Data-Oblivious Stream Productivity

Jörg Endrullis Clemens Grabmayer Dimitri Hendriks

Vrije Universiteit – Universiteit Utrecht – Vrije Universiteit

NWO Projects Infinity and ProvCyc

PAM, CWI

May 7, 2008

Outline

Productivity

- 'Productivity' first used by Dijkstra (1980).
- Slogan: *For programming with infinite structures, productivity is what termination is for programming with finite structures.*
- Productivity captures the notion of unlimited progress, of 'working' programs, producing defined values indefinitely.

Related questions:

- 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?

Productivity of Stream Specifications

- $A^\omega := \{\sigma \mid \sigma : \mathbb{N} \to A\}$ the set A^ω of streams over A
- ':' is the stream constructor symbol: $a : \sigma$ denotes the result of prepending $a \in A$ to $\sigma \in A^\omega$
- A recursive stream specification

$$M = \ldots M \ldots$$

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

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

- Productivity is undecidable in general (in fact Π^0_2-complete).
- But for restricted formats computable sufficient conditions or decidability can be obtained.

Examples

Example

$\mathsf{read}(x : \sigma) = x : \mathsf{read}(\sigma)$	
$\mathsf{fast\_read}(x : y : \sigma) = x : y : \mathsf{fast\_read}(\sigma)$	
$\mathsf{fives} = 5 : \mathsf{read}(\mathsf{fives})$	productive
$\mathsf{fives}' = 5 : \mathsf{fast\_read}(\mathsf{fives}')$	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

Productivity Recognition: Previous Approaches

- Wadge (1981): 'cyclic sum test' (limited, computable criterion).
- Sijtsma (1989): mathematical theory of productivity based on 'production moduli' (mathem., not directly computable criteria).
- Coquand (1994): 'guardedness' as a syntactic criterion for productivity (automatable, but restrictive criterion).
- Telford and Turner (1997): extend the notion of guardedness by a method in the flavour of Wadge.
- Hughes, Pareto, and Sabry (1996): introduce a type system for proving productivity (automatable criterion).
- Buchholz (2004): type system for proving productivity, two forms:
  - using unrestricted production moduli (general, not automatable);
  - a decidable subsystem with limited moduli (automable criterion, handles all examples of Telford &Turner).

Our First Paper: Productivity Decision

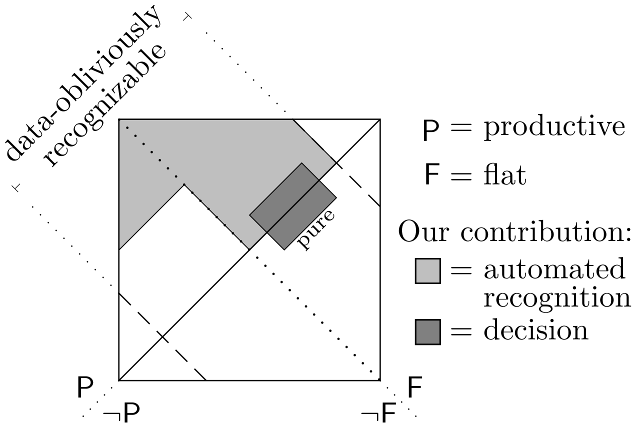
Productivity of Stream Definitions (Endrullis, Grabmayer, Hendriks, Isihara, Klop), FCT'07:

- A decision algorithm for productivity on the large and natural class of pure stream spec's.
- 'Large class': The stream functions allowed in pure stream spec's contain all automatic sequences (Allouche, Shallit).
- Idea behind the decision algorithm:
  - The process of evaluation of a pure stream spec can be modelled by dataflow of pebbles in a finite pebbleflow net.
  - The production of a pebbleflow net associated with a pure stream spec (amount of pebbles the net can produce at its output port) can be calculated by reducing nets to trivial nets.

Main New Results: Productivity Recognition/Decision

- All previous approaches use a 'quantitative' analysis that abstracts away from concrete values of stream elements. We formalise this by data-oblivious rewriting.
- We introduce the notion of data-oblivious productivity.
- We identify two syntactical classes of stream spec's: flat and (general) pure spec's.
- For flat stream spec's we obtain a decision method for data-oblivious productivity, yielding a computable, data-obliviously optimal criterion for productivity.
- For pure stream spec's we obtain a decision method for productivity.

Map of Stream Specifications



Stream Specification

Example

$\mathsf{T} \to 0 : \mathsf{zip}(\mathsf{inv}(\mathsf{T}), \mathsf{tail}(\mathsf{T}))$	*stream layer*
$\mathsf{tail}(x : \sigma) \to \sigma$ $\mathsf{zip}(x : \sigma, \tau) \to x : \mathsf{zip}(\tau, \sigma)$ $\mathsf{inv}(x : \sigma) \to \mathsf{i}(x) : \mathsf{inv}(\sigma)$	*function layer*
$\mathsf{i}(0) \to 1 \qquad \mathsf{i}(1) \to 0$	*data layer*

This is a productive stream definition obtaining the Thue-Morse sequence:

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

Stream Specification

Example

$\mathsf{J} = 0 : 1 : \mathsf{even}(\mathsf{J})$	*stream layer*
$\mathsf{even}(x : \sigma) \to x : \mathsf{odd}(\sigma)$ $\mathsf{odd}(x : \sigma) \to \mathsf{even}(\sigma)$	*function layer*
	data layer

This stream definition is not productive: $\mathsf{J} \twoheadrightarrow\!\!\!\!\to 0 : 1 : 0 : 0 : \mathsf{even}^{\omega}$

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

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

$$\begin{aligned}
\mathsf{even}(\mathsf{J}) &\to \mathsf{even}(0 : 1 : \mathsf{even}(\mathsf{J}))\\
&\to 0 : \mathsf{odd}(1 : \mathsf{even}(\mathsf{J}))\\
&\to 0 : \mathsf{even}(\mathsf{even}(\mathsf{J}))
\end{aligned}$$

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

$$\begin{aligned}
\mathsf{odd}(\mathsf{even}^2(\mathsf{J})) &\twoheadrightarrow \mathsf{odd}(0 : \mathsf{odd}(\mathsf{even}^2(\mathsf{J})))\\
&\to \mathsf{even}(\mathsf{odd}(\mathsf{even}^2(\mathsf{J})))
\end{aligned}$$

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

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

Stream TRS

A stream TRS is a

- finite $\{S, D\}$-sorted, orthogonal, constructor TRS $\langle \Sigma, R \rangle$, with
- signature partition $\Sigma = \Sigma_S \uplus \Sigma_D$ into stream symbols and data symbols, and

For the definition of stream spec's we also assume:

- stream signature partition $\Sigma_S = \{:\} \uplus \Sigma_{str} \uplus \Sigma_{fun}$, where
  - ':' the stream constructor symbol,
  - Σ_{str} a set of stream constant symbols having only data arguments;
  - Σ_{fun} a set of stream function symbols with usually at least one stream argument.

Stream Specification

Definition

Let $\mathcal{T} = \langle \Sigma, R \rangle$ a stream TRS with part's $\Sigma = \Sigma_{str} \uplus \Sigma_{fun} \uplus \{:\} \uplus \Sigma_D$ and $R = R_{str} \uplus R_{fun} \uplus R_D$. $\mathcal{T}$ is a stream specification if:

R_{str}	*stream layer*	
R_{fun}	*function layer*	stream function specification
R_D	*data layer*	

- The data-layer $\mathcal{T}_d = \langle \Sigma_D, R_D \rangle$ is a terminating D-sorted TRS.
- The underlying stream function specification $\mathcal{T}_{fun} = \langle \Sigma_{fun} \uplus \{:\} \uplus \Sigma_D, R_{fun} \uplus R_D \rangle$ is a TRS.
- $\mathcal{T}$ is exhaustive for the defined symbols in Σ.
- Σ_{str}, a set of constant symbols, contains M_0, the root of $\mathcal{T}$. R_{str} is the set of defining rules ρ_{M}: $\mathsf{M} \to s$ for every $\mathsf{M} \in \Sigma_{str}$.

Stream Specification (Layered setup)

Remark

- every layer may use symbols from a lower layer, not vice versa
- isolated data-layer:
  - data-layer symbols are stream independent, excluding stream-dependent functions like:

$$\mathsf{head}(x : \sigma) \to x$$

  - by exhaustivity for Σ_D and strong normalization of $\mathcal{T}_d$, closed data terms rewrite to constructor normal forms

Stream Specification (layered setup)

Remark

- stream layer:
  - only sortedness restrictions are imposed on how the rules in the stream layer make use of the symbols in the other layers.
- separate function layer:
  - we are interested in managable stream functions that define well-defined streams or finite stream prefixes.
    The rule

$$\mathsf{f}(\sigma) \to 0 : \mathsf{head}(\mathsf{tail}(\mathsf{f}(\sigma))) : \mathsf{f}(\sigma) \qquad \text{(excluded!)}$$

    defines an operation on streams that produces an output stream with undefined odd elements.

Production of a term. Productivity of a stream spec.

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

Definition

Let $\mathcal{T} = \langle \Sigma, R \rangle$ a stream definition.

- 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 stream definition $\mathcal{T}$ is productive if and only if

$$\mathsf{M}_0 \twoheadrightarrow\!\!\!\!\!\rightarrow u_1 : u_2 : u_3 : u_4 : \ldots$$

Stream Specifications (Properties I)

A stream spec $\mathcal{T}$ is called

- flat: all rules in the function layer R_{fun} of $\mathcal{T}$ are flat: no nested occurrences of stream function symbols on the right-hand side.

  Excludes a rule: $\mathsf{e}(x : \sigma) \to x : \mathsf{e}(\mathsf{e}(\sigma))$.

- pure: $\mathcal{T}$ is flat, and for every symbol $\mathsf{f} \in \Sigma_{fun}$ the defining rules of f in R_{fun} have the same consumption/production behaviour: they coincide (mod. renaming of variables) if all outermost data-subterms are replaced by $\bullet$.

  This excludes defining rules: $$\begin{aligned}\mathsf{f}(0 : x : \sigma) &\to x : x : \mathsf{f}(0 : \sigma)\\ \mathsf{f}(1 : x : \sigma) &\to x : \mathsf{f}(0 : \sigma)\ .\end{aligned}$$

Stream Specification (flat, non-pure)

Example

$\mathsf{Q} \to \mathsf{a} : \mathsf{R}$ $\mathsf{R} \to \mathsf{b} : \mathsf{c} : \mathsf{f}(\mathsf{R})$	*stream layer*
$\mathsf{f}(\mathsf{a} : \sigma) \to \mathsf{a} : \mathsf{b} : \mathsf{c} : \mathsf{f}(\sigma)$ $\mathsf{f}(\mathsf{b} : \sigma) \to \mathsf{a} : \mathsf{c} : \mathsf{f}(\sigma)$ $\mathsf{f}(\mathsf{c} : \sigma) \to \mathsf{b} : \mathsf{f}(\sigma)$	*function layer*
	data layer

… is productive and specifies the ternary Thue-Morse sequence:

$$\mathsf{Q} \twoheadrightarrow \mathsf{a} : \mathsf{b} : \mathsf{c} : \mathsf{a} : \mathsf{c} : \mathsf{b} : \mathsf{a} : \mathsf{b} : \mathsf{c} : \mathsf{b} : \mathsf{a} : \mathsf{c} : \ldots$$

Stream Specification (flat, pure)

Example

$$\begin{array}{rcl|l}
\mathsf{Q} & \to & \mathsf{diff}(\mathsf{M}) & \\
\mathsf{M} & \to & 0 : \mathsf{zip}(\mathsf{inv}(\mathsf{M}), \mathsf{tail}(\mathsf{M})) & \textit{stream layer} \\
\hline
\mathsf{zip}(x : \sigma, \tau) & \to & x : \mathsf{zip}(\tau, \sigma) & \\
\mathsf{inv}(x : \sigma) & \to & \mathsf{i}(x) : \mathsf{inv}(\sigma) & \textit{function layer} \\
\mathsf{tail}(x : \sigma) & \to & \sigma & \\
\mathsf{diff}(x : y : \sigma) & \to & \mathsf{x}(x, y) : \mathsf{diff}(y : \sigma) & \\
\hline
\mathsf{i}(0) \to 1 & & \mathsf{i}(1) \to 0 & \\
\mathsf{x}(0,0) \to \mathsf{b} & & \mathsf{x}(0,1) \to \mathsf{a} & \textit{data layer} \\
\mathsf{x}(1,0) \to \mathsf{c} & & \mathsf{x}(1,1) \to \mathsf{b} &
\end{array}$$

… is productive and also specifies the ternary Thue-Morse sequence.

Stream Specification (flat, non-pure)

Example (Hamming numbers)

$$\mathsf{H} \to \underline{1} : \mathsf{merge}(\mathsf{times}(\mathsf{H}, \underline{2}), \mathsf{merge}(\mathsf{times}(\mathsf{H}, \underline{3}), \mathsf{times}(\mathsf{H}, \underline{5}))) \qquad \textit{stream layer}$$

$$\begin{array}{rcl}
\mathsf{times}(x : \sigma, y) & \to & \mathsf{m}(x, y) : \mathsf{times}(\sigma, y) \\
\mathsf{merge}(x : \sigma, y : \tau) & \to & \mathsf{aux}(\sigma, \tau, x, y, \mathsf{cmp}(x, y)) \\
\mathsf{aux}(\sigma, \tau, x, y, \mathsf{lt}) & \to & x : \mathsf{merge}(\sigma, y : \tau) \\
\mathsf{aux}(\sigma, \tau, x, y, \mathsf{eq}) & \to & x : \mathsf{merge}(\sigma, \tau) \\
\mathsf{aux}(\sigma, \tau, x, y, \mathsf{gt}) & \to & y : \mathsf{merge}(x : \sigma, \tau)
\end{array} \qquad \textit{function layer}$$

$$\begin{array}{rcl@{\qquad}rcl}
\mathsf{cmp}(0, 0) & \to & \mathsf{eq} & \mathsf{a}(0, y) & \to & y \\
\mathsf{cmp}(0, \mathsf{s}(y)) & \to & \mathsf{lt} & \mathsf{a}(\mathsf{s}(x), y) & \to & \mathsf{s}(\mathsf{a}(x, y)) \\
\mathsf{cmp}(\mathsf{s}(x), 0) & \to & \mathsf{gt} & \mathsf{m}(0, y) & \to & 0 \\
\mathsf{cmp}(\mathsf{s}(x), \mathsf{s}(y)) & \to & \mathsf{cmp}(x, y) & \mathsf{m}(\mathsf{s}(x), y) & \to & \mathsf{a}(y, \mathsf{m}(x, y))
\end{array} \qquad \textit{data layer}$$

Stream Specifications (Properties II)

A stream spec $\mathcal{T}$ is called

- friendly-nesting: all rules in the function layer R_{fun} are flat, or contained in a subset $\tilde{R} \subset R_{fun}$ of friendly-nesting rules: every $\rho \in \tilde{R}$
  - consumes in each stream argument at most one element,
  - produces at least one, and
  - all defining rules of stream functions occurring on the right-hand side of ρ are again in $\tilde{R}$.

Example

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

Stream Specification (friendly-nesting)

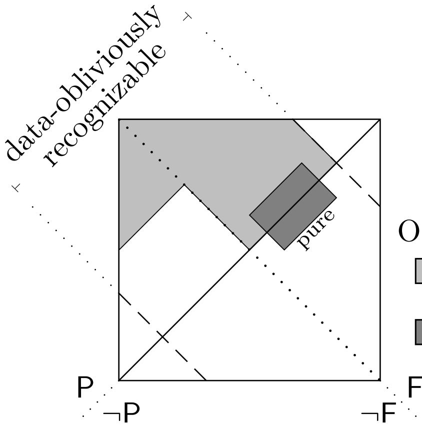
Example

$\mathsf{nats} \to 0 : \times(\mathsf{ones}, \mathsf{ones})$ $\mathsf{ones} \to \mathsf{s}(0) : \mathsf{ones}$	*stream layer*
$\times(x : \sigma, y : \tau) \to \mathsf{m}(x, y) : \mathsf{add}(\mathsf{times}(\tau, x), \times(\sigma, y : \tau))$ $\mathsf{times}(x : \sigma, y) \to \mathsf{m}(x, y) : \mathsf{times}(\sigma, y)$ $\mathsf{add}(x : \sigma, y : \tau) \to \mathsf{a}(x, y) : \mathsf{add}(\sigma, \tau)$	*function layer*
$\mathsf{a}(x, 0) \to x \quad \mathsf{a}(x, \mathsf{s}(y)) \to \mathsf{s}(\mathsf{a}(x, y))$ $\mathsf{m}(x, 0) \to 0 \quad \mathsf{m}(x, \mathsf{s}(y)) \to \mathsf{a}(\mathsf{m}(x, y), x)$	*data layer*

$\times$ defines the convolution product stream operation $\langle \sigma, \tau \rangle \mapsto \sigma \times \tau$:

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

Map of Stream Specifications



$\mathsf{P} =$ productive

$\mathsf{F} =$ flat

Our contribution:


= automated recognition


= decision

Data-Oblivious Analysis

	$\mathsf{T} \to \mathsf{f}(0 : 1 : \mathsf{T})$	*stream layer*
$(\rho_{\mathsf{f}0})$:	$\mathsf{f}(0 : x : \sigma) \to 0 : 1 : \mathsf{f}(\sigma)$	
$(\rho_{\mathsf{f}1})$:	$\mathsf{f}(1 : x : \sigma) \to x : \mathsf{f}(\sigma)$	*function layer*
		data layer

This specification is productive:

$$\mathsf{T} \twoheadrightarrow 0 : 1 : \mathsf{f}(\mathsf{T}) \twoheadrightarrow 0 : 1 : 0 : 1 : \mathsf{f}(\mathsf{f}(\mathsf{T})) \twoheadrightarrow \ldots \twoheadrightarrow\!\!\!\!\!\twoheadrightarrow 0 : 1 : 0 : 1 : \ldots \;,$$

but, disregarding the identity of data, the rewrite sequence:

$$\mathsf{T} \to \mathsf{f}(\bullet : \bullet : \mathsf{T}) \to^{\rho_{\mathsf{f}1}} \bullet : \mathsf{f}(\mathsf{T}) \twoheadrightarrow \ldots \twoheadrightarrow\!\!\!\!\!\twoheadrightarrow \bullet : \mathsf{f}(\bullet : \mathsf{f}(\bullet : \mathsf{f}(\ldots))) \;.$$

is possible. Hence the specification is not data-obliviously productive.

What is a data-oblivious analysis of productivity?

- 'quantitative reasoning'
- knowledge about concrete values of data elements is ignored
- abstract from the concrete data values

Example

$$\mathsf{f}(0 : x : \sigma) \to x : x : \mathsf{f}(0 : \sigma) \qquad \mathsf{f}(\bullet : \bullet : \sigma) \to \bullet : \bullet : \mathsf{f}(\bullet : \sigma)$$
$$\mathsf{f}(1 : x : \sigma) \to x : \mathsf{f}(0 : \sigma) \qquad \mathsf{f}(\bullet : \bullet : \sigma) \to \bullet : \mathsf{f}(\bullet : \sigma)$$

Taking into account that 0 is supplied to the recursive call:

- $n \mapsto 2n \dot{-} 3$ is the tight lower bound on the production relation

However, a data-oblivious analysis ignores this information:

- $n \mapsto n \dot{-} 1$ is the data-oblivious lower bound

We formalise data-oblivious term rewriting as two-player game:

- rewrite player $\mathcal{R}$ performs the usual term rewriting
- data-exchange player $\mathcal{G}$ arbitrarily exchanges data elements

Example ($\mathsf{f}(0 : x : \sigma) \to x : x : \mathsf{f}(0 : \sigma)$, $\mathsf{f}(1 : x : \sigma) \to x : \mathsf{f}(0 : \sigma)$)

Data-oblivious rewriting of the term $\mathsf{f}(0 : 1 : 0 : \sigma)$:

$$
\begin{tikzcd}
\mathsf{f}(0:1:0:\sigma) \arrow[r, dashed, "\mathcal{G}"] & \mathsf{f}(1:0:0:\sigma) \arrow[dl, "\mathcal{R}"'] \\
0:\mathsf{f}(0:0:\sigma) \arrow[r, dashed, "\mathcal{G}"] & 0:\mathsf{f}(1:1:\sigma) \arrow[dl, "\mathcal{R}"'] \\
0:1:\mathsf{f}(0:\sigma) \arrow[r, dashed, "\mathcal{G}"] & 0:1:\mathsf{f}(0:\sigma)
\end{tikzcd}
$$

Definition (data-oblivious lower (upper) bound on the production)

...of a term s is the infimum (supremum) of the production of s with respect to all possible strategies for the data-exchange player $\mathcal{G}$.

It is sufficient to use ‘data-value-invariant‘ strategies for $\mathcal{G}$:

- for $\mathcal{G}$ we can abstract from the data elements in favour of $\bullet$

Definition (The data abstraction $[\![s]\!]$ of a term s)

… is obtained from s by replacing all data terms in s with $\bullet$.

Example

$$\begin{tikzcd}
\mathsf{f}(\bullet:\bullet:\bullet:\sigma) \arrow[r, dashed, "\mathcal{G}"] & \mathsf{f}(1:0:0:\sigma) \arrow[dl, "{[\![\cdot]\!]\circ\mathcal{R}}"'] \\
\bullet:\mathsf{f}(\bullet:\bullet:\sigma) \arrow[r, dashed, "\mathcal{G}"] & 0:\mathsf{f}(1:1:\sigma) \arrow[dl, "{[\![\cdot]\!]\circ\mathcal{R}}"'] \\
\bullet:\bullet:\mathsf{f}(\bullet:\sigma) \arrow[r, dashed, "\mathcal{G}"] & 0:1:\mathsf{f}(0:\sigma)
\end{tikzcd}$$

Definition (The data-guess function $\mathcal{G}$)

…instantiates all $\bullet$ by closed data terms in constructor normal form.

Definition

The data-oblivious production range of a term $s \in Ter(\Sigma)_S$ is:

$$\overline{\underline{do}}_{\mathcal{T}}(s) := \{\Pi_{\mathcal{G}}([\![s]\!]) \mid \mathcal{G} \text{ a data-guess function on } \mathcal{T}\} \,.$$

The data-oblivious lower and upper bound on the production of s:

$$\underline{do}_{\mathcal{T}}(s) := \inf(\overline{\underline{do}}_{\mathcal{T}}(s))$$
$$\overline{do}_{\mathcal{T}}(s) := \sup(\overline{\underline{do}}_{\mathcal{T}}(s))$$

Definition

An stream specification $\mathcal{T}$ is called

- data-obliviously productive if $\underline{do}_{\mathcal{T}}(\mathsf{M}_0) = \infty$
- data-obliviously non-productive if $\overline{do}_{\mathcal{T}}(\mathsf{M}_0) < \infty$

Data-Oblivious Productivity versus Productivity

Proposition

Let $\mathcal{T} = \langle \Sigma, R \rangle$ be a stream specification.

- *For all stream terms $s \in Ter(\Sigma)_S$, we have*

$$\underline{do}_{\mathcal{T}}(s) \leq \Pi_{\mathcal{T}}(s) \leq \overline{do}_{\mathcal{T}}(s) \ .$$

Hence:

- *data-oblivious productivity implies productivity;*
- *data-oblivious non-productivity implies non-productivity.*

Periodically Increasing Functions

Definition

Let $f : \mathbb{N} \to \overline{\mathbb{N}}$.
f is eventually periodic if $\langle f(0), f(1), f(2), ...\rangle$ is eventually periodic.
f is periodically increasing if it is non-decreasing, and its derivative $f' : \mathbb{N} \to \overline{\mathbb{N}}$ with $f'(n) = f(n+1) - f(n)$ is eventually periodic.

- Representation: pairs $\langle \alpha, \beta \rangle \in \mathcal{I}$ with $\alpha \in \{-,+\}^*$, $\beta \in \{-,+\}^+$ where $+$ stands for output, $-$ for input.
- production function $\pi_{\langle \alpha, \beta \rangle}$ for $\langle \alpha, \beta \rangle \in \mathcal{I}$: if $s = \alpha\beta\beta\beta \ldots$ then

$$\pi_{\langle \alpha, \beta \rangle}(n) := \begin{cases} \text{number of "$+$" from left until the $(n+1)$-th "$-$" in } I \\ \infty \quad \ldots \text{ if there are less than } n+1 \text{ symbols in } I \end{cases}$$

- Abbreviation: $\alpha\overline{\beta}$ for $\langle \alpha, \beta \rangle$.
  Example: identity function is represented by $\overline{-+}$.

Production Terms

Definition

For $\mathcal{V}$ a set of recursion variables, the set $\mathcal{P}$ of production terms is generated by:

$$p ::= \underline{k} \mid x \mid \sigma(p) \mid \mu x.p \mid \min(p, p)$$

where $x \in \mathcal{V}$, $\sigma \in \mathcal{I}$, and $\underline{k}$ is a numeral for $k \in \overline{\mathbb{N}}$.

The production $\Pi(p) \in \overline{\mathbb{N}}$ of a closed production term $p \in \mathcal{P}$ is defined by induction on the term structure, interpreting:

- μ as the least fixed point operator,
- σ as π_σ,
- $\underline{k}$ as k, and
- min as min.

r-ary Gates: production term contexts $\min_r(\sigma_1(\Box_1), \ldots, \sigma_r(\Box_r))$.

Reduction $\to_{\mathsf{R}}$

Definition

The reduction relation $\to_{\mathsf{R}}$ on production terms is defined as the compatible closure of:

$$
\begin{aligned}
\sigma_1(\sigma_2(p)) &\to \sigma_1 \circ \sigma_2(p) \\
\sigma(\min(p_1, p_2)) &\to \min(\sigma(p_1), \sigma(p_2)) \\
\mu x.\min(p_1, p_2) &\to \min(\mu x.p_1, \mu x.p_2) \\
\mu x.p &\to p \qquad \text{if } x \notin \mathrm{FV}(p) \\
\mu x.\sigma(x) &\to \underline{\mathrm{fix}(\sigma)} \\
\mu x.x &\to \underline{0} \\
\sigma(\underline{k}) &\to \underline{\pi_\sigma(k)} \\
\min(\underline{k_1}, \underline{k_2}) &\to \underline{\min(k_1, k_2)}
\end{aligned}
$$

Reduction $\to_{\mathsf{R}}$

Properties of $\to_{\mathsf{R}}$:

- production preserving;
- confluent and terminating;
- normal forms are numerals.

Theorem

For all $p \in \mathcal{P}$:

$$\Pi(p) = k \ ,$$

where $\underline{k}$ is the uniquely determined $\to_{\mathsf{R}}$-normal form of p.

Translation into Production Terms

We use a translation that maps every flat stream spec $\mathcal{T}$ with root M_0 to a production term $[\mathsf{M}_0]$ such that

$$\underline{do}_{\mathcal{T}}(\mathsf{M}_0) = \Pi([\mathsf{M}_0]) \; .$$

- function layer translation: Obtain a family $\{[\mathsf{f}]\}_{\mathsf{f} \in \Sigma_{fun}}$ of gates such that, for every $\mathsf{f} \in \Sigma_{fun}$, the gate γ_{f} represents the data-oblivious lower bound of f in $\mathcal{T}$.

  (Involves solving an originally infinite 'io-term specification'.)

- stream layer translation: Using the family $\mathsf{f} \in \Sigma_{fun}$ of gates, obtain a production term $[\mathsf{M}_0]^{\mathcal{F}}$ such that $\Pi([\mathsf{M}_0]^{\mathcal{F}}) = \underline{do}_{\mathcal{T}}(\mathsf{M}_0)$.

  (Involves expanding the stream layer rules step by step and a finite loop-checking procedure.)

Algorithm DOP: Deciding Data-Oblivious Productivity

1. Take as input: a flat stream specification $\mathcal{T} = \langle \Sigma, R \rangle$.
2. Compute the translation of stream function symbols f into gates [f], yielding a family $\mathcal{F} := \{[\mathsf{f}]\}_{\mathsf{f} \in \Sigma_{fun}}$ of gates.
3. Construct the production term $[\mathsf{M}_0]^{\mathcal{F}}$ of the root M_0 of $\mathcal{T}$ with respect to the family of gates $\mathcal{F}$.
4. Compute the production $\underline{k}$ of $[\mathsf{M}_0]^{\mathcal{F}}$ using the reduction rel. $\to_{\mathsf{R}}$.
5. Give the following output:
   If $k = \infty$: "$\mathcal{T}$ `is data-obliviously productive`";
   else $k \in \mathbb{N}$: "$\mathcal{T}$ `is not data-obliviously productive`".

Deciding D-O Productivity. Recognising Productivity.

Theorem

Data-oblivious productivity of flat stream specifications is decidable. I.p., the algorithm `DOP` *decides data-oblivious productivity of flat stream specifications.*

Since data-oblivious productivity implies productivity, we get a computable, data-obliviously optimal criterion for productivity:

Corollary

A flat stream specification $\mathcal{T}$ is productive if the algorithm `DOP` *recognizes $\mathcal{T}$ as data-obliviously productive.*

Recognising and Deciding Productivity.

For pure stream spec's productivity coincides with data-oblivious productivity. Hence DOP gives rise to a decision algorithm.

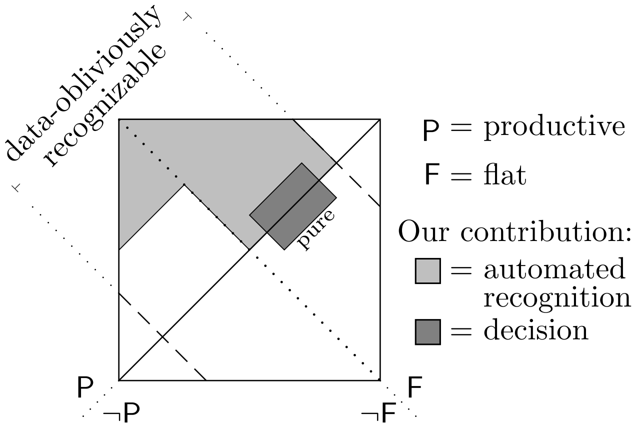
Theorem

Productivity is decidable for pure stream spec's.

Furthermore, a variant of DOP can be used as a:

- computable criterion of productivity for friendly-nesting stream spec's.

Map of Stream Specifications



Summary

- Previous Approaches: sufficient conditions for productivity, not automatable or only for a limited subclass
- FCT'07-Paper: decision algorithm for productivity of pure stream spec's
- New Results:

  1. a computable, data-obliviously optimal, sufficient condition for productivity of flat stream spec's;
  2. a decision method for productivity on pure stream spec's with duplication/additional supply in stream arg's (extension of FCT);
  3. an extension of 1 to stream spec's with friendly nesting, disregarding data-oblivious optimality;
  4. a tool automating 1, 2 and 3 available at: `http://infinity.few.vu.nl/productivity`.

Present and Future Work

- A precise complexity analysis of our algorithms.
- Data-aware methods for recognising productivity.
- A refined pebbleflow semantics that accounts for the delay of evaluation of stream elements as made possible by lazy evaluation strategies. Think of Sijtsma's example: $\mathsf{S} \to \mathsf{0} : \mathsf{head}(\mathsf{tail}^2(\mathsf{S})) : \mathsf{S}$.
- A theory of reducibility between streams.
- Can our results be used to obtained general results clarifying under which conditions term graph rewriting can be viewed as a semantics for infinitary rewriting?

Our Papers and Tools.

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

- Endrullis, Grabmayer, Hendriks, Isihara, Klop:
  Productivity of Stream Definitions, Proceedings of FCT 2007, LNCS 4637, pages 274–287, 2007;
- Endrullis, Grabmayer, Hendriks, Isihara, Klop:
  Productivity of Stream Definitions, journal submission;
- Endrullis, Grabmayer, Hendriks:
  Data-Oblivious Stream Productivity, extended abstract;
- access to our tools:
  - Endrullis: tool implementing the
    decision algorithm for data-oblivious productivity;
  - Isihara: pebbleflow visualization tool.

Thanks for your attention!