadrightarrow 0 : \mathsf{id}(0 : \mathsf{id}(0 : \mathsf{id}(\ldots)))$
- still productive, since for all max. outermost-fair rewrite sequences:
  $\mathsf{zeros} \twoheadrightarrow 0 : 0 : 0 : \ldots$

Even for well-behaved spec's (orthogonal TRSs), productivity should be based on a fair treatment of outermost redexes.

Productivity and variants

1 $\mathsf{zeros} \to 0 : \mathsf{zeros}$

- productive: there is only one maximal rewrite sequence:
  $\mathsf{zeros} \to 0 : \mathsf{zeros} \to 0 : 0 : \mathsf{zeros} \to \ldots \twoheadrightarrow 0 : 0 : 0 : \ldots$

2 $\mathsf{zeros} \to 0 : \mathsf{id}(\mathsf{zeros}) \qquad \mathsf{id}(\sigma) \to \sigma$

- $\mathsf{zeros} \twoheadrightarrow 0 : \mathsf{id}(0 : \mathsf{id}(0 : \mathsf{id}(\ldots)))$
- still productive, since for all max. outermost-fair rewrite sequences:
  $\mathsf{zeros} \twoheadrightarrow 0 : 0 : 0 : \ldots$

Even for well-behaved spec's (orthogonal TRSs), productivity should be based on a fair treatment of outermost redexes.

Productivity and variants

1 $$\mathsf{zeros} \to 0 : \mathsf{zeros}$$

- productive: there is only one maximal rewrite sequence:
  $\mathsf{zeros} \to 0 : \mathsf{zeros} \to 0 : 0 : \mathsf{zeros} \to \ldots \twoheadrightarrow 0 : 0 : 0 : \ldots$

2 $$\mathsf{zeros} \to 0 : \mathsf{id}(\mathsf{zeros}) \qquad\qquad \mathsf{id}(\sigma) \to \sigma$$

- $\mathsf{zeros} \twoheadrightarrow 0 : \mathsf{id}(0 : \mathsf{id}(0 : \mathsf{id}(\ldots)))$
- still productive, since for all max. outermost-fair rewrite sequences:
  $\mathsf{zeros} \twoheadrightarrow 0 : 0 : 0 : \ldots$

Even for well-behaved spec's (orthogonal TRSs), productivity should be based on a fair treatment of outermost redexes.

Productivity and variants

3 $\mathsf{maybe} \to 0 : \mathsf{maybe}$ $\mathsf{maybe} \to \mathsf{sink}$ $\mathsf{sink} \to \mathsf{sink}$

- productive or not, dependent on the chosen strategy
- 'weakly productive': $\mathsf{maybe} \twoheadrightarrow 0 : 0 : 0 : \ldots$
- not 'strongly productive': e.g. $\mathsf{maybe} \to \mathsf{sink} \to \mathsf{sink} \to \ldots$

4 $\mathsf{bitstream} \to 0 : \mathsf{bitstream}$ $\mathsf{bitstream} \to 1 : \mathsf{bitstream}$

- productive independent of the strategy chosen
- 'weakly' and 'strongly productive'
- infinite normal forms not unique

Productivity and variants

3 $\mathsf{maybe} \to 0 : \mathsf{maybe} \qquad \mathsf{maybe} \to \mathsf{sink} \qquad \mathsf{sink} \to \mathsf{sink}$

- productive or not, dependent on the chosen strategy
- 'weakly productive': $\mathsf{maybe} \twoheadrightarrow 0 : 0 : 0 : \ldots$
- not 'strongly productive': e.g. $\mathsf{maybe} \to \mathsf{sink} \to \mathsf{sink} \to \ldots$

4 $\mathsf{bitstream} \to 0 : \mathsf{bitstream} \qquad \mathsf{bitstream} \to 1 : \mathsf{bitstream}$

- productive independent of the strategy chosen
- 'weakly' and 'strongly productive'
- infinite normal forms not unique

Productivity w.r.t. computable strategies

Let R be a TRS.
A strategy for a rewrite relation $\to_R$ is a relation $\leadsto \subseteq \to_R$ with the same normal forms as $\to_R$.

Definition

A term t is called productive w.r.t. a strategy $\leadsto$ if all maximal $\leadsto$-rewrite sequences starting from t end in a constructor normal form.

Strong and weak productivity

Definition

A term *t* in a TRS *R* is called

- strongly productive: all maximal outermost-fair rewrite sequences starting from *t* end in a constructor normal form.
- weakly productive: if there exists a rewrite sequence starting from *t* that ends in a constructor normal form.

Definition of productivity in general TRSs

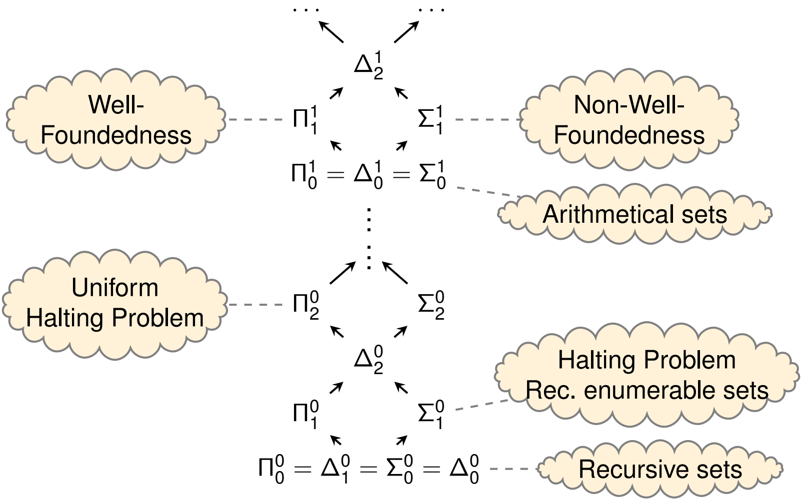
We think:

- For non-well-behaved spec's (non-orthogonal TRSs), productivity has to be defined relative to a given rewrite strategy.
- Strategy-independent variants (strong, weak productivity) are of limited general interest.
- Uniqueness of (infinite) normal form UN^{∞} should be considered to be a separate property, independent of productivity. (In orthogonal TRSs, UN^{∞} is guaranteed.)

Overview

1. Extended stream formats
2. Variants of Productivity
3. Computational Complexity

The arithmetical and analytical hierarchies



Productivity w.r.t. computable strategies

PRODUCTIVITY PROBLEM w.r.t. a family $\mathcal{S}$ of computable strategies

Instance: Encodings of a finite TRS R, a strategy $\leadsto \in \mathcal{S}(R)$, and a term t in R.

Question: Is t productive w.r.t. $\leadsto$?

We say that:

- such a family $\mathcal{S}$ is admissible: if R is orthogonal, $\mathcal{S}(R) \neq \emptyset$.

Productivity w.r.t. computable strategies

PRODUCTIVITY PROBLEM w.r.t. a family $\mathcal{S}$ of computable strategies

Instance: Encodings of a finite TRS R, a strategy $\leadsto \in \mathcal{S}(R)$, and a term t in R.

Question: Is t productive w.r.t. $\leadsto$?

We say that:

- such a family $\mathcal{S}$ is admissible: if R is orthogonal, $\mathcal{S}(R) \neq \emptyset$.

Productivity w.r.t. computable strategies

Theorem

For every family of admissible, computable strategies $\mathcal{S}$, the productivity problem w.r.t. $\mathcal{S}$ is Π^0_2-complete.

Proof.

Contained in Π^0_2: a term t is productive w.r.t. $\leadsto \in \mathcal{S}(R)$ iff

$$\left.\begin{array}{l}\forall d \in \mathbb{N}.\ \exists n \in \mathbb{N}.\ \text{every } n\text{-step} \leadsto\text{-reduct of } t \\ \qquad\qquad \text{is a constructor normal form up to depth } d\end{array}\right\} \in \Pi^0_2$$

Π^0_2-complete: By reducing the totality problem for Turing-machines, which is Π^0_2-complete, to the productivity problem here. □

Corollary

In orthogonal TRSs, productivity w.r.t. lazy (outermost-fair) evaluation is Π^0_2-complete.

Productivity w.r.t. computable strategies

Theorem

For every family of admissible, computable strategies $\mathcal{S}$, the productivity problem w.r.t. $\mathcal{S}$ is Π^0_2-complete.

Proof.

Contained in Π^0_2: a term t is productive w.r.t. $\rightsquigarrow \in \mathcal{S}(R)$ iff

$$\left.\begin{array}{l}\forall d \in \mathbb{N}.\ \exists n \in \mathbb{N}.\ \text{every } n\text{-step } \rightsquigarrow\text{-reduct of } t \\ \qquad\qquad \text{is a constructor normal form up to depth } d\end{array}\right\} \in \Pi^0_2$$

Π^0_2-complete: By reducing the totality problem for Turing-machines, which is Π^0_2-complete, to the productivity problem here. □

Corollary

In orthogonal TRSs, productivity w.r.t. lazy (outermost-fair) evaluation is Π^0_2-complete.

Productivity w.r.t. computable strategies

Theorem

For every family of admissible, computable strategies $\mathcal{S}$, the productivity problem w.r.t. $\mathcal{S}$ is Π^0_2-complete.

Proof.

Contained in Π^0_2: a term t is productive w.r.t. $\leadsto \in \mathcal{S}(R)$ iff

$$\left.\begin{array}{l}\forall d \in \mathbb{N}.\ \exists n \in \mathbb{N}.\ \text{every } n\text{-step} \leadsto\text{-reduct of } t \\ \qquad\qquad\qquad\quad \text{is a constructor normal form up to depth } d\end{array}\right\} \in \mathbf{\Pi^0_2}$$

Π^0_2-complete: By reducing the totality problem for Turing-machines, which is Π^0_2-complete, to the productivity problem here. □

Corollary

In orthogonal TRSs, productivity w.r.t. lazy (outermost-fair) evaluation is Π^0_2-complete.

Strong and weak productivity

Theorem

The recognition problem for

- *strong productivity is Π^1_1-complete;*
- *weak productivity is Σ^1_1-complete.*

Proof (Idea).

Π^1_1-hardness (Σ^1_1-hardness): reducing the
– recognition problem for well-founded (for non-well-founded) binary relations over $\mathbb{N}$, which is Π^1_1-complete (Σ^1_1-complete), to the
– to the recognition problem of strong (weak) productivity. □

Strong and weak productivity

Theorem

The recognition problem for

- *strong productivity is Π^1_1-complete;*
- *weak productivity is Σ^1_1-complete.*

Proof (Idea).

Π^1_1-hardness (Σ^1_1-hardness): reducing the
– recognition problem for well-founded (for non-well-founded) binary relations over $\mathbb{N}$, which is Π^1_1-complete (Σ^1_1-complete), to the
– to the recognition problem of strong (weak) productivity. □

Uniqueness of infinite normal form

Theorem

The problem of recognising, for TRSs R and terms t in R, whether t has a unique (finite or infinite) normal form is Π^1_1-complete.

Changes due to adding the condition uniqueness of normal form:

(i) w.r.t. family of strategies:
- uniqueness of normal forms w.r.t. $\rightsquigarrow$: Π^0_2-complete.
- uniqueness of normal forms generally: Π^1_1-complete.

(ii) strong productivity: Π^1_1-complete

(iii) weak productivity: now $(\Pi^1_1 \cup \Sigma^1_1)$-hard

Uniqueness of infinite normal form

Theorem

The problem of recognising, for TRSs R and terms t in R, whether t has a unique (finite or infinite) normal form is Π^1_1-complete.

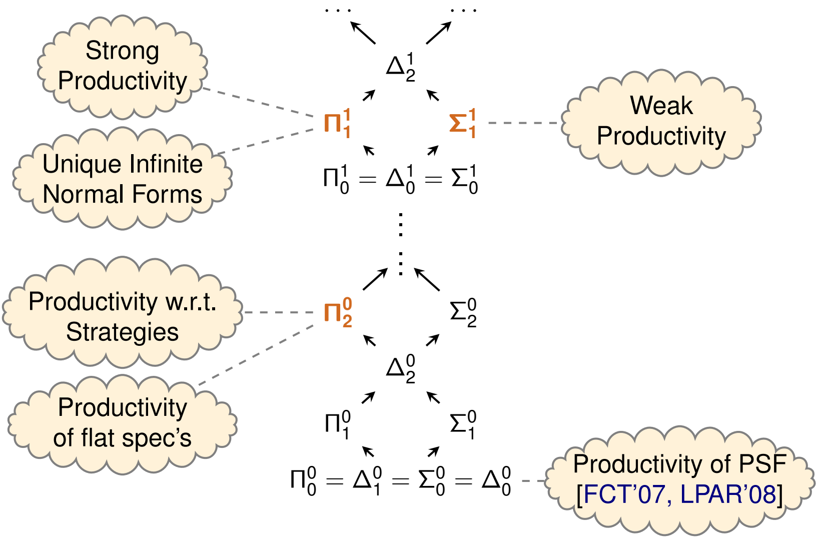
Changes due to adding the condition uniqueness of normal form:

(i) w.r.t. family of strategies:
- uniqueness of normal forms w.r.t. $\rightsquigarrow$: Π^0_2-complete.
- uniqueness of normal forms generally: Π^1_1-complete.

(ii) strong productivity: Π^1_1-complete

(iii) weak productivity: now $(\Pi^1_1 \cup \Sigma^1_1)$-hard

Complexity of productivity



Summary

- Extended stream formats
  - for a special class of stream functions
    (simulation by open pebbleflow nets)
  - for larger classes of stream specifications:
    flat, extended-pure, friendly-nesting
    (using data-oblivious productivity)
- Productivity and variant definitions in TRSs
  - productivity with respect to strategies
  - weak and strong productivity
- Complexity of productivity and its variants

The End of Infinity?

The End of Infinity? Yes, but the idea catches on . . .

Infinity Groep BV, Toetsenbordweg 48, 1033 MZ Amsterdam

Realising Optimal Sharing (ROS)

NWO-Project (2009–2012/13) at Utrecht University linking:

- Dept. of Philosophy (Theor. Philosophy)
- Dept. of Computer Science (Functional Languages)

Aims

- Study optimal-sharing implementations of the λ-calculus
- Try to incorporate optimal-sharing techniques in the Utrecht Haskell Compiler (UHC)

People

- Phil: Vincent van Oostrom (principal investigator), CG (postdoc/3 years)
- CS: Doaitse Swierstra and Atze Dijkstra, Jan Rochel (PhD student)